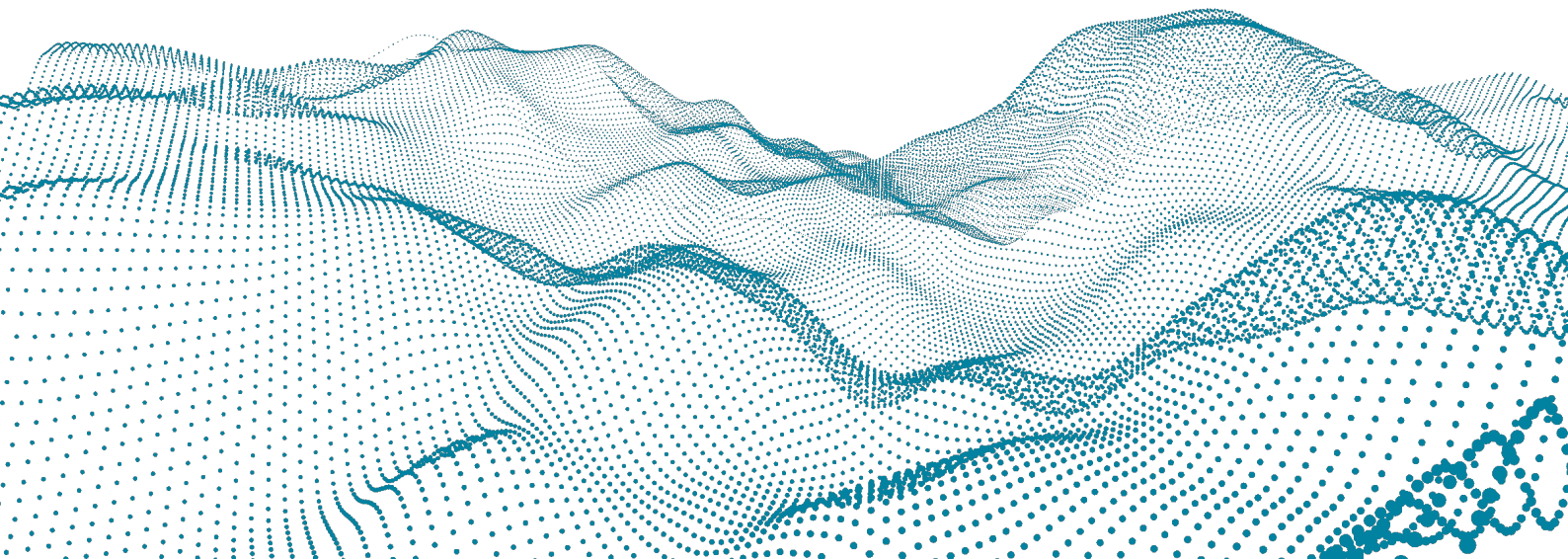
roboception

Roboception GmbH | Januar 2026

rc_reason_stack

3D Vision Software Platform

INSTALLATIONS- UND BEDIENUNGSANLEITUNG



Revisionen

Dieses Produkt kann bei Bedarf jederzeit ohne Vorankündigung geändert werden, um es zu verbessern, zu optimieren oder an eine überarbeitete Spezifikation anzupassen. Werden solche Änderungen vorgenommen, wird auch das vorliegende Handbuch überarbeitet. Beachten Sie die angegebene Versionsnummer.

DOKUMENTATIONSVERSION 26.01.4 30.01.2026

Gültig für *rc_reason_stack* Firmware 26.01.x

HERSTELLER

Roboception GmbH

Kaflerstraße 2

81241 München

Deutschland

KUNDENSUPPORT: support@roboception.de | +49 89 889 50 79-0 (09:00-17:00 CET)

Betriebsanleitung bitte vollständig lesen und produktnah aufbewahren.

Inhaltsverzeichnis

1 Einführung

Hinweise im Handbuch

Um Schäden an der Ausrüstung zu vermeiden und die Sicherheit der Benutzer zu gewährleisten, enthält das vorliegende Handbuch Sicherheitshinweise, die mit dem Symbol *Warnung* gekennzeichnet werden. Zusätzliche Informationen sind als *Bemerkung* gekennzeichnet.

Warnung: Die mit *Warnung* gekennzeichneten Sicherheitshinweise geben Verfahren und Maßnahmen an, die befolgt bzw. ergriffen werden müssen, um Verletzungsgefahren für Bediener/Benutzer oder Schäden am Gerät zu vermeiden. Beziehen sich die angegebenen Sicherheitshinweise auf Softwaremodule, dann weisen diese auf Verfahren hin, die befolgt werden müssen, um Störungen oder ein Fehlverhalten der Software zu vermeiden.

Bemerkung: Bemerkungen werden in diesem Handbuch eingesetzt, um zusätzliche relevante Informationen zu vermitteln.

1.1 Überblick

Der *rc_reason_stack* ist ein Docker Software Stack zur performanten 3D-Bildverarbeitung. Er erweitert die Rechenkapazitäten der Roboception Stereokamera *rc_visard* und unterstützt den *rc_viscore*, die Basler *Stereo ace*, die *Orbbec* Kamera und die *zivid* Kamera.

Der *rc_reason_stack* stellt Echtzeit-Kamerabilder und Tiefenbilder bereit, die zur Berechnung von 3D-Punktwolken verwendet werden können. Zudem erstellt er Konfidenz- und Fehlerbilder, mit denen sich die Qualität der Bilderfassung messen lässt. Dank der standardisierten Schnittstellen ist er mit allen großen Bildverarbeitungsbibliotheken kompatibel und bietet darüber hinaus eine intuitive, web-basierte Bedienoberfläche an.

Mit optional erhältlichen Softwaremodulen bietet der *rc_reason_stack* Standardlösungen für Anwendungen in der Objekterkennung oder für robotische Pick-and-Place-Applikationen.

Bemerkung: Das vorliegende Handbuch nutzt das metrische System und verwendet vorrangig die Maßeinheiten Meter und Millimeter. Sofern nicht anders angegeben, sind Abmessungen in technischen Zeichnungen in Millimetern angegeben.

2 Sicherheit

Warnung: Vor Inbetriebnahme des *rc_reason_stack*-Produkts muss der Bediener alle Anweisungen in diesem Handbuch gelesen und verstanden haben.

Warnung: Wird der *rc_reason_stack* zusammen mit *rc_visard*-Produkten betrieben, muss der Bediener alle Anweisungen zur Sicherheit, Inbetriebnahme und Wartung im *rc_visard*-Bedienhandbuch gelesen und verstanden haben.

Bemerkung: Der Begriff „Bediener“ bezieht sich auf jede Person, die in Verbindung mit dem *rc_reason_stack* mit einer der folgenden Aufgaben betraut ist:

- Installation
- Wartung
- Inspektion
- Kalibrierung
- Programmierung
- Außerbetriebnahme

Das vorliegende Handbuch geht auf die verschiedenen Softwaremodule des *rc_reason_stack* ein und erläutert allgemeine Aspekte zum Lebenszyklus des Produkts: von der Installation über die Verwendung bis hin zur Außerbetriebnahme.

Die im vorliegenden Handbuch enthaltenen Zeichnungen und Fotos sind Beispiele zur Veranschaulichung. Das ausgelieferte Produkt kann hiervon abweichen.

2.1 Allgemeine Sicherheitshinweise

Bemerkung: Wird der *rc_reason_stack* entgegen den hierin angegebenen Sicherheitshinweisen verwendet, so kann dies zu Personen- oder Sachschäden sowie zum Verlust der Garantie führen.

2.2 Bestimmungsgemäße Verwendung

Warnung: Der *rc_reason_stack* ist **NICHT** für sicherheitskritische Anwendungen bestimmt.

Der vom *rc_reason_stack* verwendete Schnittstellenstandard GigE Vision® unterstützt weder Authentifizierung noch Verschlüsselung. Alle von diesem und an dieses Gerät gesandten Daten werden ohne Authentifizierung und Verschlüsselung übermittelt und könnten daher von einem Dritten abgefangen

oder manipuliert werden. Es liegt in der Verantwortung des Bedieners, den *rc_reason_stack* nur an ein gesichertes internes Netzwerk anzuschließen.

Warnung: Der *rc_reason_stack* muss an gesicherte interne Netzwerke angeschlossen werden.

Der *rc_reason_stack* darf nur im Rahmen seiner technischen Spezifikation verwendet werden. Jede andere Verwendung des Produkts gilt als nicht bestimmungsgemäße Verwendung. Roboception haftet nicht für Schäden, die aus unsachgemäßer oder nicht bestimmungsgemäßer Verwendung entstehen.

Warnung: Die lokalen und/oder nationalen Gesetze, Vorschriften und Richtlinien zu Automationssicherheit und allgemeiner Maschinensicherheit sind stets einzuhalten.

3 Installation

Warnung: Vor Installation müssen die Hinweise zur *Sicherheit* (Abschnitt 2) des *rc_reason_stack* gelesen und verstanden werden.

Der *rc_reason_stack* ist ein Docker-basierter Software-Stack, der auf Rechnern installiert werden kann, die die unter *Voraussetzungen* genannten Systemvoraussetzungen erfüllen. Dieses Kapitel enthält detaillierte Informationen zur Installation der *rc_reason_stack*-Software.

3.1 Offline Installationsanleitung

Dieser Abschnitt beschreibt die manuelle Installation des *rc_reason_stack* auf einem Hostsystem. Im Gegensatz zum automatisierten Docker-Compose-Workflow werden die Docker-Images zunächst auf den Hostrechner kopiert und anschließend manuell in Docker geladen. Führen Sie die folgenden Schritte aus, um den Stack für Ihre Entwicklungs- oder Produktionsumgebung einzurichten und auszuführen.

Alle Befehle müssen auf dem Host-Rechner ausgeführt werden (nicht innerhalb eines Containers).

3.1.1 Voraussetzungen

Komponente	Minimalversion
Ubuntu	24.04 LTS
NVIDIA GPU	Jede RTX mit mindestens 8GB VRAM, oder besser **[1]**
Docker	20.10+
NVIDIA Driver	535+ (diese Anleitung nutzt *nvidia-driver-575-server*)

[1] Getestet mit Nvidia RTX A4000, RTX 4000 Ada, RTX 3080, RTX 4070, RTX 4080

Die folgenden Dateien werden von Roboception bereitgestellt und werden für die Installation benötigt.

Datei	Beschreibung
rc_container-xx.yy.zz.tar	rc_container docker image
tritonserver-xx.yy.tar	Tritonserver Docker Image
docker-compose.yml	Die Docker-Compose-Datei
docker-compose.json	Die Docker-Compose-Datei im JSON-Format

xx.yy.zz steht für die gewünschten Versionen von rc_container und tritonserver.

3.1.2 Installation von Ubuntu 24.04

Dieser Abschnitt kann übersprungen werden, wenn eine funktionierende Ubuntu 24.04-Installation vorhanden ist.

Folgen Sie zur Installation von Ubuntu der offiziellen Ubuntu-Installationsanleitung unter https://ubuntu.com/download/desktop oder https://ubuntu.com/download/server.

3.1.3 NVIDIA-Treiberinstallation

Der NVIDIA-Treiber ist erforderlich, damit der Host die GPU für Docker-Container freigeben kann. Nach der Installation des Treibers sollten die GPU und ihre Funktionen mit `nvidia-smi` sichtbar sein. Ist der Treiber nicht installiert oder nicht korrekt geladen, findet `nvidia-smi` die GPU entweder nicht oder meldet „No devices were found“.

```
# Update package lists
sudo apt update

# run nvidia-detector to find the correct driver
sudo nvidia-detector

# Install the latest NVIDIA driver (replace 570 with the version that matches your GPU)
sudo apt install -y nvidia-driver-570-server

# Reboot to load the driver
sudo reboot
```

Überprüfen Sie nach dem Neustart, ob der Treiber aktiv ist:

```
$ nvidia-smi
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 570.195.03             Driver Version: 570.195.03     CUDA Version: 12.8     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA RTX A4000               Off |   00000000:06:00.0 Off |                  Off |
| 41%   60C    P0             37W /  140W |   10719MiB /  16376MiB |      9%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|...
+-----------------------------------------------------------------------------------------+
```

Die Tabelle zeigt:

- **GPU**: die Geräte-ID (0, 1, . . .)
- **Name**: das GPU Modell (z.B., GeForce RTX 3080)
- **Driver Version**: der installierte NVIDIA-Treiber
- **CUDA Version**: das CUDA Toolkit, das mit dem Treiber ausgeliefert wurde
- **Memory-Usage**: dem Grafikprozessor zugewiesener Gesamtspeicher
- **GPU-Util**: aktueller Prozentsatz der GPU-Auslastung

Wenn diese Ausgabe angezeigt wird, ist der Treiber korrekt installiert und die GPU kann vom NVIDIA Container Toolkit und den Containern verwendet werden.

3.1.4 Docker-Installation

```
# Update the apt package index and install packages to allow apt to use a repository over HTTPS
sudo apt-get update
sudo apt-get install \
   ca-certificates \
   curl \
   gnupg \
   lsb-release

# Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/
↪keyrings/docker-archive-keyring.gpg

# Set up the stable repository
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]␣
↪https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker Engine
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli docker-compose-plugin containerd.io

# Verify Docker installation
sudo docker --version
```

3.1.5 Installation des NVIDIA Container Toolkit

Das NVIDIA Container Toolkit ermöglicht Docker, NVIDIA-GPUs innerhalb von Containern zu erkennen, bereitzustellen und in einer Sandbox auszuführen. Ohne dieses Toolkit können CUDA-Workloads nicht im Container ausgeführt werden. Es bildet die Brücke zwischen der Docker-Container-Laufzeitumgebung und dem NVIDIA-Treiberstack auf dem Hostsystem.

```
# Add the NVIDIA GPG key
sudo curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | \
   sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg

# Add the NVIDIA Container Toolkit repository
sudo curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.
↪list | \
   sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg]␣
↪https://#g' | \
   sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list

# Update package lists and install
sudo apt update && sudo apt install -y nvidia-container-toolkit

# Restart Docker to apply changes
sudo systemctl restart docker

# Modify /etc/docker/daemon.json. This is necessary for older docker versions
sudo nvidia-ctk runtime configure --runtime=docker

# Verify that nvidia is now available as docker runtime
docker info | grep -i runtime
```

Um ein Problem zu beheben, das dazu führt, dass die GPU nach einiger Zeit im Container ausfällt (erkennbar an einem Fehler von `nvidia-smi` im Container), öffnen Sie die Datei `/etc/nvidia-container-runtime/config.toml` und setzen Sie `no-cgroups = false`. Starten Sie Docker nach der Konfigurationsänderung mit folgendem Befehl:

```
sudo systemctl restart docker
```

Testen Sie, ob Docker auf die GPU zugreifen kann:

```
sudo docker run --rm --gpus all nvidia/cuda:12.1.1-base-ubuntu22.04 nvidia-smi
```

3.1.6 Begrenzung der Docker Logdateigröße

Docker verwendet standardmäßig den JSON-Datei-Logging-Treiber ohne Beschränkung der Logdateigröße. Wir empfehlen, auf den lokalen Logging-Treiber (https://docs.docker.com/engine/logging/drivers/local/) umzusteigen, der die maximale Logdateigröße standardmäßig begrenzt.

Dazu muss in */etc/docker/daemon.json* folgendes hinzugefügt werden:

```
{
"log-driver": "local",
}
```

3.1.7 Installation der WIBU CodeMeter Runtime

Installieren Sie die CodeMeter Runtime (https://www.wibu.com/de/support/anwendersoftware/anwendersoftware.html) auf dem Hostsystem.

Nach der Installation, stoppen Sie die Runtime:

```
sudo service codemeter stop
```

Aktivieren Sie die Netzwerklizenzierung, indem Sie `IsNetworkServer` in der Datei `/etc/wibu/CodeMeter/Server.ini` auf 1 setzen.

Starten Sie die Runtime:

```
sudo service codemeter start
```

Um zu verhindern, dass der WIBU-Netzwerklizenzserver im externen Netzwerk (WIBU verwendet die Ports 22350-22352) sichtbar ist, kann eine Firewall verwendet werden, da der Lizenzserver nur für den Docker-Container sichtbar sein darf.

3.1.8 Erstellen von Netzwerkinterfaces

Dieses Beispiel zeigt eine Netzwerkkonfiguration mit einem separatem Ethernet-Port (enp9s0) für die `sensor0`-Schnittstelle über macvlan. Passen Sie den Namen des Ethernet-Ports entsprechend an; in diesem Beispiel wird Port enp9s0 verwendet. Für mehrere Kameras (z.B. den *rc_viscore* oder mehrere |rc_visard|s) muss ein separates Netzwerk für jede Kamera erstellt werden.

Erstellen Sie `/etc/netplan/40-sensor0.yaml` mit dem Inhalt

```
network:
version: 2
renderer: networkd
ethernets:
   enp9s0:
      dhcp4: false
      dhcp6: false
      addresses:
      - 172.23.42.1/28
```

Eine alternative Netzwerkkonfiguration für mehrere Kameras wie folgt erfolgen. In diesem Beispiel werden die Schnittstellen *enp7s0* für einen *rc_visard* und die Schnittstellen *ens4f0* und *ens4f1* für einen *rc_viscore* verwendet.

```
network:
   version: 2
   renderer: networkd
   ethernets:
      enp7s0:
         dhcp4: false
         dhcp6: false
         addresses:
         - 172.23.42.1/28
      ens4f0:
         dhcp4: false
         dhcp6: false
         addresses:
         - 172.23.43.1/28
      ens4f1:
         dhcp4: false
         dhcp6: false
         addresses:
         - 172.23.44.1/28
```

Berechtigungen ändern und anwenden mit

```
sudo chmod 600 /etc/netplan/40-sensor0.yaml
sudo netplan apply
```

Docker-Netzwerk mit dem macvlan-Treiber erstellen:

```
sudo docker network create -d macvlan --subnet=172.23.42.0/28 --gateway=172.23.42.1 --ip-
↪range=172.23.42.8/29 -o parent=enp9s0 sensor0
# or for multiple interfaces
sudo docker network create -d macvlan --subnet=172.23.42.0/28 --gateway=172.23.42.1 --ip-
↪range=172.23.42.8/29 -o parent=enp7s0 sensor0
sudo docker network create -d macvlan --subnet=172.23.43.0/28 --gateway=172.23.43.1 --ip-
↪range=172.23.43.8/29 -o parent=ens4f0 sensor1
sudo docker network create -d macvlan --subnet=172.23.44.0/28 --gateway=172.23.44.1 --ip-
↪range=172.23.44.8/29 -o parent=ens4f1 sensor2
```

Überprüfen Sie dies in der docker-compose.yml (oder docker-compose.json):

```
# docker-compose.yml with multiple sensor interfaces
#... config truncated - for readability
networks:
back-tier:
   driver: bridge
sensor0:
   external: true
   name: sensor0
#... config truncated - for readability
services:
rc-container:
#... config truncated - for readability
   networks:
      - back-tier
      - sensor0
#... config truncated - for readability
```

Die docker-compose.yml für mehrere Schnittstellen lautet wie folgt:

3.1.9 Netzwerkeinstellungen für GigE Vision sicherstellen

GigE-Vision-Kameras streamen Bilder mit hoher Bandbreite über UDP-Pakete. Paketverluste führen zu Bildausfällen und beeinträchtigen die Anwendungsleistung. Um dies zu vermeiden, sollten die Ethernet-Lesepuffer auf dem Host erhöht werden. Erstellen Sie unter Ubuntu die Datei `/etc/sysctl.d/10-gev-perf.conf` mit folgendem Inhalt:

```
# Increase readbuffer size for GigE Vision
net.core.rmem_max=33554432
```

Übernehmen Sie die Einstellungen mit

```
sudo sysctl -p /etc/sysctl.d/10-gev-perf.conf
```

3.1.10 Laden der Container-Images

```
# replace xx.yy.zz with the desired rc_container and tritonserver version
gunzip -c ./rc_container-xx.yy.zz.tar.gz | docker load
gunzip -c ./tritonserver-xx.yy.tar.gz | docker load
```

3.1.11 Starten des Docker-Stacks

Die bevorzugte Art, den Docker-Compose Stack zu starten, ist

```
cd /path/to/rc_container/
# use docker-compose.yml
docker compose up -d --pull never
```

Falls das Hostsystem eine Docker-Compose Datei im JSON-Format benötigt, kann der folgende Befehl genutzt werden.

```
cd /path/to/rc_container/
# use docker-compose.json
docker compose -f docker-compose.json up -d --pull never
```

Warten Sie einige Minuten, bis alle Container gestartet sind. Der Status kann wie folgt überwacht werden:

```
docker compose ps
```

3.1.12 Zugriff auf die Web GUI

Sobald der Stack läuft, kann die Web GUI wie folgt aufgerufen werden:

```
http://<host-ip>:8080/
```

3.1.13 Fehlerbehebung

Symptom	Wahrscheinliche Ursache	Fehlerbehebung
docker: Fehler: Der Treiber nvidia unterstützt das angeforderte Gerät nicht.	NVIDIA-Treiber- / Docker-Integrationskonflikt	Führen Sie die Installation des NVIDIA Container Toolkits erneut aus und starten Sie den Computer neu.
Container starten nicht	Falscher Netzwerkname	Stellen Sie sicher, dass ein Docker-Netzwerk mit dem Namen `sensor0` existiert
Web GUI nicht erreichbar	Container laufen nicht	`docker compose logs` um Fehler zu untersuchen
Sehr niedrige Bildwiederholfrequenz	GPU funktioniert nicht im Container	Überprüfen, indem `nvidia-smi` auf dem Host und innerhalb des Containers ausgeführt wird, und Probleme beheben [2].

[2] Falls `nvidia-smi` auf dem Host fehlschlägt, stellen Sie sicher, dass die Pakete konsistent sind, da ein unbeaufsichtigtes Upgrade unter Ubuntu möglicherweise den Nvidia-Treiber, aber nicht das Nvidia-Toolkit aktualisiert. Dies lässt sich beheben, indem Sie manuell `sudo apt update && sudo apt upgrade` ausführen. Unbeaufsichtigte Upgrades können deaktiviert werden. Falls `nvidia-smi` im Container fehlschlägt, stellen Sie sicher, dass `no-cgroups = false` in `/etc/nvidia-container-runtime/config.toml` gesetzt ist, und starten Sie Docker neu, falls die Konfiguration geändert werden musste. Diese Konfigurationsdatei wurde möglicherweise durch ein Update des Nvidia-Container-Toolkits überschrieben.

3.2 Softwarelizenz

Jeder *rc_reason_stack* wird mit einem USB-Dongle zur Lizenzierung und zum Schutz der installierten Softwarepakete ausgeliefert. Die erworbenen Lizenzen sind auf diesem Dongle installiert und somit an ihn und seine ID gebunden.

Die Funktionalität des *rc_reason_stack* kann jederzeit durch ein *Upgrade der Lizenz* (Abschnitt 8.2) erweitert werden – zum Beispiel für zusätzlich erhältliche, optionale Softwaremodule.

Bemerkung: Der *rc_reason_stack* muss neu gestartet werden, sobald die Softwarelizenz geändert wurde.

Bemerkung: Der Status der Softwarelizenz kann über die verschiedenen Schnittstellen des *rc_reason_stack* abgefragt werden, zum Beispiel über die Seite *System* → *Firmware & Lizenz* in der *Web GUI* (Abschnitt 7.1).

Bemerkung: Damit die Lizenzierung der Softwaremodule ordnungsgemäß funktioniert, muss der USB-Dongle an den *rc_reason_stack* angesteckt werden, bevor dieser gestartet wird.

Bemerkung: Der *rc_reason_stack* muss neu gestartet werden, sobald der Dongle eingesteckt oder abgezogen wurde.

3.3 Anschluss von Kameras

Der *rc_reason_stack* bietet bis zu vier Software-Kamerapipelines für die Prozessierung der Daten der angeschlossenen Sensoren. Die Konfiguration der Kamerapipelines wird in *Kamerapipelines* (siehe Abschnitt 4.2.3) beschrieben.

4 Messprinzipien

Der *rc_reason_stack* ist ein performanter 3D-Bildverarbeitungs-Software-Stack, der zusammen mit einer oder mehreren 3D-Kameras wie dem *rc_visard* oder *rc_viscore* von Roboception betrieben wird. Gemeinsam erstellen und verarbeiten sie rektifizierte Bilder, sowie Disparitäts-, Konfidenz- und Fehlerbilder, mit denen sich die Tiefenwerte der Aufnahme berechnen lassen.

Im Folgenden sind die zugrunde liegenden Messprinzipien genauer dargestellt.

4.1 Stereovision

Bei der *Stereovision* werden 3D-Informationen gewonnen, indem zwei aus verschiedenen Blickwinkeln aufgenommene Bilder miteinander verglichen werden. Das zugrunde liegende Prinzip ist darin begründet, dass Objektpunkte je nach Abstand vom Kamerapaar an unterschiedlichen Stellen in beiden Kameras erscheinen. Während sehr weit entfernte Objektpunkte in beiden Kamerabildern etwa an der gleichen Position erscheinen, liegen sehr nahe Objektpunkte an unterschiedlichen Stellen im linken und rechten Kamerabild. Dieser Versatz der Objektpunkte in beiden Kamerabildern wird auch „Disparität“ genannt. Je größer die Disparität, desto näher ist das Objekt der Kamera. Das Prinzip der Stereovision wird in Abb. 4.1 genauer dargestellt.

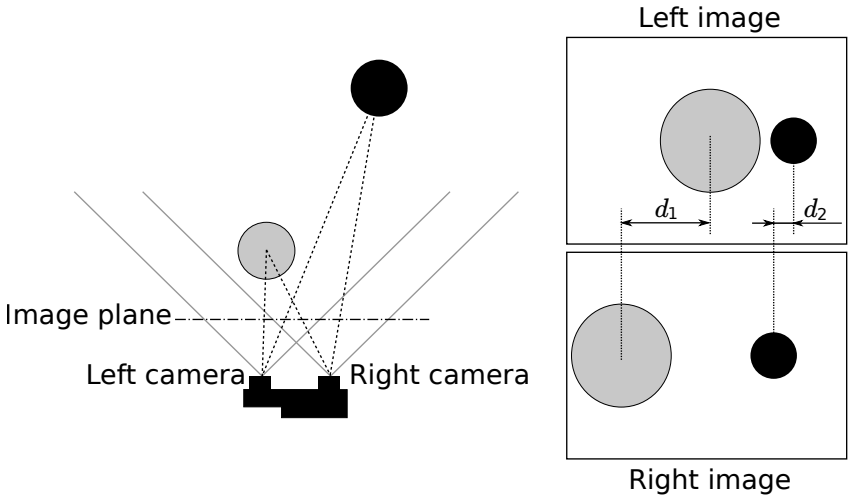


Abb. 4.1: Schematische Darstellung des Prinzips der Stereovision: Die Disparität d_2 des weiter entfernten (schwarzen) Objekts ist kleiner als die Disparität d_1 des nahe liegenden (grauen) Objekts.

Stereovision beruht auf passiver Wahrnehmung. Dies bedeutet, dass keine Licht- oder sonstigen Signale zur Distanzmessung ausgesandt werden, sondern nur das von der Umgebung ausgehende oder reflektierte Licht genutzt wird. Dadurch können die Roboception-Sensoren sowohl im Innen- als auch im Außenbereich eingesetzt werden. Zudem können problemlos mehrere Sensoren störungsfrei zusammen auf engem Raum betrieben werden.

Um die 3D-Informationen berechnen zu können, muss der Stereo-Matching-Algorithmus die zusammengehörenden Objektpunkte im linken und rechten Kamerabild finden. Hierfür bedient er sich der Bildtextur, d.h. der durch Muster oder Oberflächenstrukturen der Objekte verursachten Schwankungen in der Bildintensität. Das Stereo-Matching-Verfahren kann bei Oberflächen ohne jede Textur, wie z.B. bei glatten, weißen Wänden, keine Werte liefern. Das Stereo-Matching-Verfahren, das der *rc_reason_stack* verwendet, ist *SGM* (*Semi-Global Matching*), welches selbst bei feineren Strukturen den bestmöglichen Kompromiss aus Laufzeit und Genauigkeit bietet.

Für die Berechnung der 3D-Informationen werden folgende Softwaremodule benötigt:

- *Kamera Modul*: Dieses Modul dient dazu, synchronisierte Bildpaare aufzunehmen und diese in Bilder umzuwandeln, die weitestgehend den Aufnahmen einer idealen Kamera entsprechen (Rektifizierung).
- *Stereo-Matching Modul*: Dieses Modul errechnet mithilfe des Stereo-Matching-Verfahrens *SGM* die Disparitäten der rektifizierten Stereo-Bildpaare (Abschnitt 6.2.2).

4.2 Allgemeine Informationen zu 3D Daten

Während auf Stereo-Pipelines, wie `rc_visard`, `rc_viscore` und `stereo_ace` Disparitätsbilder durch das Matching aus linkem und rechtem Kamerabild berechnet werden, werden 3D Daten auf Pipelines vom Typ `zivid` oder `orbbec` intern in Disparitätsbilder umgerechnet, die dann mit Hilfe eines virtuellen Basisabstands zur Berechnung von Tiefendaten verwendet werden können.

Die folgenden Abschnitte beschreiben, wie Disparitätsbilder aus Stereobildpaaren berechnet werden, und wie Disparitäts-, Fehler- und Konfidenzbilder verwendet werden können, um daraus Tiefendaten und -fehler zu berechnen.

4.2.1 Berechnung von Disparitätsbildern

Nach der Rektifizierung haben das linke und das rechte Kamerabild die Eigenschaft, dass ein Objektpunkt in beiden Bildern auf die gleiche Pixelreihe projiziert wird. Die Pixelspalte des Objektpunkts ist im rechten Bild maximal so groß wie die Pixelspalte des Objektpunkts im linken Bild. Der Begriff Disparität bezeichnet den Unterschied zwischen den Pixelspalten im rechten und linken Bild und gibt indirekt die Tiefe des Objektpunkts, d.h. dessen Abstand zur Kamera an. Das Disparitätsbild speichert die Disparitätswerte aller Pixel des linken Kamerabilds.

Je größer die Disparität, desto näher liegt der Objektpunkt. Beträgt die Disparität 0, bedeutet dies, dass die Projektionen des Objektpunkts in der gleichen Bildspalte liegen und der Objektpunkt sich in unendlicher Distanz befindet. Häufig gibt es Pixel, für welche die Disparität nicht bestimmt werden kann. Dies ist der Fall bei Verdeckungen auf der linken Seite von Objekten, da diese Bereiche von der rechten Kamera nicht eingesehen werden können. Zudem lässt sich die Disparität auch bei texturlosen Bereichen nicht bestimmen. Pixel, für welche die Disparität nicht bestimmt werden kann, werden mit dem besonderen Disparitätswert 0 als ungültig markiert. Um zwischen ungültigen Disparitätsmessungen und Messungen, bei denen die Disparität aufgrund der unendlich weit entfernten Objekte 0 beträgt, unterscheiden zu können, wird der Disparitätswert für den letztgenannten Fall auf den kleinstmöglichen Disparitätswert über 0 gesetzt.

Um Disparitätswerte zu berechnen, muss der Stereo-Matching-Algorithmus die zugehörigen Objektpunkte im linken und rechten Kamerabild finden. Diese Punkte stellen jeweils den gleichen Objektpunkt in der Szene dar. Für das Stereo-Matching nutzt der *rc_reason_stack SGM* (*Semi-Global Matching*). Dieser Algorithmus zeichnet sich durch eine kurze Laufzeit aus und bietet, insbesondere an Objekträndern, bei feinen Strukturen und in schwach texturierten Bereichen, eine hohe Genauigkeit.

Unabhängig vom eingesetzten Verfahren ist es beim Stereo-Matching wichtig, dass das Bild über eine gewisse Textur verfügt, durch Muster oder Oberflächenstrukturen. Bei einer gänzlich untexturierten Szene, wie einer weißen Wand ohne jede Struktur, können Disparitätswerte entweder nicht berechnet werden, oder aber die Ergebnisse sind fehlerhaft oder von geringer Konfidenz (siehe *Konfidenz- und Fehlerbilder*, Abschnitt 4.2.3). Bei der Textur in der Szene sollte es sich nicht um ein künstliches,

regelmäßig wiederkehrendes Muster handeln, da diese Strukturen zu Mehrdeutigkeiten und damit zu falschen Disparitätsmessungen führen können.

Für schwach texturierte Objekte oder in untexturierten Umgebungen lässt sich mithilfe eines externen Musterprojektors eine statische künstliche Struktur auf die Szene projizieren. Dieses projizierte Muster sollte zufällig sein und keine wiederkehrenden Strukturen enthalten. Der *rc_reason_stack* bietet das IOControl-Modul als optionales Softwaremodul (siehe *IOControl und Projektor-Kontrolle*, Abschnitt 6.4.4), das einen Musterprojektor ansteuern kann.

4.2.2 Berechnung von Tiefenbildern und Punktwolken

Die folgenden Gleichungen zeigen, wie sich die tatsächlichen 3D-Koordinaten P_x, P_y, P_z eines Objektpunkts bezogen auf das Kamera-Koordinatensystem aus den Pixelkoordinaten p_x, p_y des Disparitätsbilds und dem Disparitätswert d in Pixeln berechnen lassen:

$$\begin{aligned} P_x &= \frac{p_x \cdot t}{d} \\ P_y &= \frac{p_y \cdot t}{d} \\ P_z &= \frac{f \cdot t}{d}, \end{aligned} \tag{4.1}$$

wobei f die Brennweite nach der Rektifizierung (in Pixeln) und t der während der Kalibrierung ermittelte Stereo-Basisabstand (in Metern) ist.

Bemerkung: Der *rc_reason_stack* stellt über seine verschiedenen Schnittstellen einen Brennweitenfaktor bereit. Er bezieht sich auf die Bildbreite, um verschiedene Bildauflösungen zu unterstützen. Die Brennweite f in Pixeln lässt sich leicht bestimmen, indem der Brennweitenfaktor mit der Bildbreite (in Pixeln) multipliziert wird.

Es ist zu beachten, dass für Gleichungen (4.1) davon ausgegangen wird, dass das Bildkoordinatensystem im Bildhauptpunkt zentriert ist, der üblicherweise in der Bildmitte liegt, und dass sich p_x, p_y auf die Mitte des Pixels bezieht, durch Addieren von 0.5 auf die ganzzahligen Pixelkoordinaten. In der folgenden Abbildung ist die Definition des Bildkoordinatensystems dargestellt.

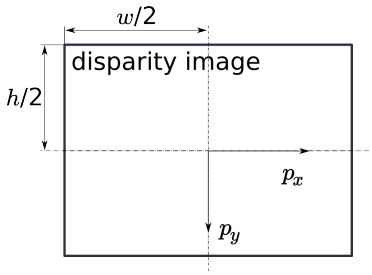


Abb. 4.2: Bildkoordinatensystem: Der Ursprung des Bildkoordinatensystems befindet sich in der Bildmitte – w ist die Bildbreite und h die Bildhöhe.

Die Gesamtheit aller aus dem Disparitätsbild errechneten Objektpunkte ergibt eine Punktwolke, die für 3D-Modellierungsanwendungen verwendet werden kann. Das Disparitätsbild kann in ein Tiefenbild umgewandelt werden, indem der Disparitätswert jedes Pixels durch den Wert P_z ersetzt wird.

Bemerkung: Auf der Homepage von Roboception (http://www.roboception.com/download) stehen Software und Beispiele zur Verfügung, um Disparitätsbilder, welche über GigE Vision vom *rc_reason_stack* empfangen werden, in Tiefenbilder und Punktwolken umzuwandeln.

4.2.3 Konfidenz- und Fehlerbilder

Für jedes Disparitätsbild wird zusätzlich ein Fehler- und ein Konfidenzbild zur Verfügung gestellt, um die Unsicherheit jedes einzelnen Disparitätswerts anzugeben. Fehler- und Konfidenzbilder besitzen die gleiche Auflösung und Bildwiederholrate wie das Disparitätsbild. Im Fehlerbild ist der Disparitätsfehler d_{eps} in Pixeln angegeben. Er bezieht sich auf den Disparitätswert an der gleichen Bildkoordinate im Disparitätsbild. Das Konfidenzbild gibt den entsprechenden Konfidenzwert c zwischen 0 und 1 an. Die Konfidenz gibt an, wie wahrscheinlich es ist, dass der wahre Disparitätswert innerhalb des Intervalls des dreifachen Fehlers um die gemessene Disparität d liegt, d.h. $[d - 3d_{eps}, d + 3d_{eps}]$. So lässt sich das Disparitätsbild mit Fehler- und Konfidenzwerten in Anwendungen einsetzen, für die probabilistische Folgerungen nötig sind. Die Konfidenz- und Fehlerwerte für eine ungültige Disparitätsmessung betragen 0.

Der Disparitätsfehler d_{eps} (in Pixeln) lässt sich mithilfe der Brennweite f (in Pixeln), des Basisabstands t (in Metern) und des Disparitätswerts d (in Pixeln) desselben Pixels im Disparitätsbild in einen Tiefenfehler z_{eps} (in Metern) umrechnen:

$$z_{eps} = \frac{d_{eps} \cdot f \cdot t}{d^2}. \tag{4.2}$$

Durch Kombination der Gleichungen (4.1) und (4.2) kann der Tiefenfehler zur Tiefe in Bezug gebracht werden:

$$z_{eps} = \frac{d_{eps} \cdot {P_z}^2}{f \cdot t}.$$

5 Kamerapipelines

Der *rc_reason_stack* unterstützt mehrere Kameras zur selben Zeit. Dazu bietet er bis zu vier *Kamerapipelines*, die vom Benutzer konfiguriert werden können.

Eine Kamerapipeline beinhaltet verschiedene Softwaremodule für die Datenaufnahme der mit der Pipeline verbundenen Kamera, für Detektionen und für die Konfiguration der Module in dieser Pipeline, z.B. durch eine Hand-Auge-Kalibrierung.

Der *rc_reason_stack* unterstützt Kameras vom Typ *rc_visard*, *rc_viscore*, *zivid*, *Orbbec* und *Stereo ace*. Der Typ der zugehörigen Kamerapipeline muss so konfiguriert werden, dass er zum angeschlossenen Gerät passt.

5.1 Konfiguration der Kamerapipelines

Die Kamerapipelines können über die *Web GUI* (Abschnitt 7.1) unter *System* → *Kamera Pipelines* konfiguriert werden. Diese Seite zeigt die laufenden Pipelines mit ihrem Typ und dem verbundenen Gerät an.

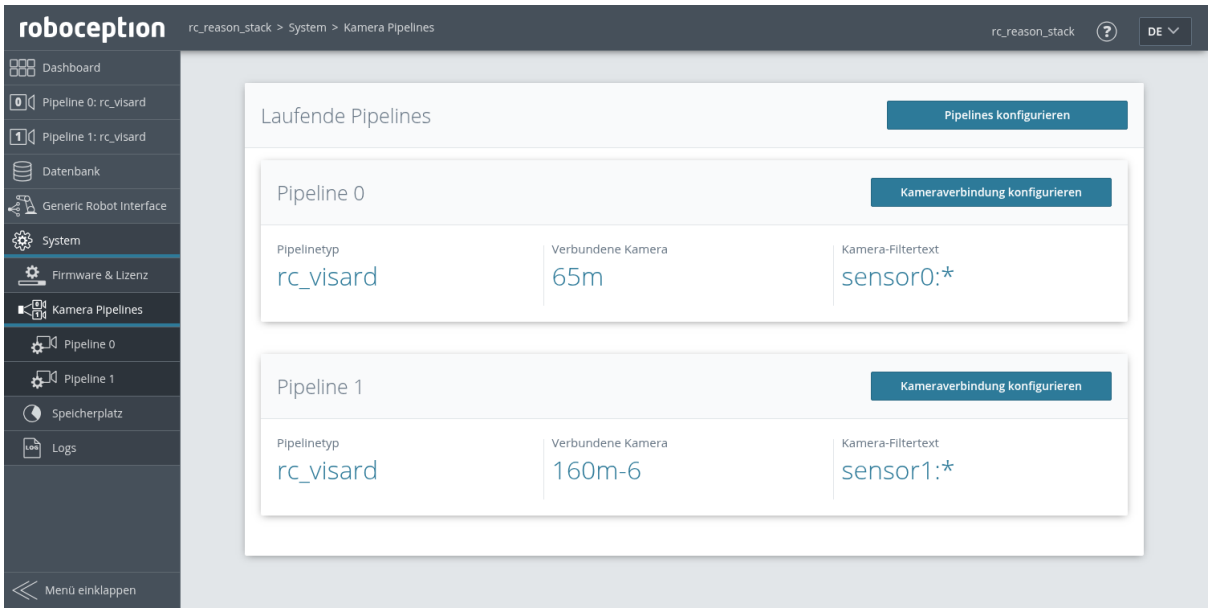


Abb. 5.1: Beispiel der Seite *Kamera Pipelines* auf einem *rc_reason_stack* mit zwei laufenden Pipelines vom Typ `rc_visard`

Durch Klick auf *Pipelines konfigurieren* kann die Anzahl und der Typ der laufenden Kamerapipelines wie in der nächsten Abbildung gezeigt konfiguriert werden.

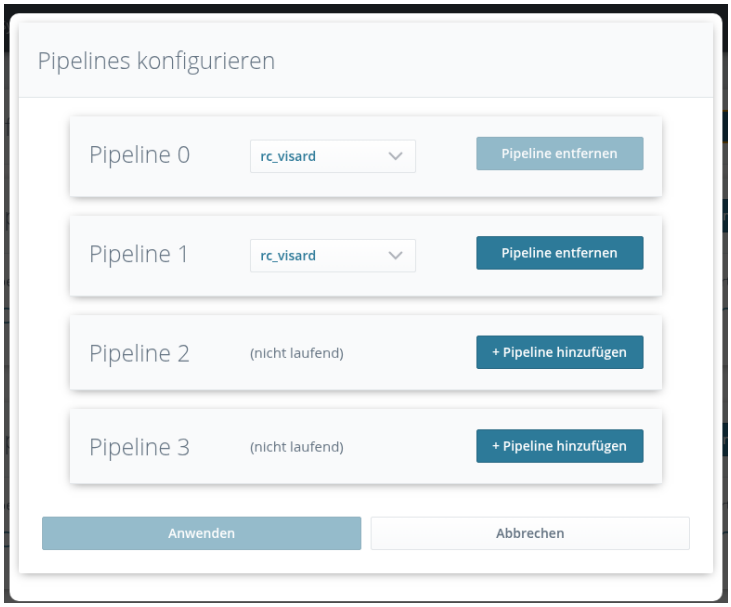


Abb. 5.2: Konfiguration der Kamerapipelines

Der Typ einer laufenden Pipeline kann geändert werden, indem ein anderer Typ im Dropdown-Feld der jeweiligen Pipeline ausgewählt wird. Eine laufende Pipeline kann entfernt werden, indem man auf *Pipeline entfernen* klickt. Einzig die Pipeline 0 kann nie gelöscht werden, da sie die primäre Pipeline ist. Durch Klick auf *+ Pipeline hinzufügen* und anschließendes Auswählen des Pipelinetyps kann eine neue Pipeline erstellt werden.

Sobald alle Pipelines wie gewünscht konfiguriert sind, können die Änderungen durch Klick auf *Anwenden* angewendet werden. Danach muss der *rc_reason_stack* neugestartet werden, damit die Änderungen wirksam werden.

5.2 Konfiguration der verbundenen Kameras

Eine Pipeline eines bestimmten Typs kann nur Geräte desselben Typs erkennen. Das bedeutet, dass eine Pipeline vom Typ `rc_visard` nur mit einem *rc_visard* verbunden werden kann. Falls mehrere Kameras desselben Typs am *rc_reason_stack* angeschlossen sind, kann durch Setzen eines Filtertexts eine bestimmte Kamera für jede Pipeline ausgewählt werden. Der aktuelle Filtertext wird für jede laufende Pipeline angezeigt, wie in Abb. 5.1 dargestellt. Standardmäßig ist der Filtertext auf $*$ gesetzt, was bedeutet, dass jedes Gerät, das zum Pipelinetyp passt, automatisch verbunden wird, aber nur, wenn es eindeutig ist. Andernfalls wird keine Kamera mit dieser Pipeline verbunden und ein Fehler angezeigt.

Um den Filtertext anzupassen und eine Kamera für eine Pipeline auszuwählen, klickt man auf *Kameraverbindung konfigurieren* auf der Seite *Kamera Pipelines*, oder wählt die entsprechende Pipeline im Menü unter, z.B., *System* → *Kamera Pipelines* → *Pipeline 1*. Diese Seite zeigt den aktuellen Filtertext und weitere Informationen über die verbundene Kamera an.

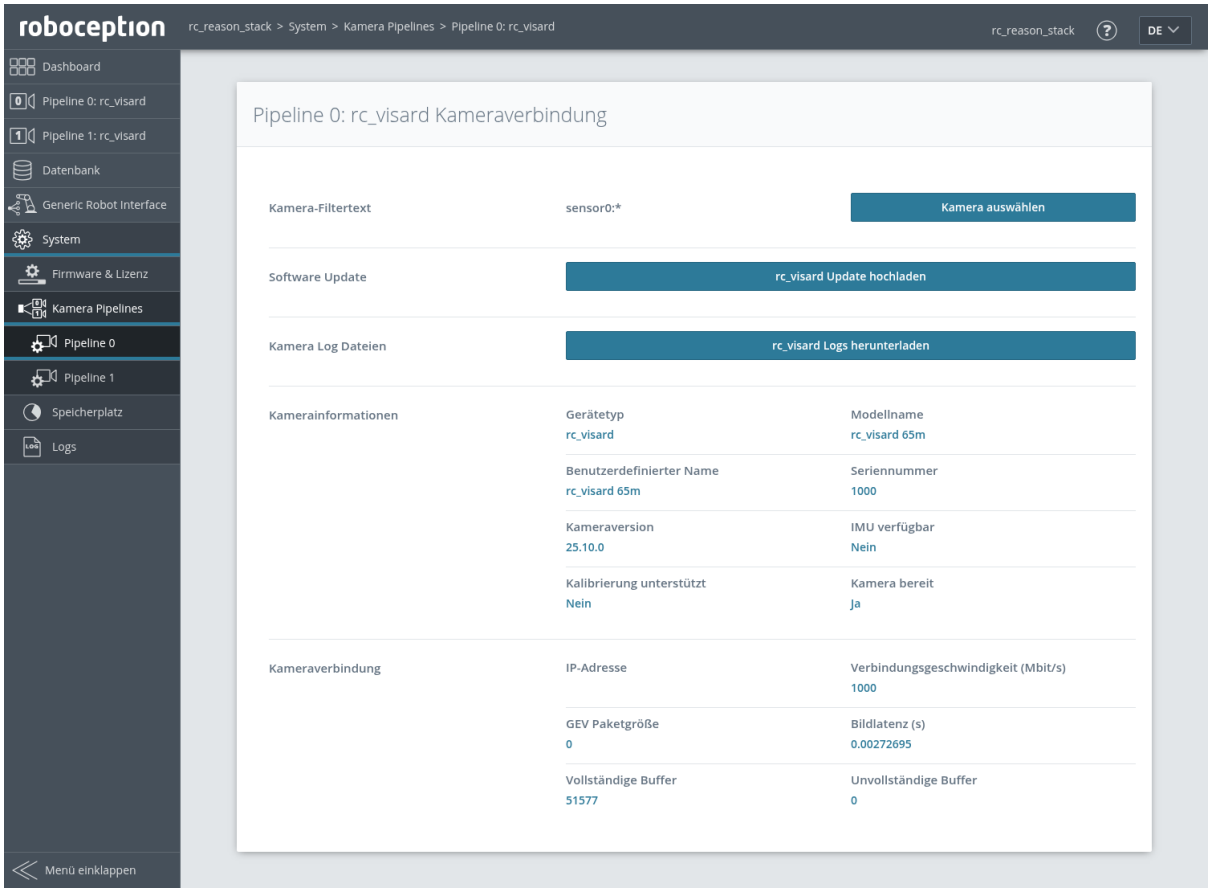


Abb. 5.3: Konfigurieren der Kameraverbindung von Pipeline 0

Ein Klick auf *Kamera auswählen* öffnet einen Dialog zum Editieren des Filtertexts.

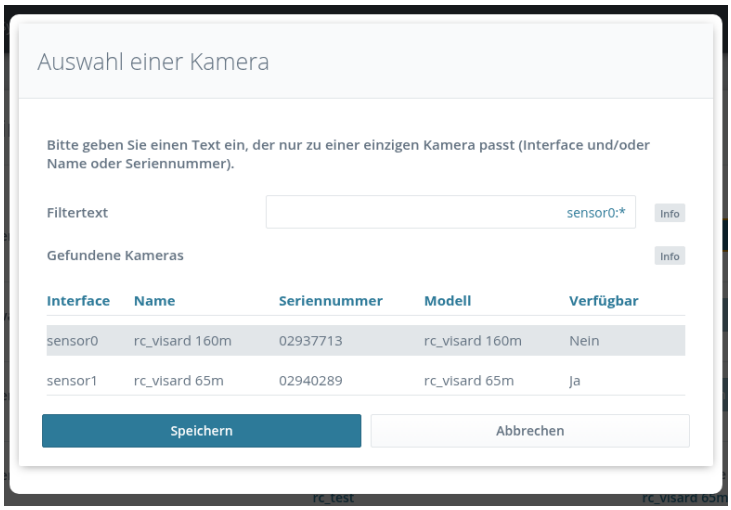


Abb. 5.4: Auswahl der Kamera durch Setzen eines Filtertexts

Dieser Dialog zeigt weiterhin eine Liste aller erkannten Geräte, die zum Pipelinetyp passen, und markiert diejenigen, die zum aktuell eingetragenen Filtertext passen. Es wird auch angezeigt, ob die Geräte bereits von einer anderen Pipeline verwendet werden. Filtertexte können durch Klicken auf das *Interface*, den *Namen* oder die *Seriennummer* des gewünschten Geräts ausgewählt werden. Die folgende Tabelle zeigt mögliche Filtertexte.

Tab. 5.1: Mögliche Kamera-Filtertexte

Filtertext	Beschreibung
*	wählt jedes Gerät aus, das zum Pipelinetyp passt
sensor<n>:*	wählt jedes Gerät aus, das über das sensor<n> Interface angeschlossen ist und dem Pipelinetyp entspricht.
<Name>	wählt das Gerät mit diesem benutzerdefinierten Namen aus
<Seriennummer>	wählt das Gerät mit dieser Seriennummer aus
sensor<n>:<Seriennummer>	wählt das Gerät aus, das über das sensor<n> Interface angeschlossen sind, und diese Seriennummer hat
sensor<n>:<Name>	wählt das Gerät aus, das über das sensor<n> Interface angeschlossen sind, und diesen benutzerdefinierten Namen hat
	falls leer, wird keine Kamera verbunden

Durch Klick auf *Speichern* wird der eingegebene Filtertext angewendet und eine passende Kamera mit dieser Pipeline verbunden, wenn möglich. Das Ändern des Filtertext ist ohne Neustart des *rc_reason_stack* möglich.

6 Softwaremodule

Der *rc_reason_stack* beinhaltet eine Reihe von Softwaremodulen mit verschiedenen Funktionalitäten. Jedes Softwaremodul bietet über seine zugehörige *Node* eine Schnittstelle über *REST-API-Schnittstelle* (Abschnitt 7.2) oder das *Generic Robot Interface* (Abschnitt 7.3) an.

Der *rc_reason_stack* bietet die Möglichkeit, mehrere 3D Kameras wie den *rc_visard* anzuschließen. Die Bilddaten jedes Geräts werden in separaten *Kamerapipelines* verarbeitet, welche jeweils aus mehreren verschiedenen Softwaremodulen bestehen. Die Module, die innerhalb einer Pipeline laufen, sind pipelinespezifisch. Das heißt, sie können verschiedene Parameterwerte für jede Pipeline haben. Die Softwaremodule, die außerhalb der Pipelines laufen, sind global und stellen globale Daten für alle Pipelines bereit. Eine Übersicht ist in Abb. 6.1 dargestellt. Die Portnummern sind für die Standardinstallation wie in *Installation* (Abschnitt **??**) beschrieben angegeben und können in der Docker Compose-Datei geändert werden.

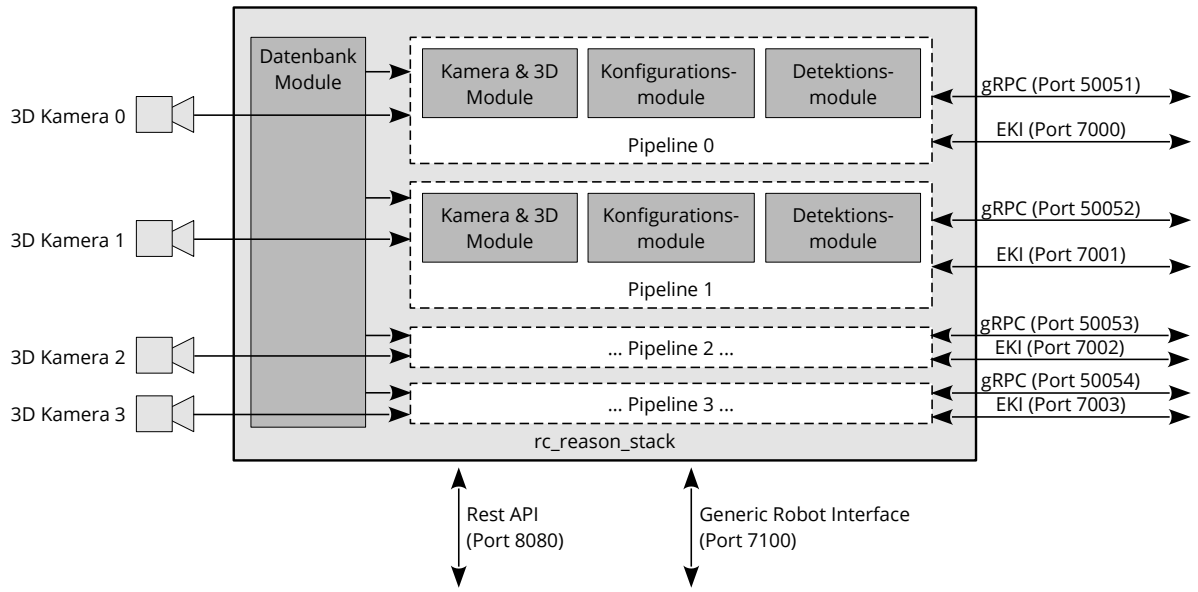


Abb. 6.1: Übersicht über die pipelinespezifischen und globalen Softwaremodule auf dem *rc_reason_stack*

Die pipelinespezifischen Softwaremodule des *rc_reason_stack* können unterteilt werden in

- ***Kamera Modul*** **(Abschnitt 6.1)** erfasst Bildpaare und führt die planare Rektifizierung durch, wodurch die Kamera als Messinstrument verwendet werden kann. In Abhängigkeit vom ausgewählten Kamerapipelinetyp bietet dieses Modul unterschiedliche Laufzeitparameter.
- ***3D-Module*** **(Abschnitt 6.2)** welche 3D Tiefeninformationen, wie Disparitäts-, Fehler- und Konfidenzbilder, bereitstellen,
- ***Detektions- und Messmodule*** **(Abschnitt 6.3)** welche eine Vielzahl verschiedener Detektionsfunktionen, wie Greifpunktberechnungen und Objekterkennung anbieten.

- ***Konfigurationsmodule*** **(Abschnitt 6.4)** welche es dem Nutzer ermöglichen, Kalibrierungen durchzuführen und den *rc_reason_stack* für spezielle Anwendungen zu konfigurieren.

Die Softwaremodule, die global für alle Kamerapipelines auf dem *rc_reason_stack* laufen, sind

- ***Datenbankmodule*** **(Abschnitt 6.5)** welche dem Nutzer die Konfiguration globaler Daten ermöglichen, die in allen anderen Modulen verfügbar sind, wie Load Carrier, Regions of Interest und Greifer.

6.1 Kamera Modul

Das Kameramodul ist ein Basismodul welches auf jedem *rc_reason_stack* verfügbar ist. Es ist für die Bildakquise und die Rektifizierung der Bilder verantwortlich. Das Modul bietet diverse Parameter um z.B. die Belichtungszeit oder die Bildwiederholrate zu verändern.

Bemerkung: In Abhängigkeit vom ausgewählten Kamerapipelinetyp bietet dieses Modul unterschiedliche Laufzeitparameter.

6.1.1 Rektifizierung

Um die Bildverarbeitung zu vereinfachen rektifiziert das Modul alle Kamerabilder basierend auf der Kamerakalibrierung. Dies bedeutet, dass die Verzeichnung entfernt und der Bildhauptpunkt genau in die Mitte des Bildes gelegt wird.

Eine rektifizierte Kamera kann mit der Brennweite als einzigen Modellparameter beschrieben werden. Der *rc_reason_stack* stellt über seine verschiedenen Schnittstellen einen Brennweitenfaktor bereit. Er bezieht sich auf die Bildbreite, um verschiedene Bildauflösungen zu unterstützen. Die Brennweite f in Pixeln lässt sich leicht bestimmen, indem der Brennweitenfaktor mit der Bildbreite (in Pixeln) multipliziert wird.

Im Fall einer Stereokamera richtet die Rektifizierung die Bilder so aus, dass Objektpunkte in beiden Bildern immer in die gleiche Bildzeile projiziert werden. Die optischen Achsen der Kameras werden dadurch exakt parallel ausgerichtet.

Bemerkung: Falls ein `zivid` oder `orbbec` Sensor statt einer Stereokamera benutzt wird, dann steht nur ein Kamerabild zur Verfügung. Dieses Kamerabild ist allerdings ebenfalls rektifiziert, d.h. es ist verzeichnungsfrei und der Bildhauptpunkt befindet sich in der Bildmitte.

6.1.2 Anzeigen und Herunterladen von Bildern

Der *rc_reason_stack* bietet über *gRPC Bilddatenschnittstelle* (siehe Abschnitt 7.6) zeitgestempelte rektifizierte Kamerabilder.

Live-Streams in geringerer Qualität werden in der *Web GUI* (Abschnitt 7.1) bereitgestellt.

Die Web GUI bietet weiterhin die Möglichkeit, einen Schnappschuss der aktuellen Szene als .tar.gz-Datei zu speichern, wie in *Herunterladen von Kamerabildern* (Abschnitt 7.1.4) beschrieben wird.

6.1.3 Pipelinetypen *rc_visard* und *rc_core*

6.1.3.1 Parameter

Das Kamera-Modul wird in der REST-API als `rc_camera` bezeichnet und in der *Web GUI* (Abschnitt 7.1) auf der Seite *Kamera* in der gewünschten Pipeline dargestellt. Der Benutzer kann die Stereo-Matching-Parameter entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.2) ändern.

Übersicht über die Parameter

Bemerkung: Das Minimum, Maximum und die Defaultwerte in der Tabelle unten zeigen Werte des *rc_visard*. Diese Werte unterscheiden sich bei anderen Kameramodellen und bei einer `rc_viscore` Pipeline.

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.1: Laufzeitparameter des `rc_camera`-Moduls für eine Pipeline vom Typ `rc_visard`

Name	Typ	Min.	Max.	Default	Beschreibung
`acquisition_mode`	string	-	-	Continuous	Aufnahmemodus: [Continuous, Trigger]
`exp_auto`	bool	false	true	true	Umschalten zwischen automatischer und manueller Belichtung (veraltet, nutzen Sie stattdessen exp_control)
`exp_auto_average_max`	float64	0.0	1.0	0.75	Maximaler Belichtungsmittelwert im Auto Belichtungsmodus
`exp_auto_average_min`	float64	0.0	1.0	0.25	Maximaler Belichtungsmittelwert im Auto Belichtungsmodus
`exp_auto_mode`	string	-	-	Normal	Modus für automatische Belichtung: [Normal, Out1High, AdaptiveOut1]
`exp_control`	string	-	-	Auto	Art der Belichtungsregelung: [Manual, Auto, HDR]
`exp_height`	int32	0	959	0	Höhe der Region für automatische Belichtung, 0 für das ganze Bild
`exp_max`	float64	6.6e-05	0.018	0.018	Maximale Belichtungszeit in Sekunden im Auto Belichtungsmodus
`exp_offset_x`	int32	0	1279	0	Erste Spalte der Region für automatische Belichtung
`exp_offset_y`	int32	0	959	0	Erste Zeile der Region für automatische Belichtung
`exp_value`	float64	6.6e-05	0.018	0.005	Maximale Belichtungszeit in Sekunden im Auto Belichtungsmodus
`exp_width`	int32	0	1279	0	Breite der Region für automatische Belichtung, 0 für das ganze Bild
`fps`	float64	1.0	25.0	25.0	Bildwiederholrate in Hertz
`gain_value`	float64	0.0	18.0	0.0	Verstärkung in Dezibel, wenn nicht im Auto Belichtungsmodus
`gamma`	float64	0.1	10.0	1.0	Gammafaktor
`trigger_activation`	string	-	-	RisingEdge	Triggeraktivierung: [RisingEdge, FallingEdge, AnyEdge]
`trigger_source`	string	-	-	Software	Triggerquelle: [Software, In1, In2, In3, In4]
`wb_auto`	bool	false	true	true	Ein- und Ausschalten des manuellen Weißabgleichs (nur für Farbkameras)
`wb_ratio_blue`	float64	0.125	8.0	2.4	Blau-zu-Grün-Verhältnis, falls wb_auto auf false gesetzt ist (nur für Farbkameras)
`wb_ratio_red`	float64	0.125	8.0	1.2	Rot-zu-Grün-Verhältnis, falls wb_auto auf false gesetzt ist (nur für Farbkameras)

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Seite *Tiefenbild* der Web GUI repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben und die Parameter werden in der Reihenfolge, in der sie in der Web GUI erscheinen, aufgelistet.

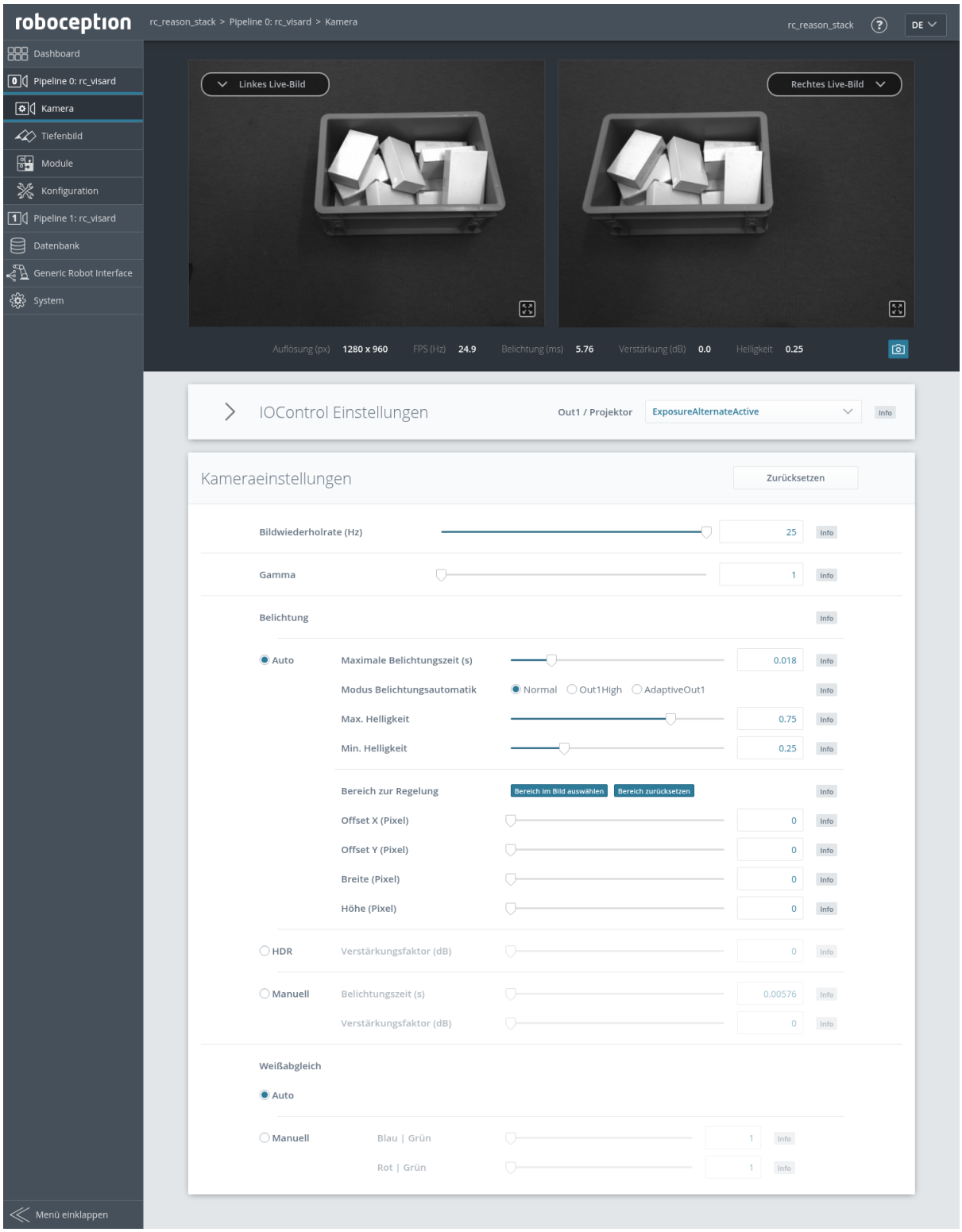


Abb. 6.2: Seite *Kamera* in der Web GUI

`fps` (*Bildwiederholrate (Hz)*)

Dieser Wert bezeichnet die Bildwiederholrate der Kamera in Bildern pro Sekunde und begrenzt zugleich die Frequenz, mit der Tiefenbilder berechnet werden können. Die Bildwiederholrate entspricht auch der Frequenz, mit welcher der *rc_reason_stack* Bilder über GigE Vision bereitstellt. Wird diese Frequenz verringert, reduziert sich auch die zur Übertragung der Bilder benötigte Bandbreite des Netzwerks.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?fps=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?fps=<value>
```

`gamma` (*Gamma*)

Der Gammawert bestimmt, wie das gemessene Licht auf die Helligkeit eines Pixels abgebildet wird. Ein Gammawert von 1 entspricht einem linearen Zusammenhang. Kleinere Gammawerte lassen dunkle Bildbereiche heller erscheinen. Ein Wert um 0.5 entspricht der menschlichen Wahrnehmung.

Bemerkung: Für eine Pipeline vom Typ `rc_visard` kann dieser Wert nur geändert werden, wenn der verbundene *rc_visard* mindestens die Firmwareversion 22.07 hat. Andernfalls ist der Gammawert immer 1.0.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?gamma=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?gamma=<value>
```

`exp_control` (Belichtung *Auto*, *HDR* oder *Manual*)

Die Belichtungsregelung kann auf *Auto*, *HDR* oder *Manual* gesetzt werden. Dies ersetzt den veralteten *exp_auto* Parameter.

Auto: Dies ist der Standard Modus der die die Belichtungszeit und Verstärkung automatisch anpasst, um Unter- und Überbelichtung zu vermeiden. Wenn die Automatik abgeschaltet wird, werden *exp_value* und *gain_value* auf die letzten von der Automatik ermittelten Werte für Belichtungszeit und Verstärkung gesetzt.

HDR: Der HDR Modus berechnet Bilder mit hohem Dynamikbereich durch Kombination von Bildern mit unterschiedlichen Belichtungszeiten um über- und unterbelichtete Bereiche zu vermeiden. Dieser Modus verringert die Bildwiederholrate und ist nur für statische Szenen geeignet.

Manual: Im manuellen Belichtungsmodus werden die Belichtungszeit und die Verstärkung konstant gehalten unabhängig von der resultierenden Bildhelligkeit.

Bemerkung: Für eine Pipeline vom Typ `rc_visard` kann der *HDR* Modus nur genutzt werden, wenn der verbundene *rc_visard* mindestens die Firmwareversion 23.01 hat.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?exp_control=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_control=<value>
```

`exp_auto_mode` (*Modus Belichtungszeitautomatik*)

Der Modus für automatische Belichtung kann auf *Normal*, *Out1High* oder *AdaptiveOut1* gesetzt werden. Diese Modi sind nur relevant, wenn der *rc_reason_stack* mit einer externen Lichtquelle oder einem Projektor betrieben wird, der an den GPIO-Ausgang 1 der Kamera angeschlossen ist. Dieser Ausgang kann durch das IOControl-Modul (*IOControl und Projektor-Kontrolle*, Abschnitt 6.4.4) gesteuert werden.

Normal: Alle Bilder werden für die Regelung der Belichtungszeit in Betracht gezogen, außer wenn der IOControl-Modus für den GPIO-Ausgang 1 *ExposureAlternateActive* ist: Dann werden nur Bilder berücksichtigt, bei denen GPIO-Ausgang 1 HIGH ist, da diese Bilder heller sein können, falls dieser GPIO-Ausgang benutzt wird um einen externen Projektor auszulösen.

Out1High: Die Belichtungszeit wird nur anhand der Bilder mit GPIO-Ausgang 1 HIGH angepasst. Bilder bei denen GPIO-Ausgang 1 LOW ist, werden für die Belichtungszeitregelung nicht berücksichtigt. Das bedeutet, die Belichtungszeit ändert sich nicht, solange nur Bilder mit GPIO-Ausgang 1 LOW aufgenommen werden. Dieser Modus wird für die Benutzung mit dem Single+Out1 Tiefenbild Aufnahmemodus (siehe *Stereo Matching Parameters*, 6.2.2.1 und externem Projektor empfohlen, wenn die Helligkeit der Szene nur zu den Zeitpunkten berücksichtigt werden soll, wenn GPIO-Ausgang 1 HIGH ist. Das ist zum Beispiel der Fall, wenn kurz vor einer Objekterkennung ein heller Teil des Roboters durch das Bild fährt, der die Belichtungseinstellungen jedoch nicht beeinflussen soll.

AdaptiveOut1: Dieser Modus nutzt alle Kamerabilder und speichert die Differenz der Belichtung zwischen Bildern mit GPIO Ausgang 1 HIGH und LOW. Während der IOControl-Modus für GPIO-Ausgang 1 LOW ist, werden die Bilder um diese Differenz unterbelichtet, um eine Überbelichtung zu verhindern, sobald der externe Projektor über GPIO-Ausgang 1 ausgelöst wird. Die Differenz der Belichtung wird als *Out1 Reduktion* unter den Livebildern angezeigt. Dieser Modus wird empfohlen, wenn im Stereo-Matching-Modul der Parameter `acquisition_mode` auf `SingleFrameOut1` (*Einzelbild+Out1*) gesetzt ist (*Parameter des Stereo-Matching-Moduls*, Abschnitt 6.2.2.1), und ein externer Projektor an den GPIO-Ausgang 1 angeschlossen ist, und wenn die Helligkeit der Szene zu jeder Zeit zur Belichtungszeitregelung berücksichtigt werden soll.Das ist zum Beispiel in Anwendungen mit veränderlichen äußeren Lichtbedingungen der Fall.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?exp_auto_mode=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_auto_mode=<value>
```

`exp_max` (*Maximale Belichtungszeit*)

Dieser Wert gibt die maximale Belichtungszeit im automatischen Modus in Sekunden an. Die tatsächliche Belichtungszeit wird automatisch angepasst, sodass das Bild korrekt belichtet wird. Sind die Bilder trotz maximaler Belichtungszeit noch immer unterbelichtet, erhöht der *rc_reason_stack* schrittweise die Verstärkung, um die Helligkeit der Bilder zu erhöhen. Es ist sinnvoll, die Belichtungszeit zu begrenzen, um die bei schnellen Bewegungen auftretende Bildunschärfe zu vermeiden oder zu verringern. Jedoch führt eine höhere Verstärkung auch zu mehr Bildrauschen. Welcher Kompromiss der beste ist, hängt immer auch von der Anwendung ab.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?exp_max=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_max=<value>
```

`exp_auto_average_max` (*Maximale Helligkeit*) und `exp_auto_average_min` (*Minimale Helligkeit*)

Die automatische Belichtungszeitsteuerung versucht die Belichtungszeit und den Verstärkungsfaktor so einzustellen, dass die mittlere Bildhelligkeit im Bild oder im *Bereich zur Regelung* zwischen der maximalen und minimalen Helligkeit liegt. Die maximale Helligkeit wird benutzt, wenn keine Bildteile in der Sättigung sind, d.h. keine Überbelichtung durch helle Oberflächen oder Reflexionen vorhanden sind. Falls Sättigungen auftreten, werden die Belichtungszeit und der Verstärkungsfaktor verringert, aber nur bis zur eingestellten minimalen Helligkeit.

Der Parameter für die maximale Helligkeit hat Vorrang über den Parameter der minimalen Helligkeit. Falls die minimale Helligkeit größer als die maximale ist, versucht die automatische Belichtungszeitsteuerung die mittlere Bildhelligkeit auf die maximale Helligkeit zu setzen.

Die aktuelle Helligkeit wird in der Statuszeile unter den Bildern angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?<exp_auto_
↪average_max|exp_auto_average_min>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?<exp_auto_average_max|exp_auto_
↪average_min>=<value>
```

`exp_offset_x`, `exp_offset_y`, `exp_width`, `exp_height` (*Bereich zur Regelung*)

Diese Werte definieren eine rechteckige Region im linken rektifizierten Bild, um den von der automatischen Belichtung überwachten Bereich zu limitieren. Die Belichtungszeit und der Verstärkungsfaktor werden so gewählt, dass die definierte Region optimal belichtet wird. Dies kann zu Über- oder Unterbelichtung in anderen Bildbereichen führen. Falls die Breite oder Höhe auf 0 gesetzt werden, dann wird das gesamte linke und rechte Bild von der automatischen Belichtungsfunktion berücksichtigt. Dies ist die Standardeinstellung.

Die Region wird in der Web GUI mit einem Rechteck im linken rektifizierten Bild visualisiert. Sie kann über Slider oder direkt im Bild mithilfe der Schaltfläche `Bereich im Bild auswählen` verändert werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?<exp_offset_
↪x|exp_offset_y|exp_width|exp_height>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?<exp_offset_x|exp_offset_y|exp_
↪width|exp_height>=<value>
```

`exp_value` (*Belichtungszeit*)

Dieser Wert gibt die Belichtungszeit im manuellen Modus in Sekunden an. Diese Belichtungszeit wird konstant gehalten, auch wenn die Bilder unterbelichtet sind.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?exp_value=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_value=<value>
```

`gain_value` (*Verstärkungsfaktor (dB)*)

Dieser Wert gibt den Verstärkungsfaktor im manuellen Modus in Dezibel an. Höhere Verstärkungswerte reduzieren die Belichtungszeit, führen aber zu Rauschen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?gain_value=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?gain_value=<value>
```

`wb_auto` (*Weißabgleich Auto* oder *Manuell*)

Dieser Wert kann auf *true* gesetzt werden, um den automatischen Weißabgleich anzuschalten. Bei *false* kann das Verhältnis der Farben manuell mit `wb_ratio_red` und `wb_ratio_blue` gesetzt werden. `wb_ratio_red` und `wb_ratio_blue` werden auf die letzten von der Automatik ermittelten Werte gesetzt, wenn diese abgeschaltet wird. Der Weißabgleich ist bei monochromen Kameras ohne Funktion und wird in diesem Fall in der Web GUI nicht angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?wb_auto=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?wb_auto=<value>
```

`wb_ratio_blue` und `wb_ratio_red` (*Blau* / *Grün* and *Rot* / *Grün*)

Mit diesen Werten kann das Verhältnis von Blau zu Grün bzw. Rot zu Grün für einen manuellen Weißabgleich gesetzt werden. Der Weißabgleich ist bei monochromen Kameras ohne Funktion und wird in diesem Fall in der Web GUI nicht angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?<wb_ratio_blue|wb_ratio_
↪red>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?<wb_ratio_blue|wb_ratio_red>=<value>
```

6.1.3.2 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 6.2: Statuswerte des rc_camera-Moduls

Name	Beschreibung
baseline	Basisabstand t der Stereokamera in Metern
brightness	Aktuelle Helligkeit als Wert zwischen 0 und 1
color	0 für monochrome Kameras, 1 für Farbkameras
exp	Aktuelle Belichtungszeit in Sekunden. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Belichtung (ms)* angezeigt.
device_trigger_sources	Angabe der verfügbaren Triggerquellen für den Fall, dass das Gerät getriggert werden kann
focal	Brennweitenfaktor, normalisiert auf eine Bildbreite von 1
fps	Aktuelle Bildwiederholrate der Kamerabilder in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als *FPS (Hz)* angezeigt.
gain	Aktueller Verstärkungsfaktor in Dezibel. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Verstärkung (dB)* angezeigt.
gamma	Aktueller Gammawert
height	Höhe des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als zweiter Teil von *Auflösung (px)* angezeigt.
last_timestamp_grabbed	Zeitstempel des letzten aufgenommenen Bildes, wenn die Kamera im Triggermodus ist
out1_reduction	Anteil der Helligkeits-Reduktion (0.0 - 1.0) für Bilder mit GPIO-Ausgang 1=LOW, wenn exp_auto_mode=AdaptiveOut1 oder exp_auto_mode=Out1High. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Out1 Reduktion (%)* angezeigt.
params_override_active	1 wenn die Parameter temporär durch einen laufenden Kalibrierprozess überschrieben werden
selfcalib_counter	Wie oft eine Korrektur durch die Selbstkalibrierung vorgenommen wurde
selfcalib_offset	Aktueller Offset, der durch die Selbstkalibrierung bestimmt wurde
test	0 for Live-Bilder und 1 für Test-Bilder
width	Breite des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als erster Teil von *Auflösung (px)* angezeigt.

6.1.3.3 Services

Das Kamera-Modul bietet folgende Services.

reset_defaults

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.1.4 Pipelinetyp *stereo_ace*

6.1.4.1 Parameter

Das Kamera-Modul auf einer Pipeline vom Typ `stereo_ace` wird in der REST-API als `rc_camera` bezeichnet und in der *Web GUI* (Abschnitt 7.1) auf der Seite *Kamera* in der gewünschten Pipeline dargestellt. Der Benutzer kann die Stereo-Matching-Parameter entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.3: Das `rc_camera`-Modul meldet folgende Statuswerte für eine Pipeline vom Typ `stereo_ace`:

Name	Typ	Min.	Max.	Default	Beschreibung
`acquisition_mode`	string	-	-	Continuous	Aufnahmemodus: [Continuous, Trigger]
`brightness`	float64	-1.0	1.0	0.0	Helligkeit
`contrast`	float64	-1.0	1.0	0.0	Kontrast
`contrast_mode`	string	-	-	Linear	Kontrastmodus [Linear, SCurve]
`exp_auto`	bool	false	true	false	Umschalten zwischen automatischer und manueller Belichtung (veraltet, nutzen Sie stattdessen exp_control)
`exp_auto_average_max`	float64	0.0	1.0	0.75	Maximaler Belichtungsmittelwert im Auto Belichtungsmodus
`exp_auto_average_min`	float64	0.0	1.0	0.25	Maximaler Belichtungsmittelwert im Auto Belichtungsmodus
`exp_auto_mode`	string	-	-	Normal	Modus für automatische Belichtung: [Normal, Out1High, AdaptiveOut1]
`exp_control`	string	-	-	Manual	Art der Belichtungsregelung: [Manual, Auto, HDR]
`exp_height`	int32	0	2047	0	Höhe der Region für automatische Belichtung, 0 für das ganze Bild
`exp_max`	float64	6.6e-05	0.1	0.018	Maximale Belichtungszeit in Sekunden im Auto Belichtungsmodus
`exp_offset_x`	int32	0	2447	0	Erste Spalte der Region für automatische Belichtung
`exp_offset_y`	int32	0	2047	0	Erste Zeile der Region für automatische Belichtung
`exp_value`	float64	6.6e-05	0.1	0.005	Maximale Belichtungszeit in Sekunden im Auto Belichtungsmodus
`exp_width`	int32	0	2447	0	Breite der Region für automatische Belichtung, 0 für das ganze Bild
`fps`	float64	1.0	50.0	25.0	Bildwiederholrate in Hertz
`gain_value`	float64	0.0	48.0	0.0	Verstärkung in Dezibel, wenn nicht im Auto Belichtungsmodus
`gamma`	float64	0.1	3.99998	1.0	Gammafaktor
`light_source_preset`	string	-	-	Daylight6500K	Voreinstellung der Lichtquelle [Off, Tungsten, Daylight5000K, Daylight6500K, FactoryLED6000K]
saturation‘	float64	0.0	2.0	1.0	Sättigung
`trigger_activation`	string	-	-	RisingEdge	Triggeraktivierung: [RisingEdge, FallingEdge, AnyEdge]
`trigger_source`	string	-	-	Software	Triggerquelle: [Software, In1, In2, In3, In4]
`wb_auto`	bool	false	true	true	Ein- und Ausschalten des manuellen Weißabgleichs (nur für Farbkameras)
`wb_ratio_blue`	float64	0.125	16.0	2.4	Blauanteil, falls wb_auto auf false gesetzt ist (nur für Farbkameras)
`wb_ratio_green`	float64	0.125	16.0	1.0	Grünanteil, falls wb_auto auf false gesetzt ist (nur für Farbkameras)
`wb_ratio_red`	float64	0.125	16.0	1.2	Rotanteil, falls wb_auto auf false gesetzt ist (nur für Farbkameras)

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Seite *Tiefenbild* der Web GUI repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben und die Parameter werden in der Reihenfolge, in der sie in der Web GUI erscheinen, aufgelistet.

`acquisition_mode` (*Aufnahmemodus*)

Dieser Wert bestimmt den Aufnahmemodus der Kamera. Im Modus *Kontinuierlich* (`Continuous`) nimmt die Kamera Bilder mit der in `fps` angegebenen Bildwiederholrate auf. Im Modus *Trigger* (`Trigger`) werden nur Bilder aufgenommen, wenn die Kamera ein Triggersignal empfängt.

Bemerkung: Dieser Parameter hat nur eine Auswirkung, wenn er in einer Pipeline mit einem *rc_viscore* oder *rc_visard NG* verwendet wird.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?acquisition_
↪mode=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?acquisition_mode=<value>
```

`trigger_source` (*Triggerquelle*)

Dieser Wert wird nur verwendet, wenn der Aufnahmemodus auf `Trigger` gesetzt ist und bestimmt die Triggerquelle. Im `Software`-Modus kann ein Trigger über den `rc_camera/acquisition_trigger` Service gesendet werden. Wenn der Aufnahmemodus `acquisition_mode` für die Tiefenbilder auf `SingleFrame` oder `SingleFrameOut1` gesetzt ist (siehe *Parameter*, Abschnitt 6.2.2.1), wird der Kamera-Softwaretrigger automatisch bei jeder Tiefenbildaufnahme gesendet. Die Modi `In1` und `In2` sind Hardwaretriggermodi. Ein Bild wird aufgenommen, sobald ein Signal auf dem jeweiligen Eingang empfangen wird.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?trigger_
↪source=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?trigger_source=<value>
```

`trigger_activation` (*Triggeraktivierung*)

Dieser Wert wird nur verwendet, wenn der Aufnahmemodus auf `Trigger` gesetzt ist und die Triggerquelle auf `In1` oder `In2` steht. Er bestimmt die Signalflanke, die genutzt werden soll, um eine Bildaufnahme auszulösen. Mögliche Werte sind `RisingEdge` (steigende Flanke), `FallingEdge` (fallende Flanke) oder `AnyEdge` (steigende und fallende Flanke).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?trigger_
↪activation=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?trigger_activation=<value>
```

`fps` (*Bildwiederholrate (Hz)*)

Dieser Wert bezeichnet die Bildwiederholrate der Kamera in Bildern pro Sekunde und begrenzt zugleich die Frequenz, mit der Tiefenbilder berechnet werden können. Die Bildwiederholrate entspricht auch der Frequenz, mit welcher der *rc_reason_stack* Bilder über GigE Vision bereitstellt. Wird diese Frequenz verringert, reduziert sich auch die zur Übertragung der Bilder benötigte Bandbreite des Netzwerks.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?fps=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?fps=<value>
```

`gamma` (*Gamma*)

Der Gammawert bestimmt, wie das gemessene Licht auf die Helligkeit eines Pixels abgebildet wird. Ein Gammawert von 1 entspricht einem linearen Zusammenhang. Kleinere Gammawerte lassen dunkle Bildbereiche heller erscheinen. Ein Wert um 0.5 entspricht der menschlichen Wahrnehmung.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?gamma=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?gamma=<value>
```

`contrast_mode` (*Kontrastmodus*)

Der Kontrastmodus kann auf „Linear“ (*Linear*) oder „SCurve“ (*S-Curve*) eingestellt werden und bestimmt, wie die Bildintensitätswerte skaliert werden, wenn der Kontrast angepasst wird. Im Modus `Linear` verwendet die Kamera eine lineare Funktion zur Anpassung des Kontrasts. Durch Erhöhen oder Verringern des Kontrasts wird der Gradient der linearen Funktion erhöht oder verringert. Wenn der Kontrast erhöht wird, erscheinen die dunkelsten und hellsten Bereiche des Bildes vollständig schwarz oder vollständig weiß, die anderen Bereiche erscheinen jedoch definierter. Eine Verringerung des Kontrasts hat den gegenteiligen Effekt.Im Modus `SCurve` nutzt die Kamera eine S-Kurven-Funktion zur Anpassung des Kontrasts. Durch Erhöhen des Kontrasts werden dunkle Pixel abgedunkelt und helle Pixel aufgehellt, der Dynamikumfang des Bildes bleibt jedoch erhalten.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?contrast_mode=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?contrast_mode=<value>
```

`contrast` (*Kontrast*)

Durch Anpassen des Kontrasts wird der Unterschied zwischen hellen und dunklen Bereichen im Bild erhöht oder verringert. Die Art und Weise, wie sich die hellen und dunklen Bereiche beim Anpassen des Kontrasts ändern, hängt vom ausgewählten Kontrastmodus (`contrast_mode`) ab.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?contrast=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?contrast=<value>
```

`exp_control` (Belichtung *Auto*, *HDR* oder *Manual*)

Die Belichtungsregelung kann auf *Auto*, *HDR* oder *Manual* gesetzt werden.

Auto: Dies ist der Standard Modus der die die Belichtungszeit und Verstärkung automatisch anpasst, um Unter- und Überbelichtung zu vermeiden. Wenn die Automatik abgeschaltet wird, werden *exp_value* und *gain_value* auf die letzten von der Automatik ermittelten Werte für Belichtungszeit und Verstärkung gesetzt.

HDR: Der HDR Modus berechnet Bilder mit hohem Dynamikbereich durch Kombination von Bildern mit unterschiedlichen Belichtungszeiten um über- und unterbelichtete Bereiche zu vermeiden. Dieser Modus verringert die Bildwiederholrate und ist nur für statische Szenen geeignet.

Manual: Im manuellen Belichtungsmodus werden die Belichtungszeit und die Verstärkung konstant gehalten unabhängig von der resultierenden Bildhelligkeit.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?exp_control=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_control=<value>
```

`exp_auto_mode` (*Modus Belichtungszeitautomatik*)

Der Modus für automatische Belichtung kann auf *Normal*, *Out1High* oder *AdaptiveOut1* gesetzt werden. Diese Modi sind nur relevant, wenn der *rc_reason_stack* mit einer externen Lichtquelle oder einem Projektor betrieben wird, der an den GPIO-Ausgang 1 der Ka-

mera angeschlossen ist. Dieser Ausgang kann durch das IOControl-Modul (*IOControl und Projektor-Kontrolle*, Abschnitt 6.4.4) gesteuert werden.

Normal: Alle Bilder werden für die Regelung der Belichtungszeit in Betracht gezogen, außer wenn der IOControl-Modus für den GPIO-Ausgang 1 *ExposureAlternateActive* ist: Dann werden nur Bilder berücksichtigt, bei denen GPIO-Ausgang 1 HIGH ist, da diese Bilder heller sein können, falls dieser GPIO-Ausgang benutzt wird um einen externen Projektor auszulösen.

Out1High: Die Belichtungszeit wird nur anhand der Bilder mit GPIO-Ausgang 1 HIGH angepasst. Bilder bei denen GPIO-Ausgang 1 LOW ist, werden für die Belichtungszeitregelung nicht berücksichtigt. Das bedeutet, die Belichtungszeit ändert sich nicht, solange nur Bilder mit GPIO-Ausgang 1 LOW aufgenommen werden. Dieser Modus wird für die Benutzung mit dem Single+Out1 Tiefenbild Aufnahmemodus (siehe *Stereo Matching Parameters*, 6.2.2.1 und externem Projektor empfohlen, wenn die Helligkeit der Szene nur zu den Zeitpunkten berücksichtigt werden soll, wenn GPIO-Ausgang 1 HIGH ist. Das ist zum Beispiel der Fall, wenn kurz vor einer Objekterkennung ein heller Teil des Roboters durch das Bild fährt, der die Belichtungseinstellungen jedoch nicht beeinflussen soll.

AdaptiveOut1: Dieser Modus nutzt alle Kamerabilder und speichert die Differenz der Belichtung zwischen Bildern mit GPIO Ausgang 1 HIGH und LOW. Während der IOControl-Modus für GPIO-Ausgang 1 LOW ist, werden die Bilder um diese Differenz unterbelichtet, um eine Überbelichtung zu verhindern, sobald der externe Projektor über GPIO-Ausgang 1 ausgelöst wird. Die Differenz der Belichtung wird als *Out1 Reduktion* unter den Livebildern angezeigt. Dieser Modus wird empfohlen, wenn im Stereo-Matching-Modul der Parameter `acquisition_mode` auf `SingleFrameOut1` (*Einzelbild+Out1*) gesetzt ist (*Parameter des Stereo-Matching-Moduls*, Abschnitt 6.2.2.1), und ein externer Projektor an den GPIO-Ausgang 1 angeschlossen ist, und wenn die Helligkeit der Szene zu jeder Zeit zur Belichtungszeitregelung berücksichtigt werden soll.Das ist zum Beispiel in Anwendungen mit veränderlichen äußeren Lichtbedingungen der Fall.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?exp_auto_mode=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_auto_mode=<value>
```

`exp_max` (*Maximale Belichtungszeit*)

Dieser Wert gibt die maximale Belichtungszeit im automatischen Modus in Sekunden an. Die tatsächliche Belichtungszeit wird automatisch angepasst, sodass das Bild korrekt belichtet wird. Sind die Bilder trotz maximaler Belichtungszeit noch immer unterbelichtet, erhöht der *rc_reason_stack* schrittweise die Verstärkung, um die Helligkeit der Bilder zu erhöhen. Es ist sinnvoll, die Belichtungszeit zu begrenzen, um die bei schnellen Bewegungen auftretende Bildunschärfe zu vermeiden oder zu verringern. Jedoch führt eine höhere Verstärkung auch zu mehr Bildrauschen. Welcher Kompromiss der beste ist, hängt immer auch von der Anwendung ab.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?exp_max=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_max=<value>
```

`exp_auto_average_max` (*Maximale Helligkeit*) und `exp_auto_average_min` (*Minimale Helligkeit*)

Die automatische Belichtungszeitsteuerung versucht die Belichtungszeit und den Verstärkungsfaktor so einzustellen, dass die mittlere Bildhelligkeit im Bild oder im *Bereich zur Regelung* zwischen der maximalen und minimalen Helligkeit liegt. Die maximale Helligkeit wird benutzt, wenn keine Bildteile in der Sättigung sind, d.h. keine Überbelichtung durch helle Oberflächen oder Reflexionen vorhanden sind. Falls Sättigungen auftreten, werden die Belichtungszeit und der Verstärkungsfaktor verringert, aber nur bis zur eingestellten minimalen Helligkeit.

Der Parameter für die maximale Helligkeit hat Vorrang über den Parameter der minimalen Helligkeit. Falls die minimale Helligkeit größer als die maximale ist, versucht die automatische Belichtungszeitsteuerung die mittlere Bildhelligkeit auf die maximale Helligkeit zu setzen.

Die aktuelle Helligkeit wird in der Statuszeile unter den Bildern angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?<exp_auto_
↪average_max|exp_auto_average_min>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?<exp_auto_average_max|exp_auto_
↪average_min>=<value>
```

`exp_offset_x`, `exp_offset_y`, `exp_width`, `exp_height` (*Bereich zur Regelung*)

Diese Werte definieren eine rechteckige Region im linken rektifizierten Bild, um den von der automatischen Belichtung überwachten Bereich zu limitieren. Die Belichtungszeit und der Verstärkungsfaktor werden so gewählt, dass die definierte Region optimal belichtet wird. Dies kann zu Über- oder Unterbelichtung in anderen Bildbereichen führen. Falls die Breite oder Höhe auf 0 gesetzt werden, dann wird das gesamte linke und rechte Bild von der automatischen Belichtungsfunktion berücksichtigt. Dies ist die Standardeinstellung.

Die Region wird in der Web GUI mit einem Rechteck im linken rektifizierten Bild visualisiert. Sie kann über Slider oder direkt im Bild mithilfe der Schaltfläche `Bereich im Bild auswählen` verändert werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?<exp_offset_
↪x|exp_offset_y|exp_width|exp_height>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?<exp_offset_x|exp_offset_y|exp_
↪width|exp_height>=<value>
```

`exp_value` (*Belichtungszeit*)

Dieser Wert gibt die Belichtungszeit im manuellen Modus in Sekunden an. Diese Belichtungszeit wird konstant gehalten, auch wenn die Bilder unterbelichtet sind.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?exp_value=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_value=<value>
```

`gain_value` (*Verstärkungsfaktor (dB)*)

Dieser Wert gibt den Verstärkungsfaktor im manuellen Modus in Dezibel an. Höhere Verstärkungswerte reduzieren die Belichtungszeit, führen aber zu Rauschen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?gain_value=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?gain_value=<value>
```

`brightness` (*Helligkeit*)

Das Anpassen der Helligkeit hellt das gesamte Bild auf oder verdunkelt es.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?brightness=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?brightness=<value>
```

`wb_auto` (*Weißabgleich Auto* oder *Manuell*, nur verfügbar für Farbkameras)

Dieser Wert kann auf *true* gesetzt werden, um den automatischen Weißabgleich anzuschalten. Bei *false* kann das Verhältnis der Farben manuell mit `wb_ratio_red` und `wb_ratio_blue` gesetzt werden. `wb_ratio_red` und `wb_ratio_blue` werden auf die letzten von der Automatik ermittelten Werte gesetzt, wenn diese abgeschaltet wird. Der Weißabgleich ist bei monochromen Kameras ohne Funktion und wird in diesem Fall in der Web GUI nicht angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?wb_auto=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?wb_auto=<value>
```

`wb_ratio_blue`, `wb_ratio_red` and `wb_ratio_green` (*Blauanteil*, *Rotanteil* und *Grünanteil*, nur verfügbar für Farbkameras)

Mit diesen Werten können der Blau-, Rot- und Grünanteil für einen manuellen Weißabgleich gesetzt werden. Der Weißabgleich ist bei monochromen Kameras ohne Funktion und wird in diesem Fall in der Web GUI nicht angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?<wb_ratio_blue|wb_ratio_
↪red|wb_ratio_green>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?<wb_ratio_blue|wb_ratio_red|wb_ratio_green>=
↪<value>
```

`light_source_preset` (*Voreinstellung Lichtquelle*, nur verfügbar für Farbkameras)

Mit dem Parameter `light_source_preset` können Farbverschiebungen korrigiert werden, die durch bestimmte Lichtquellen verursacht werden. Abhängig von der Farbtemperatur kann das für die Bildaufnahme verwendete Licht Farbverschiebungen im Bild verursachen. Diese Farbverschiebungen können durch Auswahl der entsprechenden Voreinstellung für die Lichtquelle korrigiert werden. Mögliche Werte sind: *Off*, *Tungsten*, *Daylight5000K*, *Daylight6500K* und *FactoryLED6000K*.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?light_source_
↪preset=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?light_source_preset=<value>
```

`saturation` (*Sättigung*, nur verfügbar für Farbkameras)

Durch Anpassen der Sättigung ändert sich die Buntheit (Intensität) der Farben. Eine höhere Sättigung erleichtert beispielsweise die Unterscheidung von Farben.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?saturation=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?saturation=<value>
```

6.1.4.2 Statuswerte

Das `rc_camera`-Modul meldet folgende Statuswerte für eine Pipeline vom Typ `stereo_ace`:

Tab. 6.4: Statuswerte des `rc_camera`-Moduls

Name	Beschreibung
`baseline`	Basisabstand t der Stereokamera in Metern
`brightness`	Aktuelle Helligkeit des Bildes als Wert zwischen 0 und 1
`color`	0 für monochrome Kameras, 1 für Farbkameras
`exp`	Aktuelle Belichtungszeit in Sekunden. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Belichtung (ms)* angezeigt.
`device_trigger_sources`	Angabe der verfügbaren Triggerquellen für den Fall, dass das Gerät getriggert werden kann
`focal`	Brennweitenfaktor, normalisiert auf eine Bildbreite von 1
`fps`	Aktuelle Bildwiederholrate der Kamerabilder in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als *FPS (Hz)* angezeigt.
`gain`	Aktueller Verstärkungsfaktor in Dezibel. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Verstärkung (dB)* angezeigt.
`gamma`	Aktueller Gammawert
`height`	Höhe des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als zweiter Teil von *Auflösung (px)* angezeigt.
`last_timestamp_grabbed`	Zeitstempel des letzten aufgenommenen Bildes, wenn die Kamera im Triggermodus ist
`out1_reduction`	Anteil der Helligkeits-Reduktion (0.0 - 1.0) für Bilder mit GPIO-Ausgang 1=LOW, wenn exp_auto_mode=AdaptiveOut1 oder exp_auto_mode=Out1High. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Out1 Reduktion (%)* angezeigt.
`params_override_active`	1 wenn die Parameter temporär durch einen laufenden Kalibrierprozess überschrieben werden
`selfcalib_counter`	Wie oft eine Korrektur durch die Selbstkalibrierung vorgenommen wurde
`selfcalib_offset`	Aktueller Offset, der durch die Selbstkalibrierung bestimmt wurde
`test`	0 for Live-Bilder und 1 für Test-Bilder
`width`	Breite des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als erster Teil von *Auflösung (px)* angezeigt.

6.1.4.3 Services

Das `rc_camera`-Modul bietet folgende Services für eine Pipeline vom Typ `stereo_ace`.

`acquisition_trigger`

triggert eine Bildaufnahme, wenn der Aufnahmemodus auf `Trigger` und die Triggerquelle auf `Software` gesetzt sind.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/acquisition_trigger
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/services/acquisition_trigger
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "acquisition_trigger",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.1.5 Pipelinetyp *orbbec*

Bemerkung: Die Firmwareversion der angeschlossenen *Orbbec* Kamera muss mindestens 1.6.00 sein, damit die Kamera genutzt werden kann.

6.1.5.1 Parameter

Das Kamera-Modul auf einer Pipeline vom Typ `orbbec` wird in der REST-API als `rc_camera` bezeichnet und in der *Web GUI* (Abschnitt 7.1) auf der Seite *Kamera* in der gewünschten Pipeline dargestellt. Der Benutzer kann die Stereo-Matching-Parameter entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.5: Laufzeitparameter des `rc_camera`-Moduls für eine Pipeline vom Typ `orbbec`

Name	Typ	Min.	Max.	Default	Beschreibung
`exp_control`	string	-	-	Auto	Art der Belichtungsregelung: [Manual, Auto]
`exp_height`	int32	0	799	0	Höhe der Region für automatische Belichtung, 0 für das ganze Bild
`exp_max`	float64	1.0	1999.0	665.0	Maximale Belichtungszeit in Sekunden im Auto Belichtungsmodus
`exp_offset_x`	int32	0	1279	0	Erste Spalte der Region für automatische Belichtung
`exp_offset_y`	int32	0	799	0	Erste Zeile der Region für automatische Belichtung
`exp_value`	float64	1.0	1999.0	156.0	Maximale Belichtungszeit in Sekunden im Auto Belichtungsmodus
`exp_width`	int32	0	1279	0	Breite der Region für automatische Belichtung, 0 für das ganze Bild
`gain_value`	float64	0.0	128.0	16.0	Verstärkung in Dezibel, wenn nicht im Auto Belichtungsmodus
`gamma`	float64	100.0	500.0	300.0	Gammafaktor
`wb_auto`	bool	false	true	true	Ein- und Ausschalten des manuellen Weißabgleichs (nur für Farbkameras)
`wb_value`	float64	2800.0	6500.0	4600.0	Weißabgleich Wert

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Seite *Tiefenbild* der Web GUI repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben und die Parameter werden in der Reihenfolge, in der sie in der Web GUI erscheinen, aufgelistet.

`gamma` (*Gamma*)

Der Gammawert bestimmt, wie das gemessene Licht auf die Helligkeit eines Pixels abgebildet wird. Kleinere Gammawerte lassen dunkle Bildbereiche heller erscheinen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?gamma=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?gamma=<value>
```

`exp_control` (Belichtung *Auto* oder *Manual*)

Die Belichtungsregelung kann auf *Auto* oder *Manual* gesetzt werden.

Auto: Dies ist der Standard Modus der die die Belichtungszeit und Verstärkung automatisch anpasst, um Unter- und Überbelichtung zu vermeiden. Wenn die Automatik abgeschaltet wird, werden *exp_value* und *gain_value* auf die letzten von der Automatik ermittelten Werte für Belichtungszeit und Verstärkung gesetzt.

Manual: Im manuellen Belichtungsmodus werden die Belichtungszeit und die Verstärkung konstant gehalten unabhängig von der resultierenden Bildhelligkeit.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?exp_control=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_control=<value>
```

`exp_max` (*Maximale Belichtung*)

Dieser Wert gibt die maximale Belichtung im automatischen Modus in Sekunden an. Die tatsächliche Belichtung wird automatisch angepasst, sodass das Bild korrekt belichtet wird. Sind die Bilder trotz maximaler Belichtung noch immer unterbelichtet, erhöht der *rc_reason_stack* schrittweise die Verstärkung, um die Helligkeit der Bilder zu erhöhen. Es ist sinnvoll, die Belichtung zu begrenzen, um die bei schnellen Bewegungen auftretende Bildunschärfe zu vermeiden oder zu verringern. Jedoch führt eine höhere Verstärkung auch zu mehr Bildrauschen. Welcher Kompromiss der beste ist, hängt immer auch von der Anwendung ab.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?exp_max=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_max=<value>
```

`exp_offset_x`, `exp_offset_y`, `exp_width`, `exp_height` (*Bereich zur Regelung*)

Diese Werte definieren eine rechteckige Region im linken rektifizierten Bild, um den von der automatischen Belichtung überwachten Bereich zu limitieren. Die Belichtungszeit und der Verstärkungsfaktor werden so gewählt, dass die definierte Region optimal belichtet wird.

Dies kann zu Über- oder Unterbelichtung in anderen Bildbereichen führen. Falls die Breite oder Höhe auf 0 gesetzt werden, dann wird das gesamte linke und rechte Bild von der automatischen Belichtungsfunktion berücksichtigt. Dies ist die Standardeinstellung.

Die Region wird in der Web GUI mit einem Rechteck im linken rektifizierten Bild visualisiert. Sie kann über Slider oder direkt im Bild mithilfe der Schaltfläche `Bereich im Bild auswählen` verändert werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?<exp_offset_
↪x|exp_offset_y|exp_width|exp_height>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?<exp_offset_x|exp_offset_y|exp_
↪width|exp_height>=<value>
```

`exp_value` (*Belichtungszeit*)

Dieser Wert gibt die Belichtung im manuellen Modus an. Diese Belichtung wird konstant gehalten, auch wenn die Bilder unterbelichtet sind.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?exp_value=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?exp_value=<value>
```

`gain_value` (*Verstärkungsfaktor*)

Dieser Wert gibt den Verstärkungsfaktor im manuellen Modus an. Höhere Verstärkungswerte reduzieren die Belichtungszeit, führen aber zu Rauschen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?gain_value=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?gain_value=<value>
```

`wb_auto` (*Weißabgleich Auto* oder *Manuell*)

Dieser Wert kann für den automatischen Weißabgleich auf *true* gesetzt werden. Ist dieser Wert *false*, wird der Weißabgleich über `wb_value` bestimmt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?wb_auto=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?wb_auto=<value>
```

`wb_value` (*Weißabgleich Manuell Wert*)

Dieser Wert bestimmt den Weißabgleich, wenn `wb_auto` auf *false* gesetzt ist.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?<wb_ratio_blue|wb_ratio_
↪red>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?<wb_ratio_blue|wb_ratio_red>=<value>
```

6.1.5.2 Statuswerte

Das `rc_camera`-Modul meldet die folgenden Statuswerte für eine Pipeline vom Typ `orbbec`:

Tab. 6.6: Statuswerte des `rc_camera`-Moduls

Name	Beschreibung
`baseline`	Intern angenommener Stereo-Basisabstand t in Metern zur Berechnung von Disparitätsbildern
`brightness`	Aktuelle Helligkeit als Wert zwischen 0 und 1
`color`	0 für monochrome Kameras, 1 für Farbkameras
`exp`	Aktuelle Belichtung. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Belichtung* angezeigt.
`focal`	Brennweitenfaktor, normalisiert auf eine Bildbreite von 1
`fps`	Aktuelle Bildwiederholrate der Kamerabilder in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als *FPS (Hz)* angezeigt.
`gain`	Aktueller Verstärkungsfaktor. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Verstärkung* angezeigt.
`height`	Höhe des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als zweiter Teil von *Auflösung (px)* angezeigt.
`last_capture_ok`	1 wenn die letzte Bildaufnahme erfolgreich war
`last_timestamp_grabbed`	Zeitstempel des letzten aufgenommenen Bildes
`test`	0 for Live-Bilder und 1 für Test-Bilder
`width`	Breite des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als erster Teil von *Auflösung (px)* angezeigt.

6.1.5.3 Services

In einer Pipeline vom Typ `orbbec` bietet das `rc_camera` Modul folgende Services.

`reset_defaults`

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.1.6 Pipelinetyp *zivid*

Bemerkung: Die Firmwareversion der angeschlossenen *zivid* Kamera muss der vom *rc_reason_stack* geforderten Firmwareversion entsprechen, sonst kann die *zivid* nicht verwendet werden. Um die *zivid* Firmware auf die benötigte Version zu aktualisieren, öffnen Sie die *Web GUI* (Abschnitt 7.1), navigieren Sie zu *System* → *Kamerapipelines* und wählen Sie die *zivid* Pipeline. Danach wird das Update durch Klicken auf *Zivid Firmware updaten* durchgeführt.

6.1.6.1 Benutzerdefinierte Voreinstellungen

Die *zivid* Kamera verfügt über mehrere vorkonfigurierte Einstellungen für die Bildaufnahme, sogenannte Presets. Die 2D-Presets sind speziell auf die 2D-Bildaufnahme zugeschnitten und beinhalten vor allem Einstellungen wie Auflösung, Belichtungszeit, Helligkeit und Verstärkung. Sie sind für Anwendungen optimiert, die detaillierte Farb- oder Monochrombilder erfordern.

Benutzer können mit der Software *Zivid Studio* (https://www.zivid.com/zivid-studio-software) auch eigene 2D-Voreinstellungen erstellen und als .yml-Dateien speichern. Diese Voreinstellungsdateien können auf der Seite *Kamera* der Web GUI auf den *rc_reason_stack* hochgeladen werden. Alternativ können Voreinstellungen über die REST-API verwaltet werden, wie in *Presets API* beschrieben. Benutzerdefinierte Voreinstellungen können dann wie die vordefinierten Voreinstellungen über den Laufzeitparameter `preset_name` für die Bildaufnahme ausgewählt werden. Auch 3D-Voreinstellungen mit 2D-Einstellungen können hochgeladen und als 2D-Voreinstellung verwendet werden. In diesem Fall werden nur die 2D-Einstellungen angewendet.

6.1.6.2 Parameter

Das Kamera-Modul auf einer Pipeline vom Typ `zivid` wird in der REST-API als `rc_camera` bezeichnet und in der *Web GUI* (Abschnitt 7.1) auf der Seite *Kamera* in der gewünschten Pipeline dargestellt. Der Benutzer kann die Stereo-Matching-Parameter entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.7: Laufzeitparameter des `rc_camera`-Moduls in einer Pipeline vom Typ `zivid`

Name	Typ	Min.	Max.	Default	Beschreibung
`acquisition_mode`	string	-	-	Trigger	Aufnahmemodus: [Continuous, Trigger]
`fps`	float64	1.0	25.0	25.0	Bildwiederholrate in Hertz
`preset_name`	string	-	-	-	Name der Voreinstellung

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Seite *Kamera* der Web GUI repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben und die Parameter werden in der Reihenfolge, in der sie in der Web GUI erscheinen, aufgelistet:

`acquisition_mode` (*Aufnahmemodus*)

Dieser Parameter bestimmt den Aufnahmemodus der 2D-Kamerabilder. Im Modus *Kontinuierlich* (`Continuous`) nimmt die Kamera Bilder mit der in `fps` angegebenen Bildwiederholrate auf. Im Modus *Trigger* (`Trigger`) werden nur Bilder aufgenommen, wenn die Kamera ein Software-Triggersignal empfängt, entweder durch Drücken des *Aufnehmen*-Knopfes in der Web GUI oder durch Aufrufen des Services `rc_camera/acquisition_trigger` (siehe *Services* (Abschnitt 6.1.6.4)).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/parameters?acquisition_
↪mode=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?acquisition_mode=<value>
```

`fps` (*FPS (Hz)*)

Dieser Wert bezeichnet die Bildwiederholrate der Kamera in Bildern pro Sekunde und begrenzt die Frequenz, mit der Kamerabilder aufgenommen werden können.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?fps=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?fps=<value>
```

`preset_name` (*Voreinstellung*)

Mit diesem Parameter kann eine Voreinstellung für die 2D-Bildaufnahme ausgewählt werden. Die Voreinstellung kann eine der vorkonfigurierten *zivid*-Voreinstellungen sein, die vom *zivid*-Modell abhängen und vom angeschlossenen Gerät gelesen werden, oder eine benutzerdefinierte Voreinstellung, die auf den *rc_reason_stack* hochgeladen wurde (siehe *Benutzerdefinierte Voreinstellungen*, Abschnitt 6.1.6.1).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/parameters?preset_name=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/parameters?preset_name=<value>
```

6.1.6.3 Statuswerte

Das `rc_camera`-Modul meldet die folgenden Statuswerte für eine Pipeline vom Typ `zivid`:

Tab. 6.8: Die Statuswerte des `rc_camera`-Moduls

Name	Beschreibung
`baseline`	Intern angenommener Stereo-Basisabstand t in Metern zur Berechnung von Disparitätsbildern
`brightness`	Aktuelle Helligkeit des Bildes als Wert zwischen 0 und 1
`color`	0 für monochrome Kameras, 1 für Farbkameras
`exp`	Aktuelle Belichtungszeit in Sekunden. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Belichtung (ms)* angezeigt.
`focal`	Brennweitenfaktor, normalisiert auf eine Bildbreite von 1
`fps`	Aktuelle Bildwiederholrate der Kamerabilder in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als *FPS (Hz)* angezeigt.
`height`	Höhe des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als zweiter Teil von *Auflösung (px)* angezeigt.
`last_capture_ok`	1 wenn die letzte Bildaufnahme erfolgreich war
`last_timestamp_grabbed`	Zeitstempel des letzten aufgenommenen Bildes
`test`	0 for Live-Bilder und 1 für Test-Bilder
`width`	Breite des Kamerabilds in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als erster Teil von *Auflösung (px)* angezeigt.

6.1.6.4 Services

In einer Pipeline vom Typ `zivid` bietet das `rc_camera` Modul folgende Services.

`acquisition_trigger`

triggert eine Bildaufnahme, wenn der Aufnahmemodus auf `Trigger` gesetzt sind.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/0/nodes/rc_camera/services/acquisition_trigger
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/services/acquisition_trigger
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "acquisition_trigger",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

reset_defaults‘

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_camera/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_camera/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.1.6.5 Presets API

Die 2D Voreinstellungen können über die folgenden REST-API Endpunkte gesetzt, abgefragt und gelöscht werden.

GET /presets/rc_zivid/2d_presets
Listet zivid 2D Voreinstellungen.

Musteranfrage

```
GET /api/v2/presets/rc_zivid/2d_presets HTTP/1.1
```

Antwort-Header

- Content-Type – application/json

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung

GET /presets/rc_zivid/2d_presets/{id}
Gibt eine zivid 2D Voreinstellungsdatei (yml) zurück.

Musteranfrage

```
GET /api/v2/presets/rc_zivid/2d_presets/<id> HTTP/1.1
```

Parameter

- **id** (*string*) – ID/Dateiname ohne Endung *(obligatorisch)*

Antwort-Header

- Content-Type – application/octet-stream

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung
- 404 Not Found – yml Datei nicht gefunden

PUT /presets/rc_zivid/2d_presets/{id}
Erstellt oder aktualisiert eine zivid 2D Voreinstellungsdatei.

Musteranfrage

```
PUT /api/v2/presets/rc_zivid/2d_presets/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Parameter

- **id** (*string*) – ID/Dateiname ohne Endung *(obligatorisch)*

Formularparameter

- **file** – yml Voreinstellungsdatei *(obligatorisch)*

Anfrage-Header

- Accept – multipart/form-data application/json

Antwort-Header

- Content-Type – application/json

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung

- 400 Bad Request – yml ist ungültig oder maximale Anzahl von Elementen erreicht
- 413 Request Entity Too Large – Datei zu groß

DELETE /presets/rc_zivid/2d_presets/{id}
Entfernt eine zivid 2D Voreinstellungsdatei

Musteranfrage

```
DELETE /api/v2/presets/rc_zivid/2d_presets/<id> HTTP/1.1
Accept: application/json
```

Parameter

- **id** (*string*) – ID/Dateiname ohne Endung *(obligatorisch)*

Anfrage-Header

- Accept – application/json

Antwort-Header

- Content-Type – application/json

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung
- 404 Not Found – Element nicht gefunden

6.2 3D-Module

Die 3D-Kamera-Software des *rc_reason_stack* enthält die folgenden Module:

- ***Stereo-Matching Modul*** **(`rc_stereomatching`, Abschnitt 6.2.2)** nutzt die rektifizierten Bildpaare der verbundenen Stereokamera, z.B. des *rc_visard*, um 3D-Tiefeninformationen, z.B. für Disparitäts-, Fehler- und Konfidenzbilder, zu berechnen. Dieses Modul läuft nur auf Pipelines für Stereokameras, d.h. `rc_visard`, `rc_viscore` und `stereo_ace`.
- ***Zivid Modul*** **(`rc_zivid`, Abschnitt 6.2.3)** stellt 3D-Tiefeninformationen, z.B. Disparitäts-, Fehler- und Konfidenzbilder, der angeschlossenen *zivid* Structured Light Kamera zur Verfügung. Dieses Modul läuft nur auf Pipelines vom Typ `zivid`.
- ***Orbbec Modul*** **(`rc_orbbec`, Abschnitt 6.2.4)** stellt 3D-Tiefeninformationen, z.B. Disparitäts-, Fehler- und Konfidenzbilder, der angeschlossenen *Orbbec* Gemini 335Le Stereokamera zur Verfügung. Dieses Modul läuft nur auf Pipelines vom Typ `orbbec`.

Diese Softwaremodule sind pipelinespezifisch, was heißt, dass sie innerhalb jeder Kamerapipeline laufen. Änderungen ihrer Einstellungen oder Parameter gelten nur für die zugehörige Pipeline und haben keinen Einfluss auf andere Kamerapipelines auf dem *rc_reason_stack*.

6.2.1 Anzeigen und Herunterladen von Tiefenbildern und Punktwolken

Der *rc_reason_stack* stellt zeitgestempelte Disparitäts-, Fehler- und Konfidenzbilder über *gRPC Bilddatenschnittstelle* (siehe Abschnitt 7.6) zur Verfügung.

Live-Streams in geringerer Qualität werden auf der *Tiefenbild* Seite in der gewünschten Pipeline in der *Web GUI* (Abschnitt 7.1) bereitgestellt.

Die Web GUI bietet weiterhin die Möglichkeit, einen Schnappschuss der aktuellen Szene mit den Tiefen-, Fehler und Konfidenzbildern, sowie der Punktwolke als .tar.gz-Datei zu speichern, wie in *Herunterladen von Kamerabildern* (Abschnitt 7.1.5) beschrieben wird.

6.2.2 Stereo-Matching Modul

Das Stereo-Matching-Modul ist ein Basismodul, das auf jedem *rc_reason_stack* verfügbar ist, und berechnet auf Grundlage des rektifizierten Stereobildpaars Disparitäts-, Fehler- und Konfidenzbilder.

Bemerkung: Dieses Modul ist nicht verfügbar in Kamerapipelines vom Typ `zivid` oder `orbbec`.

Um Disparitäts-, Fehler- und Konfidenzbilder in voller Auflösung zu berechnen, wird eine gesonderte StereoPlus *Lizenz* (Abschnitt 8.2) benötigt. Diese Lizenz ist auf jedem *rc_reason_stack* vorhanden, der nach dem 31.01.2019 gekauft wurde.

6.2.2.1 Parameter

Das Stereo-Matching-Modul wird in der REST-API als `rc_stereomatching` bezeichnet und in der *Web GUI* (Abschnitt 7.1) auf der Seite *Tiefenbild* in der gewünschten Pipeline dargestellt. Der Benutzer kann die Stereo-Matching-Parameter entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.9: Laufzeitparameter des `rc_stereomatching`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`acquisition_mode`	string	-	-	Continuous	Aufnahmemodus: [Continuous, SingleFrame, SingleFrameOut1]
`double_shot`	bool	false	true	false	Kombination zweier Disparitätsbilder von zwei Stereobildpaaren
`exposure_adapt_timeout`	float64	0.0	2.0	0.0	Maximale Zeit in Sekunden, die nach Auslösen einer Aufnahme im Einzelbild-Modus gewartet wird, bis die Belichtung angepasst ist
`fill`	int32	0	4	3	Disparitätstoleranz (für das Füllen von Löchern) in Pixeln
`maxdepth`	float64	0.1	100.0	100.0	Maximaler Abstand in Metern
`maxdeptherr`	float64	0.01	100.0	100.0	Maximaler Tiefenfehler in Metern
`minconf`	float64	0.5	1.0	0.5	Mindestkonfidenz
`mindepth`	float64	0.1	100.0	0.1	Minimaler Abstand in Metern
`quality`	string	-	-	High	Full (Voll), High (Hoch), Medium (Mittel), oder Low (Niedrig). Full benötigt eine ‚StereoPlus'-Lizenz.
`seg`	int32	0	4000	200	Mindestgröße der gültigen Disparitätssegmente in Pixeln
`smooth`	bool	false	true	true	Glättung von Disparitätsbildern (benötigt eine ‚StereoPlus'-Lizenz)
`static_scene`	bool	false	true	false	Mitteln von Bildern in statischen Szenen, um Rauschen zu reduzieren

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Seite *Tiefenbild* der Web GUI repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben und die Parameter werden in der Reihenfolge, in der sie in der Web GUI erscheinen, aufgelistet:

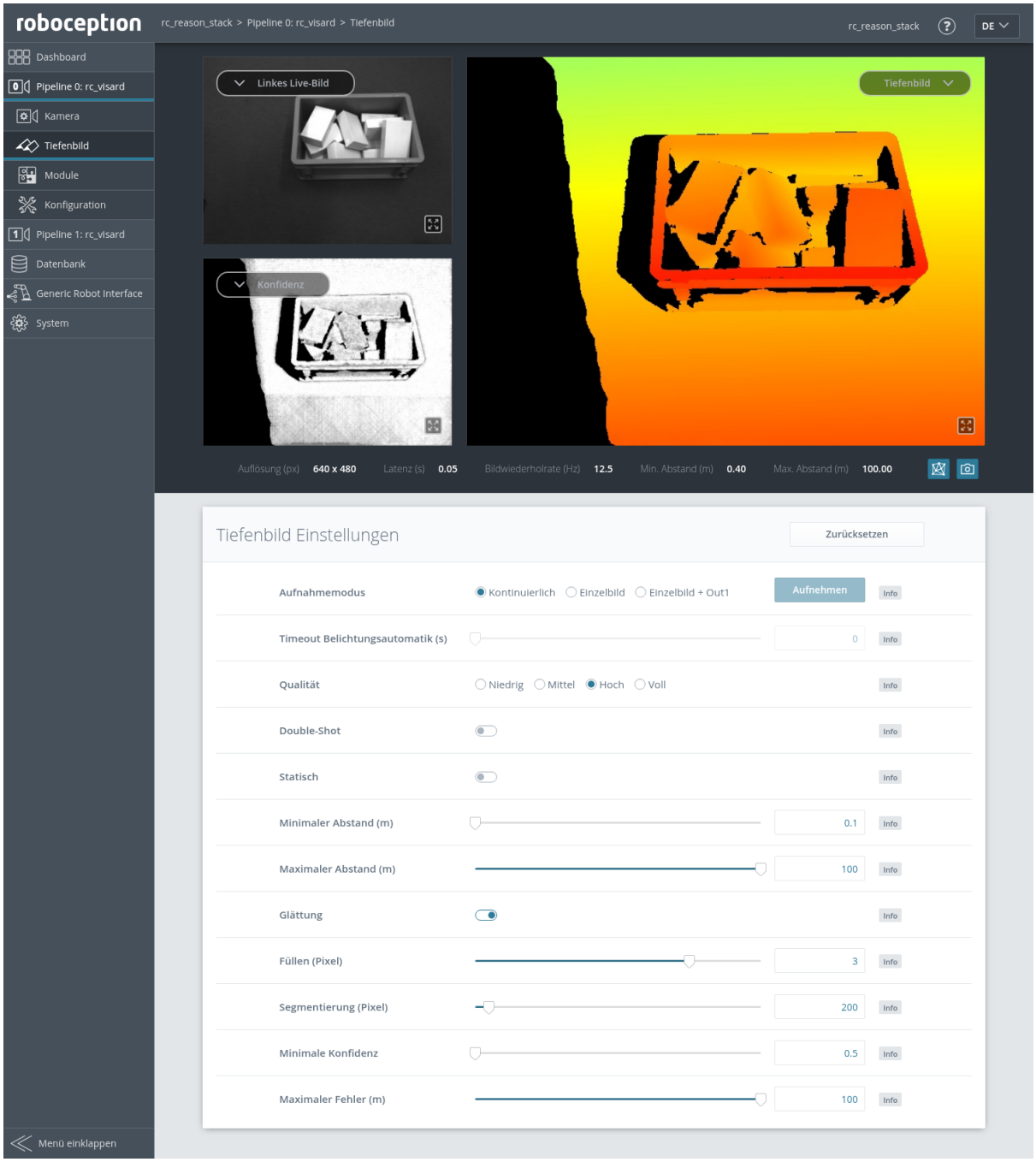


Abb. 6.3: Seite *Tiefenbild* der Web GUI

`acquisition_mode` (*Aufnahmemodus*)

Der Aufnahmemodus kann auf `Continuous` (*Kontinuierlich*), `SingleFrame` (*Einzelbild*) oder `SingleFrameOut1` (*Einzelbild + Out1*) eingestellt werden. *Kontinuierlich* ist die Standardeinstellung, bei der das Stereo-Matching kontinuierlich mit der vom Benutzer eingestellten Bildwiederholrate, entsprechend der verfügbaren Rechenressourcen, durchgeführt wird. Bei den beiden anderen Modi wird das Stereo-Matching bei jedem Drücken des *Aufnehmen*-Knopfes durchgeführt. Der *Einzelbild + Out1* Modus kontrolliert zusätzlich einen externen Projektor, falls dieser an GPIO-Ausgang 1 angeschlossen ist (*IOControl und Projektor-Kontrolle*, Abschnitt 6.4.4). In diesem Modus wird `out1_mode` des IOControl-Moduls automa-

tisch bei jedem Trigger auf `ExposureAlternateActive` und nach dem Aufnehmen der Bilder für das Stereo-Matching auf `Low` gesetzt.

Bemerkung: Der *Einzelbild + Out1* Modus kann nur dann über `out1_mode` einen Projektor steuern, wenn die IOControl-Lizenz auf dem *rc_reason_stack* verfügbar ist.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?
↪acquisition_mode=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?acquisition_mode=<value>
```

`exposure_adapt_timeout` (*Timeout Belichtungsautomatik*)

Der Timeout für die Belichtungsautomatik gibt die maximale Zeitspanne in Sekunden an, die das System nach dem Auslösen einer Bildaufnahme warten wird, bis die Belichtungsautomatik die optimale Belichtungszeit gefunden hat. Dieser Timeout wird nur im Modus `SingleFrame` (*Einzelbild*) oder `SingleFrameOut1` (*Einzelbild + Out1*) bei aktiver Belichtungsautomatik verwendet. Dieser Wert sollte erhöht werden, wenn in Anwendungen mit veränderlichen Lichtbedingungen Bilder unter- oder überbelichtet werden, und das resultierende Disparitätsbild nicht dicht genug ist. In diesem Fall werden mehrere Bilder aufgenommen, bis sich die Belichtungsautomatik angepasst hat oder der Timeout erreicht ist, und erst dann wird die eigentliche Bildaufnahme ausgelöst.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?
↪exposure_adapt_timeout=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?exposure_adapt_timeout=
↪<value>
```

`quality` (*Qualität*)

Disparitätsbilder lassen sich in verschiedenen Auflösungen berechnen: `Full` (*Voll*, volle Bildauflösung), `High` (*Hoch*, halbe Bildauflösung), `Medium` (*Mittel*, Viertel-Bildauflösung) und `Low` (*Niedrig*, Sechstel-Bildauflösung). Stereo-Matching mit voller Auflösung (`Full`) ist nur mit einer gültigen StereoPlus Lizenz möglich. Je niedriger die Auflösung, desto höher die Bildwiederholrate des Disparitätsbilds. Es ist zu beachten, dass die Bildwiederholrate der Disparitäts-, Konfidenz- und Fehlerbilder immer höchstens der Bildwiederholrate der Kamera entspricht. Falls die Projektoreinstellung `ExposureAlternateActive` ist, kann die Wiederholrate der Bilder höchstens die halbe Bildwiederholrate der Kamera sein.

Wenn volle Auflösung eingestellt ist, dann ist der mögliche Tiefenbereich intern limitiert, aufgrund von beschränktem on-board Speicherplatz. Es wird empfohlen

`mindepth` and `maxdepth` auf den Tiefenbereich anzupassen der für die Applikation benötigt wird.

Tab. 6.10: Auflösung des Tiefenbilds (Pixel) in Abhängigkeit von der gewählten Qualität

Verbundene Kamera	Volle Qualität (`Full`)	Hohe Qualität (`High`)	Mittlere Qualität (`Medium`)	Niedrige Qualität (`Low`)
rc_visard	1280 x 960	640 x 480	320 x 240	214 x 160
rc_visard_ng	1440 x 1080	720 x 540	360 x 270	240 x 180
rc_viscore	4112 x 3008	2056 x 1504	1028 x 752	686 x 502

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?
↪quality=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?quality=<value>
```

`double_shot` (*Double-Shot*)

Das Aktivieren dieses Modus führt zu dichteren Disparitätsbildern, aber einer erhöhten Verarbeitungszeit.

Bei Szenen, die mit einem Projektor im `Single + Out1` Modus oder im kontinuierlichen Modus mit der Projektoreinstellung `ExposureAlternateActive` aufgenommen werden, werden Löcher, die durch Projektor-Reflexionen verursacht werden, gefüllt mit Tiefeninformationen aus den Bildern ohne Projektormuster. In diesem Fall darf der `double_shot` Modus nur verwendet werden, wenn sich die Szene während der Aufnahme der Bilder nicht verändert.

Bei allen anderen Szenen werden Löcher im Disparitätsbild mit Tiefeninformationen aus demselben, herunterskalierten Bild gefüllt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?double_
↪shot=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?double_shot=<value>
```

`static_scene` (*Statisch*)

Mit dieser Option werden acht aufeinanderfolgende Kamerabilder vor dem Matching gemittelt. Dies reduziert Rauschen, was die Qualität des Stereo-Matching-Resultats verbessert. Allerdings erhöht sich auch die Latenz deutlich. Der Zeitstempel des ersten Bildes wird als Zeitstempel für das Disparitätsbild verwendet. Diese Option betrifft nur das Matching in voller und hoher Qualität. Sie darf nur verwendet werden, wenn sich die Szene während der Aufnahme der acht Bilder nicht verändert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?static_
↪scene=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?static_scene=<value>
```

`mindepth` (*Minimaler Abstand*)

Dieser Wert bezeichnet den geringsten Abstand zur Kamera, ab dem Messungen möglich sind. Größere Werte verringern implizit den Disparitätsbereich, wodurch sich auch die Rechenzeit verkürzt. Der minimale Abstand wird in Metern angegeben.

Abhängig von den Eigenschaften des Sensors kann der tatsächliche minimale Abstand größer sein als die Benutzereinstellung. Der tatsächliche minimale Abstand wird in den Statuswerten angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?
↪mindepth=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?mindepth=<value>
```

`maxdepth` (*Maximaler Abstand*)

Dieser Wert ist der größte Abstand zur Kamera, bis zu dem Messungen möglich sind. Pixel mit größeren Distanzwerten werden auf „ungültig“ gesetzt. Wird dieser Wert auf das Maximum gesetzt, so sind Abstände bis Unendlich möglich. Der maximale Abstand wird in Metern angegeben.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?
↪maxdepth=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?maxdepth=<value>
```

`smooth` (*Glättung*)

Diese Option aktiviert die Glättung von Disparitätswerten. Sie ist nur mit gültiger StereoPlus-Lizenz verfügbar.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?smooth=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?smooth=<value>
```

`fill` (*Füllen*)

Diese Option wird verwendet, um Löcher im Disparitätsbild durch Interpolation zu füllen. Der Füllwert gibt die maximale Disparitätsabweichung am Rand des Lochs an. Größere Füllwerte können die Anzahl an Löchern verringern, aber die interpolierten Werte können größere Fehler aufweisen. Maximal 5% der Pixel werden interpoliert. Kleine Löcher werden dabei bevorzugt interpoliert. Die Konfidenz für die interpolierten Pixel wird auf einen geringen Wert von 0,5 eingestellt. Das Auffüllen lässt sich deaktivieren, wenn der Wert auf 0 gesetzt wird.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?fill=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?fill=<value>
```

`seg` (*Segmentierung*)

Der Segmentierungsparameter wird verwendet, um die Mindestanzahl an Pixeln anzugeben, die eine zusammenhängende Disparitätsregion im Disparitätsbild ausfüllen muss. Isolierte Regionen, die kleiner sind, werden im Disparitätsbild auf ungültig gesetzt. Der Wert bezieht sich immer auf ein Disparitätsbild mit hoher Qualität (halbe Auflösung) und muss nicht verändert werden, wenn andere Qualitäten gewählt werden. Die Segmentierung eignet sich, um Disparitätsfehler zu entfernen. Bei größeren Werten kann es jedoch vorkommen, dass real vorhandene Objekte entfernt werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?seg=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?seg=<value>
```

`minconf` (*Minimale Konfidenz*)

Die minimale Konfidenz lässt sich einstellen, um potenziell falsche Disparitätsmessungen herauszufiltern. Dabei werden alle Pixel, deren Konfidenz unter dem gewählten Wert liegt, im Disparitätsbild auf „ungültig“ gesetzt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?
↪minconf=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?minconf=<value>
```

`maxdeptherr` (*Maximaler Fehler*)

Der maximale Fehler wird verwendet, um Messungen, die zu ungenau sind, herauszufiltern. Alle Pixel mit einem Tiefenfehler, der den gewählten Wert überschreitet, werden im Disparitätsbild auf „ungültig“ gesetzt. Der maximale Tiefenfehler wird in Metern angegeben. Der Tiefenfehler wächst in der Regel quadratisch mit dem Abstand eines Objekts zur Kamera (siehe *Konfidenz- und Fehlerbilder*, Abschnitt 4.2.3).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/parameters?
↪maxdeptherr=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/parameters?maxdeptherr=<value>
```

6.2.2.2 Statuswerte

Dieses Modul meldet folgende Statuswerte:

Tab. 6.11: Statuswerte des `rc_stereomatching`-Moduls

Name	Beschreibung
`fps`	Tatsächliche Bildwiederholrate der Disparitäts-, Fehler- und Konfidenzbilder. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Bildwiederholrate (Hz)* angezeigt.
`latency`	Zeit zwischen Bildaufnahme und Weitergabe des Disparitätsbildes in Sekunden. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Latenz (s)* angezeigt.
`width`	Aktuelle Breite von Disparitäts-, Fehler- und Konfidenzbild in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als erster Wert von *Auflösung (px)* angezeigt.
`height`	Aktuelle Höhe von Disparitäts-, Fehler- und Konfidenzbild in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als zweiter Wert von *Auflösung (px)* angezeigt.
`mindepth`	Tatsächlicher minimaler Arbeitsabstand in Metern. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Min. Abstand (m)* angezeigt.
`maxdepth`	Tatsächlicher maximaler Arbeitsabstand in Metern. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Max. Abstand (m)* angezeigt.
`time_matching`	Zeit in Sekunden für die Durchführung des Stereo-Matchings mittels *SGM* auf der GPU
`time_postprocessing`	Zeit in Sekunden für die Nachbearbeitung des Matching-Ergebnisses auf der CPU
`reduced_depth_range`	Gibt an, ob der Tiefenbereich aufgrund von Rechenressourcen verringert ist ist

6.2.2.3 Services

Das Stereo-Matching-Modul bietet folgende Services.

acquisition_trigger

signalisiert dem Modul, das Stereo-Matching auf den nächsten Stereobildern durchzuführen, falls `acquisition_mode` auf `SingleFrame` (*Einzelbild*) oder `SingleFrameOut1` (*Einzelbild+Out1*) eingestellt ist.

Details

Es wird ein Fehler zurückgegeben, falls `acquisition_mode` auf `Continuous` (*Kontinuierlich*) eingestellt ist.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/services/
↪acquisition_trigger
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/services/acquisition_trigger
```

Request

Dieser Service hat keine Argumente.

Response

Mögliche Rückgabewerte sind in der Tabelle unten aufgeführt.

Tab. 6.12: Mögliche Rückgabewerte des `acquisition_trigger` Serviceaufrufs.

Code	Beschreibung
0	Erfolgreich
-8	Triggern ist nur im Einzelbild-Modus möglich.
101	Triggern wird ignoriert, da bereits ein anderer Triggeraufruf stattfindet.
102	Triggern wird ignoriert, da keine Empfänger registriert sind.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "acquisition_trigger",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_stereomatching/services/reset_
↪defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_stereomatching/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.2.3 Zivid Modul

Bemerkung: Die Firmwareversion der angeschlossenen *zivid* Kamera muss der vom *rc_reason_stack* geforderten Firmwareversion entsprechen, sonst kann die *zivid* nicht verwendet werden. Um die *zivid* Firmware auf die benötigte Version zu aktualisieren, öffnen Sie die *Web GUI* (Abschnitt 7.1), navigieren Sie zu *System* → *Kamerapipelines* und wählen Sie die *zivid* Pipeline. Danach wird das Update durch Klicken auf *Zivid Firmware updaten* durchgeführt.

Das zivid Modul ist ein Basismodul, das auf jedem *rc_reason_stack* verfügbar ist und liefert Disparitäts-, Konfidenz- und Fehlerbilder einer angeschlossenen *zivid* Structured Light Kamera. Es läuft nur in Kamera-Pipelines vom Typ `zivid`.

6.2.3.1 Benutzerdefinierte Voreinstellungen

Die *zivid*-Kamera verfügt über zahlreiche vorkonfigurierte Einstellungen zur Bildaufnahme, sogenannte Presets.

Die mit der *zivid*-Kamera mitgelieferten 3D-Voreinstellungen umfassen sowohl 2D- als auch 3D-Einstellungen und ermöglichen die gleichzeitige Erfassung von Farbbildern und Tiefendaten. Die 2D-Bildeinstellungen werden jedoch ignoriert. Stattdessen wird das 2D-Bild mit der in `rc_camera` gewählten Voreinstellung aufgenommen (siehe *Benutzerdefinierte Voreinstellungen*, Abschnitt 6.1.6.1). Die 3D-Voreinstellungen sind nach Anwendungsanforderungen kategorisiert, z.B. *Consumer Goods*, *Manufacturing* usw.

Benutzer können mit der Software *Zivid Studio* (https://www.zivid.com/zivid-studio-software) auch eigene 3D-Presets erstellen und als .yml-Dateien speichern. Diese Preset-Dateien können auf der Seite *Tiefenbild* der Web GUI auf den *rc_reason_stack* hochgeladen werden. Alternativ können Voreinstellungen über die REST-API verwaltet werden, wie in *Presets API* beschrieben. Benutzerdefinierte Presets können dann wie die vordefinierten Presets über den Laufzeitparameter `preset_name` für die Tiefenbildaufnahme ausgewählt werden. Soll das im benutzerdefinierten 3D-Preset enthaltene 2D-Preset verwendet werden, muss dieses ebenfalls als 2D-Preset hochgeladen und als `preset_name` in `rc_camera` ausgewählt werden.

6.2.3.2 Parameter

Das zivid Modul wird in der REST-API als `rc_zivid` bezeichnet und in der *Web GUI* (Abschnitt 7.1) auf der Seite *Tiefenbild* in der gewünschten Pipeline dargestellt, wenn eine *zivid* Kamera an der entsprechenden Pipeline angeschlossen ist. Der Benutzer kann die zivid Parameter entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.13: Laufzeitparameter des `rc_zivid` Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`acquisition_mode`	string	-	-	SingleFrame	Aufnahmemodus: [Continuous, SingleFrame]
`maxdepth`	float64	0.3	100.0	100.0	Maximaler Abstand in Metern
`mindepth`	float64	0.3	100.0	0.3	Minimaler Abstand in Metern
`preset_name`	string	-	-	-	Name der Voreinstellung

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Seite *Tiefenbild* der Web GUI repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben und die Parameter werden in der Reihenfolge, in der sie in der Web GUI erscheinen, aufgelistet:

`acquisition_mode` (*Aufnahmemodus*)

Dieser Parameter bestimmt den Aufnahmemodus der 3D-Tiefenbilder. Der Aufnahmemodus kann auf *Kontinuierlich* (`Continuous`) oder *Einzelbild* (`SingleFrame`) gestellt werden. Letzteres ist der Standardwert, wobei ein Tiefenbild bei jedem Klick auf den *Aufnehmen*-Knopf oder beim Aufruf des Services `rc_zivid/acquisition_trigger` (siehe *Services des rc_zivid Moduls*, Abschnitt 6.2.3.4)) aufgenommen wird. Im Modus *Kontinuierlich* (`Continuous`) werden Tiefenbilder kontinuierlich aufgenommen, wenn auch der 2D Bildaufnahmemodus auf *Kontinuierlich* (`Continuous`) gestellt ist. Andernfalls werden Tiefenbilder nur aufgenommen, wenn eine 2D Bildaufnahme ausgelöst wird.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_zivid/parameters?acquisition_
↪mode=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_zivid/parameters?acquisition_mode=<value>
```

`preset_name` (*Voreinstellung*)

Mit diesem Parameter kann eine Voreinstellung für die 3D-Bildaufnahme ausgewählt werden. Die Voreinstellung kann eine der vorkonfigurierten *zivid*-Voreinstellungen sein, die vom *zivid*-Modell abhängen und vom angeschlossenen Gerät gelesen werden, oder eine benutzerdefinierte Voreinstellung, die auf den *rc_reason_stack* hochgeladen wurde (siehe *Benutzerdefinierte Voreinstellungen*, Abschnitt 6.2.3.1).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_zivid/parameters?preset_name=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_zivid/parameters?preset_name=<value>
```

6.2.3.3 Statuswerte

Das `rc_zivid` Modul meldet die folgenden Statuswerte:

Tab. 6.14: Die Statuswerte des `rc_zivid`-Moduls

Name	Beschreibung
`fps`	Aktuelle Bildwiederholrate der Disparitäts-, Fehler- und Konfidenzbilder in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als *FPS (Hz)* angezeigt.
`height`	Höhe der Disparitäts-, Fehler- und Konfidenzbilder in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als zweiter Teil von *Auflösung (px)* angezeigt.
`last_capture_ok`	1 wenn die letzte Bildaufnahme erfolgreich war
`last_timestamp_grabbed`	Zeitstempel des letzten aufgenommenen Tiefenbilds
`latency`	Aktuelle Bildwiederholrate der Disparitäts-, Fehler- und Konfidenzbilder in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als *FPS (Hz)* angezeigt.
`width`	Breite der Disparitäts-, Fehler- und Konfidenzbilder in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als erster Teil von *Auflösung (px)* angezeigt.

6.2.3.4 Services des rc_zivid Moduls

Das ‘‘rc_zivid‘ Modul bietet folgende Services.

`acquisition_trigger`

signalisiert dem Modul, ein Tiefenbild aufzunehmen, falls `acquisition_mode` auf `SingleFrame` (*Einzelbild*) eingestellt ist.

Details

Es wird ein Fehler zurückgegeben, falls `acquisition_mode` auf `Continuous` (*Kontinuierlich*) eingestellt ist.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_zivid/services/acquisition_
↪trigger
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_zivid/services/acquisition_trigger
```

Request

Dieser Service hat keine Argumente.

Response

Mögliche Rückgabewerte sind in der Tabelle unten aufgeführt.

Tab. 6.15: Mögliche Rückgabewerte des `acquisition_trigger` Serviceaufrufs.

Code	Beschreibung
0	Erfolgreich
-8	Triggern ist nur im Einzelbild-Modus möglich.
101	Triggern wird ignoriert, da bereits ein anderer Triggeraufruf stattfindet.
102	Triggern wird ignoriert, da keine Empfänger registriert sind.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "acquisition_trigger",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

reset_defaults‘

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_zivid/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_zivid/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.2.3.5 Presets API

Die 3D Voreinstellungen können über die folgenden REST-API Endpunkte gesetzt, abgefragt und gelöscht werden.

GET /presets/rc_zivid/3d_presets
Listet zivid 3D Voreinstellungen.

Musteranfrage

```
GET /api/v2/presets/rc_zivid/3d_presets HTTP/1.1
```

Antwort-Header

- Content-Type – application/json

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung

GET /presets/rc_zivid/3d_presets/{id}
Gibt eine zivid 3D Voreinstellungsdatei (yml) zurück.

Musteranfrage

```
GET /api/v2/presets/rc_zivid/3d_presets/<id> HTTP/1.1
```

Parameter

- **id** (*string*) – ID/Dateiname ohne Endung *(obligatorisch)*

Antwort-Header

- Content-Type – application/octet-stream

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung
- 404 Not Found – yml Datei nicht gefunden

PUT /presets/rc_zivid/3d_presets/{id}
Erstellt oder aktualisiert eine zivid 3D Voreinstellungsdatei.

Musteranfrage

```
PUT /api/v2/presets/rc_zivid/3d_presets/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Parameter

- **id** (*string*) – ID/Dateiname ohne Endung *(obligatorisch)*

Formularparameter

- **file** – yml Voreinstellungsdatei *(obligatorisch)*

Anfrage-Header

- Accept – multipart/form-data application/json

Antwort-Header

- Content-Type – application/json

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung

- 400 Bad Request – yml ist ungültig oder maximale Anzahl von Elementen erreicht
- 413 Request Entity Too Large – Datei zu groß

DELETE /presets/rc_zivid/3d_presets/{id}
Entfernt eine zivid 3D Voreinstellungsdatei

Musteranfrage

```
DELETE /api/v2/presets/rc_zivid/3d_presets/<id> HTTP/1.1
Accept: application/json
```

Parameter

- **id** (*string*) – ID/Dateiname ohne Endung *(obligatorisch)*

Anfrage-Header

- Accept – application/json

Antwort-Header

- Content-Type – application/json

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung
- 404 Not Found – Element nicht gefunden

6.2.4 Orbbec Modul

Bemerkung: Die Firmwareversion der angeschlossenen *Orbbec* Kamera muss mindestens 1.6.00 sein, damit die Kamera genutzt werden kann.

Das Orbbec Modul ist ein Basismodul, das auf jedem *rc_reason_stack* verfügbar ist und liefert Disparitäts-, Konfidenz- und Fehlerbilder einer angeschlossenen *Orbbec* Stereokamera. Es läuft nur in Kamera-Pipelines vom Typ `orbbec`.

6.2.4.1 Parameter

Das Orbbec Modul wird in der REST-API als `rc_orbbec` bezeichnet und in der *Web GUI* (Abschnitt 7.1) auf der Seite *Tiefenbild* in der gewünschten Pipeline dargestellt, wenn eine *Orbbec* Kamera an der entsprechenden Pipeline angeschlossen ist. Der Benutzer kann die Orbbec Parameter entweder dort oder über die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.16: Laufzeitparameter des `rc_orbbec` Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`fill`	int32	0	4	3	Disparitätstoleranz (für das Füllen von Löchern) in Pixeln
`maxdepth`	float64	0.1	100.0	100.0	Maximaler Abstand in Metern
`mindepth`	float64	0.1	100.0	0.1	Minimaler Abstand in Metern
`seg`	int32	0	4000	200	Mindestgröße der gültigen Disparitätssegmente in Pixeln
`smooth`	bool	false	true	true	Glättung von Disparitätsbildern (benötigt eine ‚StereoPlus'-Lizenz)

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile auf der Seite *Tiefenbild* der Web GUI repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben und die Parameter werden in der Reihenfolge, in der sie in der Web GUI erscheinen, aufgelistet:

`mindepth` (*Minimaler Abstand*)

Dieser Wert bezeichnet den geringsten Abstand zur Kamera, ab dem Messungen möglich sind. Der minimale Abstand wird in Metern angegeben.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_orbbec/parameters?mindepth=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_orbbec/parameters?mindepth=<value>
```

`maxdepth` (*Maximaler Abstand*)

Dieser Wert ist der größte Abstand zur Kamera, bis zu dem Messungen möglich sind. Pixel mit größeren Distanzwerten werden auf „ungültig" gesetzt. Wird dieser Wert auf das Maximum gesetzt, so sind Abstände bis Unendlich möglich. Der maximale Abstand wird in Metern angegeben.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_orbbec/parameters?maxdepth=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_orbbec/parameters?maxdepth=<value>
```

`smooth` (*Glättung*)

Diese Option aktiviert die Glättung von Disparitätswerten.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_orbbec/parameters?smooth=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_orbbec/parameters?smooth=<value>
```

`fill` (*Füllen*)

Diese Option wird verwendet, um Löcher im Disparitätsbild durch Interpolation zu füllen. Der Füllwert gibt die maximale Disparitätsabweichung am Rand des Lochs an. Größere Füllwerte können die Anzahl an Löchern verringern, aber die interpolierten Werte können größere Fehler aufweisen. Maximal 5% der Pixel werden interpoliert. Kleine Löcher werden dabei bevorzugt interpoliert. Die Konfidenz für die interpolierten Pixel wird auf einen geringen Wert von 0,5 eingestellt. Das Auffüllen lässt sich deaktivieren, wenn der Wert auf 0 gesetzt wird.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_orbbec/parameters?fill=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_orbbec/parameters?fill=<value>
```

`seg` (*Segmentierung*)

Der Segmentierungsparameter wird verwendet, um die Mindestanzahl an Pixeln anzugeben, die eine zusammenhängende Disparitätsregion im Disparitätsbild ausfüllen muss. Isolierte Regionen, die kleiner sind, werden im Disparitätsbild auf ungültig gesetzt. Der Wert bezieht sich immer auf ein Disparitätsbild mit hoher Qualität (halbe Auflösung) und muss nicht verändert werden, wenn andere Qualitäten gewählt werden. Die Segmentierung eignet sich, um Disparitätsfehler zu entfernen. Bei größeren Werten kann es jedoch vorkommen, dass real vorhandene Objekte entfernt werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_orbbec/parameters?seg=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_orbbec/parameters?seg=<value>
```

6.2.4.2 Statuswerte

Das `rc_orbbec` Modul meldet die folgenden Statuswerte:

Tab. 6.17: Die Statuswerte des `rc_orbbec`-Moduls

Name	Beschreibung
`fps`	Aktuelle Bildwiederholrate der Disparitäts-, Fehler- und Konfidenzbilder in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als *FPS (Hz)* angezeigt.
`height`	Höhe der Disparitäts-, Fehler- und Konfidenzbilder in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als zweiter Teil von *Auflösung (px)* angezeigt.
`last_capture_ok`	1 wenn die letzte Bildaufnahme erfolgreich war
`last_timestamp_grabbed`	Zeitstempel des letzten aufgenommenen Tiefenbilds
`latency`	Aktuelle Bildwiederholrate der Disparitäts-, Fehler- und Konfidenzbilder in Hertz. Dieser Wert wird unter der Bildvorschau in der Web GUI als *FPS (Hz)* angezeigt.
`mindepth`	Tatsächlicher minimaler Arbeitsabstand in Metern. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Min. Abstand (m)* angezeigt.
`maxdepth`	Tatsächlicher maximaler Arbeitsabstand in Metern. Dieser Wert wird unter der Bildvorschau in der Web GUI als *Max. Abstand (m)* angezeigt.
`width`	Breite der Disparitäts-, Fehler- und Konfidenzbilder in Pixeln. Dieser Wert wird unter der Bildvorschau in der Web GUI als erster Teil von *Auflösung (px)* angezeigt.

6.2.4.3 Services des rc_orbbec Moduls

Das ‘‘rc_orbbec‘ Modul bietet folgende Services.

reset_defaults‘

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_orbbec/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_orbbec/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.3 Detektions- und Messmodule

Der *rc_reason_stack* bietet Softwaremodule für unterschiedliche Detektions- und Messanwendungen:

- ***Measure*** **(`rc_measure`, Abschnitt 6.3.1)** ermöglicht die Messung von Tiefenwerten
- ***LoadCarrier*** **(`rc_load_carrier`, Abschnitt 6.3.2)** ermöglicht die Erkennung von Load Carriern (Behältern) und ihres Füllstands.
- ***TagDetect*** **(`rc_april_tag_detect` und `rc_qr_code_detect`, Abschnitt 6.3.3)** ermöglicht die Erkennung von AprilTags und QR-Codes sowie die Schätzung von deren Pose.
- ***ItemPick und ItemPickAI*** **(`rc_itempick`, Abschnitt 6.3.4)** bietet eine Standardlösung für robotische Pick-and-Place-Anwendungen für Objekte einer Objektkategorie oder unbekannte Objekte.
- ***BoxPick*** **(`rc_boxpick`, Abschnitt 6.3.5)** bietet eine Standardlösung für robotische Pick-and-Place-Anwendungen für Boxen oder texturierte Boxen.
- ***SilhouetteMatch und SilhouetteMatchAI*** **(`rc_silhouettematch`, Abschnitt 6.3.6)** bietet eine Objekterkennungslösung für Objekte, die sich auf einer Ebene befinden, oder gestapelte planare Objekt.
- ***CADMatch*** **(`rc_cadmatch`, Abschnitt 6.3.7)** bietet eine Objekterkennungslösung für 3D-Objekte.

Diese Softwaremodule sind pipelinespezifisch, was heißt, dass sie innerhalb jeder Kamerapipeline laufen. Änderungen ihrer Einstellungen oder Parameter gelten nur für die zugehörige Pipeline und haben keinen Einfluss auf andere Kamerapipelines auf dem *rc_reason_stack*.

Diese Module sind optional und können durch Kauf einer separaten *Lizenz* (Abschnitt 8.2) aktiviert werden.

6.3.1 Measure

6.3.1.1 Einleitung

Das Measure Modul ermöglicht die Messung von Tiefenwerten in einer Region of Interest.

Das Measure Modul ist ein Basismodul, welches auf jedem *rc_reason_stack* verfügbar ist.

6.3.1.2 Tiefe messen

Das Measure Modul bietet Funktionen zum Messen von Tiefenwerten in der aktuellen Szene in einer 2D Region of Interest. Optional kann die Region of Interest in bis zu 100 Zellen unterteilt werden, für die separate Tiefenmessungen erfolgen, die zusätzlich zu den Tiefenmessungen für die gesamte Region of Interest zurückgegeben werden.

Die Tiefenmessung besteht aus der durchschnittlichen Tiefe `mean_z`, der minimalen Tiefe `min_z` und der maximalen Tiefe `max_z`, die jeweils 3D-Koordinaten enthalten. Die Koordinaten der Messungen `min_z` und `max_z` entsprechen dem Punkt in der Zelle oder der gesamten Region of Interest mit dem minimalen bzw. maximalen Abstand von der Kamera. Die `x`- und `y`-Koordinaten der `mean_z`-Messungen definieren einen Punkt in der Mitte der Zelle oder der gesamten Region of Interest und die `z`-Koordinate wird aus dem Durchschnitt aller Tiefenwerte (Entfernungen von der Kamera) in diesem Bereich bestimmt. Darüber hinaus wird für jede Zelle und die gesamte Region of Interest ein `coverage` Wert zurückgegeben. Dabei handelt es sich um eine Zahl zwischen 0 und 1, die den Anteil der gültigen Tiefenwerte innerhalb der jeweiligen Region widerspiegelt. Ein `coverage` Wert von 0 bedeutet, dass die Zelle ungültig ist und kein Tiefenwert berechnet werden konnte.

Wenn als Referenzkoordinatensystem (`pose_frame`) `external` für die Tiefenmessungen verwendet wird, werden alle 3D-Koordinaten wie oben beschrieben berechnet, dann aber in das externe Koordinatensystem transformiert. Das heißt, die Tiefe wird immer entlang der Sichtlinie der Kamera gemessen, unabhängig vom gewählten Referenzkoordinatensystem.

6.3.1.3 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_reason_stack* laufenden Module liefern Daten für das Measure Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieser Module kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des Measure Moduls haben.

Tiefenbilder

Folgende Daten werden vom Measure Modul verarbeitet:

- Die Disparitätsbilder des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 6.2.2), falls eine Stereokamera verwendet wird
- Die Disparitätsbilder der *Orbbec Modul* (`rc_orbbec`, Abschnitt 6.2.4), falls eine *Orbbec* Kamera verwendet wird
- die Disparitätsbilder des *Zivid Modul* (`rc_zivid`, Abschnitt 6.2.3), falls eine *zivid* Kamera verwendet wird

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das Measure Modul automatisch Punkte im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 6.3.1.6) kann das Koordinatensystem der berechneten Punkte mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Punkte sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Punkte der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Punkte sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Measure Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

6.3.1.4 Parameter

Das Measure Modul wird in der REST-API als `rc_measure` bezeichnet und in der *Web GUI* (Abschnitt 7.1) in der gewünschten Pipeline unter *Module* → *Measure* dargestellt.

Übersicht über die Parameter

Dieses Modul besitzt keine Laufzeitparameter.

6.3.1.5 Statuswerte

Das Measure Modul meldet folgende Statuswerte:

Tab. 6.18: Statuswerte des `rc_measure` Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Tiefenbilddaten gewartet werden musste
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Tiefenbilds
`processing_time`	Berechnungszeit für die letzte Messung in Sekunden

6.3.1.6 Services

Die angebotenen Services des Measure Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der *rc_reason_stack Web GUI* (Abschnitt 7.1) auf der Seite *Measure* unter dem Menüpunkt *Module* ausprobiert und getestet werden.

Das Measure Modul stellt folgende Services zur Verfügung.

`measure_depth`

Berechnet die durchschnittliche, minimale und maximale Tiefe in einer angegebenen Region of Interest, die optional in Zellen unterteilt werden kann.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_measure/services/measure_depth
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_measure/services/measure_depth
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.1.3).

Optionale Serviceargumente:

`region_of_interest_2d_id`: Die ID der 2D Region of Interest (siehe *RoiDB*, Abschnitt 6.5.2), innerhalb welcher die Tiefenmessung durchgeführt werden soll.

`region_of_interest_2d` ist eine alternative Definition der Region of Interest für die Tiefenmessung. Diese Region of Interest wird ignoriert, falls eine `region_of_interest_2d_id` gesetzt ist. Die Region of Interest wird immer auf dem Kamerabild mit voller Auflösung definiert, wobei `offset_x` und `offset_y` die Pixelkoordinaten der oberen linken Ecke der rechteckigen Region of Interest sind, und `width` und `height` die Breite und Höhe des Rechtecks in Pixeln angeben. Der Standardwert ist eine Region of Interest, die das gesamte Bild abdeckt.

`cell_count` ist die Anzahl der Zellen in x und y Richtung, in die die Region of Interest für die Tiefenmessung unterteilt wird. Falls nicht angegeben, wird ein `cell_count` von 0, 0 angenommen und es werden nur die Gesamtwerte `overall` berechnet. Die Gesamtanzahl der Zellen, die als Produkt aus den x und y Werten des `cell_count` berechnet werden kann, darf nicht größer sein als 100.

`data_acquisition_mode`: Falls der Aufnahmemodus auf `CAPTURE_NEW` (Standardwert) gesetzt ist, wird ein neuer Bild-Datensatz für die Messung aufgenommen. Falls der Modus auf `USE_LAST` gesetzt wird, wird der Datensatz der vorherigen Messung erneut verwendet.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.1.3).

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "cell_count": {
      "x": "uint32",
      "y": "uint32"
    },
    "data_acquisition_mode": "string",
    "pose_frame": "string",
    "region_of_interest_2d": {
      "height": "uint32",
      "offset_x": "uint32",
      "offset_y": "uint32",
      "width": "uint32"
    },
    "region_of_interest_2d_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

Response

`cells` enthält die Tiefenmessungen aller gewünschter Zellen. Die Zellen sind immer von links nach rechts und oben nach unten in Bildkoordinaten sortiert.

`overall` enthält die Tiefenmessung der gesamten Region of Interest.

`coverage` enthält den Anteil der Pixel mit gültigen Tiefenmesswerten, wie in *Tiefe messen* (Abschnitt 6.3.1.2) beschrieben.

`mean_z`, `min_z` und `max_z` enthalten die gemessenen Koordinaten wie in *Tiefe messen* (Abschnitt 6.3.1.2).

`region_of_interest_2d` gibt die Definition der angeforderten Region of Interest für die Tiefenmessung zurück.

`pose_frame` enthält das Koordinatensystem der Tiefenmessungen.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "measure_depth",
  "response": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "cell_count": {
      "x": "uint32",
      "y": "uint32"
    },
    "cells": [
      {
        "coverage": "float64",
        "max_z": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "mean_z": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "min_z": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    ],
    "overall": {
      "coverage": "float64",
      "max_z": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "mean_z": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "min_z": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "pose_frame": "string",
    "region_of_interest_2d": {
      "height": "uint32",
      "offset_x": "uint32",
      "offset_y": "uint32",
      "width": "uint32"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

trigger_dump

speichert die Detektion auf dem angeschlossenen USB Speicher, die dem übergebenen Zeitstempel entspricht, oder die letzte, falls kein Zeitstempel angegeben wurde.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_measure/services/trigger_dump
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_measure/services/trigger_dump
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "comment": "string",
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "trigger_dump",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.3.1.7 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 6.19: Rückgabecodes der Services des Measure Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)

6.3.2 LoadCarrier

6.3.2.1 Einleitung

Das LoadCarrier Modul ermöglicht die Erkennung von Load Carriern, was oftmals der erste Schritt für die Erkennung von Objekten oder Berechnung von Greifpunkten in einem Behälter ist. Die Modelle der zu erkennenden Load Carrier müssen im *LoadCarrierDB* (Abschnitt 6.5.1) Modul definiert werden.

Das LoadCarrier Modul ist ein optionales Modul, welches intern auf dem *rc_reason_stack* läuft, und ist freigeschaltet, sobald eine gültige Lizenz für eines der Module *ItemPick und ItemPickAI* (Abschnitt 6.3.4) und *BoxPick* (Abschnitt 6.3.5) oder *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6) vorhanden ist. Andernfalls benötigt es eine gesonderte LoadCarrier *Lizenz* (Abschnitt 8.2), welche erworben werden muss.

Bemerkung: Dieses Softwaremodul ist pipelinespezifisch. Änderungen seiner Einstellungen oder Parameter betreffen nur die zugehörige Kamerapipeline und haben keinen Einfluss auf die anderen Pipelines, die auf dem *rc_reason_stack* laufen.

6.3.2.2 Erkennung von Load Carriern

Der Algorithmus zur Erkennung von Load Carriern detektiert Load Carrier, die einem speziellen Load Carrier Modell entsprechen, welches im *LoadCarrierDB* (Abschnitt 6.5.1) Modul definiert werden muss. Das Load Carrier Modell wird über seine ID referenziert, die bei der Load Carrier Detektion übergeben wird. Die Erkennung von Load Carriern basiert auf der Erkennung des rechteckigen Load Carrier Rands. Dazu werden detektierte Linien im linken Kamerabild und die Tiefenwerte des Load Carrier Randes genutzt. Daher sollte der Rand einen Kontrast zum Hintergrund bilden und das Disparitätsbild auf dem Rand dicht sein.

Wenn mehrere Load Carrier mit der angegeben Load Carrier ID in der Szene sichtbar sind, werden alle von ihnen detektiert und zurückgeliefert.

Standardmäßig, wenn `assume_gravity_aligned` aktiv ist und Gravitationsmessungen verfügbar sind, wird nach Load Carriern gesucht, deren Randebene senkrecht zur gemessenen Gravitationsrichtung ausgerichtet ist. Um geneigte Load Carrier zu erkennen, muss `assume_gravity_aligned` deaktiviert werden oder deren ungefähre Orientierung als Pose `pose` in einem Referenzkoordinatensystem `pose_frame` angegeben werden, und der Posentyp `pose_type` auf `ORIENTATION_PRIOR` gesetzt werden.

Load Carrier können höchstens bis zu einer Entfernung von 3 Metern von der Kamera erkannt werden.

Wenn eine 3D Region of Interest (siehe *RoiDB*, Abschnitt 6.5.2) genutzt wird, um das Volumen für die Load Carrier Erkennung einzuschränken, müssen nur die Ränder der Load Carrier vollständig in der Region of Interest enthalten sein.

Die Erkennung liefert die Posen der Load Carrier Koordinatensysteme (siehe *Load Carrier Definition*, Abschnitt 6.5.1.2) im gewünschten Referenzkoordinatensystem zurück.

Bei der Erkennung wird auch ermittelt, ob die Load Carrier überfüllt (`overfilled`) sind, was bedeutet, dass Objekte aus dem Load Carrier herausragen.

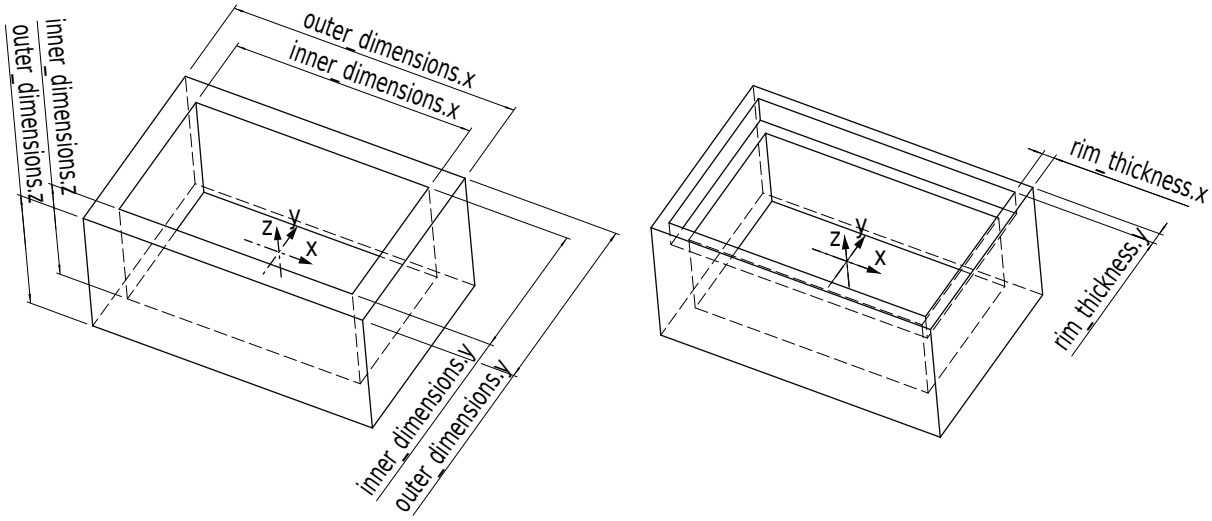


Abb. 6.4: Illustration verschiedener Load Carrier Modelle und deren Koordinatensysteme

6.3.2.3 Füllstandserkennung

Das LoadCarrier Modul bietet den Service `detect_filling_level` an, um den Füllstand aller erkannten Load Carrier zu berechnen.

Dazu werden die Load Carrier in eine konfigurierbare Anzahl von Zellen unterteilt, welche in einem 2D-Raster angeordnet sind. Die maximale Anzahl beträgt 200x200 Zellen. Für jede Zelle werden folgende Werte ermittelt:

- `level_in_percent`: Minimum, Maximum und Mittelwert des Füllstands vom Boden in Prozent. Diese Werte können größer als 100% sein, falls die Zelle überfüllt ist.
- `level_free_in_meters`: Minimum, Maximum und Mittelwert in Metern des freien Teils der Zelle vom Rand des Load Carriers gemessen. Diese Werte können negativ sein, falls die Zelle überfüllt ist.
- `cell_size`: Abmessungen der 2D-Zelle in Metern.
- `cell_position`: Position des Mittelpunkts der Zelle in Metern (entweder im Koordinatensystem `camera` oder `external`, siehe *Hand-Auge-Kalibrierung*, Abschnitt 6.3.4.4). Die z-Koordinate liegt auf der Ebene des Load Carrier Randes.
- `coverage`: Anteil der gültigen Pixel in dieser Zelle. Dieser Wert reicht von 0 bis 1 in Schritten von 0.1. Ein niedriger Wert besagt, dass die Zelle fehlende Daten beinhaltet (d.h. nur wenige Punkte konnten in der Zelle gemessen werden).

Diese Werte werden auch für den gesamten Load Carrier berechnet. Falls keine Zellunterteilung angegeben ist, wird nur der Gesamtfüllstand (`overall_filling_level`) berechnet.

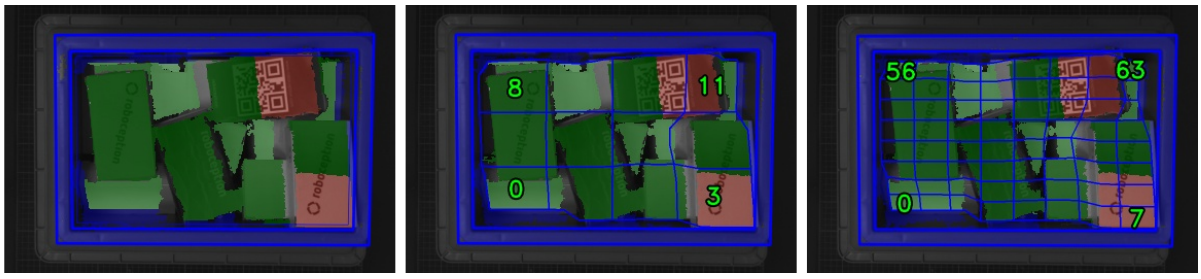


Abb. 6.5: Visualisierungen des Behälterfüllstands: Gesamtfüllstand ohne Raster (links), 4x3 Raster (Mitte), 8x8 Raster (rechts). Der Inhalt wird mit einem Farbverlauf von weiß (auf dem Boden) nach dunkelgrün dargestellt. Die überfüllten Bereiche sind rot dargestellt. Die Nummern stellen die Zell-IDs dar.

6.3.2.4 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_reason_stack* laufenden Module liefern Daten für das LoadCarrier Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieser Module kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des LoadCarrier Moduls haben.

Kamera- und Tiefendaten

Folgende Daten werden vom LoadCarrier Modul verarbeitet:

- Die rektifizierten Bilder des *Kamera Modul* (`rc_camera`, Abschnitt 6.1);
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 6.2.2), falls eine Stereokamera verwendet wird.
- die Disparitäts-, Konfidenz- und Fehlerbilder der *Orbbec Modul* (`rc_orbbec`, Abschnitt 6.2.4), falls eine *Orbbec* Kamera verwendet wird
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Zivid Modul* (`rc_zivid`, Abschnitt 6.2.3), falls eine *zivid* Kamera verwendet wird

Für alle genutzten Bilder ist garantiert, dass diese nach dem Auslösen des Services aufgenommen wurden.

IOControl und Projektor-Kontrolle

Für den Anwendungsfall, dass der *rc_reason_stack* zusammen mit einem externen Musterprojektor und dem Modul für *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 6.4.4) betrieben wird, wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 6.2.2.1), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 6.4.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren.

Darüber hinaus sind keine weiteren Änderungen für diesen Anwendungsfall notwendig.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann die Load Carrier Komponente automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 6.3.2.7) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Load Carrier Modul alle notwendigen

Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

6.3.2.5 Parameter

Das LoadCarrier Modul wird in der REST-API als `rc_load_carrier` bezeichnet und in der *Web GUI* (Abschnitt 7.1) in der gewünschten Pipeline unter *Module* → *LoadCarrier* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 7.2) ändern.

Übersicht über die Parameter

Bemerkung: Die Defaultwerte in der Tabelle unten zeigen die Werte des *rc_visard*. Diese Werte können sich bei anderen Sensoren unterscheiden.

Dieses Modul bietet folgende Laufzeitparameter:

Tab. 6.20: Laufzeitparameter des `rc_load_carrier` Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`assume_gravity_aligned`	bool	false	true	true	Wenn dieser Parameter aktiv ist, werden nur waagerechte Load Carrier erkannt falls eine Gravitationsmessung verfügbar ist.
`crop_distance`	float64	0.0	0.05	0.005	Sicherheitsspielraum um den das Load Carrier Innenmaß verringert wird, um eine Region of Interest für die Erkennung zu definieren
`min_plausibility`	float64	0.0	0.99	0.8	Gibt an, wie viel von der Ebene um den Load Carrier Rand herum mindestens frei sein muss, um als gültige Erkennung zu zählen.
`model_tolerance`	float64	0.003	0.025	0.008	Gibt an, wie weit die Abmessungen des Load Carriers von den Werten im Modell abweichen dürfen

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden in der Web GUI zeilenweise im Abschnitt *LoadCarrier Einstellungen* auf der Seite *LoadCarrier* unter *Module* dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet. Wenn die Parameter außerhalb des `rc_load_carrier` Moduls über die *REST-API-Schnittstelle* (Abschnitt 7.2) eines anderen Moduls verwendet werden, sind sie durch den Präfix `load_carrier_` gekennzeichnet.

`assume_gravity_aligned` (*Gravitationsausgerichtet*)

Wenn dieser Parameter aktiv ist, werden nur waagerechte Load Carrier erkannt. Dies kann die Erkennung beschleunigen. Wenn dieser Parameter nicht aktiv ist, werden auch Load Carrier mit Verkippung detektiert.

Für Load Carrier mit einem Orientierungsprior wird dieser Parameter ignoriert.

Bemerkung: Die Ausrichtung an der Gravitation ist nur für Kamerapipelines vom Typ `rc_visard` verfügbar. Die Richtung des Gravitationsvektors wird durch Messungen der linearen Beschleunigung der IMU bestimmt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_load_carrier/parameters?assume_gravity_
↪aligned=<value>
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/parameters?assume_gravity_aligned=<value>
```

`model_tolerance` (*Modelltoleranz*)

Gibt an, wie weit die Abmessungen des Load Carriers von den Werten im Modell abweichen dürfen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_load_carrier/parameters?model_
↪tolerance=<value>
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/parameters?model_tolerance=<value>
```

`crop_distance` (*Cropping*)

setzt den Sicherheitsspielraum, um den das Load Carrier Innenmaß verringert wird, um eine Region of Interest für die Erkennung zu definieren (siehe Abb. 6.37).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_load_carrier/parameters?crop_
↪distance=<value>
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/parameters?crop_distance=<value>
```

`min_plausibility` (*Minimale Plausibilität*):

Die minimale Plausibilität gibt an, wie viel von der Ebene um den Load Carrier Rand herum mindestens frei sein muss, um als gültige Erkennung zu zählen. Erhöhen Sie die minimale Plausibilität um falsch-positive Erkennungen zu vermeiden, und verringern Sie den Wert, wenn ein deutlich sichtbarer Load Carrier nicht erkannt wird.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_load_carrier/parameters?min_
↪plausibility=<value>
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/parameters?min_plausibility=<value>
```

6.3.2.6 Statuswerte

Das LoadCarrier Modul meldet folgende Statuswerte:

Tab. 6.21: Statuswerte des `rc_load_carrier` Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`load_carrier_detection_time`	Berechnungszeit für die letzte Load Carrier Detektion in Sekunden

6.3.2.7 Services

Die angebotenen Services des LoadCarrier Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der *rc_reason_stack Web GUI* (Abschnitt 7.1) auf der Seite *LoadCarrier* unter dem Menüpunkt *Module* ausprobiert und getestet werden.

Das LoadCarrier Modul stellt folgende Services zur Verfügung.

`detect_load_carriers`

löst die Erkennung von Load Carriern aus, wie in *Erkennung von Load Carriern* (Abschnitt 6.3.2.2) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_load_carrier/services/detect_
↪load_carriers
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/detect_load_carriers
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.2.4).

`load_carrier_ids`: IDs der zu erkennenden Load Carrier. Aktuell kann nur eine ID angegeben werden.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.2.4).

Optionale Serviceargumente:

`region_of_interest_id`: Die ID der 3D Region of Interest, innerhalb welcher nach den Load Carriern gesucht wird.

`region_of_interest_2d_id`: Die ID der 2D Region of Interest, innerhalb welcher nach den Load Carriern gesucht wird.

Bemerkung: Es kann nur eine Art von Region of Interest kann angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "load_carrier_ids": [
      "string"
    ],
    "pose_frame": "string",
    "region_of_interest_2d_id": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "detect_load_carriers",
  "response": {
    "load_carriers": [
      {
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


detect_filling_level

löst eine Load Carrier Füllstandserkennung aus, wie in *Füllstandserkennung* (Abschnitt 6.3.2.3) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_load_carrier/services/detect_
↪filling_level
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/detect_filling_level
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.2.4).

`load_carrier_ids`: IDs der zu erkennenden Load Carrier. Aktuell kann nur eine ID angegeben werden.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.2.4).

Optionale Serviceargumente:

`filling_level_cell_count`: Anzahl der Zellen im Füllstandsraster.

`region_of_interest_id`: Die ID der 3D Region of Interest, innerhalb welcher nach den Load Carriern gesucht wird.

`region_of_interest_2d_id`: Die ID der 2D Region of Interest, innerhalb welcher nach den Load Carriern gesucht wird.

Bemerkung: Es kann nur eine Art von Region of Interest kann angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "filling_level_cell_count": {
      "x": "uint32",
      "y": "uint32"
    },
    "load_carrier_ids": [
      "string"
    ],
    "pose_frame": "string",
    "region_of_interest_2d_id": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```


Response

`load_carriers`: Liste an erkannten Load Carriern und deren Füllstand.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "detect_filling_level",
  "response": {
    "load_carriers": [
      {
        "cells_filling_levels": [
          {
            "cell_position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "cell_size": {
              "x": "float64",
              "y": "float64"
            },
            "coverage": "float64",
            "level_free_in_meters": {
              "max": "float64",
              "mean": "float64",
              "min": "float64"
            },
            "level_in_percent": {
              "max": "float64",
              "mean": "float64",
              "min": "float64"
            }
          }
        ],
        "filling_level_cell_count": {
          "x": "uint32",
          "y": "uint32"
        },
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overall_filling_level": {
          "cell_position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "cell_size": {
            "x": "float64",
            "y": "float64"
          },
          "coverage": "float64",
          "level_free_in_meters": {
            "max": "float64",
            "mean": "float64",
            "min": "float64"
          },
          "level_in_percent": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "max": "float64",
            "mean": "float64",
            "min": "float64"
          }
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


reset_defaults

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_load_carrier/services/reset_
↪defaults
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

trigger_dump

speichert die Detektion auf dem angeschlossenen USB Speicher, die dem übergebenen Zeitstempel entspricht, oder die letzte, falls kein Zeitstempel angegeben wurde.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_load_carrier/services/trigger_
↪dump
```

API version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/trigger_dump
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "comment": "string",
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "trigger_dump",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`set_load_carrier` (veraltet)

speichert einen Load Carrier auf dem *rc_reason_stack*.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *set_load_carrier* (Abschnitt 6.5.1.5) in `rc_load_carrier_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/set_load_carrier
```

Die Definitionen von Request und Response sind dieselben wie in *set_load_carrier* (Abschnitt 6.5.1.5) in `rc_load_carrier_db` beschrieben.

`get_load_carriers` (veraltet)

gibt die mit `load_carrier_ids` spezifizierten, gespeicherten Load Carrier zurück.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *get_load_carriers* (Abschnitt 6.5.1.5) in `rc_load_carrier_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/get_load_carriers
```

Die Definitionen von Request und Response sind dieselben wie in *get_load_carriers* (Abschnitt 6.5.1.5) in `rc_load_carrier_db` beschrieben.

`delete_load_carriers` (veraltet)

löscht die mit `load_carrier_ids` spezifizierten, gespeicherten Load Carrier.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *delete_load_carriers* (Abschnitt 6.5.1.5) in `rc_load_carrier_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/delete_load_carriers
```

Die Definitionen von Request und Response sind dieselben wie in *delete_load_carriers* (Abschnitt 6.5.1.5) in `rc_load_carrier_db` beschrieben.

`set_region_of_interest` (veraltet)

speichert eine 3D Region of Interest auf dem *rc_reason_stack*.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *set_region_of_interest* (Abschnitt 6.5.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/set_region_of_interest
```

Die Definitionen von Request und Response sind dieselben wie in *set_region_of_interest* (Abschnitt 6.5.2.4) in `rc_roi_db` beschrieben.

`get_regions_of_interest` (veraltet)

gibt die mit `region_of_interest_ids` spezifizierten, gespeicherten 3D Regions of Interest zurück.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *get_regions_of_interest* (Abschnitt 6.5.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/get_regions_of_interest
```

Die Definitionen von Request und Response sind dieselben wie in *get_regions_of_interest* (Abschnitt 6.5.2.4) in `rc_roi_db` beschrieben.

`delete_regions_of_interest` (veraltet)

löscht die mit `region_of_interest_ids` spezifizierten, gespeicherten 3D Regions of Interest.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *delete_regions_of_interest* (Abschnitt 6.5.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/delete_regions_of_interest
```

Die Definitionen von Request und Response sind dieselben wie in *delete_regions_of_interest* (Abschnitt 6.5.2.4) in `rc_roi_db` beschrieben.

`set_region_of_interest_2d` (veraltet)

speichert eine 2D Region of Interest auf dem *rc_reason_stack*.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *set_region_of_interest_2d* (Abschnitt 6.5.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/set_region_of_interest_2d
```

Die Definitionen von Request und Response sind dieselben wie in *set_region_of_interest_2d* (Abschnitt 6.5.2.4) in `rc_roi_db` beschrieben.

`get_regions_of_interest_2d` (veraltet)

gibt die mit `region_of_interest_2d_ids` spezifizierten, gespeicherten 2D Regions of Interest zurück.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *get_regions_of_interest_2d* (Abschnitt 6.5.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/get_region_of_interest_2d
```

Die Definitionen von Request und Response sind dieselben wie in *get_regions_of_interest_2d* (Abschnitt 6.5.2.4) in `rc_roi_db` beschrieben.

`delete_regions_of_interest_2d` (veraltet)

löscht die mit `region_of_interest_2d_ids` spezifizierten, gespeicherten 2D Regions of Interest.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *delete_regions_of_interest_2d* (Abschnitt 6.5.2.4) in `rc_roi_db`.

API version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_load_carrier/services/delete_regions_of_interest_2d
```

Die Definitionen von Request und Response sind dieselben wie in *delete_regions_of_interest_2d* (Abschnitt 6.5.2.4) in `rc_roi_db` beschrieben.

6.3.2.8 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 6.22: Rückgabecodes der Services des LoadCarrier Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-4	Die maximal erlaubte Zeitspanne für die interne Akquise der Bilddaten wurde überschritten.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern überschritten wurde.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit
-12	Ressource ausgelastet, z.B. wenn `trigger_dump` zu häufig aufgerufen wird
-302	Es wurde mehr als ein Load Carrier an den `detect_load_carriers` oder `detect_filling_level` Service übergeben, aber nur einer wird unterstützt.
3	Der Timeout während der Load Carrier Erkennung wurde erreicht. Die Modelltoleranz sollte reduziert werden.
10	Die maximal speicherbare Anzahl an Load Carriern wurde erreicht.
11	Mit dem Aufruf von `set_load_carrier` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
102	Der erkannte Load Carrier enthält keine 3D-Punkte
300	Ein gültiges `robot_pose`-Argument wurde angegeben, ist aber nicht erforderlich.

6.3.3 TagDetect

6.3.3.1 Einführung

Die TagDetect-Module sind optionale Module, die intern auf dem *rc_reason_stack* laufen, und benötigen gesonderte *Lizenzen* (Abschnitt 8.2), die erworben werden müssen. Diese Lizenzen sind auf jedem *rc_reason_stack*, der nach dem 01.07.2020 gekauft wurde, vorhanden.

Die TagDetect-Module laufen intern auf dem *rc_reason_stack* und ermöglichen es, 2D-Matrixcodes und Marker (Tags) zu erkennen. Derzeit gibt es TagDetect-Module für *QR-Codes* und *AprilTags*. Neben der Erkennung berechnen die Module die Position und Orientierung jedes Tags im 3D-Kamerakoordinatensystem, um diesen beispielsweise mit einem Roboter zu manipulieren oder die Pose der Kamera in Bezug auf den Tag zu berechnen.

Bemerkung: Diese Module sind nicht verfügbar in Kamerapipelines vom Typ `zivid` oder `orbbec`.

Bemerkung: Diese Softwaremodule sind pipelinespezifisch. Änderungen ihrer Einstellungen oder Parameter betreffen nur die zugehörige Kamerapipeline und haben keinen Einfluss auf die anderen Pipelines, die auf dem *rc_reason_stack* laufen.

Die Tagerkennung besteht aus drei Schritten:

1. Tagerkennung auf dem 2D-Bildpaar (siehe *Tagerkennung*, Abschnitt 6.3.3.2).
2. Schätzung der Pose jedes Tags (siehe *Posenschätzung*, Abschnitt 6.3.3.3).
3. Wiedererkennung von bisher gesehenen Tags (siehe *Tag-Wiedererkennung*, Abschnitt 6.3.3.4).

Im Folgenden werden die zwei unterstützten Tagtypen näher beschrieben, gefolgt von einem Vergleich.

QR-Code

Abb. 6.6: Beispiel eines QR-Codes

QR-Codes sind zweidimensionale Matrixcodes, welche beliebige, benutzerspezifizierte Daten enthalten können. Viele Alltagsgeräte, wie beispielsweise Smartphones, unterstützen die Erkennung von QR-Codes. Zusätzlich stehen Online- und Offlinetools zur Verfügung, um QR-Codes zu generieren.

Die „Pixel“ eines QR-Codes werden *Module* genannt. Das Aussehen und die Auflösung von QR-Codes ändert sich mit der Menge der in ihnen gespeicherten Daten. Während die speziellen Muster in den drei Ecken immer 7 Module breit sind, erhöht sich die Anzahl der Module dazwischen, je mehr Daten gespeichert sind. Der am niedrigsten aufgelöste QR-Code besitzt eine Größe von 21x21 Modulen und kann bis zu 152 Bits speichern.

Auch wenn viele QR-Code-Generatoren speziell designte QR-Codes erzeugen können (bspw. mit einem Logo, mit runden Ecken oder mit Punkten als Module), wird eine zuverlässige Erkennung solcher Tags mit dem TagDetect-Modul nicht garantiert. Gleiches gilt für QR-Codes, welche Zeichen außerhalb des ASCII-Zeichensatzes beinhalten.

AprilTag

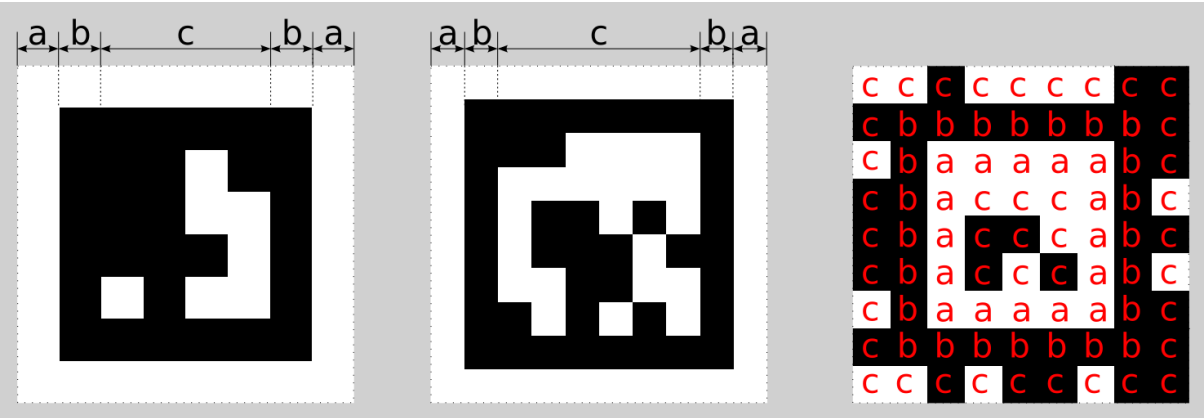


Abb. 6.7: Ein 16h5 Tag (links), ein 36h11 Tag (Mitte) und ein 41h12 Tag (rechts). AprilTags bestehen aus einem obligatorischen weißen (a) und schwarzen (b) Rahmen und einer variablen Menge an Datenmodulen (c).

AprilTags sind ähnlich zu QR-Codes. Sie wurden allerdings speziell zur robusten Identifikation auf weite Entfernungen entwickelt. Wie bei QR-Codes werden die „Pixel“ *Module* genannt. Abb. 6.7 veranschaulicht den Aufbau von AprilTags. Sie haben einen obligatorischen weißen und schwarzen Rahmen,

welcher jeweils ein Modul breit ist. Tags der Familien 16h5, 25h9, 36h10 und 36h11 sind von diesem Rahmen umschlossen und enthalten innen eine variable Menge an Datenmodulen. Bei Tags der Familie 41h12 ist der Rahmen nach innen verschoben und die Datenmodule befinden sich sowohl innerhalb als auch außerhalb des Rahmens. Anders als QR-Codes speichern AprilTags keine benutzerdefinierten Informationen, sondern werden durch eine vordefinierte *Familie* und *ID* identifiziert. Die Tags in Abb. 6.7 sind zum Beispiel aus Familie 16h5, 36h11 bzw. 41h12 und besitzen ID 0, 11 bzw. 0. Alle unterstützten Familien werden in Tab. 6.23 aufgelistet.

Tab. 6.23: AprilTag-Familien

Familie	Anzahl IDs	Empfohlen
16h5	30	-
25h9	35	o
36h10	2320	o
36h11	587	+
41h12	2115	+

Die Zahl vor dem „h“ jeder Familie bezeichnet die Anzahl der Datenmodule, welche im Tag enthalten sind: Während ein 16h5 Tag 16 (4x4) Datenmodule enthält ((c) in Abb. 6.7) und ein 36h11 Tag 36 (6x6), beinhaltet ein 41h12 Tag 41 Datenmodule (3x3 innen und 4x8 außen). Die Zahl hinter dem „h“ bezeichnet den Hamming-Abstand zwischen zwei Tags der Familie. Je höher, desto höher ist die Robustheit, aber desto weniger IDs stehen bei gleicher Anzahl an Datenmodulen zur Verfügung (siehe Tab. 6.23).

Der Vorteil von Familien mit weniger Modulen (bspw. 16h5 im Vergleich zu 36h11) ist die niedrigere Auflösung der Tags. Jedes Modul ist somit größer, weshalb der Tag auf eine größere Distanz erkannt werden kann. Dies hat allerdings auch Nachteile: Zum einen stehen bei niedrigerer Zahl an Datenmodulen auch weniger IDs zur Verfügung. Wichtiger aber ist, dass die Robustheit der Tagerkennung signifikant reduziert wird, da es zu einer höheren Falsch-Positiv-Rate kommt. Dies bedeutet, dass Tags verwechselt werden oder nicht existierende Tags in zufälliger Bildtextur oder im Bildrauschen erkannt werden. Die 41h12 Familie hat ihren Rahmen nach innen verschoben, was im Vergleich zur 36h11 Familie mehr Datenmodule bei einer geringen Gesamtmodulanzahl ermöglicht.

Aus diesen Gründen empfehlen wir die Verwendung der 42h12 und 36h11-Familien und raten ausdrücklich von der Familie 16h5 ab. Letztgenannte Familie sollten nur benutzt werden, wenn eine große Erkennungsdistanz für die Anwendung unbedingt erforderlich ist. Jedoch ist die maximale Erkennungsdistanz nur ca. 25% größer, wenn anstelle der 36h11-Familie die 16h5-Familie verwendet wird.

Vorgenerierte AprilTags können von der Webseite https://github.com/AprilRobotics/apriltag-imgs heruntergeladen werden. Jede Familie besteht aus mehreren PNGs, welche jeweils einen AprilTag enthalten. Jedes Pixel im PNG entspricht dabei einem Modul des AprilTags. Beim Drucken der Tags der Familien 36h11, 36h10, 25h9 und 16h5 sollte darauf geachtet werden, den weißen Rand um den AprilTag mit einzuschließen – dieser ist in den PNGs enthalten (siehe (a) in Abb. 6.7). Die Tags müssen außerdem ohne Interpolation auf die Druckgröße skaliert werden, sodass die scharfen Kanten erhalten bleiben.

Vergleich

Sowohl QR-Codes als auch AprilTags haben ihre Vor- und Nachteile. Während QR-Codes die Speicherung von benutzerdefinierten Daten erlauben, sind die Tags bei AprilTags vordefiniert und in ihrer Anzahl limitiert. Andererseits haben AprilTags eine niedrigere Auflösung und können daher auf eine größere Distanz erkannt werden. Zusätzlich hilft die durchgängige weiß-zu-schwarz-Kante in jedem AprilTag bei einer präziseren Posenschätzung.

Bemerkung: Falls die Speicherung von benutzerdefinierten Daten nicht benötigt wird, sollten AprilTags QR-Codes vorgezogen werden.

6.3.3.2 Tagerkennung

Der erste Schritt der Tagerkennung ist die Detektion der Tags auf dem Stereo-Bildpaar. Dieser Schritt benötigt die meiste Zeit und seine Präzision ist entscheidend für die Präzision der finalen Tagpose. Um die Dauer dieses Schritts zu kontrollieren, kann der Parameter `quality` vom Benutzer konfiguriert werden. Er hat ein Herunterskalieren des Stereo-Bildpaares vor der Tagerkennung zur Folge. *Hoch* (`High`) ergibt die höchste maximale Erkennungsdistanz und Präzision, aber auch die längste Dauer der Erkennung. *Niedrig* (`Low`) führt zur kleinsten maximalen Erkennungsdistanz und Präzision, aber benötigt auch nur weniger als die halbe Zeit. *Mittel* (`Medium`) liegt dazwischen. Es sollte beachtet werden, dass dieser `quality`-Parameter keine Verbindung zum `quality`-Parameter des *Stereo-Matching Modul* (Abschnitt 6.2.2) hat.

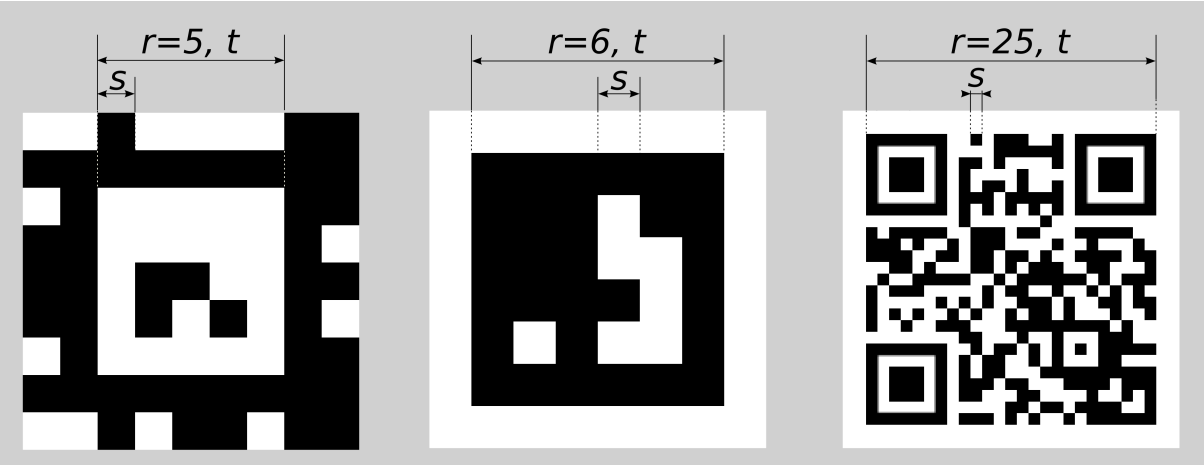


Abb. 6.8: Visualisierung der Modulgröße s, der Größe eines Tags in Modulen r und der Größe eines Tags in Metern t für AprilTags (links und Mitte) und QR-Codes (rechts)

Die maximale Erkennungsdistanz z für Qualität *Hoch* (`High`) kann mit folgenden Formeln angenähert werden:

$$z = \frac{fs}{p},$$

$$s = \frac{t}{r},$$

wobei f die *Brennweite* (Abschnitt 6.1.1) in Pixeln und s die Größe jedes Moduls in Metern bezeichnet. s kann leicht mit letztgenannter Formel berechnet werden, in welcher t der Taggröße in Metern und r der Breite des Tags in Modulen entspricht (bei AprilTags ohne den weißen Rahmen). Abb. 6.8 veranschaulicht diese Variablen. p bezeichnet die Zahl der Bildpixel pro Modul, welche für eine Erkennung erforderlich sind. Sie unterscheidet sich zwischen QR-Codes und AprilTags. Auch der Winkel des Tags zur Kamera und die Beleuchtung spielen eine Rolle. Ungefähre Werte für eine robuste Erkennung sind:

- AprilTag: $p = 5$ Pixel/Modul
- QR-Code: $p = 6$ Pixel/Modul

Die folgenden Tabellen enthalten Beispiele für die maximale Erkennungsdistanz in unterschiedlichen Situationen. Die Brennweite des *rc_visard* wird dafür mit 1075 Pixeln, die Qualität mit `High` angenommen.

Tab. 6.24: Beispiele zur maximalen Erkennungsdistanz für AprilTags mit einer Breite von $t = 4$ cm

AprilTag-Familie	Tagbreite	Maximale Distanz
36h11 (empfohlen)	8 Module	1.1 m
16h5	6 Module	1.4 m
41h12 (empfohlen)	5 Module	1.7 m

Tab. 6.25: Beispiele zur maximalen Erkennungsdistanz für QR-Codes mit einer Breite von $t = 8$ cm

Tagbreite	Maximale Distanz
29 Module	0.49 m
21 Module	0.70 m

6.3.3.3 Posenschätzung

Für jeden erkannten Tag wird dessen Pose im Kamerakoordinatensystem geschätzt. Eine Bedingung dafür ist, dass der Tag vollständig im linken und rechten Bild zu sehen ist. Das Koordinatensystem ist wie unten gezeigt am Tag ausgerichtet.

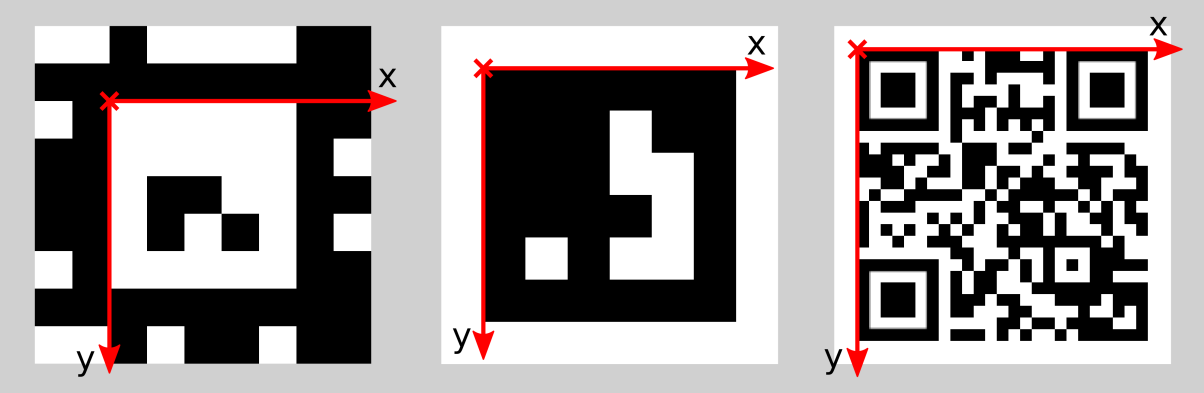


Abb. 6.9: Koordinatensysteme für AprilTags (links und Mitte) bzw. QR-Codes (rechts)

Die z-Achse zeigt „in“ den Tag. Es ist zu beachten, dass, auch wenn AprilTags den weißen Rand in ihrer Definition enthalten, der Ursprung des Koordinatensystems trotzdem am Übergang des weißen zum schwarzen Rand liegt. Da AprilTags keine offensichtliche Orientierung haben, liegt der Ursprung in der oberen linken Ecke des vorgenerierten AprilTags.

Während der Posenschätzung wird auch die Größe des Tags geschätzt unter der Annahme, dass der Tag quadratisch ist. Bei QR-Codes bezieht sich die Größe auf den gesamten Tag, bei AprilTags dagegen nur auf den Bereich innerhalb des Übergangs vom schwarzen zum weißen Rand. Das heißt, dass bei Tags der Familien 16h5, 25h9, 36h10 und 36h11 der äußere weiße Rand ignoriert wird.

Der Benutzer kann auch die ungefähre Größe ($\pm 10\%$) eines Tags angeben. Alle Tags, die dieser Einschränkung nicht entsprechen, werden automatisch herausgefiltert. Weiter hilft diese Information in bestimmten Situationen, Mehrdeutigkeiten in der Posenschätzung aufzulösen, die entstehen können, wenn mehrere Tags mit derselben ID im linken und rechten Bild sichtbar und diese Tags parallel zu den Bildzeilen ausgerichtet sind.

Bemerkung: Für beste Ergebnisse der Posenschätzung sollte der Tag sorgfältig gedruckt und auf einem steifen und möglichst ebenen Untergrund angebracht werden. Jegliche Verzerrung des Tags oder Unebenheit der Oberfläche verschlechtert die geschätzte Pose.

Bemerkung: Wir empfehlen, die ungefähre Größe der Tags anzugeben. Ansonsten, falls mehrere Tags mit derselben ID im linken oder rechten Bild sichtbar sind, kann es zu einer fehlerhaften Posenschätzung kommen, wenn die Tags gleich orientiert sind und sie ungefähr parallel zu den Bildzeilen angeordnet sind. Auch wenn die Größe nicht angegeben sein sollte, versuchen die TagDetect-Module jedoch, solche Situationen zu erkennen und verwerfen betroffene Tags.

Unten stehende Tabellen enthalten grobe Angaben zur Präzision der geschätzten Posen von AprilTags. Wir unterscheiden zwischen lateraler Präzision (also in x- und y-Richtung) und Präzision in z-Richtung.

Es wird angenommen, dass `quality` auf `High` gesetzt ist, und dass die Blickrichtung der Kamera parallel zur Normalen des Tags ist. Die Größe eines Tags hat keinen signifikanten Einfluss auf die Präzision in lateraler und z-Richtung. Im Allgemeinen verbessert ein größerer Tag allerdings die Präzision. Im Bezug auf die Präzision der Rotation, im speziellen um die x- und y-Achsen, übertreffen große Tags kleinere deutlich.

Tab. 6.26: Ungefähre Präzision der Orientierung von AprilTag Messungen mit Qualität `Hoch` in einem idealen Szenario für verschiedene Tag-Größen

Distanz	60 x 60 mm	120 x 120 mm
0.5 m	0.2°	–
1.0 m	0.8°	0.3°
2.0 m	2.0°	0.8°
3.0 m	–	1.8°

6.3.3.4 Tag-Wiedererkennung

Jeder Tag besitzt eine ID: bei AprilTags ist dies die *Familie* zusammen mit der AprilTag-*ID*, bei QR-Codes die enthaltenen Daten. Diese IDs sind jedoch nicht einzigartig, da mehrere Tags mit derselben ID in einer Szene vorkommen können.

Zur Unterscheidung dieser Tags weisen die TagDetect-Module jedem Tag einen eindeutigen Identifikator zu. Um den Benutzer dabei zu unterstützen, denselben Tag über mehrere Tagerkennungsläufe hinweg zu identifizieren, versucht das TagDetect-Modul Tags wiederzuerkennen. Falls erfolgreich, wird einem Tag derselbe Identifikator zugewiesen.

Die Tag-Wiedererkennung vergleicht die Positionen der Ecken der Tags im Kamera-Koordinatensystem, um identische Tags wiederzufinden. Tags werden als identisch angenommen, falls sie sich nicht oder nur geringfügig in diesem Koordinatensystem bewegt haben.

Über den `max_corner_distance`-Parameter kann der Benutzer festlegen, wie weit ein Tag sich zwischen zwei Erkennungsläufen bewegen darf, um als identisch zu gelten. Der Parameter definiert die maximale Distanz zwischen den Ecken zweier Tags, was in Abb. 6.10 dargestellt ist. Die euklidischen Abstände der vier zusammengehörenden Tagecken in 3D werden berechnet. Falls keiner dieser Abstände den Grenzwert überschreitet, gilt der Tag als wiedererkannt.

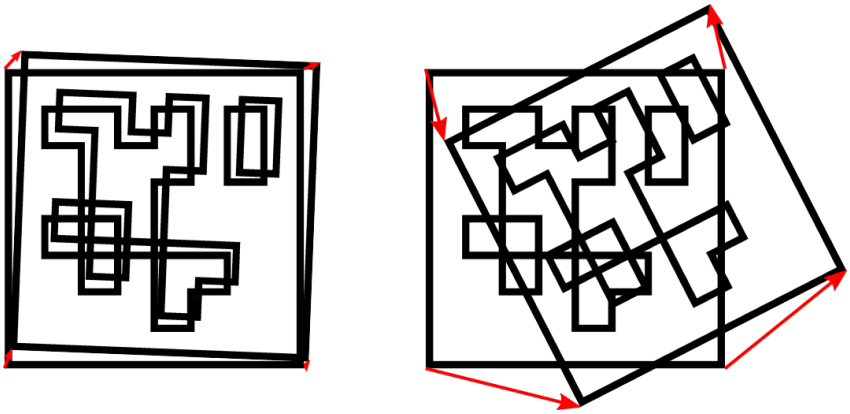


Abb. 6.10: Vereinfachte Darstellung der Tag-Wiedererkennung. Die euklidischen Abstände zwischen zusammengehörigen Tagecken in 3D werden berechnet (rote Pfeile).

Nach einer bestimmten Anzahl von Tagerkennungsläufen werden vorher gesehene Tags verworfen, falls diese in der Zwischenzeit nicht mehr erkannt wurden. Dies kann über den Parameter `forget_after_n_detections` festgelegt werden.

6.3.3.5 Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das TagDetect-Modul automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 6.3.3.8) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene `pose_frame`-Werte können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben.
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

6.3.3.6 Parameter

Es stehen zwei getrennte Module für die Tagerkennung zur Verfügung, eines für AprilTag- und eines für QR-Code-Erkennung: `rc_april_tag_detect` bzw. `rc_qr_code_detect`. Abgesehen vom Modulnamen teilen beide die gleiche Schnittstellendefinition.

Neben der *REST-API-Schnittstelle* (Abschnitt 7.2) stellen die TagDetect-Module außerdem Seiten in der Web GUI in der gewünschten Pipeline unter *Module* → *AprilTag* und *Module* → *QR Code* bereit, über welche sie manuell ausprobiert und konfiguriert werden können.

Im Folgenden sind die Parameter am Beispiel von `rc_qr_code_detect` aufgelistet. Sie gleichen denen von `rc_april_tag_detect`.

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.27: Laufzeitparameter des `rc_qr_code_detect`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`detect_inverted_tags`	bool	false	true	false	Erkennt Tags, bei denen Schwarz und Weiß vertauscht sind
`forget_after_n_detections`	int32	1	1000	30	Anzahl an Erkennungsläufen, nach denen ein vorher gesehener Tag während der Tag-Wiedererkennung verworfen wird
`max_corner_distance`	float64	0.001	0.01	0.005	Maximale Distanz zusammengehöriger Ecken zweier Tags während der Tag-Wiedererkennung
`quality`	string	-	-	High	Qualität der Tagerkennung: [Low, Medium, High]
`use_cached_images`	bool	false	true	false	Benutze das zuletzt empfangene Stereo-Bildpaar, anstatt auf ein neues zu warten

Über die REST-API können diese Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/<rc_qr_code_detect|rc_april_tag_
↪detect>/parameters?<parameter-name>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/parameters?
↪<parameter-name>=<value>
```

6.3.3.7 Statuswerte

Die TagDetect-Module melden folgende Statuswerte:

Tab. 6.28: Statuswerte der `rc_qr_code_detect`- und `rc_april_tag_detect`-Module

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`processing_time`	Berechnungszeit für die letzte Erkennung in Sekunden
`state`	Der aktuelle Zustand des Moduls

Der Parameter `state` kann folgende Werte annehmen:

Tab. 6.29: Mögliche Zustände der TagDetect-Module

Zustandsname	Beschreibung
IDLE	Das Modul ist inaktiv.
RUNNING	Das Modul läuft und ist bereit zur Tagerkennung.
FATAL	Ein schwerwiegender Fehler ist aufgetreten.

6.3.3.8 Services

Die TagDetect-Module implementieren einen Zustandsautomaten, welcher zum Starten und Stoppen genutzt werden kann. Die eigentliche Tagerkennung kann mit `detect` ausgelöst werden.

Die angebotenen Services von `rc_qr_code_detect` bzw. `rc_april_tag_detect` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der *rc_reason_stack Web GUI* (Abschnitt 7.1) ausprobiert und getestet werden.

`detect`

löst eine Tagerkennung aus.

Details

Abhängig vom `use_cached_images`-Parameter arbeitet das Modul auf dem zuletzt empfangenen Bildpaar (wenn *true*) oder wartet auf ein Bildpaar, das nach dem Auslösen des Services aufgenommen wurde (wenn *false*, dies ist das Standardverhalten). Auch wenn der Parameter auf *true* steht, arbeitet die Tagerkennung niemals mehrmals auf einem Bildpaar.

Es wird empfohlen, `detect` nur im Zustand `RUNNING` aufzurufen. Es ist jedoch auch im Zustand `IDLE` möglich, was zu einem Autostart und -stop des Moduls führt. Dies hat allerdings Nachteile: Erstens dauert der Aufruf deutlich länger, zweitens funktioniert die Tag-Wiedererkennung nicht. Es wird daher ausdrücklich empfohlen, das Modul manuell zu starten, bevor `detect` aufgerufen wird.

Tags können vom `detect`-Ergebnis aus mehreren Gründen ausgeschlossen werden, z.B. falls ein Tag nur in einem der Kamerabilder sichtbar war, oder falls die Posenschätzung fehlschlug. Diese herausgefilterten Tags werden im Log aufgelistet, auf welches wie in *Download der Logdateien* (Abschnitt 8.3) beschrieben zugegriffen werden kann.

Auf den Web GUI-Seiten der TagDetect-Module wird eine Visualisierung der letzten Tagerkennung bereitgestellt. Diese Visualisierung wird allerdings erst angezeigt, sobald die Tagerkennung mindestens einmal ausgeführt wurde. In der Web GUI kann die Tagerkennung außerdem manuell ausprobiert werden, indem die *Detektieren*-Schaltfläche betätigt wird.

Aufgrund von Änderungen der Systemzeit auf dem *rc_reason_stack* können Zeitsprünge auftreten, sowohl vorwärts als auch rückwärts. Während Vorwärtssprünge keinen Einfluss auf die TagDetect-Module haben, invalidieren Rücksprünge die bereits empfangenen Bilder. Deshalb wird, wenn ein Rücksprung erkannt wird, Fehler -102 beim nächsten `detect`-Aufruf zurückgegeben. Dies geschieht auch, um den Benutzer darauf hinzuweisen, dass die Zeitstempel in der `detect`-Antwort ebenso zurückspringen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/<rc_qr_code_detect|rc_
↪april_tag_detect>/services/detect
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪services/detect
```

Request

Optionale Serviceargumente:

`tags` bezeichnet die Liste der Tag-IDs, welche erkannt werden sollen. Bei QR-Codes ist die ID gleich den enthaltenen Daten. Bei AprilTags ist es *„<Familie>_<ID>“*, also beispielsweise *„36h11_5“* für Familie 36h11 und ID 5. Natürlich kann das AprilTag-Modul nur zur Erkennung von AprilTags und das QR-Code-Modul nur zur Erkennung von QR-Codes genutzt werden.

Die `tags`-Liste kann auch leer gelassen werden. In diesem Fall werden alle erkannten Tags zurückgegeben. Dieses Feature sollte nur während der Entwicklung einer Applikation oder zur Fehlerbehebung benutzt werden. Wann immer möglich sollten die konkreten Tag-IDs aufgelistet werden, zum einen zur Vermeidung von Fehldetektionen, zum anderen auch um die Tagerkennung zu beschleunigen, da nicht benötigte Tags aussortiert werden können.

Bei AprilTags kann der Benutzer nicht nur einzelne Tags, sondern auch eine gesamte Familie spezifizieren, indem die ID auf „<family>“ gesetzt wird, bspw. „36h11“. Dadurch werden alle Tags dieser Familie erkannt. Es ist auch möglich, mehrere Familien oder eine Kombination aus Familien und einzelnen Tags anzugeben. Zum Beispiel kann `detect` mit „36h11“, „25h9_3“ und „36h10“ zur gleichen Zeit aufgerufen werden.

Zusätzlich zur ID kann auch die ungefähre Größe ($\pm 10\%$) eines Tags angegeben werden. Wie in *Posenschätzung* (Abschnitt 6.3.3.3) erklärt, verhilft dies Mehrdeutigkeiten aufzulösen, die in bestimmten Situationen auftreten können, und kann zum Herausfiltern von Tags genutzt werden, die nicht der angegebenen Größe entsprechen.

Die `tags`-Liste ist ODER-verknüpft. Es werden alle Tags zurückgegeben, die mit einem der `id-size`-Paare in der `tags`-Liste übereinstimmen.

Das Feld `pose_frame` gibt an, ob die Posen im Kamera- oder im externen Koordinatensystem zurückgegeben werden (siehe *Hand-Auge-Kalibrierung*, Abschnitt 6.3.3.5). Der Standardwert ist `camera`.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "pose_frame": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "tags": [
      {
        "id": "string",
        "size": "float64"
      }
    ]
  }
}
```


Response

`timestamp` wird auf den Zeitstempel des Bildpaares gesetzt, auf dem die Tagerkennung gearbeitet hat.

`tags` enthält alle erkannten Tags.

`id` ist die ID des Tags, vergleichbar zur `id` in der Anfrage.

`instance_id` ist der zufällige, eindeutige Identifikator eines Tags, welcher von der Tag-Wiedererkennung zugewiesen wird.

`pose` enthält `position` und `orientation`. Die Orientierung ist im Quaternionen-Format angegeben.

`pose_frame` bezeichnet das Koordinatensystem, auf welches obige Pose bezogen ist, und hat den Wert `camera` oder `external`.

`size` wird auf die gemessene Taggröße gesetzt.

`return_code` enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "detect",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "tags": [
      {
        "id": "string",
        "instance_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "size": "float64",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        }
      }
    ],
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


start

startet das Modul durch einen Übergang von `IDLE` nach `RUNNING`.

Details

Wenn das Modul läuft, empfängt es die Bilder der Stereokamera und ist bereit, Tags zu erkennen. Um Rechenressourcen zu sparen, sollte das Modul nur laufen, wenn dies nötig ist.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/<rc_qr_code_detect|rc_april_tag_
↪detect>/services/start
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/services/start
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "start",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

stoppt das Modul durch einen Übergang zu `IDLE`.

Details

Dieser Übergang kann auf dem Zustand `RUNNING` und `FATAL` durchgeführt werden. Alle Tag-Wiedererkennungs-Informationen werden beim Stoppen gelöscht.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/<rc_qr_code_detect|rc_
↪april_tag_detect>/services/stop
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪services/stop
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

restart

startet das Modul neu.

Details

Wenn im Zustand `RUNNING` oder `FATAL`, wird das Modul erst gestoppt und dann wieder gestartet. In `IDLE` wird das Modul nur gestartet.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/<rc_qr_code_detect|rc_
↪april_tag_detect>/services/restart
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪services/restart
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "restart",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

trigger_dump

speichert die Detektion auf dem angeschlossenen USB Speicher, die dem übergebenen Zeitstempel entspricht, oder die letzte, falls kein Zeitstempel angegeben wurde.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/<rc_qr_code_detect|rc_april_tag_
↪detect>/services/trigger_dump
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/services/
↪trigger_dump
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "comment": "string",
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "trigger_dump",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/<rc_qr_code_detect|rc_april_tag_
↪detect>/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/services/reset_
↪defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.3.3.9 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Code	Beschreibung
0	Erfolg
-1	Ein ungültiges Argument wurde übergeben.
-4	Die maximale Wartezeit auf ein Stereo-Bildpaar wurde überschritten.
-9	Die Lizenz ist ungültig.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit
-12	Ressource ausgelastet, z.B. wenn `trigger_dump` zu häufig aufgerufen wird
-101	Ein interner Fehler trat während der Tagerkennung auf.
-102	Ein Rückwärtssprung der Systemzeit trat auf
-103	Ein interner Fehler trat während der Posenschätzung auf.
-200	Ein schwerwiegender interner Fehler trat auf.
200	Mehrere Warnungen traten auf. Siehe die Auflistung in `message`.
201	Das Modul war nicht im Zustand `RUNNING`.

6.3.4 ItemPick und ItemPickAI

6.3.4.1 Einführung

Das ItemPick und ItemPickAI Modul liefert eine gebrauchsfertige Perzeptionslösung, um robotische Pick-and-Place-Anwendungen zu realisieren. ItemPick detektiert ebene Oberflächen unbekannter Objekte für die Positionierung eines Sauggreifers. :cubeonly:'ItemPickAI nutzt neuronale Netze, um Objekte einer bestimmten Objektkategorie zu segmentieren und orientierte und objektzentrierte Greifpunkte für Sauggreifer zu berechnen.

Darüber hinaus bietet das Modul:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc_reason_stack* *Web GUI* (Abschnitt 7.1)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche der Szene auszuwählen (siehe *RoiDB*, Abschnitt 6.5.2)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 6.3.2), um in Bin-Picking-Anwendungen („Griff in die Kiste“) Greifpunkte nur für Objekte in dem erkannten Load Carrier zu berechnen
- die Unterstützung von Load Carriern mit Fächern, sodass Greifpunkte für Objekte nur in einem definierten Teilvolumen des Load Carriers berechnet werden
- eine Kollisionsprüfung zwischen Greifer und Load Carrier und/oder der Punktwolke
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1) kombiniert werden, um Greifposen in einem benutzerdefinierten externen Koordinatensystem zu liefern.
- einen Qualitätswert für jeden vorgeschlagenen Greifpunkt, der die Ebenheit der für das Greifen verfügbaren Oberfläche bewertet
- Auswahl einer Strategie zum Sortieren der zurückgelieferten Greifpunkte
- eine 3D Visualisierung des Detektionsergebnisses mit Greifpunkten und einer Greiferanimation in der Web GUI

Bemerkung: Dieses Softwaremodul ist pipelinespezifisch. Änderungen seiner Einstellungen oder Parameter betreffen nur die zugehörige Kamerapipeline und haben keinen Einfluss auf die anderen Pipelines, die auf dem *rc_reason_stack* laufen.

Bemerkung: In diesem Kapitel werden die Begriffe Cluster und Oberfläche synonym verwendet und bezeichnen eine Menge von Punkten (oder Pixeln) mit ähnlichen geometrischen Eigenschaften.

Das Modul ist ein optional erhältliches Module, welches intern auf dem *rc_reason_stack* läuft und eine gesonderte ItemPick bzw. ItemPickAI *Lizenz* (Abschnitt 8.2) benötigt.

6.3.4.2 Berechnung der Greifpunkte

Das ItemPick und ItemPickAI Modul bietet einen Service, um Greifpunkte für Sauggreifer zu berechnen. Der Sauggreifer ist durch die Länge und Breite der Greiffläche definiert.

Das ItemPick-Modul identifiziert ebene Flächen in der Szene und unterstützt flexible und/oder deformierbare Objekte. Der Typ (`type`) dieser Objektmodelle (`item_models`) ist als unbekannt (`UNKNOWN`) definiert, da sie keine gebräuchliche geometrische Form aufweisen müssen. Optional kann eine minimale und maximale Größe angegeben werden.

Für ItemPickAI werden die Greifpunkte in der Mitte der oberen Oberfläche der segmentierten Objekte (`items`) der angegebenen Objektkategorie berechnet. Die Objektkategorie wird durch Festlegen des Typs (`type`) der Objektmodelle (`item_models`) ausgewählt. Derzeit werden die Typen `BAG`,

CONSUMER_GOODS und SHEET_METAL unterstützt. BAG bezieht sich auf verformbare und flexible beutelartige Objekte mit unterschiedlichen Füllgraden, wie z.B. Beutelpackungen, Päckchen, Schüttgutsäcke, Versandtaschen, Papiertüten und Säcke. CONSUMER_GOODS umfasst allgemeine verpackte Konsumgüter wie verpackte Lebensmittel, Getränke, Toilettenartikel, Reinigungsmittel und andere erschwingliche Haushaltswaren. SHEET_METAL segmentiert flache, ebene Metallteile, z.B. lasergeschnittene Bleche.

Bemerkung: Der erste Aufruf der Erkennung mit dem Objektmodelltyp BAG, CONSUMER_GOODS oder SHEET_METAL dauert jeweils länger als die nachfolgenden Aufrufe, weil das Modell erst in das ItemPickAI-Modul geladen werden muss.

Optional können den Modulen zu einer Greifpunktberechnung weitere Informationen übergeben werden:

- die ID des Load Carriers, welcher die zu greifenden Objekte enthält
- ein Unterabteil (load_carrier_compartment) innerhalb eines Load Carriers, in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteile*, Abschnitt 6.5.1.3).
- die ID der 3D Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die 3D Region of Interest, innerhalb der Greifpunkte berechnet werden
- Informationen für die Kollisionsprüfung: Die ID des Greifers, um die Kollisionsprüfung zu aktivieren, und optional ein Greif-Offset, der die Vorgreifposition definiert. Details zur Kollisionsprüfung sind in *CollisionCheck* (Abschnitt 6.3.4.4) gegeben.

Ein vom BoxPick-Modul ermittelter Greifpunkt repräsentiert die empfohlene Pose des TCP (Tool Center Point) des Sauggreifers. Der Greifpunkt type ist immer auf SUCTION gesetzt.

Für ItemPick mit dem Objektmodelltyp UNKNOWN liegt der Ursprung der berechneten Greifposen pose im Mittelpunkt der größten von der jeweiligen Greiffläche umschlossenen Ellipse.

Für ItemPickAI mit dem Objektmodelltyp BAG, CONSUMER_GOODS oder SHEET_METAL liegen die Greifpunkte im Mittelpunkt der oberen Fläche der segmentierten Objekte.

Die Orientierung des Greifpunkts ist ein rechtshändiges Koordinatensystem, sodass die z-Achse orthogonal zur Greiffläche in das zu greifende Objekt zeigt und die x-Achse entlang der längsten Ausdehnung ausgerichtet ist.Da die x-Achse zwei mögliche Richtungen haben kann, wird diejenige ausgewählt, die besser zur bevorzugten TCP-Ausrichtung passt (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 6.3.4.3). Wenn der Laufzeitparameter allow_any_grasp_z_rotation auf True gesetzt ist, wird die x-Achse nicht zwangsweise an der maximalen Dehnung der greifbaren Ellipse ausgerichtet, sondern kann eine beliebige Drehung um die z-Achse aufweisen. In diesem Fall hat der zurückgegebene Greifpunkt die Ausrichtung, die am besten zur bevorzugten TCP-Ausrichtung passt und kollisionsfrei ist, wenn die Kollisionsprüfung aktiviert ist.

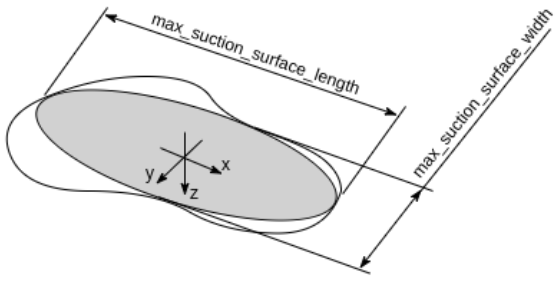


Abb. 6.11: Veranschaulichung eines berechneten Greifpunktes mit Koordinatensystem und der zugehörigen Ellipse, welche die Greiffläche bestmöglich beschreibt.

Zusätzlich enthält jeder Greifpunkt die Abmessungen der maximal verfügbaren Greiffläche, die als Ellipse mit den Achslängen max_suction_surface_length und max_suction_surface_width beschrie-

ben wird. Der Nutzer kann Greifpunkte mit zu kleinen Greifflächen herausfiltern, indem die minimalen Abmessungen der Greiffläche, die vom Sauggreifer benötigt wird, angegeben werden. Wenn der Laufzeitparameter `allow_any_grasp_z_rotation` auf True gesetzt ist, dann sind die Achslängen `max_suction_surface_length` und `max_suction_surface_width` gleich und entsprechen der kürzeren Achse der größtmöglichen Greifellipse.

Jeder Greifpunkt enthält auch einen Qualitätswert (`quality`), der einen Hinweis auf die Ebenheit der Greiffläche gibt. Dieser Wert reicht von 0 bis 1, wobei höhere Werte für eine ebenere rekonstruierte Oberfläche stehen.

Jeder berechnete Greifpunkt lässt sich anhand einer `uuid` (Universally Unique Identifier) eindeutig identifizieren und enthält zusätzlich den Zeitstempel der ältesten Bildaufnahme, auf der die Greifpunktberechnung durchgeführt wurde.

Die Sortierung der Greifpunkte basiert auf der ausgewählten Sortierstrategie. Folgende Sortierstrategien sind verfügbar und können über die *Web GUI* (Abschnitt 7.1) oder über den `set_sorting_strategies` Service gesetzt werden:

- `gravity`: höchste Greifpunkte entlang der Gravitationsrichtung werden zuerst zurückgeliefert.
- `surface_area`: Greifpunkte mit den größten Oberflächen werden zuerst zurückgeliefert.
- `direction`: Greifpunkte mit dem kleinsten Abstand entlang der gesetzten Richtung `vector` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.
- `distance_to_point`: Greifpunkte mit dem kleinsten oder größten (falls `farthest_first` auf true gesetzt ist) Abstand von einem gesetzten Sortierpunkt `point` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.

Wenn keine Sortierstrategie gesetzt ist, oder die Standard-Sortierstrategie in der Web GUI ausgewählt ist, geschieht die Sortierung der Greifpunkte basierend auf einer Kombination von `gravity` und `surface_area`.

6.3.4.3 Setzen der bevorzugten TCP-Orientierung

Das ItemPick und ItemPickAI-Modul berechnet die Erreichbarkeit von Greifpunkten basierend auf der bevorzugten Orientierung des TCPs. Die bevorzugte Orientierung kann über den Service `set_preferred_orientation` oder über die *CADMatch*-Seite in der Web GUI gesetzt werden. Die bevorzugten Orientierung des TCPs wird genutzt, um Greifpunkte zu verwerfen, die der Greifer nicht erreichen kann, und kann auch zur Sortierung der Greifpunkte genutzt werden.

Die bevorzugte TCP-Orientierung kann im Kamerakoordinatensystem oder im externen Koordinatensystem gesetzt werden, wenn eine Hand-Auge-Kalibrierung verfügbar ist. Wenn die bevorzugte TCP-Orientierung im externen Koordinatensystem definiert ist, und die Kamera am Roboter montiert ist, muss bei jedem Aufruf der Objekterkennung die aktuelle Roboterpose angegeben werden. Wenn keine bevorzugte TCP-Orientierung gesetzt wird, wird die Orientierung der linken Kamera (siehe Coordinate frames im rc_visard Handbuch) als die bevorzugte TCP-Orientierung genutzt.

6.3.4.4 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_reason_stack* laufenden Module liefern Daten für das ItemPick und ItemPickAI-Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieser Module kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des ItemPick und ItemPickAI-Moduls haben.

Kamera- und Tiefendaten

Folgende Daten werden vom ItemPick und ItemPickAI-Modul verarbeitet:

- die rektifizierten Bilder des *Kamera Modul* (`rc_camera`, Abschnitt 6.1)
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 6.2.2), falls eine Stereokamera verwendet wird.
- die Disparitäts-, Konfidenz- und Fehlerbilder der *Orbbec Modul* (`rc_orbbec`, Abschnitt 6.2.4), falls eine *Orbbec* Kamera verwendet wird
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Zivid Modul* (`rc_zivid`, Abschnitt 6.2.3), falls eine *zivid* Kamera verwendet wird

Für alle genutzten Bilder ist garantiert, dass diese nach dem Auslösen des Services aufgenommen wurden.

IOControl und Projektor-Kontrolle

Für den Anwendungsfall, dass der *rc_reason_stack* zusammen mit einem externen Musterprojektor und dem Modul für *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 6.4.4) betrieben wird, wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 6.2.2.1), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 6.4.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das ItemPick und ItemPickAI-Modul automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 6.3.4.7) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das ItemPick- oder BoxPick-Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

Für den Fall einer robotergeführten Kamera ist es abhängig von `pose_frame` und der Sortierrichtung bzw. des Sortierpunktes nötig, zusätzlich die aktuelle Roboterpose (`robot_pose`) zur Verfügung zu stellen:

- Wenn `external` als `pose_frame` ausgewählt ist, ist die Angabe der Roboterpose obligatorisch.

- Wenn die Sortierrichtung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn der Sortierpunkt für die Abstandssortierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- In allen anderen Fällen ist die Angabe der Roboterpose optional.

LoadCarrier

Das ItemPick und ItemPickAI-Module nutzt die Funktionalität zur Load Carrier Erkennung aus dem *LoadCarrier* Modul (`rc_load_carrier`, Abschnitt 6.3.2) mit den Laufzeitparametern, die für dieses Modul festgelegt wurden. Wenn sich jedoch mehrere Load Carrier in der Szene befinden, die zu der angegebenen Load Carrier ID passen, wird nur einer davon zurückgeliefert. In diesem Fall sollte eine 3D Region of Interest gesetzt werden, um sicherzustellen, dass immer derselbe Load Carrier für das ItemPick und ItemPickAI-Modul verwendet wird.

CollisionCheck

Die Kollisionsprüfung kann für die Greifpunktberechnung des ItemPick und ItemPickAI-Moduls aktiviert werden, indem das `collision_detection` Argument an den `compute_grasps` oder `compute_grasps_extended` Service übergeben wird. Es enthält die ID des benutzten Greifers und optional einen Greif-Offset. Der Greifer muss im GripperDB Modul definiert werden (siehe *Erstellen eines Greifers*, Abschnitt 6.5.3.2) und Details über die Kollisionsprüfung werden in *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 6.4.2.2) gegeben.

Wenn die Kollisionsprüfung aktiviert ist, werden nur kollisionsfreie Greifpunkte zurückgeliefert. Jedoch werden in den Visualisierungen auf der ItemPick:cubeonly:*bzw. ItemPickAI* -Seite der Web GUI kollidierende Greifpunkte als schwarze Ellipsen dargestellt.

Die Laufzeitparameter des CollisionCheck-Moduls beeinflussen die Kollisionserkennung wie in *CollisionCheck-Parameter* (Abschnitt 6.4.2.3) beschrieben.

6.3.4.5 Parameter

Das ItemPick und ItemPickAI-Modul wird in der REST-API als `rc_itempick` bezeichnet und in der *Web GUI* (Abschnitt 7.1) in der gewünschten Pipeline unter *Module* → *ItemPick* und *Modules* → *ItemPickAI* dargestellt. Wenn beide Lizenzen, ItemPick und ItemPickAI, auf einem Gerät vorhanden sind, wird die ItemPick-Funktionalität in die *ItemPickAI* Seite der Web GUI integriert. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 7.2) ändern.

Die angebotenen Services von `rc_itempick` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der *rc_reason_stack Web GUI* (Abschnitt 7.1) ausprobiert und getestet werden.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.30: Laufzeitparameter des `rc_itempick` Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`allow_any_grasp_pose`	bool	false	true	false	Bestimmt, ob die Greifpunkte beliebig dort auf dem Objekt platziert sein dürfen, wo ebene Greifflächen detektiert werden
`allow_any_grasp_z_rotation`	bool	false	true	false	Bestimmt, ob die Greifpunkte beliebige Orientierung haben dürfen, anstatt an der Hauptachse der greifbaren Ellipse ausgerichtet zu sein
`check_collisions_with_point_cloud`	bool	false	true	false	Gibt an, ob Kollisionen zwischen Greifer und anderen Matches geprüft werden
`cluster_max_curvature`	float64	0.005	0.5	0.11	Maximal erlaubte Krümmung für Greifflächen
`cluster_max_dimension`	float64	0.05	2.0	0.3	Maximum allowed diameter for a cluster in meters. Clusters with a diameter larger than this value are not used for grasp computation.
`clustering_discontinuity_factor`	float64	0.1	5.0	1.0	Erlaubte Unebenheit von Greifflächen
`clustering_max_surface_rmse`	float64	0.0005	0.01	0.004	Maximal erlaubte Abweichung (Root Mean Square Error, RMSE) von Punkten zur Greiffläche in Metern
`clustering_patch_size`	int32	3	10	4	Pixelgröße der Patches für die Unterteilung des Tiefenbildes im ersten Clustering-Schritt
`grasp_filter_orientation_threshold`	float64	0.0	180.0	45.0	Maximal erlaubte Orientierungsabweichung zwischen Greifpunkt und bevorzugter TCP-Orientierung in Grad
`max_grasps`	int32	1	100	5	Maximale Anzahl von bereitgestellten Greifpunkten

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf der ItemPick bzw. ItemPickAI Seite in der Web GUI dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet:

`max_grasps` (*Anzahl Greifpunkte*)

ist die maximale Anzahl von bereitgestellten Greifpunkten.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?max_grasps=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?max_grasps=<value>
```

`cluster_max_dimension` (*Maximale Größe*, Nur für ItemPick)

is the maximum allowed diameter for a cluster in meters. Clusters with a diameter larger than this value are not used for grasp computation.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?cluster_max_
↪dimension=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?cluster_max_dimension=<value>
```

`cluster_max_curvature` (*Maximale Krümmung*, Nur für ItemPick)

ist die maximal erlaubte Krümmung für Greifflächen. Je kleiner dieser Wert ist, desto mehr mögliche Greifflächen werden in kleinere Flächen mit weniger Krümmung aufgeteilt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?cluster_max_
↪curvature=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?cluster_max_curvature=<value>
```

`clustering_patch_size` (*Patchgröße*, Nur für ItemPick)

ist die Pixelgröße der Patches für die Unterteilung des Tiefenbildes im ersten Clustering-Schritt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?clustering_
↪patch_size=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?clustering_patch_size=<value>
```

`clustering_discontinuity_factor` (*Unstetigkeitsfaktor*, Nur für ItemPick)

beschreibt die erlaubte Unebenheit von Greifflächen. Je kleiner dieser Wert ist, umso mehr werden mögliche Greifflächen in kleinere Flächen mit weniger Unebenheiten aufgeteilt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?clustering_
↪discontinuity_factor=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?clustering_discontinuity_factor=
↪<value>
```

`clustering_max_surface_rmse` (*Maximaler RMSE*, Nur für ItemPick)

ist die maximal erlaubte Abweichung (Root Mean Square Error, RMSE) von Punkten zur Greiffläche in Metern.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?clustering_
↪max_surface_rmse=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?clustering_max_surface_rmse=
↪<value>
```

grasp_filter_orientation_threshold‘ (*Grasp Orientation Threshold*)

ist die maximale Abweichung der TCP-z-Achse am Greifpunkt von der z-Achse der bevorzugten TCP-Orientierung in Grad. Es werden nur Greifpunkte zurückgeliefert, deren Orientierungsabweichung kleiner als der angegebene Wert ist. Falls der Wert auf Null gesetzt wird, sind alle Abweichungen valide.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?grasp_filter_
↪orientation_threshold=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?grasp_filter_orientation_
↪threshold=<value>
```

`allow_any_grasp_z_rotation` (*Beliebige Greifrotation um Z*)

Wenn der Wert auf True gesetzt ist, werden die x-Achsen der zurückgegebenen Greifpunkte nicht mehr notwendigerweise an der maximalen Ausdehnung der greifbaren Ellipse ausgerichtet, sondern können eine beliebige Drehung um die z-Achse haben. Die zurückgegebenen Werte von `max_suction_surface_length` und `max_suction_surface_width` sind dann gleich und entsprechen dem kleinsten Durchmesser der größten greifbaren Ellipsenfläche. Dieser Parameter eröffnet dem Roboter mehr Optionen zum Greifen von Objekten, insbesondere in Szenen, in denen es zu Kollisionen kommen kann. Da der Greifpunkt jedoch nicht mehr mit

der greifbaren Ellipse ausgerichtet ist, muss bei Objektmodellen vom Typ `UNKNOWN` die korrekte Ausrichtung zum Platzieren des Objekts auf andere Weise bestimmt werden. Im Fall von ItemPickAI kann die Pose des zugehörigen "item" verwendet werden, um die richtige Greiforientierung für die Platzierung zu bestimmen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?allow_any_
↪grasp_z_rotation=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?allow_any_grasp_z_rotation=
↪<value>
```

`allow_any_grasp_pose` (*Beliebige Greifposen*)

Wenn dieser Parameter aktiv ist, werden die Greifpunkte nicht mehr zwangsläufig auf dem Objekt zentriert und an der Hauptachse des Objekts ausgerichtet, sondern können sich an beliebiger Stelle des Objekts befinden, wo greifbare Oberflächen vorhanden sind. Dazu werden die segmentierten Objektoberflächen mithilfe der Cluster Parameter unterteilt, um die greifbaren Flächen eines Objekts zu ermitteln. Dieser Parameter hat keine Auswirkung, wenn der Objektmodell Typ `UNKNOWN` verwendet wird.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?allow_any_
↪grasp_pose=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?allow_any_grasp_pose=<value>
```

`check_collisions_with_point_cloud` (*Kollisionsprüfung mit Punktwolke*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `compute_grasps` oder `compute_grasps_extended` Service aktiviert ist. Wenn `check_collisions_with_point_cloud` auf true gesetzt ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit dieser Punktwolke wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/parameters?check_
↪collisions_with_point_cloud=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?check_collisions_with_point_
↪cloud=<value>
```

6.3.4.6 Statuswerte

Das `rc_itempick` Modul meldet folgende Statuswerte:

Tab. 6.31: Statuswerte des `rc_itempick` Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste.
`grasp_computation_time`	Laufzeit für die Greifpunktberechnung beim letzten Aufruf in Sekunden
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`load_carrier_detection_time`	Laufzeit für die letzte Load Carrier Erkennung in Sekunden
`processing_time`	Laufzeit für die letzte Erkennung (einschließlich Load Carrier Detektion) in Sekunden
`state`	Aktueller Zustand des rc_itempick Moduls

Folgende `state`-Werte werden gemeldet.

Tab. 6.32: Mögliche Werte für den Zustand des ItemPick und ItemPickAI Moduls

Zustand	Beschreibung
IDLE	Das Modul ist inaktiv.
RUNNING	Das Modul wurde gestartet und ist bereit, Load Carrier zu erkennen und Greifpunkte zu berechnen.
FATAL	Ein schwerwiegender Fehler ist aufgetreten.

6.3.4.7 Services

Die angebotenen Services von `rc_itempick` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der *rc_reason_stack Web GUI* (Abschnitt 7.1) ausprobiert und getestet werden.

Das ItemPick und ItemPickAI Modul stellt folgende Services zur Verfügung.

`compute_grasps`

löst die Erkennung von Greifpunkten für einen Sauggreifer aus, wie in *Berechnung der Greifpunkte* (Abschnitt 6.3.4.2) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/compute_grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/compute_grasps
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.4.4).

`suction_surface_length`: Länge der Greiffläche des verwendeten Vakuum-Greifsystems.

`suction_surface_width`: Breite der Greiffläche des verwendeten Vakuum-Greifsystems.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.4.4).

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, welcher die zu greifenden Objekte enthält.

`load_carrier_compartment`: Teilvolumen (Fach oder Abteil) in einem zu detektierenden Load Carrier (Behälter), in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteile*, Abschnitt 6.5.1.3).

`region_of_interest_id`: Falls `load_carrier_id` gesetzt ist, die ID der 3D Region of Interest, innerhalb welcher nach dem Load Carrier gesucht wird. Andernfalls die ID der 3D Region of Interest, innerhalb der Greifpunkte berechnet werden.

`item_models`: Liste von Objektmodellen, die erkannt werden sollen. Im Fall von ItemPick wird aktuell nur ein einzelnes Objektmodell vom Typ `UNKNOWN` mit minimaler und maximaler Größe unterstützt, wobei die minimale Größe kleiner als die maximale Größe sein muss.

Im Fall von ItemPickAI werden aktuell Objektmodelle vom Typ `BAG`, `CONSUMER_GOODS` und `SHEET_METAL` unterstützt.

`collision_detection`: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 6.4.2.2)

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "item_models": [
      {
        "type": "string",
        "unknown": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        }
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "suction_surface_length": "float64",
    "suction_surface_width": "float64"
  }
}
```


Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`grasps`: sortierte Liste von Sauggreifpunkten.

`items`: Liste von erkannten Objekten, die zu den zurückgelieferten Greifpunkten gehören. Im Fall von ItemPick ist diese Liste immer leer.

Im Fall von ItemPickAI enthält `items` die segmentierten Objekte vom Typ `BAG`, `CONSUMER_GOODS` oder `SHEET_METAL` mit ihren Posen bezogen auf den Mittelpunkt der Bounding Box (kleinste umschließende Box) des sichtbaren Teils des Objekts und die Abmessungen dieser Bounding Box.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "compute_grasps",
  "response": {
    "grasps": [
      {
        "item_uuid": "string",
        "max_suction_surface_length": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      "max_suction_surface_width": "float64",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "quality": "float64",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "type": "string",
      "uuid": "string"
    }
  ],
  "items": [
    {
      "bounding_box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "grasp_uuids": [
        "string"
      ],
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "type": "string",
      "uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


compute_grasps_extended

löst die Erkennung von Greifpunkten für einen Sauggreifer aus. Dieser Service verhält sich analog zu `compute_grasps`, gibt aber die Objektinformationen `item` für jeden Greifpunkt direkt zurück, anstatt sie in einer separaten Liste zu speichern. Dies ermöglicht ein einfacheres Parsen, wenn Objektinformationen für die Greifpunkte benötigt werden.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/compute_grasps_
↪extended
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/compute_grasps_extended
```

Request

Siehe `compute_grasps` Service.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "item_models": [
      {
        "type": "string",
        "unknown": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        }
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "suction_surface_length": "float64",
    "suction_surface_width": "float64"
  }
}
```

Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`grasps`: sortierte Liste von Sauggreifpunkten. Jeder Greifpunkt enthält auch die `item` Information, falls verfügbar.

Im Fall von ItemPickAI enthält jedes `item` das segmentierte Objekt vom Typ `BAG`, `CONSUMER_GOODS` oder `SHEET_METAL` mit seiner Pose bezogen auf den Mittelpunkt der Bounding Box (kleinste umschließende Box) des sichtbaren Teils des Objekts und die Abmessungen dieser Bounding Box.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "compute_grasps_extended",
  "response": {
    "grasps": [
      {
        "item": {
          "bounding_box": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "pose_frame": "string",
          "type": "string",
          "uuid": "string"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      },
      "max_suction_surface_length": "float64",
      "max_suction_surface_width": "float64",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "quality": "float64",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "type": "string",
      "uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "overfilled": "bool",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "rim_ledge": {
        "x": "float64",
        "y": "float64"
      },
      "rim_step_height": "float64",
      "rim_thickness": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


set_preferred_orientation

speichert die bevorzugte TCP-Orientierung zum Berechnen der Erreichbarkeit der Greifpunkte, die zur Filterung und optional zur Sortierung der vom `compute_grasps` und `compute_grasps_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 6.3.4.3).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/set_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/set_preferred_orientation
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_preferred_orientation",
  "response": {
    "return_code": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_preferred_orientation

gibt die bevorzugte TCP-Orientierung zurück, die für die Filterung und optional für die Sortierung der vom `compute_grasps` und `compute_grasps_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 6.3.4.3).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/get_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/get_preferred_orientation
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_preferred_orientation",
  "response": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

set_sorting_strategies

speichert die gewählte Strategie zur Sortierung der Greifpunkte, die vom `compute_grasps` und `compute_grasps_extended` Service zurückgeliefert werden (siehe *Berechnung der Greifpunkte*, Abschnitt 6.3.4.2).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/set_sorting_
↪strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/set_sorting_strategies
```

Request

Nur eine Sortierstrategie darf einen Gewichtswert `weight` größer als 0 haben. Wenn alle Werte für `weight` auf 0 gesetzt sind, wird die Standardsortierstrategie verwendet.

Wenn der Wert `weight` für `direction` gesetzt ist, muss `vector` den Richtungsvektor enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Wenn der Wert `weight` für `distance_to_point` gesetzt ist, muss `point` den Sortierpunkt enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "gravity": {
      "weight": "float64"
    },
    "surface_area": {
      "weight": "float64"
    }
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_sorting_strategies",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

get_sorting_strategies

gibt die gewählte Sortierstrategie zurück, die zur Sortierung der vom `compute-grasps` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Berechnung der Greifpunkte*, Abschnitt 6.3.4.2).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/get_sorting_
↪strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/get_sorting_strategies
```

Request

Dieser Service hat keine Argumente.

Response

Wenn alle Werte für `weight` 0 sind, wird die Standardsortierstrategie verwendet.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_sorting_strategies",
  "response": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "gravity": {
      "weight": "float64"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    },
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "surface_area": {
      "weight": "float64"
    }
  }
}
```

start

startet das Modul und versetzt es in den Zustand `RUNNING`.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/start
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/start
```

Request

Dieser Service hat keine Argumente.

Response

Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `IDLE` unterscheidenden Zustand zurück.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "start",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

stoppt das Modul und versetzt es in den Zustand `IDLE`.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/stop
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/stop
```

Request

Dieser Service hat keine Argumente.

Response

Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `IDLE` unterscheidenden Zustand zurück.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

trigger_dump

speichert die Detektion auf dem angeschlossenen USB Speicher, die dem übergebenen Zeitstempel entspricht, oder die letzte, falls kein Zeitstempel angegeben wurde.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/trigger_dump
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/trigger_dump
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "comment": "string",
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "trigger_dump",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`reset_defaults`

stellt die Werkseinstellungen der Parameter und der Sortierstrategie dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.3.4.8 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 6.33: Rückgabecodes der Services des ItemPick und ItemPickAI Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-3	Ein interner Timeout ist aufgetreten, beispielsweise während der Boxerkennung, wenn der Bereich der angegebenen Abmessungen zu groß ist.
-4	Die maximal erlaubte Zeitspanne für die interne Akquise der Bilddaten wurde überschritten.
-8	Das Template wurde während der Detektion gelöscht.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern, ROIs oder Templates überschritten wurde.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit
-12	Ressource ausgelastet, z.B. wenn `trigger_dump` zu häufig aufgerufen wird
-200	Ein schwerwiegender interner Fehler ist aufgetreten.
-301	Für die Anfrage zur Greifpunktberechnung `compute_grasps` oder `compute_grasps_extended` wurden mehrere Objektmodelle (`item_models`) übergeben.
10	Die maximal speicherbare Anzahl an Load Carriern, ROIs oder Templates wurde erreicht.
11	Mit dem Aufruf von `set_load_carrier` oder `set_region_of_interest` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
101	Es wurden keine gültigen Greifflächen in der Szene gefunden.
102	Der detektierte Load Carrier ist leer.
103	Alle berechneten Greifpunkte sind in Kollision.
112	Die Detektionen eines oder mehrerer Cluster wurden verworfen, da die minimale Clusterabdeckung nicht erreicht wurde.
300	Ein gültiges `robot_pose`-Argument wurde angegeben, ist aber nicht erforderlich.
999	Zusätzliche Hinweise für die Anwendungsentwicklung

6.3.5 BoxPick

6.3.5.1 Einführung

Das BoxPick Modul liefert eine gebrauchsfertige Perzeptionslösung, um robotische Pick-and-Place-Anwendungen zu realisieren. Es erkennt rechteckige Oberflächen und bestimmt ihre Position, Orientierung und Größe für das Greifen. Mit der +Match-Erweiterung kann BoxPick zur Detektion von texturierten Rechtecken mit konsistenten Orientierungen verwendet werden, z.B. für bedruckte Produktverpackungen, Etiketten, Broschüren oder Bücher.

Darüber hinaus bietet das Modul:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc_reason_stack* *Web GUI* (Abschnitt 7.1)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche der Szene auszuwählen (siehe *RoiDB*, Abschnitt 6.5.2)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 6.3.2), um in Bin-Picking-Anwendungen („Griff in die Kiste“) Greifpunkte nur für Objekte in dem erkannten Load Carrier zu berechnen
- die Unterstützung von Load Carriern mit Fächern, sodass Greifpunkte für Objekte nur in einem definierten Teilvolumen des Load Carriers berechnet werden
- eine Kollisionsprüfung zwischen Greifer und Load Carrier und/oder der Punktwolke
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1) kombiniert werden, um Greifposen in einem benutzerdefinierten externen Koordinatensystem zu liefern.

- einen Qualitätswert für jeden vorgeschlagenen Greifpunkt, der die Ebenheit der für das Greifen verfügbaren Oberfläche bewertet
- Auswahl einer Strategie zum Sortieren der zurückgelieferten Greifpunkte
- eine 3D Visualisierung des Detektionsergebnisses mit Greifpunkten und einer Greiferanimation in der Web GUI

Bemerkung: Dieses Softwaremodul ist pipelinespezifisch. Änderungen seiner Einstellungen oder Parameter betreffen nur die zugehörige Kamerapipeline und haben keinen Einfluss auf die anderen Pipelines, die auf dem *rc_reason_stack* laufen.

Bemerkung: In diesem Kapitel werden die Begriffe Cluster und Oberfläche synonym verwendet und bezeichnen eine Menge von Punkten (oder Pixeln) mit ähnlichen geometrischen Eigenschaften.

Das ItemPick Modul ist ein optional erhältliches Module, welches intern auf dem *rc_reason_stack* läuft und eine gesonderte BoxPick-*Lizenzen* (Abschnitt 8.2) benötigt. Die +Match-Erweiterung von BoxPick bedarf einer separaten Lizenz.

6.3.5.2 Erkennung von Rechtecken

Es gibt zwei verschiedene Typen von Objektmodellen für die Erkennung von Rechtecken im BoxPick Modul.

Standardmäßig unterstützt BoxPick nur Objektmodelle (`item_models`) des Typs (`type`) `RECTANGLE`. Mit der +Match-Erweiterung können auch Objektmodelle des Typs `TEXTURED_BOX` detektiert werden. Die Erkennung der verschiedenen Objektmodelltypen wird weiter unten beschrieben.

Optional können dem BoxPick-Modul folgende Informationen übergeben werden:

- die ID des Load Carriers, welcher die Objekte enthält
- ein Teilbereich innerhalb eines Load Carriers, in dem Objekte detektiert werden sollen
- die ID der Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die Region of Interest, in der nach Objekten gesucht wird
- die aktuelle Roboterpose, wenn die Kamera am Roboter montiert ist und als Koordinatensystem `external` gewählt wurde, oder die gewählte Region of Interest im externen Koordinatensystem definiert ist

Die zurückgegebene Pose `pose` eines detektierten Objekts `item` ist die Pose des Mittelpunkts des erkannten Rechtecks im gewünschten Koordinatensystem `pose_frame`, wobei die z-Achse in Richtung der Kamera zeigt und die x-Achse parallel zu langen Seite des Rechtecks ausgerichtet ist. Diese Pose hat eine 180° Mehrdeutigkeit in der Rotation um die z-Achse, welche durch Nutzung der +Match-Erweiterung im BoxPick Modul aufgelöst werden kann. Jedes erkannte Rechteck beinhaltet eine `uuid` (Universally Unique Identifier) und den Zeitstempel `timestamp` des ältesten Bildes, das für die Erkennung benutzt wurde.

Erkennung von Objekten des Typs RECTANGLE

Das BoxPick-Modul unterstützt mehrere Objektmodelle (`item_models`) vom Typ (`type`) Rechteck (`RECTANGLE`). Jedes Rechteck ist durch seine minimale und maximale Größe definiert, wobei die minimale Größe kleiner als die maximale Größe sein muss. Die Abmessungen sollten relativ genau angegeben werden, um Fehldetektionen zu verhindern, jedoch eine gewisse Toleranz beinhalten, um Messunsicherheiten und mögliche Produktionsabweichungen zu berücksichtigen.

Die Erkennung der Rechtecke läuft in mehreren Schritten ab. Zuerst wird die Punktwolke in möglichst ebene Segmente (Cluster) unterteilt. Dann werden gerade Liniensegmente in den 2D Bildern erkannt und auf die zugehörigen Clusterflächen projiziert. Die Cluster und die erkannten Linien werden in der

„Zwischenergebnis“ Visualisierung auf der *BoxPick* Seite in der Web GUI angezeigt. Schließlich werden für jedes Cluster die am besten zu den erkannten Linien passenden Rechtecke extrahiert.

Erkennung von Objekten des Typs RECTANGLE (BoxPick+Match)

Mit der +Match-Erweiterung unterstützt BoxPick zusätzlich Objektmodelle (`item_models`) des Typs (`type`) `TEXTURED_BOX`. Wenn dieser Objektmodelltyp verwendet wird, kann nur ein einzelnes Objektmodell pro Anfrage angegeben werden.

Das `TEXTURED_BOX` Objektmodell sollte für die Detektion mehrerer Rechtecke mit gleicher Textur, d.h. gleichem Aussehen oder Aufdruck, verwendet werden, wie zum Beispiel bedruckte Produktverpackungen, Etiketten, Broschüren oder Bücher. Es wird vorausgesetzt, dass die Textur bei allen Objekten gleich positioniert ist in Bezug auf die Objektgeometrie. Weiterhin sollte die Textur nicht repetitiv sein.

Ein Objekt vom Typ `TEXTURED_BOX` wird definiert durch die exakten Abmessungen `dimensions` des Objekts in `x`, `y` und `z` (wobei nur `z` 0 sein darf) sowie eine Toleranz `dimensions_tolerance_m` die angibt, wie stark die Abmessungen der erkannten Rechtecke von den gegebenen Dimensionen abweichen dürfen. Als Standardwert wird eine Toleranz von 0.01 m angenommen. Des Weiteren muss eine `template_id` angegeben werden, über die die spezifizierten Abmessungen und die Texturen der erkannten Rechtecke referenziert werden. Zusätzlich können die maximal mögliche Verformung der Objekte (`max_deformation_m`) in Metern angegeben werden (Standardwert 0.004 m), um steifere oder flexiblere Objekte zu beschreiben.

Wird eine `template_id` zum ersten Mal verwendet, dann detektiert BoxPick die Rechtecke so wie für den Objektmodelltyp `RECTANGLE` beschrieben, und nutzt die angegebene Toleranz um den Abmessungsbereich für die Erkennung festzulegen. Wenn zusätzlich zu x und y auch die z Abmessungen gegeben sind, werden Rechtecke mit allen möglichen Kombinationen der drei Abmessungen erkannt. Aus den erkannten Rechtecken werden sogenannte *Views* erzeugt, die die Form und die Bildintensitätswerte der Rechtecke beinhalten, und werden in einem neu erzeugten Template mit der angegebenen `template_id` gespeichert. Die Views werden schrittweise erzeugt: Beginnend bei dem Rechteck mit dem höchsten Erkennungs-Score wird ein View erzeugt und direkt verwendet, um weitere Rechtecke mit derselben Textur zu finden. Dann werden in allen verbleibenden Clustern weitere Rechtecke mit den angegebenen Abmessungen detektiert und es wird wiederum aus dem besten Rechteck ein View generiert, der für weitere Erkennungen genutzt wird. Jedes Template kann bis zu 10 verschiedene Views speichern, zum Beispiel um verschiedene Sorten derselben Produktverpackung abzubilden. Jeder View hat eine eindeutige ID (`view_uuid`) und alle Rechtecke mit gleicher Textur erhalten dieselbe `view_uuid`. Das bedeutet auch, dass alle Objekte (`items`) mit derselben `view_uuid` konsistente Orientierungen haben, da die Orientierung jedes Objekts an der Textur ausgerichtet ist. Die Views können angezeigt und gelöscht werden, und ihre Orientierungen können über die *Web GUI* (Abschnitt 7.1) geändert werden, indem das Template oder sein Editierbutton in der Templateübersicht angeklickt wird. Jedes erkannte Objekt hat ein Feld `view_pose_set`, welches angibt, ob die Orientierung des zum Objekt zugeordneten Views explizit gesetzt wurde, oder ob sie unbestimmt in einem zufälligen Zustand ist, welcher eine 180° Mehrdeutigkeit hat. Weiterhin kann ein benutzerdefinierter Name für jeden View gesetzt werden, der gemeinsam mit der `view_uuid` zurückgegeben wird und die Identifikation bestimmter Views vereinfacht. Der Typ `type` eines zurückgelieferten Objekts mit einer `view_uuid` lautet `TEXTURED_RECTANGLE`.

Wenn ein Template mit der angegebenen `template_id` bereits existiert, werden die vorhandenen Views verwendet, um Rechtecke anhand ihrer Textur zu erkennen. Wenn weitere Rechtecke gefunden werden, die ebenfalls passende Abmessungen haben, aber eine andere Textur, dann werden neue Views generiert und dem Template hinzugefügt. Wenn die maximale Anzahl Views erreicht ist, werden zu selten detektierte Views gelöscht, damit neu generierte Views dem Template hinzugefügt werden können, und das Template aktuell gehalten wird. Um zu verhindern, dass ein Template durch neue Views aktualisiert wird, kann das automatische Updaten der Views in der Web GUI aus- und eingeschaltet werden. Die Dimensionstoleranz `dimensions_tolerance_m` und die maximale Verformung `max_deformation_m` können dort ebenso für jedes Template geändert werden. Die maximale Verformung bestimmt die Toleranz für die Texturerkennung, die nötig ist, wenn sich durch flexible Objektoberflächen Teile der Textur relativ zueinander verschieben. Für steife Objekte sollte die maximale Verformung möglichst niedrig gesetzt werden, um eine hohe Erkennungsgenauigkeit zu erreichen.

Die Abmessungen `dimensions` des Templates können nur beim Erstellen eines neuen Templates angegeben werden. Sobald das Template erzeugt wurde, können die Abmessungen nicht mehr geändert werden und müssen beim Aufruf der Erkennung auch nicht angegeben werden. Wenn die Abmessungen dennoch beim Aufruf angegeben werden, müssen sie mit den Abmessungen im vorhandenen Template übereinstimmen. Die Toleranz `dimensions_tolerance_m` und die maximale Verformung `max_deformation_m` können jedoch für jeden Detektionsaufruf unterschiedlich angegeben werden und ihre Werte werden auch im gespeicherten Template entsprechend aktualisiert.

6.3.5.3 Berechnung der Greifpunkte

Das BoxPick-Modul bietet einen Service, um Greifpunkte für Sauggreifer zu berechnen. Der Sauggreifer ist durch die Länge und Breite der Greiffläche definiert.

Die Greifpunkte werden auf den erkannten Rechtecken `items` berechnet (siehe *Erkennung von Rechtecken*, Abschnitt 6.3.5.2).

Optional können dem Modul weitere Informationen zur Greifpunktberechnung übergeben werden:

- die ID des Load Carriers, welcher die zu greifenden Objekte enthält
- ein Unterabteil (`load_carrier_compartment`) innerhalb eines Load Carriers, in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteile*, Abschnitt 6.5.1.3).
- die ID der 3D Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die 3D Region of Interest, innerhalb der Greifpunkte berechnet werden
- Informationen für die Kollisionsprüfung: Die ID des Greifers, um die Kollisionsprüfung zu aktivieren, und optional ein Greif-Offset, der die Vorgreifposition definiert. Details zur Kollisionsprüfung sind in *CollisionCheck* (Abschnitt 6.3.5.5) gegeben.

Ein vom BoxPick-Modul ermittelter Greifpunkt repräsentiert die empfohlene Pose des TCP (Tool Center Point) des Sauggreifers. Der Greifpunkt `type` ist immer auf `SUCTION` gesetzt. Für jeden Greifpunkt liegt der Ursprung der Greifpose `pose` im Mittelpunkt der größten von der jeweiligen Greiffläche umschlossenen Ellipse. Die Orientierung des Greifpunkts ist ein rechtshändiges Koordinatensystem, sodass die z-Achse orthogonal zur Greiffläche in das zu greifende Objekt zeigt und die x-Achse entlang der längsten Ausdehnung ausgerichtet ist.Da die x-Achse zwei mögliche Richtungen haben kann, wird diejenige ausgewählt, die besser zur bevorzugten TCP-Ausrichtung passt (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 6.3.5.4). Wenn der Laufzeitparameter `allow_any_grasp_z_rotation` auf True gesetzt ist, wird die x-Achse nicht zwangsweise an der maximalen Dehnung der greifbaren Ellipse ausgerichtet, sondern kann eine beliebige Drehung um die z-Achse aufweisen. In diesem Fall hat der zurückgegebene Greifpunkt die Ausrichtung, die am besten zur bevorzugten TCP-Ausrichtung passt und kollisionsfrei ist, wenn die Kollisionsprüfung aktiviert ist.

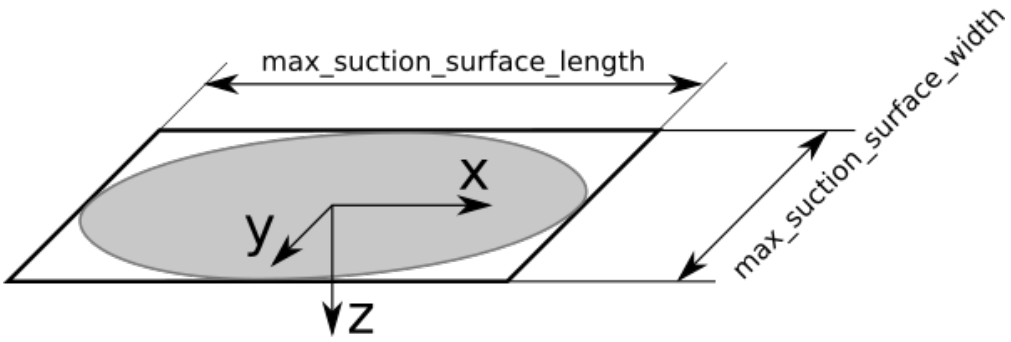


Abb. 6.12: Veranschaulichung eines berechneten Greifpunktes mit Koordinatensystem und der zugehörigen Ellipse, welche die größtmögliche Greiffläche beschreibt.

Zusätzlich enthält jeder Greifpunkt die Abmessungen der maximal verfügbaren Greiffläche, die als Ellipse mit den Achslängen `max_suction_surface_length` und `max_suction_surface_width` beschrie-

ben wird. Der Nutzer kann Greifpunkte mit zu kleinen Greifflächen herausfiltern, indem die minimalen Abmessungen der Greiffläche, die vom Sauggreifer benötigt wird, angegeben werden. Wenn der Laufzeitparameter `allow_any_grasp_z_rotation` auf True gesetzt ist, dann sind die Achslängen `max_suction_surface_length` und ‘‘max_suction_surface_width‘‘gleich und entsprechen der kürzeren Achse der größtmöglichen Greifellipse.

Im BoxPick-Modul entspricht der Greifpunkt dem Zentrum des detektierten Rechtecks. Wenn BoxPick mit einem Objektmodell vom Typ `RECTANGLE` aufgerufen wird, entsprechen die Achslängen der Greiffläche der Länge und Breite des erkannten Rechtecks. In diesem Fall erhalten Rechtecke keinen Greifpunkt, wenn mehr als 15% ihrer Fläche durch andere Objekte verdeckt ist oder ungültige Datenpunkte hat.

Wenn BoxPick mit einem Objektmodell vom Typ `TEXTURED_BOX` aufgerufen wird, können Greifpunkt auch auf teilweise verdeckten Rechtecken berechnet werden. Die zurückgegebene maximale Sauggreiffläche entspricht dann der freien Oberfläche des Rechtecks, die nicht durch andere Cluster verdeckt ist.

Jeder Greifpunkt enthält auch einen Qualitätswert (`quality`), der einen Hinweis auf die Ebenheit der Greiffläche gibt. Dieser Wert reicht von 0 bis 1, wobei höhere Werte für eine ebenere rekonstruierte Oberfläche stehen.

Jeder berechnete Greifpunkt lässt sich anhand einer `uuid` (Universally Unique Identifier) eindeutig identifizieren und enthält zusätzlich den Zeitstempel der ältesten Bildaufnahme, auf der die Greifpunktberechnung durchgeführt wurde.

Die Sortierung der Greifpunkte basiert auf der ausgewählten Sortierstrategie. Folgende Sortierstrategien sind verfügbar und können über die *Web GUI* (Abschnitt 7.1) oder über den `set_sorting_strategies` Service gesetzt werden:

- `gravity`: höchste Greifpunkte entlang der Gravitationsrichtung werden zuerst zurückgeliefert.
- `surface_area`: Greifpunkte mit den größten Oberflächen werden zuerst zurückgeliefert.
- `direction`: Greifpunkte mit dem kleinsten Abstand entlang der gesetzten Richtung `vector` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.
- `distance_to_point`: Greifpunkte mit dem kleinsten oder größten (falls `farthest_first` auf true gesetzt ist) Abstand von einem gesetzten Sortierpunkt `point` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.

Wenn keine Sortierstrategie gesetzt ist, oder die Standard-Sortierstrategie in der Web GUI ausgewählt ist, geschieht die Sortierung der Greifpunkte basierend auf einer Kombination von `gravity` und `surface_area`.

6.3.5.4 Setzen der bevorzugten TCP-Orientierung

Das BoxPick-Modul berechnet die Erreichbarkeit von Greifpunkten basierend auf der bevorzugten Orientierung des TCPs. Die bevorzugte Orientierung kann über den Service `set_preferred_orientation` oder über die *CADMatch*-Seite in der Web GUI gesetzt werden. Die bevorzugten Orientierung des TCPs wird genutzt, um Greifpunkte zu verwerfen, die der Greifer nicht erreichen kann, und kann auch zur Sortierung der Greifpunkte genutzt werden.

Die bevorzugte TCP-Orientierung kann im Kamerakoordinatensystem oder im externen Koordinatensystem gesetzt werden, wenn eine Hand-Auge-Kalibrierung verfügbar ist. Wenn die bevorzugte TCP-Orientierung im externen Koordinatensystem definiert ist, und die Kamera am Roboter montiert ist, muss bei jedem Aufruf der Objekterkennung die aktuelle Roboterpose angegeben werden. Wenn keine bevorzugte TCP-Orientierung gesetzt wird, wird die Orientierung der linken Kamera (siehe Coordinate frames im rc_visard Handbuch) als die bevorzugte TCP-Orientierung genutzt.

6.3.5.5 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_reason_stack* laufenden Module liefern Daten für das BoxPick-Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieses Moduls kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des Boxpick-Moduls haben.

Kamera- und Tiefendaten

Folgende Daten werden vom BoxPick-Modul verarbeitet:

- die rektifizierten Bilder des *Kamera Modul* (`rc_camera`, Abschnitt 6.1)
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 6.2.2), falls eine Stereokamera verwendet wird.
- die Disparitäts-, Konfidenz- und Fehlerbilder der *Orbbec Modul* (`rc_orbbec`, Abschnitt 6.2.4), falls eine *Orbbec* Kamera verwendet wird
- die Disparitäts-, Konfidenz- und Fehlerbilder der *Zivid Modul* (`rc_zivid`, Abschnitt 6.2.3), falls eine *zivid* Kamera verwendet wird

Für alle genutzten Bilder ist garantiert, dass diese nach dem Auslösen des Services aufgenommen wurden.

IOControl und Projektor-Kontrolle

Für den Anwendungsfall, dass der *rc_reason_stack* zusammen mit einem externen Musterprojektor und dem Modul für *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 6.4.4) betrieben wird, wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 6.2.2.1), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 6.4.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das BoxPick-Modul automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 6.3.5.8) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das ItemPick- oder BoxPick-Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

Für den Fall einer robotergeführten Kamera ist es abhängig von `pose_frame` und der Sortierrichtung bzw. des Sortierpunktes nötig, zusätzlich die aktuelle Roboterpose (`robot_pose`) zur Verfügung zu stellen:

- Wenn `external` als `pose_frame` ausgewählt ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die Sortierrichtung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn der Sortierpunkt für die Abstandssortierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- In allen anderen Fällen ist die Angabe der Roboterpose optional.

LoadCarrier

Das BoxPick-Modul nutzt die Funktionalität zur Load Carrier Erkennung aus dem *LoadCarrier* Modul (`rc_load_carrier`, Abschnitt 6.3.2) mit den Laufzeitparametern, die für dieses Modul festgelegt wurden. Wenn sich jedoch mehrere Load Carrier in der Szene befinden, die zu der angegebenen Load Carrier ID passen, wird nur einer davon zurückgeliefert. In diesem Fall sollte eine 3D Region of Interest gesetzt werden, um sicherzustellen, dass immer derselbe Load Carrier für das BoxPick-Modul verwendet wird.

Der Load Carrier wird verwendet um Fehldetektionen zu filtern, wenn BoxPick mit einem Objektmodell vom Typ `TEXTURED_BOX` aufgerufen wird, und alle drei Dimensionen x, y und z angegeben werden. In diesem Fall werden intern 3D Boxen generiert, indem die erkannten Rechtecken um die fehlende Dimension erweitert werden. Es werden dann nur die erkannten Rechtecke zurückgeliefert, bei denen die entsprechende 3D Box vollständig im Load Carrier enthalten ist.

CollisionCheck

Die Kollisionsprüfung kann für die Greifpunktberechnung des BoxPick-Moduls aktiviert werden, indem das `collision_detection` Argument an den `compute_grasps` oder `compute_grasps_extended` Service übergeben wird. Es enthält die ID des benutzten Greifers und optional einen Greif-Offset. Der Greifer muss im GripperDB Modul definiert werden (siehe *Erstellen eines Greifers*, Abschnitt 6.5.3.2) und Details über die Kollisionsprüfung werden in *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 6.4.2.2) gegeben.

Wenn die Kollisionsprüfung aktiviert ist, werden nur kollisionsfreie Greifpunkte zurückgeliefert. Jedoch werden in den Visualisierungen auf der *BoxPick*-Seite der Web GUI kollidierende Greifpunkte als schwarze Ellipsen dargestellt.

Die Laufzeitparameter des CollisionCheck-Moduls beeinflussen die Kollisionserkennung wie in *CollisionCheck-Parameter* (Abschnitt 6.4.2.3) beschrieben.

6.3.5.6 Parameter

Das BoxPick-Modul wird in der REST-API als `rc_boxpick` bezeichnet und in der *Web GUI* (Abschnitt 7.1) in der gewünschten Pipeline unter *Module* → *BoxPick* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 7.2) ändern.

Übersicht über die Parameter

Bemerkung: Die Defaultwerte in der Tabelle unten zeigen die Werte des *rc_visard*. Diese Werte können sich bei anderen Sensoren unterscheiden.

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.34: Laufzeitparameter des `rc_boxpick` Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`allow_any_grasp_z_rotation`	bool	false	true	false	Bestimmt, ob die Greifpunkte beliebige Orientierung haben dürfen, anstatt an der Hauptachse der greifbaren Ellipse ausgerichtet zu sein
`allow_untextured_detections`	bool	false	true	false	Gibt an, ob auch untexturierte Rechtecke zurückgegeben werden sollen, wenn ein Modell vom Typ TEXTURED_BOX angegeben wurde
`check_collisions_with_point_cloud`	bool	false	true	false	Gibt an, ob Kollisionen zwischen Greifer und anderen Matches geprüft werden
`cluster_max_curvature`	float64	0.005	0.5	0.11	Maximal erlaubte Krümmung für Greifflächen
`clustering_discontinuity_factor`	float64	0.1	5.0	1.0	Erlaubte Unebenheit von Greifflächen
`clustering_max_surface_rmse`	float64	0.0005	0.01	0.004	Maximal erlaubte Abweichung (Root Mean Square Error, RMSE) von Punkten zur Greiffläche in Metern
`grasp_filter_orientation_threshold`	float64	0.0	180.0	45.0	Maximal erlaubte Orientierungsabweichung zwischen Greifpunkt und bevorzugter TCP-Orientierung in Grad
`line_sensitivity`	float64	0.1	1.0	0.1	Empfindlichkeit des Liniendetektors
`manual_line_sensitivity`	bool	false	true	false	Gibt an, ob die benutzerdefinierte Linienempfindlichkeit oder die automatische genutzt werden soll
`max_grasps`	int32	1	100	5	Maximale Anzahl von bereitgestellten Greifpunkten
`min_cluster_coverage`	float64	0.0	0.99	0.0	Bestimmt den minimalen Anteil an Punkten pro Cluster, die durch Detektionen abgedeckt sein müssen
`mode`	string	-	-	Unconstrained	Modus der Rechteckerkennung: [Unconstrained, PackedGridLayout, PackedLayers]
`prefer_splits`	bool	false	true	false	Gibt an, ob Rechtecke in kleinere Rechtecke gesplittet werden sollen, falls möglich

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf der *BoxPick*-Seite in der Web GUI dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet:

`max_grasps` (*Anzahl Greifpunkte*)

ist die maximale Anzahl von bereitgestellten Greifpunkten.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?max_grasps=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?max_grasps=<value>
```

`cluster_max_curvature` (*Maximale Krümmung*)

ist die maximal erlaubte Krümmung für Greifflächen. Je kleiner dieser Wert ist, desto mehr mögliche Greifflächen werden in kleinere Flächen mit weniger Krümmung aufgeteilt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?cluster_max_
↪curvature=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?cluster_max_curvature=<value>
```

`clustering_discontinuity_factor` (*Unstetigkeitsfaktor*)

beschreibt die erlaubte Unebenheit von Greifflächen. Je kleiner dieser Wert ist, umso mehr werden mögliche Greifflächen in kleinere Flächen mit weniger Unebenheiten aufgeteilt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?clustering_
↪discontinuity_factor=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?clustering_discontinuity_factor=
↪<value>
```

`clustering_max_surface_rmse` (*Maximaler RMSE*)

ist die maximal erlaubte Abweichung (Root Mean Square Error, RMSE) von Punkten zur Greiffläche in Metern.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?clustering_
↪max_surface_rmse=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?clustering_max_surface_rmse=
↪<value>
```

`mode` (*Modus*)

legt den Modus der Rechteckerkennung fest. Mögliche Werte sind `Unconstrained` (*Unbeschränkt*), `PackedGridLayout` (*Dichtes Gitterlayout*) und `PackedLayer` (*Dicht geschichtet*). Im Modus `PackedGridLayout` werden Rechtecke eines Clusters in einem dichten Gittermuster erkannt. Im Modus `PackedLayers` wird angenommen, dass die Boxen Schichten (Layer) bilden, und die Erkennung der Boxen startet an den Ecken des Clusters. Dieser Modus sollte für Depalettierszenarien genutzt werden. Im Modus `Unconstrained` (Standardwert) werden Rechtecke unabhängig von ihren relativen Positionen zueinander und ihren Positionen im Cluster erkannt. Abb. 6.13 zeigt die Modi für verschiedene Szenarien.

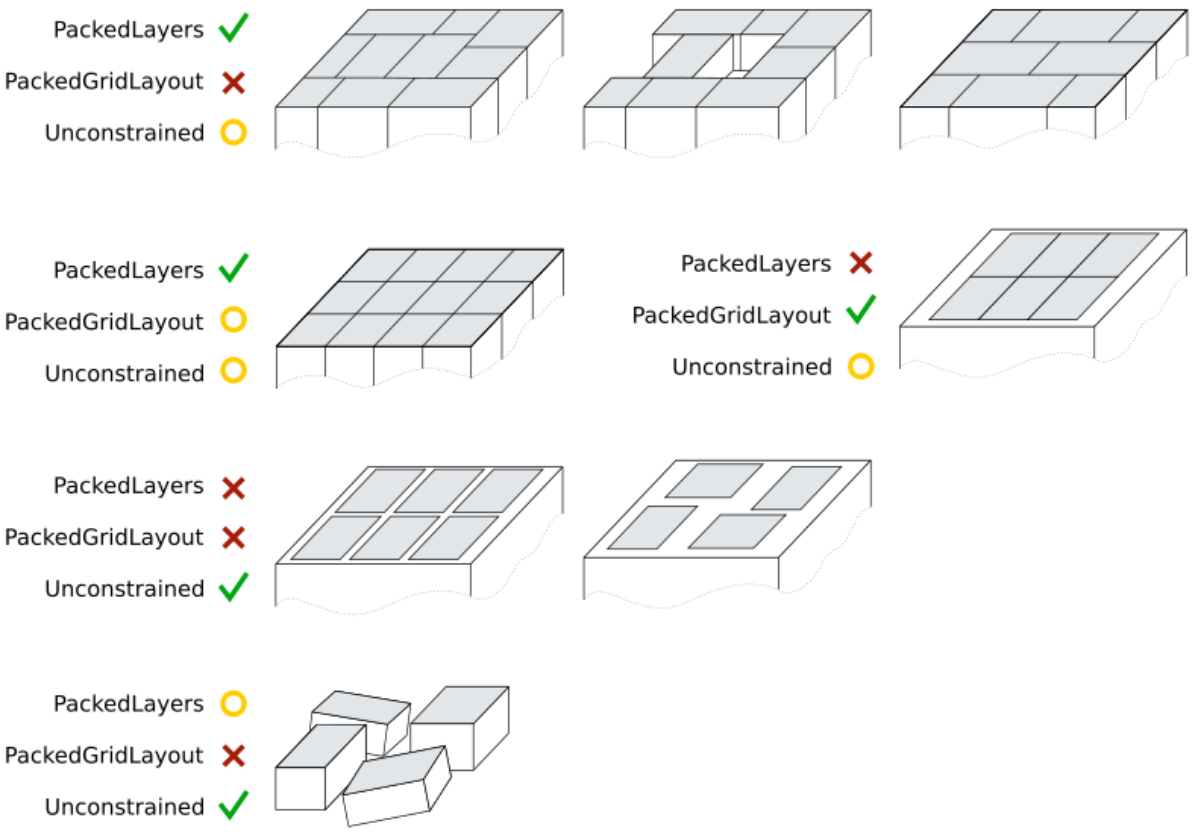


Abb. 6.13: Darstellung geeigneter BoxPick Modi für unterschiedliche Szenen. Gelb markierte Modi sind anwendbar, aber nicht empfohlen für das jeweilige Szenario. Die grauen Flächen markieren die Rechtecke, die erkannt werden sollen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?mode=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?mode=<value>
```

`manual_line_sensitivity` (*Manuelle Linienempfindlichkeit*)

legt fest, ob die benutzerdefinierte Linienempfindlichkeit für die Liniendetektion zur Rechteckerkennung verwendet werden soll. Wenn dieser Parameter auf true gesetzt ist, wird der benutzerdefinierte Wert in `line_sensitivity` (Linienempfindlichkeit) zur Detektion verwendet, andernfalls wird die Linienempfindlichkeit automatisch ermittelt. Dieser Parameter sollte auf true gesetzt werden, wenn die automatische Linienempfindlichkeit nicht genügend Linien an den Rändern der Boxen liefert, sodass Boxen nicht erkannt werden. Die detektierten Linien werden in der „Zwischenergebnis“ Visualisierung auf der *BoxPick* Seite in der Web GUI angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?manual_line_
↪sensitivity=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?manual_line_sensitivity=<value>
```

`line_sensitivity` (*Linienempfindlichkeit*)

legt die Empfindlichkeit für die Detektion von Linien für die Rechteckerkennung fest, wenn der Parameter `manual_line_sensitivity` (Manuelle Linienempfindlichkeit) auf true gesetzt ist. Andernfalls hat dieser Parameter keinen Einfluss auf die Rechteckerkennung. Höhere Werte liefern mehr Liniensegmente, aber erhöhen auch die Laufzeit der Detektion. Dieser Parameter sollte erhöht werden, wenn Boxen nicht erkannt werden können, weil ihre Ränder nicht als Linien detektiert werden. Die erkannten Linien werden in der „Zwischenergebnis“ Visualisierung auf der *BoxPick* Seite in der Web GUI angezeigt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?line_
↪sensitivity=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?line_sensitivity=<value>
```

`prefer_splits` (*Splitting bevorzugen*)

bestimmt, ob Rechtecke in kleinere Rechtecke aufgesplittet werden, falls die kleineren Rechtecke ebenfalls den angegebenen Objektmodellen entsprechen. Dieser Parameter sollte auf true gesetzt werden, wenn Boxen dicht beieinander liegen, und die Objektmodelle auch zu einem Rechteck der Größe von zwei angren-

zenden Boxen passen. Wenn dieser Parameter auf false steht, werden in solch einem Fall die Rechtecke bevorzugt, die sich aus zwei angrenzenden Boxen ergeben.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?prefer_splits=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?prefer_splits=<value>
```

`min_cluster_coverage` (*Minimale Clusterabdeckung*)

bestimmt den Anteil von Punkten in jedem segmentierten Cluster, der durch Rechtecksdetektionen abgedeckt sein muss, um diese Detektionen als valide anzunehmen. Wird die minimale Clusterabdeckung unterschritten, werden für das jeweilige Cluster keine Detektionen zurückgeliefert und eine Warnung ausgegeben. Dieser Parameter sollte genutzt werden, um in einem Depalettierszenario zu verifizieren, dass alle Objekte in einem Layer detektiert wurden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?min_cluster_
↪coverage=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?min_cluster_coverage=<value>
```

`allow_untextured_detections` (Nur für BoxPick+Match, *Untexturierte Detektionen*)

ermöglicht die Rückgabe aller Rechtecke, die den angegebenen Templateabmessungen entsprechen, auch wenn sie mit keinem vorhandenen View gematcht werden können oder wenn sie nicht über genügend Textur verfügen, um daraus einen neuen View zu generieren. Das Deaktivieren dieses Parameters führt zu schnellerer Laufzeit, wenn ein Template verwendet wird, für das automatische View Updates gesperrt sind.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?allow_
↪untextured_detections=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?allow_untextured_detections=
↪<value>
```

`grasp_filter_orientation_threshold` (*Grasp Orientation Threshold*)

ist die maximale Abweichung der TCP-z-Achse am Greifpunkt von der z-Achse der bevorzugten TCP-Orientierung in Grad. Es werden nur Greifpunkte zurückgeliefert, deren Orientierungsabweichung kleiner als der angegebene Wert ist. Falls der Wert auf Null gesetzt wird, sind alle Abweichungen valide.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?grasp_filter_
↪orientation_threshold=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?grasp_filter_orientation_
↪threshold=<value>
```

`allow_any_grasp_z_rotation` (*Allow Any Grasp Z Rotation*)

Wenn der Wert auf True gesetzt ist, werden die x-Achsen der zurückgegebenen Greifpunkte nicht mehr notwendigerweise an der maximalen Ausdehnung der greifbaren Ellipse ausgerichtet, sondern können eine beliebige Drehung um die z-Achse haben. Die zurückgegebenen Werte von `max_suction_surface_length` und `max_suction_surface_width` sind dann gleich und entsprechen dem kleinsten Durchmesser der größten greifbaren Ellipsenfläche. Dieser Parameter eröffnet dem Roboter mehr Optionen zum Greifen von Objekten, insbesondere in Szenen, in denen es zu Kollisionen kommen kann. Da der Greifpunkt jedoch nicht mehr mit der greifbaren Ellipse ausgerichtet, muss die richtige Orientierung zum Platzieren des Objekts anhand der Pose des zugehörigen `item` ermittelt werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?allow_any_
↪grasp_z_rotation=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?allow_any_grasp_z_rotation=<value>
```

`check_collisions_with_point_cloud` (*Kollisionsprüfung mit Punktwolke*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `compute_grasps` oder `compute_grasps_extended` Service aktiviert ist. Wenn `check_collisions_with_point_cloud` auf true gesetzt ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit dieser Punktwolke wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/parameters?check_
↪collisions_with_point_cloud=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?check_collisions_with_point_cloud=
↪<value>
```

6.3.5.7 Statuswerte

Das `rc_boxpick` Modul meldet folgende Statuswerte:

Tab. 6.35: Statuswerte des `rc_boxpick` Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste.
`grasp_computation_time`	Laufzeit für die Greifpunktberechnung beim letzten Aufruf in Sekunden
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`load_carrier_detection_time`	Laufzeit für die letzte Load Carrier Erkennung in Sekunden
`processing_time`	Laufzeit für die letzte Erkennung (einschließlich Load Carrier Detektion) in Sekunden
`state`	Aktueller Zustand des BoxPick-Moduls

Folgende `state`-Werte werden gemeldet.

Tab. 6.36: Mögliche Werte für den Zustand des BoxPick Moduls

Zustand	Beschreibung
IDLE	Das Modul ist inaktiv.
RUNNING	Das Modul wurde gestartet und ist bereit, Load Carrier zu erkennen und Greifpunkte zu berechnen.
FATAL	Ein schwerwiegender Fehler ist aufgetreten.

6.3.5.8 Services

Die angebotenen Services von `rc_boxpick` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der *rc_reason_stack Web GUI* (Abschnitt 7.1) ausprobiert und getestet werden.

Das BoxPick-Modul stellt folgende Services zur Verfügung.

`detect_items`

löst die Erkennung von Rechtecken aus, wie in *Erkennung von Rechtecken* (Abschnitt 6.3.5.2) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/detect_items
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/detect_items
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.5.5).

`item_models`: Liste der zu erkennenden Objektmodelle. Der Typ `type` der Modelle muss immer `RECTANGLE` oder `TEXTURED_BOX` sein. Für den Typ `RECTANGLE` muss das Feld `rectangle` gefüllt werden, wohingegen für `TEXTURED_BOX` das Feld `textured_box` angegeben werden muss. Siehe *Erkennung von Rechtecken* (Abschnitt 6.3.5.2) für eine ausführliche Beschreibung der Objektmodelle.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.5.5).

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, welcher die zu erkennenden Objekte enthält.

`load_carrier_compartment`: Teilvolumen (Fach oder Abteil) in einem zu detektierenden Load Carrier (Behälter), in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteile*, Abschnitt 6.5.1.3).

`region_of_interest_id`: Falls `load_carrier_id` gesetzt ist, die ID der 3D Region of Interest, innerhalb welcher nach dem Load Carrier gesucht wird. Andernfalls die ID der 3D Region of Interest, in der nach Objekten gesucht wird.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "item_models": [
      {
        "rectangle": {
          "max_dimensions": {
            "x": "float64",
            "y": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64"
          }
        },
        "textured_box": {
          "dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "dimensions_tolerance_m": "float64",
          "max_deformation_m": "float64",
          "template_id": "string"
        },
        "type": "string"
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`items`: Liste von erkannten Rechtecken.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "detect_items",
  "response": {
    "items": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rectangle": {
          "x": "float64",
          "y": "float64"
        },
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      "template_id": "string",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "type": "string",
      "uuid": "string",
      "view_name": "string",
      "view_pose_set": "bool",
      "view_uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "overfilled": "bool",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "rim_ledge": {
        "x": "float64",
        "y": "float64"
      },
      "rim_step_height": "float64",
      "rim_thickness": {
        "x": "float64",
        "y": "float64"
      },
      "type": "string"
    }
  ],
  "return_code": {
    "message": "string",
    "value": "int16"
  },
  "timestamp": {
    "nsec": "int32",
    "sec": "int32"
  }
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

compute_grasps

löst die Erkennung von Rechtecken und Berechnung von Greifposen für diese Rechtecke aus, wie in *Berechnung der Greifpunkte* (Abschnitt 6.3.5.3) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/compute_grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/compute_grasps
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.5.5).

`item_models`: Liste der zu erkennenden Objektmodelle. Der Typ `type` der Modelle muss immer `RECTANGLE` oder `TEXTURED_BOX` sein. Für den Typ `RECTANGLE` muss das Feld `rectangle` gefüllt werden, wohingegen für `TEXTURED_BOX` das Feld `textured_box` angegeben werden muss. Siehe *Erkennung von Rechtecken* (Abschnitt 6.3.5.2) für eine ausführliche Beschreibung der Objektmodelle.

`suction_surface_length`: Länge der Greiffläche des verwendeten Vakuum-Greifsystems.

`suction_surface_width`: Breite der Greiffläche des verwendeten Vakuum-Greifsystems.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.5.5).

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, welcher die zu greifenden Objekte enthält.

`load_carrier_compartment`: Teilvolumen (Fach oder Abteil) in einem zu detektierenden Load Carrier (Behälter), in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteile*, Abschnitt 6.5.1.3).

`region_of_interest_id`: Falls `load_carrier_id` gesetzt ist, die ID der 3D Region of Interest, innerhalb welcher nach dem Load Carrier gesucht wird. Andernfalls die ID der 3D Region of Interest, innerhalb der Greifpunkte berechnet werden.

`collision_detection`: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 6.4.2.2)

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "pre_grasp_offset": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    }
  },
  "item_models": [
    {
      "rectangle": {
        "max_dimensions": {
          "x": "float64",
          "y": "float64"
        },
        "min_dimensions": {
          "x": "float64",
          "y": "float64"
        }
      },
      "textured_box": {
        "dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "dimensions_tolerance_m": "float64",
        "max_deformation_m": "float64",
        "template_id": "string"
      },
      "type": "string"
    }
  ],
  "load_carrier_compartment": {
    "box": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  },
  "load_carrier_id": "string",
  "pose_frame": "string",
  "region_of_interest_id": "string",
  "robot_pose": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "suction_surface_length": "float64",
    "suction_surface_width": "float64"
  }
}
```

Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`grasps`: sortierte Liste von Sauggreifpunkten.

`items`: Liste von erkannten Rechtecken, die zu den zurückgelieferten Greifpunkten gehören.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "compute_grasps",
  "response": {
    "grasps": [
      {
        "item_uuid": "string",
        "max_suction_surface_length": "float64",
        "max_suction_surface_width": "float64",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "quality": "float64",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "type": "string",
        "uuid": "string"
      }
    ],
    "items": [
      {
        "grasp_uuids": [
          "string"
        ],
        "pose": {
          "orientation": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rectangle": {
          "x": "float64",
          "y": "float64"
        },
        "template_id": "string",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "type": "string",
        "uuid": "string",
        "view_name": "string",
        "view_pose_set": "bool",
        "view_uuid": "string"
      }
    ],
    "load_carriers": [
      {
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


compute_grasps_extended

löst die Erkennung von Rechtecken und Berechnung von Greifposen für diese Rechtecke aus. Dieser Service verhält sich analog zu `compute_grasps`, gibt aber die Objektinformationen für jeden Greifpunkt direkt zurück, anstatt sie in einer separaten Liste zu speichern. Dies ermöglicht ein einfacheres Parsen, wenn Objektinformationen für die Greifpunkte benötigt werden.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/compute_grasps_
↪extended
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/compute_grasps_extended
```

Request

Siehe `compute_grasps` Service.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "item_models": [
      {
        "rectangle": {
          "max_dimensions": {
            "x": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "y": "float64"
          },
          "min_dimensions": {
            "x": "float64",
            "y": "float64"
          }
        },
        "textured_box": {
          "dimensions": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "dimensions_tolerance_m": "float64",
          "max_deformation_m": "float64",
          "template_id": "string"
        },
        "type": "string"
      }
    ],
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "suction_surface_length": "float64",
    "suction_surface_width": "float64"
  }
}
```


Response

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`grasps`: sortierte Liste von Sauggreifpunkten. Jeder Greifpunkt enthält die `item` Information des zugehörigen erkannten Rechtecks.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "compute_grasps_extended",
  "response": {
    "grasps": [
      {
        "item": {
          "pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "pose_frame": "string",
          "rectangle": {
            "x": "float64",
            "y": "float64"
          },
          "template_id": "string",
          "type": "string",
          "uuid": "string",
          "view_name": "string",
          "view_pose_set": "bool",
          "view_uuid": "string"
        },
        "max_suction_surface_length": "float64",
        "max_suction_surface_width": "float64",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "quality": "float64",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "type": "string",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "uuid": "string"
      }
    ],
    "load_carriers": [
      {
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


set_preferred_orientation

speichert die bevorzugte TCP-Orientierung zum Berechnen der Erreichbarkeit der Greifpunkte, die zur Filterung und optional zur Sortierung der vom `compute_grasps` oder `compute_grasps_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe

Setzen der bevorzugten TCP-Orientierung, Abschnitt 6.3.5.4).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/set_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/set_preferred_orientation
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_preferred_orientation",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_preferred_orientation

gibt die bevorzugte TCP-Orientierung zurück, die für die Filterung und optional für die Sortierung der vom `detect_object` und `compute_grasps_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 6.3.5.4).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/get_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/get_preferred_orientation
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_preferred_orientation",
  "response": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

set_sorting_strategies

speichert die gewählte Strategie zur Sortierung der Greifpunkte, die vom `compute_grasps` und `compute_grasps_extended` Service zurückgeliefert werden (siehe *Berechnung der Greifpunkte*, Abschnitt 6.3.5.3).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/set_sorting_
↪strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/set_sorting_strategies
```

Request

Nur eine Sortierstrategie darf einen Gewichtswert `weight` größer als 0 haben. Wenn alle Werte für `weight` auf 0 gesetzt sind, wird die Standardsortierstrategie verwendet.

Wenn der Wert `weight` für `direction` gesetzt ist, muss `vector` den Richtungsvektor enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Wenn der Wert `weight` für `distance_to_point` gesetzt ist, muss `point` den Sortierpunkt enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "direction": {
      "pose_frame": "string",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "gravity": {
      "weight": "float64"
    },
    "surface_area": {
      "weight": "float64"
    }
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_sorting_strategies",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


get_sorting_strategies

gibt die gewählte Sortierstrategie zurück, die zur Sortierung der vom `compute-grasps` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Berechnung der Greifpunkte*, Abschnitt 6.3.5.3).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/get_sorting_
↪strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/get_sorting_strategies
```

Request

Dieser Service hat keine Argumente.

Response

Wenn alle Werte für `weight` 0 sind, wird die Standardsortierstrategie verwendet.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_sorting_strategies",
  "response": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "gravity": {
      "weight": "float64"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "surface_area": {
      "weight": "float64"
    }
  }
}
```


start

startet das Modul und versetzt es in den Zustand `RUNNING`.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/start
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/start
```

Request

Dieser Service hat keine Argumente.

Response

Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `IDLE` unterscheidenden Zustand zurück.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "start",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

stoppt das Modul und versetzt es in den Zustand `IDLE`.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/stop
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/stop
```

Request

Dieser Service hat keine Argumente.

Response

Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `IDLE` unterscheidenden Zustand zurück.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

trigger_dump

speichert die Detektion auf dem angeschlossenen USB Speicher, die dem übergebenen Zeitstempel entspricht, oder die letzte, falls kein Zeitstempel angegeben wurde.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/trigger_dump
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/trigger_dump
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "comment": "string",
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "trigger_dump",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter und der Sortierstrategie dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    }
  }
}
```

6.3.5.9 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 6.37: Rückgabecodes der Services des BoxPick-Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-3	Ein interner Timeout ist aufgetreten, beispielsweise während der Boxerkennung, wenn der Bereich der angegebenen Abmessungen zu groß ist.
-4	Die maximal erlaubte Zeitspanne für die interne Akquise der Bilddaten wurde überschritten.
-8	Das Template wurde während der Detektion gelöscht.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern, ROIs oder Templates überschritten wurde.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit
-12	Ressource ausgelastet, z.B. wenn `trigger_dump` zu häufig aufgerufen wird
-200	Ein schwerwiegender interner Fehler ist aufgetreten.
-301	Für die Anfrage zur Greifpunktberechnung `compute_grasps` oder `compute_grasps_extended` wurden mehrere Objektmodelle (`item_models`) übergeben.
10	Die maximal speicherbare Anzahl an Load Carriern, ROIs oder Templates wurde erreicht.
11	Mit dem Aufruf von `set_load_carrier` oder `set_region_of_interest` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
101	Es wurden keine gültigen Greifflächen in der Szene gefunden.
102	Der detektierte Load Carrier ist leer.
103	Alle berechneten Greifpunkte sind in Kollision.
112	Die Detektionen eines oder mehrerer Cluster wurden verworfen, da die minimale Clusterabdeckung nicht erreicht wurde.
300	Ein gültiges `robot_pose`-Argument wurde angegeben, ist aber nicht erforderlich.
999	Zusätzliche Hinweise für die Anwendungsentwicklung

6.3.5.10 BoxPick Template API“

BoxPick Templates sind nur mit der +Match-Erweiterung von BoxPick verfügbar. Für den Upload, Download, das Auflisten und Löschen von Templates werden spezielle REST-API-Endpunkte zur Verfügung gestellt. Templates können auch über die Web GUI hoch- und runtergeladen werden. Die Templates beinhalten die Greifpunkte und Posenvorgaben, falls Greifpunkte oder Posenvorgaben konfiguriert wurden. Bis zu 100 Templates können gleichzeitig auf dem *rc_reason_stack* gespeichert werden.

`GET /templates/rc_boxpick`
listet alle rc_cadmatch-Templates auf.

Musteranfrage

```
GET /api/v2/templates/rc_boxpick HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "string"
  }
]
```

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Array der Templates)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Template* (Abschnitt 7.2.3)

`GET /templates/rc_boxpick/{id}`
ruft ein rc_boxpick-Template ab. Falls der angefragte Content-Typ application/octet-stream ist, wird das Template als Datei zurückgegeben.

Musteranfrage

```
GET /api/v2/templates/rc_boxpick/<id> HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **`id`** (*`string`*) – ID des Templates *(obligatorisch)*

Antwort-Header

- Content-Type – application/json application/ubjson application/octet-stream

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 404 Not Found – Modul oder Template wurden nicht gefunden.

Referenzierte Datenmodelle

- *Template* (Abschnitt 7.2.3)

`PUT /templates/rc_boxpick/{id}`
erstellt oder aktualisiert ein rc_boxpick-Template.

Musteranfrage

```
PUT /api/v2/templates/rc_boxpick/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **id** (*string*) – ID des Templates *(obligatorisch)*

Formularparameter

- **file** – Template-Datei *(obligatorisch)*

Anfrage-Header

- Accept – multipart/form-data application/json

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 400 Bad Request – Template ist ungültig oder die maximale Zahl an Templates wurde erreicht.
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.
- 413 Request Entity Too Large – Template ist zu groß.

Referenzierte Datenmodelle

- *Template* (Abschnitt 7.2.3)

DELETE /templates/rc_boxpick/{id}

entfernt ein rc_boxpick-Template.

Musteranfrage

```
DELETE /api/v2/templates/rc_boxpick/<id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameter

- **id** (*string*) – ID des Templates *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.

6.3.6 SilhouetteMatch und SilhouetteMatchAI

6.3.6.1 Einführung

Das SilhouetteMatch und SilhouetteMatchAI Modul ist ein optionales Modul, welches intern auf dem *rc_reason_stack* läuft, und benötigt eine eigene *Lizenz* (Abschnitt 8.2), welche erworben werden muss.

Bemerkung: Dieses Modul ist nicht verfügbar in Kamerapipelines vom Typ `blaze`.

Das Modul erkennt Objekte, indem eine vordefinierte Silhouette („Template“) mit Kanten im Bild verglichen wird.

Das SilhouetteMatch und SilhouetteMatchAI Modul kann Objekte in zwei verschiedenen Szenarien erkennen:

Mit kalibrierter Basisebene: Die Objekte befinden sich auf einer gemeinsamen Basisebene, die vor der Objekterkennung kalibriert werden muss, und die Objekte haben prägnante Kanten auf einer gemeinsamen Ebene, welche parallel zu der Basisebene ist.

Mit Objektebenenerkennung: Die Objekte können sich auf verschiedenen, vorab unbekannten Ebenen befinden, falls die Objekte eine planare Oberfläche haben und ihre Konturen gut in den Kamerabildern sichtbar sind (z.B. gestapelte flache Objekte).

Mit Objektebenenerkennung und KI-basierter Segmentierung: SilhouetteMatchAI bietet ein KI-basiertes Objektsegmentierungsmodell, das Objekte in der Szene erkennt und mögliche Ebenen für die Objekterkennung extrahiert. Die Ergebnisse der KI-basierten Segmentierung ermöglichen zudem die Berechnung von Objektüberlappungen und das Filtern von Objekten nach ihrem Überlappungsgrad.

Templates für die Objekterkennung können erstellt werden, indem eine DXF Datei hochgeladen und die Objekthöhe angegeben wird. Die korrekte Skalierung und Einheit der Konturen wird aus der DXF Datei extrahiert. Falls die DXF Datei keine Einheit enthält, muss der Nutzer die korrekte Einheit angeben. Wenn die Außenkontur des Objekts in der DXF Datei geschlossen ist, wird automatisch ein 3D Kollisionsmodell erstellt, indem die Kontur auf die Objekthöhe extrudiert wird. Dieses Modell wird dann zur Kollisionsprüfung und 3D-Visualisierung verwendet. Das Hochladen der DXF Datei kann in der Web GUI über guilabel:*+ Neues Template erstellen* im Abschnitt *SilhouetteMatch Templates und Greifpunkte* auf der *Module* → *SilhouetteMatch* oder *Datenbank* → *Templates* Seite erfolgen.

Roboception bietet hierfür auch einen Template-Generierungsservice auf ihrer Website (https://roboception.com/de/template-request-de/) an, auf der der Benutzer CAD-Daten oder mit dem System aufgenommene Daten hochladen kann, um Templates generieren zu lassen.

Templates bestehen aus den prägnanten Kanten eines Objekts. Die Kanten des Templates werden mit den erkannten Kanten im linken und rechten Kamerabild abgeglichen, wobei die Größe der Objekte und deren Abstand zur Kamera mit einbezogen wird. Die Posen der erkannten Objekte werden zurückgegeben und können beispielsweise benutzt werden, um die Objekte zu greifen.

Bemerkung: Auf Kamerapipelines vom Typ `zivid` oder `orbbec` wird nur das linke Kamerabild zum Matching der Templatekanten verwendet.

Das SilhouetteMatch und SilhouetteMatchAI Modul bietet:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc_reason_stack* *Web GUI* (Abschnitt 7.1)

- eine *REST-API-Schnittstelle* (Abschnitt 7.2) und eine *KUKA Ethernet KRL Schnittstelle* (Abschnitt 7.5)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche des Kamerabilds auszuwählen (siehe *Setzen einer Region of Interest*, Abschnitt 6.3.6.3)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 6.3.2), um in Bin-Picking-Anwendungen („Griff in die Kiste“) Greifpunkte nur für Objekte in dem erkannten Load Carrier zu berechnen
- die Speicherung von bis zu 50 Templates
- die Definition von bis zu 50 Greifpunkten für jedes Template über eine interaktive Visualisierung in der Web GUI
- eine Kollisionsprüfung zwischen Greifer und Load Carrier, der kalibrierten Basisebene, anderen erkannten Objekten, und/oder der Punktwolke
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1) kombiniert werden, um Greifposen in einem benutzerdefinierten externen Koordinatensystem zu liefern
- Auswahl einer Strategie zum Sortieren der erkannten Objekte und zurückgelieferten Greifpunkte
- eine 3D Visualisierung des Detektionsergebnisses mit Greifpunkten und einer Greiferanimation in der Web GUI

Bemerkung: Dieses Softwaremodul ist pipelinespezifisch. Änderungen seiner Einstellungen oder Parameter betreffen nur die zugehörige Kamerapipeline und haben keinen Einfluss auf die anderen Pipelines, die auf dem *rc_reason_stack* laufen.
Achtung: Die Objekt-Templates und ihre Greifpunkte werden global gespeichert. Das Anlegen, Ändern oder Löschen eines Templates oder seiner Greifpunkte betrifft alle Kamerapipelines.

Taugliche Objekte

Das SilhouetteMatch und SilhouetteMatchAI Modul ist für Objekte ausgelegt, die prägnante Kanten auf einer Ebene besitzen, welche parallel zu der Ebene ist, auf der die Objekte liegen. Das trifft beispielsweise auf flache, nicht-transparente Objekte zu, wie gefräste, lasergeschnittene oder wasserstrahlgeschnittene Teile. Komplexere Objekte können auch erkannt werden, solange sie prägnante Kanten auf einer Ebene besitzen, z.B. ein gedrucktes Muster auf einer ebenen Fläche.

Falls die Objekte nicht auf einer gemeinsamen Ebene liegen oder die Basisebene nicht vorab kalibriert werden kann, brauchen die Objekte eine planare Oberfläche und ihre Konturen müssen gut im linken und rechten Kamerabild sichtbar sein. Weiterhin müssen die Templates für diese Objekte eine geschlossene Außenkontur haben.

Taugliche Szene

Eine für das SilhouetteMatch und SilhouetteMatchAI Modul taugliche Szene muss folgende Bedingungen erfüllen:

- Die zu erkennenden Objekte müssen, wie oben beschrieben, tauglich für das SilhouetteMatch und SilhouetteMatchAI Modul sein.
- Nur Objekte, die zum selben Template gehören, dürfen gleichzeitig sichtbar sein (sortenrein). Falls auch andere Objekte sichtbar sind, muss eine passende Region of Interest (ROI) festgelegt werden.
- Im Falle einer kalibrierten Basisebene: Die Verkippung der Basisebene zur Blickrichtung der Kamera darf 10 Grad nicht übersteigen.

- Im Falle von verschiedenen oder unbekannten Basisebenen: Die Verkippung der planaren Oberfläche der Objekte zur Blickrichtung der Kamera darf 25 Grad nicht übersteigen.
- Die Objekte sind weder teilweise noch komplett verdeckt. Mit SilhouetteMatchAI werden leichte Überlappungen toleriert und können zum Filtern von Matches mit Überlappungen verwendet werden.
- Alle sichtbaren Objekte liegen richtig herum.
- Die Objektkanten, welche abgeglichen werden sollen, sind sowohl im linken als auch im rechten Kamerabild zu sehen.

6.3.6.2 Kalibrierung der Basisebene

Falls alle Objekte auf einer gemeinsamen Ebene liegen, die vorab bekannt ist, sollte diese Ebene kalibriert werden, bevor die Objekterkennung gestartet wird. Hierbei wird die Distanz und der Winkel der Ebene, auf welcher die Objekte liegen, gemessen und persistent auf dem *rc_reason_stack* gespeichert.

Durch die Trennung der Kalibrierung der Basisebene von der eigentlichen Objekterkennung werden beispielsweise Szenarien ermöglicht, in denen die Basisebene zeitweise verdeckt ist. Darüber hinaus wird die Berechnungszeit der Objekterkennung für Szenarien verringert, in denen die Basisebene für eine gewisse Zeit fixiert ist – die Basisebene muss in diesem Fall nicht fortlaufend neu detektiert werden.

Die Kalibrierung der Basisebene kann mit drei unterschiedlichen Verfahren durchgeführt werden, auf die im Folgenden näher eingegangen wird:

- AprilTag-basiert
- Stereo-basiert
- Manuell

Die Kalibrierung ist erfolgreich, solange der Normalenvektor der Basisebene höchstens 10 Grad gegen die Blickrichtung der Kamera verkippt ist. Eine erfolgreiche Kalibrierung wird persistent auf dem *rc_reason_stack* gespeichert, bis sie entweder gelöscht wird oder eine neue Kalibrierung durchgeführt wird.

Bemerkung: Um Datenschutzproblemen entgegenzuwirken, wird die Visualisierung der Kalibrierung der Basisebene nach einem Neustart des *rc_reason_stack* verschwommen dargestellt.

In Szenarien, in denen die Basisebene nicht direkt kalibriert werden kann, ist es auch möglich, zu einer zur Basisebene parallel liegenden Ebene zu kalibrieren. In diesem Fall kann der Parameter `offset` benutzt werden, um die geschätzte Ebene auf die eigentliche Basisebene zu verschieben. Der Parameter `offset` gibt die Distanz in Metern an, um welche die geschätzte Ebene in Richtung der Kamera verschoben wird.

In der REST-API ist eine Ebene durch eine Normale (`normal`) und einen Abstand (`distance`) definiert. `normal` ist ein normalisierter 3-Vektor, welcher die Normale der Ebene spezifiziert. Die Normale zeigt immer von der Kamera weg. `distance` repräsentiert den Abstand der Ebene von der Kamera in Richtung der Normale. `normal` und `distance` können auch als a, b, c, bzw. d der Ebenengleichung interpretiert werden:

$$ax + by + cz + d = 0$$

AprilTag-basierte Kalibrierung der Basisebene

Bemerkung: Auf Kamerapipelines vom Typ `zivid` oder `orbbec` ist die AprilTag-basierte Basisebenenkalibrierung nicht verfügbar.

Die AprilTag-Erkennung (siehe *TagDetect*, Abschnitt 6.3.3) wird benutzt, um AprilTags in der Szene zu finden und eine Ebene durch diese zu legen. Mindestens drei AprilTags müssen so auf der Basisebene platziert werden, dass sie im linken und rechten Kamerabild zu sehen sind. Die AprilTags sollten ein möglichst großes Dreieck aufspannen. Je größer das Dreieck ist, desto höher wird die Genauigkeit der Schätzung der Basisebene. Diese Methode sollte benutzt werden, wenn die Basisebene untexturiert und kein externer Projektor mit Zufallsmuster angeschlossen ist. Diese Kalibriermethode ist sowohl über die *REST-API-Schnittstelle* (Abschnitt 7.2) als auch über die *rc_reason_stack* Web GUI verfügbar.

Stereo-basierte Kalibrierung der Basisebene

Die 3D-Punktwolke, welche vom Stereo-Matching-Modul berechnet wird, wird benutzt um eine Ebene in den 3D-Punkten zu finden. Die Region of Interest (ROI) sollte für diese Methode deshalb so gewählt werden, dass nur die relevante Basisebene eingeschlossen wird. Der Parameter `plane_preference` erlaubt es auszuwählen, ob die zur Kamera am nächsten gelegene oder die von der Kamera am weitesten entfernte Ebene als Basisebene benutzt wird. Die am nächsten gelegene Ebene kann in Szenarien ausgewählt werden, in denen die Basisebene vollständig von Objekten verdeckt wird oder für die Kalibrierung nicht erreichbar ist. Diese Methode sollte benutzt werden, wenn die Basisebene texturiert ist oder ein Projektor mit Zufallsmuster angeschlossen ist. Diese Kalibriermethode ist sowohl über die *REST-API-Schnittstelle* (Abschnitt 7.2) als auch über die *rc_reason_stack* Web GUI verfügbar.

Manuelle Kalibrierung der Basisebene

Die Basisebene kann manuell gesetzt werden, falls die Parameter bekannt sind – beispielsweise von einer vorangegangenen Kalibrierung. Diese Kalibriermethode ist nur über die *REST-API-Schnittstelle* (Abschnitt 7.2) und nicht über die *rc_reason_stack* Web GUI verfügbar.

6.3.6.3 Setzen einer Region of Interest

Falls Objekte nur in einem Teil des Sichtfelds der Kamera erkannt werden sollen, kann eine 2D Region of Interest (ROI) gesetzt werden, wie in *Region of Interest* (Abschnitt 6.5.2.2) beschrieben wird.

6.3.6.4 Setzen von Greifpunkten

Um das SilhouetteMatch und SilhouetteMatchAI Modul direkt in einer Roboteranwendung zu nutzen, können für jedes Template bis zu 50 Greifpunkte definiert werden. Ein Greifpunkt repräsentiert die gewünschte Position und Orientierung des Roboter-TCPs (Tool Center Point), mit der das Objekt gegriffen werden kann (siehe Abb. 6.14).

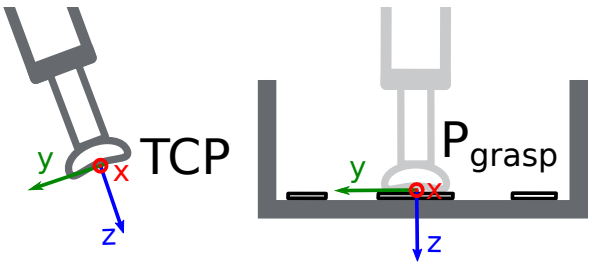


Abb. 6.14: Definition von Greifpunkten bezogen auf den Roboter-TCP

Jeder Greifpunkt enthält eine `id`, die eindeutig über alle Greifpunkte eines Objekt-Templates sein muss, die ID des Templates (`template_id`), zu dem der Greifpunkt hinzugefügt wird, und die Greifpose (`pose`) im Koordinatensystem des Templates. Greifpunkte können über die *REST-API-Schnittstelle* (Abschnitt 7.2), oder über die interaktive Visualisierung in der Web GUI definiert werden. Zudem kann einem Greifpunkt eine Priorität (von -2 für sehr niedrig bis 2 für sehr hoch) zugewiesen werden. Prioritäten können

Roboteranwendungen vereinfachen, oder die Rechenzeit der Kollisionsprüfung verkürzen, wenn der Parameter `only_highest_priority_grasp` aktiviert ist. In diesem Fall endet die Kollisionsprüfung, wenn Greifpunkte mit der höchsten Priorität gefunden sind. Weiterhin können Greifpunkte unterschiedlichen Greifern zugewiesen werden, indem die ID des Greifers (`gripper_id`) spezifiziert wird. Dieser Greifer wird dann anstelle des Greifers, welcher im `detect_object` oder `detect_object_extended` Service definiert ist, für die Kollisionsprüfung des zugehörigen Greifpunkts verwendet.

Wenn für einen Greifpunkt eine `gripper_id` angegeben wird, und der zugehörige Greifer Elemente vom Typ (`function_type`) `FINGER` besitzt, kann jeder Greifpunkt auch Werte für `stroke_per_finger_approach_mm` und `stroke_per_finger_grasp_mm` festlegen. Diese Werte geben die Verschiebung eines Fingers in Millimetern an, um die das Fingerelement und alle seine Kind-Elemente von der `zero_pose` zur `pose` des Fingerelements bewegt werden. Der Wert `stroke_per_finger_approach_mm` gibt die Greiferöffnung während der Annäherung an und wird zur Kollisionsprüfung verwendet. Der Wert `stroke_per_finger_grasp_mm` wird nicht für die Kollisionsprüfung verwendet, sondern enthält Informationen über die Greiferöffnung während des Greifens. Dieses Feld definiert somit implizit die Bewegungsrichtung des Fingers beim Greifen. Wenn weder `stroke_per_finger_approach_mm` noch `stroke_per_finger_grasp_mm` angegeben werden, dann wird der Greifer mit den Fingern in der Standardpose zur Kollisionsprüfung verwendet.

Wird ein Greifpunkt auf einem symmetrischen Objekt definiert, werden alle Greifpunkte, die zu diesem symmetrisch sind, automatisch im `detect_object` und `detect_object_extended` Service des SilhouetteMatch und SilhouetteMatchAI Moduls mit berücksichtigt. Symmetrische Greifpunkte zu einem gegebenen Greifpunkt können mittels des `get_symmetric_grasps` Services abgefragt werden und in der Web GUI visualisiert werden.

Benutzer können ebenfalls Replikationen eines Greifpunktes um eine selbst-definierte Achse definieren. Eine Replikation generiert mehrere Greifpunkte und sorgt dafür, dass Benutzer nicht zu viele Greifpunkte manuell setzen müssen. Der Ursprung der Replikation ist als Koordinatensystem im Objektkoordinatensystem definiert und die x-Achse dieses Koordinatensystems entspricht der Replikationsachse. Der Greifpunkt wird repliziert, indem er ausgehend von seiner ursprünglichen Pose um diese x-Achse gedreht wird. Die Replikation erfolgt in `step_x_deg`-Grad Schritten. Der Bereich wird durch die minimalen und maximalen Endpunkte `min_x_deg` und `max_x_deg` bestimmt. Der minimale (maximale) Endpunkt muss nicht-positiv (nicht-negativ) sein.

Setzen von Greifpunkten in der Web GUI

Die *rc_reason_stack* Web GUI bietet eine interaktive und intuitive Möglichkeit, Greifpunkte für Objekt-Templates zu setzen. Im ersten Schritt muss das Objekt-Template auf den *rc_reason_stack* hochgeladen werden. Das kann über die Web GUI in einer beliebigen Kamerapipeline unter *Module* → *SilhouetteMatch* erfolgen, indem im Abschnitt *Templates und Greifpunkte* auf *+ Neues Template hinzufügen* geklickt wird, oder unter *Datenbank* → *Templates* im Abschnitt *SilhouetteMatch Templates und Greifpunkte*. Wenn der Upload abgeschlossen ist, erscheint ein Fenster mit einer 3D-Visualisierung des Templates, in dem Greifpunkte hinzugefügt oder existierende Greifpunkte bearbeitet werden können. Dasselbe Fenster erscheint, wenn ein vorhandenes Template bearbeitet wird. Wenn das Template ein Kollisionsmodell oder ein Visualisierungsmodell enthält, wird dieses Modell ebenfalls angezeigt.

Dieses Fenster bietet zwei Möglichkeiten, um Greifpunkte zu setzen:

1. **Greifpunkte manuell hinzufügen**: Durch Klicken auf das *+* Symbol wird ein neuer Greifpunkt im Ursprung des Templates angelegt. Diesem Greifpunkt kann ein eindeutiger Name gegeben werden, der seiner ID entspricht. Die gewünschte Pose des Greifpunkts im Koordinatensystem des Templates kann in den Feldern für *Position* und *Roll/Pitch/Yaw* eingegeben werden. Die Greifpunkte können frei platziert werden, auch außerhalb oder innerhalb des Templates, und werden mit ihrer Orientierung zur Überprüfung in der Visualisierung veranschaulicht.
2. **Greifpunkte interaktiv hinzufügen**: Greifpunkte können interaktiv zu einem Template hinzugefügt werden, indem zuerst auf den Button *Greifpunkt hinzufügen* oben rechts in der Visualisierung und anschließend auf den gewünschten Punkt auf dem Template geklickt wird. Wenn ein 3D-Modell angezeigt wird, wird der Greifpunkt an die Oberfläche des Modells angeheftet, andernfalls an die Template-Oberfläche. Die Orientierung des Greifpunkts entspricht einem rechtshändigen

Koordinatensystem, sodass die z-Achse senkrecht auf der Template-Oberfläche steht und in das Template hinein gerichtet ist. Die Position und Orientierung des Greifpunkts im Koordinatensystem des Templates ist auf der rechten Seite angezeigt. Die Position und Orientierung des Greifpunkts kann auch interaktiv verändert werden. Für den Fall, dass *An Oberfläche anheften* in der Visualisierung deaktiviert ist (das ist der Standardwert), kann der Greifpunkt in allen drei Dimensionen frei verschoben und gedreht werden, indem in der Visualisierung auf *Greifpunkt bewegen* geklickt wird und der Greifpunkt dann entlang der Achse zur gewünschten Position verschoben wird. Die Orientierung des Greifpunkts kann ebenfalls interaktiv verändert werden, indem die Achse mit der Maus rotiert wird. Wenn *An Oberfläche anheften* nicht aktiv ist, kann der Greifpunkt nur auf der Objektoberfläche verschoben und rotiert werden.

Benutzer können auch eine Greifpunktpriorität festlegen, indem sie den Schieberegler *Priorität* ändern. Ein dedizierter Greifer kann im Dropdown-Feld *Greifer* ausgewählt werden.

Durch Aktivieren des Kontrollkästchens *Replizieren* können Benutzer den Greifpunkt um eine benutzerdefinierte Achse replizieren. Die Replikationsachse und die generierten Greifpunkte werden visualisiert. Die Lage und Ausrichtung der Replikationsachse relativ zum Objektkoordinatensystem kann interaktiv angepasst werden, indem im Visualisierungsmenü auf *Replikationsachse bewegen* geklickt und die Achse an die gewünschte Position und Ausrichtung gezogen wird. Die Greifpunkte werden innerhalb des angegebenen Drehbereichs mit der ausgewählten Schrittgröße repliziert. Benutzer können eine Visualisierung die replizierten Greifpunkte durchlaufen, indem sie die Leiste unter *Durchlaufen n repl. Greifpunkte* im Abschnitt *Ansichtsoptionen* des Visualisierungsmenüs ziehen. Wenn für den Greifpunkt ein Greifer ausgewählt ist oder im Visualisierungsmenü ein Greifer ausgewählt wurde, wird der Greifer auch am aktuell ausgewählten Greifpunkt angezeigt.

Wenn das Template Symmetrien hat, können die Greifpunkte, die symmetrisch zum definierten Greifpunkt sind, zusammen mit ihren Replikationen (sofern definiert) durch Aktivieren von *. . . Symmetrien* im Abschnitt *Ansichtsoptionen* des Visualisierungsmenüs angezeigt werden. Visualisierungen der symmetrischen Greifpunkte können ebenfalls durchlaufen werden, indem die Leiste unter *Durchlaufe n symm. Greifpunkte* bewegt wird.

Setzen von Greifpunkten über die REST-API

Greifpunkte können über die *REST-API-Schnittstelle* (Abschnitt 7.2) mithilfe des `set_grasp` oder `set_all_grasps` Services gesetzt werden (siehe *Interne Services*, Abschnitt 6.3.6.12). Ein Greifpunkt besteht aus der `id`, die eindeutig über alle Greifpunkte eines Objekt-Templates sein muss, der ID des Templates (`template_id`), zu dem der Greifpunkt hinzugefügt wird, und der Greifpose (`pose`). Die Pose ist im Koordinatensystem des Templates angegeben und besteht aus einer Position (`position`) in Metern und einer Orientierung (`orientation`) als Quaternion. Ein dedizierter Greifer kann durch Setzen des Feldes `gripper_id` angegeben werden. Die `priority` wird durch einen ganzzahligen Wert angegeben, der von -2 für sehr niedrig bis 2 für sehr hoch reicht. Der Replikationsursprung ist als eine Transformation im Koordinatensystem des Objekts definiert und die x-Achse der Transformation entspricht der Replikationsachse. Der Replikationsbereich wird durch die Felder `min_x_deg` und `max_x_deg` und die Schrittweite `step_x_deg` gesteuert.

6.3.6.5 Setzen der bevorzugten TCP-Orientierung

Das SilhouetteMatch und SilhouetteMatchAI Modul berechnet die Erreichbarkeit von Greifpunkten basierend auf der bevorzugten Orientierung des TCPs. Die bevorzugte Orientierung kann über den Service `set_preferred_orientation` oder über die *SilhouetteMatch*-Seite in der Web GUI gesetzt werden. Die resultierende Richtung der z-Achse des TCP wird genutzt, um Greifpunkte zu verwerfen, die der Greifer nicht erreichen kann. Weiterhin kann die bevorzugte Orientierung genutzt werden, um die erreichbaren Greifpunkte zu sortieren, indem die entsprechende Sortierstrategie ausgewählt wird.

Die bevorzugte TCP-Orientierung kann im Kamerakoordinatensystem oder im externen Koordinatensystem gesetzt werden, wenn eine Hand-Auge-Kalibrierung verfügbar ist. Wenn die bevorzugte TCP-Orientierung im externen Koordinatensystem definiert ist, und die Kamera am Roboter montiert ist, muss bei jedem Aufruf der Objekterkennung die aktuelle Roboterpose angegeben werden. Wenn keine

bevorzugte TCP-Orientierung gesetzt wird, wird die Orientierung der linken Kamera (siehe Coordinate frames im rc_visard Handbuch) als die bevorzugte TCP-Orientierung genutzt.

6.3.6.6 Setzen der Sortierstrategie

Die vom `detect_object` und `detect_object_extended` Service zurückgelieferten Objekte und Greifpunkte werden gemäß einer Sortierstrategie sortiert, die vom Nutzer gewählt werden kann. Folgende Sortierstrategien sind verfügbar und können über die *Web GUI* (Abschnitt 7.1) oder über den `set_sorting_strategies` Service gesetzt werden:

- `preferred_orientation`: Matches und Greifpunkte mit der geringsten Rotationsänderung einer gewählten Achse (`axis`), oder aller Achsen, wenn `axis` leer ist, bezogen auf die bevorzugte TCP-Orientierung werden zuerst zurückgeliefert.
- `direction`: Objekte und Greifpunkte mit dem kleinsten Abstand entlang der gesetzten Richtung `vector` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.
- `distance_to_point`: Objekte und Greifpunkte mit dem kleinsten oder größten (falls `farthest_first` auf true gesetzt ist) Abstand von einem gesetzten Sortierpunkt `point` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.

Wenn keine Sortierstrategie gesetzt ist, oder die Standard-Sortierstrategie in der Web GUI ausgewählt ist, geschieht die Sortierung der Greifpunkte basierend auf einer Kombination von `preferred_orientation` und dem kleinsten Abstand entlang der z-Achse der bevorzugten TCP-Orientierung von der Kamera.

6.3.6.7 Objekterkennung

Um eine Objekterkennung durchzuführen, müssen im Allgemeinen die folgenden Serviceargumente an das SilhouetteMatch und SilhouetteMatchAI Modul übergeben werden:

- das Template des Objekts, welches in der Szene erkannt werden soll
- das Koordinatensystem, in dem die Posen der detektierten Objekte zurückgegeben werden sollen (siehe *Hand-Auge-Kalibrierung*, Abschnitt 6.3.6.8)

Optional können auch folgende Serviceargumente an das SilhouetteMatch und SilhouetteMatchAI Modul übergeben werden:

- Ein Flag `object_plane_detection`, welches bestimmt, ob die Oberflächenebene der Objekte für die Erkennung verwendet werden soll anstelle einer kalibrierten Basisebene.
- ein Versatz `offset`, falls Objekte nicht direkt auf der Basisebene liegen, sondern auf einer zu dieser parallelen Ebene. Der Versatz bezeichnet die Distanz beider Ebenen in Richtung der Kamera. Wenn dieser Wert nicht gesetzt wird, wird ein Versatz von 0 angenommen. Der Versatz darf nicht gesetzt werden, wenn `object_plane_detection` true ist.
- die ID des Load Carriers, der die zu detektierenden Objekte enthält
- die ID der Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die Region of Interest, innerhalb der Objekte erkannt werden sollen. Wenn keine ROI gesetzt wird, werden Objekte im gesamten Kamerabild gesucht.
- die aktuelle Roboterpose, wenn die Kamera am Roboter montiert ist und als Koordinatensystem `external` gewählt wurde, oder die bevorzugte TCP-Orientierung im externen Koordinatensystem angegeben ist
- Informationen für die Kollisionsprüfung: Die ID des Greifers, um die Kollisionsprüfung zu aktivieren, und optional ein Greif-Offset, der die Vorgreifposition definiert. Details zur Kollisionsprüfung sind in *CollisionCheck* (Abschnitt 6.3.6.8) gegeben.

Wenn `object_plane_detection` nicht true ist, können Objekte erst nach einer erfolgreichen Kalibrierung der Basisebene erkannt werden. Es muss sichergestellt werden, dass sich Position und Orientierung

der Basisebene zwischen Kalibrierung und Objekterkennung nicht ändern. Anderenfalls muss die Kalibrierung erneuert werden.

Wenn `object_plane_detection` auf true gesetzt ist, ist eine Kalibrierung der Basisebene nicht nötig und eine ggf. existierende Kalibrierung wird ignoriert. Während der Erkennung wird die Szene in planare Flächen unterteilt und das Matching der Templatekanten wird für jede dieser Ebenen durchgeführt, solange sie nicht mehr als 25° in Bezug auf die Sichtachse der Kamera verkippt ist, und solange ihre Größe ausreichend ist für das gewählte Template. Wenn ein Match gefunden wird, wird dessen Position und Orientierung durch Kanten im Kamerabild und durch die Punktwolke innerhalb der Außenkontur des Templates verfeinert. Aus diesem Grund muss die Außenkontur des Templates geschlossen und die Oberfläche des Objekts planar sein.

Wenn SilhouetteMatchAI verfügbar ist und `object_plane_detection` auf true gesetzt ist, kann ein Objektsegmentierungsmodell `object_segmentation_model` angegeben werden, das für die KI-basierte Segmentierung von Objekten anstelle des Unterteilens der Szene in planare Flächen verwendet wird. Die resultierenden Objektmasken werden verwendet, um Oberflächenebenen für das Template-Matching zu extrahieren und ermöglichen zudem die Berechnung von Objektüberlappungen, die zum Filtern verwendet werden, falls `max_object_overlap` auf einen Wert kleiner als 1 gesetzt ist. Das aktuell unterstützte Objektsegmentierungsmodell ist `SHEET_METAL` (Blech).

Im *Ausprobieren*-Abschnitt der Seite *SilhouetteMatch* der Web GUI kann die Objektdetektion ausprobiert werden. Verschiedene Bild-Streams können ausgewählt werden, um Zwischenergebnisse und die finalen Matches anzuzeigen.

Das **„Template“** Bild zeigt das zu erkennende Template in Grün mit den Greifpunkten (siehe *Setzen von Greifpunkten*, Abschnitt 6.3.6.4) in Grün. Das Template wird verformt dargestellt, passend zu Abstand und Verkippung der kalibrierten Basisebene, oder - falls `object_plane_detection` auf true gesetzt war, der höchsten erkannten Ebene. Die entsprechende Ebene ist in Dunkelblau dargestellt.

Das **„Zwischenergebnis“** Bild zeigt die Kanten im linken Bild, die für die Suche nach Matches verwendet wurden, in Hellblau. Die gewählte Region of Interest wird als petrolfarbenes Rechteck dargestellt. Eine blau schattierte Fläche auf der linken Seite markiert den Teil des linken Kamerabilds, welcher nicht mit dem rechten Kamerabild überlappt. In diesem Bereich können keine Objekte erkannt werden. Wenn die Objektebenenerkennung verwendet wurde (`object_plane_detection` ist true), zeigt dieses Bild auch die erkannten planaren Cluster in der Szene. Cluster, die nicht für das Matching verwendet wurden, weil sie zu klein oder zu stark geneigt sind, werden mit einem Streifenmuster dargestellt.

Das **„Zwischenergebnis rechts“** Bild zeigt die Kanten im rechten Bild, die für die Suche nach Matches verwendet wurden, in Hellblau. Die gewählte Region of Interest wird als petrolfarbenes Rechteck dargestellt. Eine blau schattierte Fläche auf der rechten Seite markiert den Teil des rechten Kamerabilds, welcher nicht mit dem linken Kamerabild überlappt. In diesem Bereich können keine Objekte erkannt werden.

Bemerkung: Auf Kamerapipelines vom Typ `zivid` oder `orbbec` ist das **„Zwischenergebnis rechts“** nicht verfügbar.

Das **„Ergebnis“** Bild zeigt das Detektionsergebnis. Die Kanten, die zur Verfeinerung der Match Posen genutzt wurden, werden in hellem Blau und erkannte Objekte (`instances`) in Grün visualisiert. Blaue Punkte markieren jeweils den Ursprung der detektierten Objekte, wie im Template festgelegt. Kollisionsfreie Greifpunkte sind als grüne Punkte dargestellt, ungeprüfte Greifpunkte als gelbe Punkte, und kollidierende Greifpunkte werden als rote Punkte visualisiert.

Die Posen der Objektursprünge werden im gewählten Koordinatensystem als Liste (`instances`) zurückgegeben. Falls die kalibrierte Basisebene für die Erkennung genutzt wurde (`object_plane_detection` nicht oder false gesetzt), wird die Orientierung der erkannten Objekte mit mit der Normalen der Basisebene ausgerichtet. Andernfalls ist die Orientierung der Objekte an der Normalen der Ebene ausgerichtet, die in die zugehörigen Objektpunkte in der 3D Punktwolke eingepasst wurde.

Wenn das ausgewählte Template auch Greifpunkte hat, dann wird zusätzlich zu den erkannten Objekten auch eine Liste von Greifpunkten (`grasps`) für alle erkannten Objekte zurückgegeben. Die Posen der

Greifpunkte sind im gewünschten Koordinatensystem angegeben und die Liste ist gemäß der gewählten Sortierstrategie sortiert (siehe *Setzen der Sortierstrategie*, Abschnitt 6.3.6.6). Die erkannten Objekte und die Greifpunkte können über ihre UUIDs einander zugeordnet werden.

Falls das Template eine kontinuierliche Rotationssymmetrie aufweist (z.B. zylindrische Objekte), besitzen alle Ergebnisposen die gleiche Orientierung. Weiterhin werden alle Symmetrien eines Greifpunkts auf Erreichbarkeit und Kollisionsfreiheit geprüft, und anschließend nur der jeweilige beste gemäß der gewählten Sortierstrategie zurückgeliefert.

Für Objekte mit einer diskreten Symmetrie (z.B. prismatische Objekte), werden alle kollisionsfreien Symmetrien jedes Greifpunkts, die entsprechend der gesetzten bevorzugten TCP-Orientierung erreichbar sind, zurückgeliefert, und gemäß der gewählten Sortierstrategie sortiert.

Die Detektionsergebnisse und Berechnungszeiten werden durch Laufzeitparameter beeinflusst, welche weiter unten aufgezählt und beschrieben werden. Unsachgemäße Parameterwerte können zu Zeitüberschreitungen im Detektionsprozess des SilhouetteMatch und SilhouetteMatchAI Moduls führen.

6.3.6.8 Wechselwirkung mit anderen Modulen

Die folgenden auf dem *rc_reason_stack* laufenden Module liefern Daten für das SilhouetteMatch und SilhouetteMatchAI Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieser Module kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des SilhouetteMatch und SilhouetteMatchAI Moduls haben.

Kamera- und Tiefendaten

Das SilhouetteMatch und SilhouetteMatchAI Modul verarbeitet intern die rektifizierten Bilder des *Kamera Modul* (`rc_camera`, Abschnitt 6.1). Es sollte deshalb auf eine passende Belichtungszeit geachtet werden, um optimale Ergebnisse zu erhalten.

Für die Kalibrierung der Basisebene mit der Stereo-Methode, für die Load Carrier Erkennung, für die automatische Objektebenenerkennung und für die Kollisionsprüfung mit der Punktwolke wird das Disparitätsbild des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 6.2.2) verarbeitet.

Für das Erkennen von Objekten mit einer kalibrierten Basisebene, ohne Load Carrier und ohne Kollisionsprüfung mit der Punktwolke sollte das Stereo-Matching-Modul nicht parallel zum SilhouetteMatch und SilhouetteMatchAI Modul ausgeführt werden, da die Laufzeit der Objekterkennung sonst negativ beeinflusst wird.

Für beste Ergebnisse wird empfohlen, *Glättung* (Abschnitt 6.2.2.1) für das *Stereo-Matching Modul* zu aktivieren.

IOControl und Projektor-Kontrolle

Wenn der *rc_reason_stack* in Verbindung mit einem externen Musterprojektor und dem Modul *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 6.4.4) betrieben wird, sollte der Projektor für die stereobasierte Kalibrierung der Basisebene, für die automatische Objektebenenerkennung und für die Kollisionsprüfung mit der Punktwolke benutzt werden.

Das projizierte Muster darf während der Objektdetektion nicht im linken oder rechten Kamerabild sichtbar sein, da es den Detektionsvorgang behindert. Daher wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 6.2.2.1), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 6.4.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren.

Hand-Auge-Kalibrierung

Wenn die Kamera zu einem Roboter kalibriert ist, kann das SilhouetteMatch und SilhouetteMatchAI Modul die Ergebnisposen automatisch im Roboterkoordinatensystem liefern. Für die *Services* (Abschnitt 6.3.6.11) des SilhouetteMatch und SilhouetteMatchAI Moduls kann das Referenzkoordinatensystem aller Posen über das Argument `pose_frame` angegeben werden.

Es kann zwischen den folgenden zwei Werten für `pose_frame` gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen und Ebenenparameter werden im Kamera-Koordinatensystem angegeben.
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen und Ebenenparameter sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das SilhouetteMatch und SilhouetteMatchAI Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom internen Modul *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

Bemerkung: Wurde keine Hand-Auge-Kalibrierung durchgeführt, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Bemerkung: Wird die Hand-Auge-Kalibrierung nach einer Kalibrierung der Basisebene verändert, wird die Kalibrierung der Basisebene als ungültig markiert und muss erneuert werden.

Für den Fall einer robotergeführten Kamera ist es abhängig von `pose_frame`, der bevorzugten TCP-Orientierung und der Sortierrichtung bzw. des Sortierpunktes nötig, zusätzlich die aktuelle Roboterpose (`robot_pose`) zur Verfügung zu stellen:

- Wenn `external` als `pose_frame` ausgewählt ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die bevorzugte TCP-Orientierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die Sortierrichtung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn der Sortierpunkt für die Abstandssortierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- In allen anderen Fällen ist die Angabe der Roboterpose optional.

Wenn die aktuelle Roboterpose während der Kalibrierung der Basisebene angegeben wird, wird sie persistent auf dem *rc_reason_stack* gespeichert. Falls für die Services `get_base_plane_calibration`, `detect_objects` oder `detect_object_extended` die dann aktuelle Roboterpose ebenfalls angegeben wird, wird die Basisebene automatisch zu der neuen Roboterpose transformiert. Das erlaubt dem Benutzer, die Roboterpose (und damit die Pose der Kamera) zwischen Kalibrierung der Basisebene und Objekterkennung zu verändern.

Bemerkung: Eine Objekterkennung kann nur durchgeführt werden, wenn die Verkippung der Basisebene zur Sichtachse der Kamera ein 10-Grad-Limit nicht übersteigt.

LoadCarrier

Das SilhouetteMatch und SilhouetteMatchAI Modul nutzt die Funktionalität zur Load Carrier Erkennung aus dem *LoadCarrier* Modul (`rc_load_carrier`, Abschnitt 6.3.2) mit den Laufzeitparametern, die für dieses Modul festgelegt wurden. Wenn sich jedoch mehrere Load Carrier in der Szene befinden, die zu der angegebenen Load Carrier ID passen, wird nur einer davon zurückgeliefert. In diesem Fall sollte eine Region of Interest gesetzt werden, um sicherzustellen, dass immer derselbe Load Carrier für das SilhouetteMatch und SilhouetteMatchAI Modul verwendet wird.

CollisionCheck

Die Kollisionsprüfung kann für die Greifpunktberechnung des SilhouetteMatch und SilhouetteMatchAI Moduls aktiviert werden, indem das `collision_detection` Argument an den `detect_object` Service übergeben wird. Es enthält die ID des benutzten Greifers und optional einen Greif-Offset. Der Greifer muss im GripperDB Modul definiert werden (siehe *Erstellen eines Greifers*, Abschnitt 6.5.3.2) und Details über die Kollisionsprüfung werden in *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 6.4.2.2) gegeben.

Alternativ können Greifpunkten individuell Greifer IDs zugewiesen werden, und die Kollisionsprüfung kann für alle Greifpunkte mit einer Greifer ID über den Laufzeitparameter `check_collisions` eingeschaltet werden.

Zusätzlich wird auf Kollisionen zwischen dem Greifer und der kalibrierten Basisebene geprüft, wenn der Laufzeitparameter `check_collisions_with_base_plane` auf true gesetzt ist. Wenn das ausgewählte Template ein Kollisionsmodell enthält und der Laufzeitparameter `check_collisions_with_matches` true ist, wird außerdem auch auf Kollisionen zwischen dem Greifer und den anderen detektierten Objekten (nicht begrenzt auf die Anzahl `max_number_of_detected_objects`) geprüft, wobei das Objekt, auf dem sich der jeweilige Greifpunkt befindet, von der Prüfung ausgenommen ist.

Wenn der Laufzeitparameter `check_collisions_with_point_cloud` auf true gesetzt ist, werden auch Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Wenn diese Funktionalität in Kombination mit Sauggreifern genutzt wird, muss sichergestellt werden, dass sich der TCP außerhalb der Greifergeometrie befindet, oder dass die Greifpunkte oberhalb der Objektoberfläche definiert sind. Andernfalls wird für jeden Greifpunkt eine Kollision zwischen Greifer und Punktwolke erkannt.

Wenn der Laufzeitparameter `check_collisions_during_retraction` auf true gesetzt ist, und ein Load Carrier sowie ein Greif-Offset angegeben wurden, wird jeder Greifpunkt auf Kollisionen zwischen dem Objekt im Greifer und den Wänden des Load Carriers während der Entnahme geprüft. Die Prüfung findet auf der gesamten linearen Trajektorie von der Greifposition bis zurück zur Vorgreifposition statt.

Wenn die Kollisionsprüfung aktiviert ist, werden nur Greifpunkte zurückgeliefert, die kollisionsfrei sind, oder die nicht auf Kollisionen geprüft werden konnten (z.B. weil kein Greifer angegeben wurde). In der Ergebnis-Visualisierung oben auf der *SilhouetteMatch*-Seite der Web GUI werden kollisionsfreie Greifpunkte grün dargestellt, ungeprüfte Greifpunkte gelb und kollidierende Greifpunkte rot. Die erkannten Objekte, die bei der Kollisionsprüfung betrachtet werden, werden mit grünen Kanten visualisiert.

Die Laufzeitparameter des CollisionCheck-Moduls beeinflussen die Kollisionserkennung wie in *CollisionCheck-Parameter* (Abschnitt 6.4.2.3) beschrieben.

6.3.6.9 Parameter

Das SilhouetteMatch und SilhouetteMatchAI Modul wird in der REST-API als `rc_silhouettematch` bezeichnet und in der *Web GUI* (Abschnitt 7.1) in der gewünschten Pipeline unter *Module* → *SilhouetteMatch* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.38: Laufzeitparameter des `rc_silhouettematch`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`check_collisions`	bool	false	true	false	Gibt an, ob Kollisionen geprüft werden sollen, wenn ein Greifer für einen Greifpunkt definiert wurde
`check_collisions_during_retraction`	bool	false	true	false	Gibt an, ob Kollisionen zwischen dem Objekt im Greifer und dem Load Carrier während der Entnahme geprüft werden
`check_collisions_with_base_plane`	bool	false	true	true	Gibt an, ob Kollisionen zwischen Greifer und der Basisebene geprüft werden
`check_collisions_with_matches`	bool	false	true	true	Gibt an, ob Kollisionen zwischen Greifer und anderen Matches geprüft werden
`check_collisions_with_point_cloud`	bool	false	true	false	Gibt an, ob Kollisionen zwischen Greifer und der Punktwolke geprüft werden
`edge_sensitivity`	float64	0.1	1.0	0.7	Empfindlichkeit der Kantenerkennung
`match_max_distance`	float64	0.1	10.0	3.0	Der maximale tolerierte Abstand zwischen dem Template und den detektierten Kanten im Bild in Pixeln
`match_percentile`	float64	0.7	1.0	0.8	Der Anteil der Template-Pixel, die innerhalb der maximalen Matchingdistanz liegen müssen, um ein Objekt erfolgreich zu detektieren
`max_number_of_detected_objects`	int32	1	20	10	Maximale Anzahl der zu detektierenden Objekte
`max_object_overlap`	float64	0.0	1.0	0.05	Maximaler Anteil der Objektoberfläche, der von anderen segmentierten Objekten überlappt werden darf
`only_highest_priority_grasps`	bool	false	true	false	Falls aktiviert werden nur Greifpunkte der höchsten Priorität zurückgegeben.
`point_cloud_enhancement`	string	-	-	Off	Art der Verbesserung der Punktwolke mit der Basisebene: [Off, ReplaceBright]
`quality`	string	-	-	High	Quality: [Low, Medium, High]

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf der SilhouetteMatch und SilhouetteMatchAI Seite in der Web GUI dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet:

`max_number_of_detected_objects` (*Maximale Objektanzahl*)

Dieser Parameter gibt an, wie viele Objekte maximal in der Szene erkannt werden sollen. Falls mehr als die angegebene Zahl an Objekten gefunden wurden,

werden nur die am besten zur gewählten Sortierstrategie passenden Ergebnisse zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?max_
↪number_of_detected_objects=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?max_number_of_detected_
↪objects=<value>
```

`quality` (*Qualität*)

Die Objekterkennung kann auf Bildern mit unterschiedlicher Auflösung durchgeführt werden: `High` (*Hoch*, volle Auflösung), `Medium` (*Mittel*, halbe Auflösung) oder `Low` (*Niedrig*, Viertel-Auflösung). Je niedriger die Auflösung ist, desto niedriger ist die Berechnungszeit der Objekterkennung, aber desto weniger Objektdetails sind erkennbar.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?
↪quality=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?quality=<value>
```

`match_max_distance` (*Maximale Matchingdistanz*)

Dieser Parameter gibt den maximal tolerierten Abstand zwischen dem Template und den detektierten Kanten im Bild in Pixeln an. Falls das Objekt durch das Template nicht exakt genug beschrieben wird, wird es möglicherweise nicht erkannt, wenn dieser Wert zu klein ist. Höhere Werte können jedoch im Fall von komplexen Szenen und bei ähnlichen Objekten zu Fehldetektionen führen, und auch die Berechnungszeit erhöhen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?match_
↪max_distance=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?match_max_distance=<value>
```

`match_percentile` (*Matching Perzentil*)

Dieser Parameter kontrolliert, wie strikt der Detektionsprozess sein soll. Das Matching Perzentil gibt den Anteil der Template-Pixel an, die innerhalb der maximalen

Matchingdistanz liegen müssen, um ein Objekt erfolgreich zu detektieren. Je höher der Wert, desto exakter muss ein Match sein, um als gültig angesehen zu werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?match_
↪percentile=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?match_percentile=<value>
```

`edge_sensitivity` (*Kantenempfindlichkeit*)

Der Parameter beeinflusst, wie viele Kanten im linken und rechten Kamerabild gefunden werden. Umso größer dieser Parameter gewählt wird, umso mehr Kanten werden für die Erkennung benutzt. Eine große Anzahl von Kanten im Bild kann die Erkennung verlangsamen. Es muss sichergestellt werden, dass die Kanten der zu erkennenden Objekte sowohl im linken als auch im rechten Kamerabild detektiert werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?edge_
↪sensitivity=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?edge_sensitivity=<value>
```

`max_object_overlap` (*Maximale Objektüberlappung*)

Dieser Parameter ist nur für SilhouetteMatchAI verfügbar und bestimmt den maximalen Anteil eines Objekts, der von anderen Objekten überlappt sein darf. Objekte mit größeren Überlappungswerten werden verworfen. Ein Wert von 1 schaltet den Überlappungscheck aus. Überlappungen werden nur geprüft, wenn ein Modell zur Objektsegmentierung ausgewählt wurde.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?max_
↪object_overlap=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?max_object_overlap=<value>
```

`only_highest_priority_grasps` (*Nur Greifpunkte höchster Priorität*)

Wenn dieser Parameter auf *true* gesetzt ist, werden ausschließlich Greifpunkte der höchsten Priorität zurückgegeben. Sofern die Kollisionsprüfung aktiviert ist, werden ausschließlich

kollisionsfreie Greifpunkt der höchstmöglichen Priorität zurückgegeben. Dadurch kann Rechenzeit gespart und die Anzahl der applikationsseitig zu verarbeitenden Greifpunkte reduziert werden.

Ohne Kollisionsprüfung werden nur Greifpunkt der höchsten Priorität zurückgegeben.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?only_
↪highest_priority_grasps=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?only_highest_priority_
↪grasps=<value>
```

`check_collisions` (*Kollisionsprüfung*)

Wenn diese Option aktiv ist, wird die Kollisionsprüfung für alle Greifpunkte durchgeführt, denen eine Greifer ID zugewiesen wurde, auch wenn kein Standardgreifer im `detect_object` Service gesetzt wurde. Wenn ein Load Carrier verwendet wird, wird die Kollisionsprüfung immer zwischen dem Greifer und dem Load Carrier durchgeführt. Kollisionen mit der Punktwolke oder anderen Matches werden nur geprüft, wenn die zugehörigen Laufzeitparameter aktiv sind.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?check_
↪collisions=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions=<value>
```

`check_collisions_with_base_plane` (*Kollisionsprüfung mit Basisebene*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service oder durch Setzen des Parameters `check_collisions` aktiviert ist. Wenn `check_collisions_with_base_plane` auf true gesetzt ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und der kalibrierten Basisebene geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit der Basisebene wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?check_
↪collisions_with_base_plane=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_with_
↪base_plane=<value>
```

`check_collisions_with_matches` (*Kollisionsprüfung mit Matches*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service oder durch Setzen des Parameters `check_collisions` aktiviert ist. Wenn `check_collisions_with_matches` auf true gesetzt ist und die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service aktiviert ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und den anderen detektierten Objekten (nicht begrenzt auf die Anzahl `max_number_of_detected_objects`) geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit anderen detektierten Objekten wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?check_
↪collisions_with_matches=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_with_
↪matches=<value>
```

`check_collisions_with_point_cloud` (*Kollisionsprüfung mit Punktwolke*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service oder durch Setzen des Parameters `check_collisions` aktiviert ist. Wenn `check_collisions_with_point_cloud` auf true gesetzt ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit dieser Punktwolke wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?check_
↪collisions_with_point_cloud=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_with_
↪point_cloud=<value>
```

`point_cloud_enhancement` (*Verbesserung mit Basisebene*)

Dieser Parameter wird nur beachtet, wenn `check_collisions_with_point_cloud` auf true gesetzt ist und die Detektion ohne Objektebenenerkennung (`object_plane_detection` ist false) ausgelöst wurde. Standardmäßig ist `point_cloud_enhancement` auf `Off` (*Aus*) gesetzt. Wenn `point_cloud_enhancement` auf `ReplaceBright` (*Helle Bildpunkte ersetzen*) gesetzt wird, wird die kalibrierte Basisebene verwendet, um die Punktwolke für die Kollisionsprüfung zu verbessern. Dazu werden Punkte, die zu hellen Pixeln im Bild oder in der gewählten 2D Region of Interest gehören, auf den Tiefenwert der kalibrierten Basisebene gesetzt. Dieser Parameter sollte genutzt werden, wenn dunkle Objekten auf texturlosem, hellem Untergrund liegen, z.B. auf einem Lichttisch.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?point_
↪cloud_enhancement=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?point_cloud_enhancement=
↪<value>
```

`check_collisions_during_retraction` (*Kollisionsprüfung während Entnahme*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` Service oder durch Setzen des Parameters `check_collisions` aktiviert ist. Wenn `check_collisions_during_retraction` auf true gesetzt ist und ein Load Carrier sowie ein Greif-Offset angegeben wurden, wird jeder Greifpunkt auf Kollisionen zwischen dem Objekt im Greifer und den Wänden des Load Carriers während der Entnahme geprüft. Die Prüfung findet auf der gesamten linearen Trajektorie von der Greifposition bis zurück zur Vorgreifposition statt. Es werden nur kollisionsfreie Greifpunkte zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/parameters?check_
↪collisions_during_retraction=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_during_
↪retraction=<value>
```

6.3.6.10 Statuswerte

Dieses Modul meldet folgende Statuswerte.

Tab. 6.39: Statuswerte des `rc_silhouettematch`-Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`load_carrier_detection_time`	Berechnungszeit für die letzte Load Carrier Detektion in Sekunden
‘‘processing_time‘	Berechnungszeit für die letzte Detektion (einschließlich Load Carrier Detektion) in Sekunden

6.3.6.11 Services

Die angebotenen Services des `rc_silhouettematch`-Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der *rc_reason_stack Web GUI* (Abschnitt 7.1) ausprobiert und getestet werden.

Das SilhouetteMatch und SilhouetteMatchAI Modul bietet folgende Services.

detect_object

führt eine Objekterkennung durch, wie in *Objekterkennung* (Abschnitt 6.3.6.7) beschrieben. Der Service gibt die Posen aller gefundenen Objektinstanzen zurück.

Details

Das Zeitverhalten dieses Services garantiert, dass nur Bilddaten zur Erkennung benutzt werden, welche nach dem Anfragezeitpunkt generiert wurden.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/detect_
↪object
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/detect_object
```

Request

Obligatorische Serviceargumente:

`object_id` in `object_to_detect`: ID des Templates, welches erkannt werden soll.

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.6.8).

Potentiell obligatorische Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.6.8).

Optionale Serviceargumente:

`object_plane_detection`: false wenn Objekte auf einer kalibrierten Basisebene liegen, true wenn die Objekte planare Oberflächen haben und die Basisebene unbekannt ist oder die Objekte auf mehreren verschiedenen Ebenen liegen, z.B. auf Stapeln.

`offset`: Versatz in Metern, um welche die Basisebene in Richtung der Kamera verschoben werden soll.

`load_carrier_id`: ID des Load Carriers, welcher die zu erkennenden Objekte enthält.

`collision_detection`: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 6.4.2.2)

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "load_carrier_id": "string",
    "object_plane_detection": "bool",
    "object_segmentation_model": "string",
    "object_to_detect": {
      "object_id": "string",
      "region_of_interest_2d_id": "string"
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    },
    "offset": "float64",
    "pose_frame": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

Response

Die maximale Anzahl der zurückgegebenen Instanzen kann über den `max_number_of_detected_objects`-Parameter kontrolliert werden.

`object_id`: ID des erkannten Templates.

`instances`: Liste der erkannten Objektinstanzen, sortiert gemäß der gewählten Sortierstrategie.

`grasps`: Liste von Greifpunkten auf den erkannten Objekten. Die Greifpunkte sind gemäß der gewählten Sortierstrategie sortiert. Die `instance_uuid` gibt eine Referenz auf das detektierte Objekt in `instances` an, zu dem dieser Greifpunkt gehört. Die Liste der Greifpunkte wird auf die 100 besten Greifpunkte gekürzt, falls mehr erreichbare Greifpunkte gefunden werden. Jeder Greifpunkt enthält ein Flag `collision_checked` und das Feld `gripper_id` (siehe *Integrierte Kollisionsprüfung in anderen Modulen* Abschnitt 6.4.2.2).

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`timestamp`: Zeitstempel des Bildes, das für die Erkennung benutzt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "detect_object",
  "response": {
    "grasps": [
      {
        "collision_checked": "bool",
        "gripper_id": "string",
        "id": "string",
        "instance_uuid": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "priority": "int8",
        "stroke_per_finger_approach_mm": "float64",
        "stroke_per_finger_grasp_mm": "float64",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "uuid": "string"
      }
    ],
    "instances": [
      {
        "grasp_uuids": [
          "string"
        ],
        "id": "string",
        "object_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "uuid": "string"
      }
    ],
    "load_carriers": [
      {
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "object_id": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


detect_object_extended

führt eine Objekterkennung durch. Dieser Service verhält sich analog zu `detect_object`, gibt aber die Instanzinformationen für jeden Greifpunkt direkt zurück, anstatt sie in einer separaten Liste zu speichern. Dies ermöglicht ein einfacheres Parsen, wenn z.B. die Objektposen für jeden Greifpunkt benötigt werden, um das Objekt platziert abzulegen.

Details

Das Zeitverhalten dieses Services garantiert, dass nur Bilddaten zur Erkennung benutzt werden, welche nach dem Anfragezeitpunkt generiert wurden.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/detect_
↪object_extended
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/detect_object_extended
```

Request

Siehe `detect_object` Service.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "load_carrier_id": "string",
    "object_plane_detection": "bool",
    "object_segmentation_model": "string",
    "object_to_detect": {
      "object_id": "string",
      "region_of_interest_2d_id": "string"
    },
    "offset": "float64",
    "pose_frame": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```


Response

Die maximale Anzahl der zurückgegebenen Instanzen kann über den `max_number_of_detected_objects`-Parameter kontrolliert werden.

`object_id`: ID des erkannten Templates.

`grasps`: Liste von Greifpunkten auf den erkannten Objekten. Die Greifpunkte sind gemäß der gewählten Sortierstrategie sortiert. Jeder Greifpunkt enthält ein Feld `instance` mit Informationen zum detektierten Objekt, z.B. seiner Pose. Die Liste der Greifpunkte wird auf die 100 besten Greifpunkte gekürzt, falls mehr erreichbare Greifpunkte gefunden werden. Jeder Greifpunkt enthält ein Flag `collision_checked` und das Feld `gripper_id` (siehe *Integrierte Kollisionsprüfung in anderen Modulen* Abschnitt 6.4.2.2).

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`timestamp`: Zeitstempel des Bildes, das für die Erkennung benutzt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "detect_object_extended",
  "response": {
    "grasps": [
      {
        "collision_checked": "bool",
        "gripper_id": "string",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      "id": "string",
      "instance": {
        "object_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "uuid": "string"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "priority": "int8",
      "stroke_per_finger_approach_mm": "float64",
      "stroke_per_finger_grasp_mm": "float64",
      "timestamp": {
        "nsec": "int32",
        "sec": "int32"
      },
      "uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "overfilled": "bool",
      "pose": {
        "orientation": {
          "w": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "object_id": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


calibrate_base_plane

führt die Kalibrierung der Basisebene durch, wie in *Kalibrierung der Basisebene* (Abschnitt 6.3.6.2) beschrieben.

Details

Eine erfolgreiche Kalibrierung der Basisebene wird persistent auf dem *rc_reason_stack* gespeichert und vom Service zurückgegeben. Die Kalibrierung ist dauerhaft – auch über Firmware-Updates und -Wiederherstellungen hinweg – gespeichert.

Das Zeitverhalten dieses Services garantiert, dass nur Bilddaten zur Erkennung benutzt werden, welche nach dem Anfragezeitpunkt generiert wurden.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/
↪calibrate_base_plane
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/calibrate_base_plane
```

Request

Obligatorische Serviceargumente:

`plane_estimation_method`: Methode der Kalibrierung der Basisebene. Gültige Werte sind `STEREO`, `APRILTAG`, `MANUAL`.

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.6.8).

Potentiell obligatorische Serviceargumente:

`plane` wenn für `plane_estimation_method` `MANUAL` gewählt ist: Die Ebene, welche als Basisebene gesetzt wird.

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.6.8).

`region_of_interest_2d_id`: ID der Region of Interest für die Kalibrierung der Basisebene.

Optionale Serviceargumente:

`offset`: Versatz in Metern, um welchen die geschätzte Ebene in Richtung der Kamera verschoben wird.

`plane_preference` in `stereo`: Ob die der Kamera am nächsten (`CLOSEST`) gelegene oder die am weitesten entfernte (`FARTHEST`) Ebene als Basisebene benutzt wird. Diese Option kann nur gesetzt werden, falls `plane_estimation_method` auf `STEREO` gesetzt ist. Valide Werte sind `CLOSEST` und `FARTHEST`. Falls der Wert nicht gesetzt ist, wird `FARTHEST` verwendet.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "offset": "float64",
    "plane": {
      "distance": "float64",
      "normal": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "plane_estimation_method": "string",
    "pose_frame": "string",
    "region_of_interest_2d_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "stereo": {
      "plane_preference": "string"
    }
  }
}
```

Response

`plane`: kalibrierte Basisebene.

`timestamp`: Zeitstempel des Bildes, das für die Kalibrierung benutzt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "calibrate_base_plane",
  "response": {
    "plane": {
      "distance": "float64",
      "normal": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


get_base_plane_calibration

gibt die derzeitige Kalibrierung der Basisebene zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/get_
↪base_plane_calibration
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_base_plane_calibration
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.6.8).

Potentiell obligatorische Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.6.8).

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "pose_frame": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_base_plane_calibration",
  "response": {
    "plane": {
      "distance": "float64",
      "normal": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

delete_base_plane_calibration

löscht die derzeitige Kalibrierung der Basisebene.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/delete_
↪base_plane_calibration
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/delete_base_plane_
↪calibration
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_base_plane_calibration",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


set_preferred_orientation

speichert die bevorzugte TCP-Orientierung zum Berechnen der Erreichbarkeit der Greifpunkte, die zur Filterung und optional zur Sortierung der vom `detect_object` und `detect_object_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 6.3.6.5).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/set_
↪preferred_orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_preferred_orientation
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_preferred_orientation",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_preferred_orientation

gibt die bevorzugte TCP-Orientierung zurück, die für die Filterung und optional zur Sortierung der vom `detect_object` und `detect_object_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 6.3.6.5).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/get_
↪preferred_orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_preferred_orientation
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_preferred_orientation",
  "response": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

set_sorting_strategies

speichert die gewählte Strategie zur Sortierung der erkannten Objekte und Greifpunkte, die vom `detect_object` und `detect_object_extended` Service zurückgeliefert werden (siehe *Objekterkennung*, Abschnitt 6.3.6.7).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/set_
↪sorting_strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_sorting_strategies
```

Request

Nur eine Sortierstrategie darf einen Gewichtswert `weight` größer als 0 haben. Wenn alle Werte für `weight` auf 0 gesetzt sind, wird die Standardsortierstrategie verwendet.

Wenn der Wert `weight` für `direction` gesetzt ist, muss `vector` den Richtungsvektor enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Wenn der Wert `weight` für `distance_to_point` gesetzt ist, muss `point` den Sortierpunkt enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Wenn der Wert `weight` für `preferred_orientation` gesetzt ist, kann `axis` auf `x`, `y` oder `z` gesetzt werden, um nur Rotationsunterschiede zwischen diesen Achsen zu berücksichtigen. Wenn `axis` nicht gesetzt wird, wird die volle Rotationsdifferenz zur Sortierung verwendet.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "preferred_orientation": {
      "axis": "string",
      "weight": "float64"
    }
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_sorting_strategies",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


get_sorting_strategies

gibt die gewählte Sortierstrategie zurück, die zur Sortierung der vom `detect_object` und `detect_object_extended` Service zurückgelieferten Objekte und Greifpunkte verwendet

wird (siehe *Objekterkennung*, Abschnitt 6.3.6.7).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/get_
↪sorting_strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_sorting_strategies
```

Request

Dieser Service hat keine Argumente.

Response

Wenn alle Werte für `weight` 0 sind, wird die Standardsortierstrategie verwendet.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_sorting_strategies",
  "response": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "preferred_orientation": {
      "axis": "string",
      "weight": "float64"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


trigger_dump

speichert die Detektion auf dem angeschlossenen USB Speicher, die dem übergebenen Zeitstempel entspricht, oder die letzte, falls kein Zeitstempel angegeben wurde.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/trigger_
↪dump
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/trigger_dump
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "comment": "string",
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "trigger_dump",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter und die bevorzugte TCP-Orientierung sowie die Sortierstrategie dieses Moduls wieder her und wendet sie an („factory reset“). Dies betrifft nicht die konfigurierten Templates und die Kalibrierung der Basisebene.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/reset_
↪defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


6.3.6.12 Interne Services

Die folgenden Services für die Konfiguration von Greifpunkten können sich in Zukunft ohne weitere Ankündigung ändern. Es wird empfohlen, das Setzen, Abrufen und Löschen von Greifpunkten über die Web GUI vorzunehmen.

Bemerkung: Das Konfigurieren von Greifpunkten ist global für alle Templates auf dem *rc_reason_stack* und hat Einfluss auf alle Kamerapipelines.

`set_grasp`

speichert einen Greifpunkt für das angegebene Template auf dem *rc_reason_stack*. Alle Greifpunkte sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/set_
↪grasp
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_grasp
```

Request

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 6.3.6.4) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "grasp": {
      "gripper_id": "string",
      "id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "y": "float64",
          "z": "float64"
        }
      },
      "priority": "int8",
      "replication": {
        "max_x_deg": "float64",
        "min_x_deg": "float64",
        "origin": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "step_x_deg": "float64"
      },
      "stroke_per_finger_approach_mm": "float64",
      "stroke_per_finger_grasp_mm": "float64",
      "template_id": "string"
    }
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_grasp",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


set_all_grasps

Ersetzt die gesamte Liste von Greifpunkten auf dem *rc_reason_stack* für das angegebene Template.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/set_all_
↪grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_all_grasps
```

Request

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 6.3.6.4) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "grasps": [
      {
        "gripper_id": "string",
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "priority": "int8",
        "replication": {
          "max_x_deg": "float64",
          "min_x_deg": "float64",
          "origin": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "step_x_deg": "float64"
        },
        "stroke_per_finger_approach_mm": "float64",
        "stroke_per_finger_grasp_mm": "float64",
        "template_id": "string"
      }
    ],
    "template_id": "string"
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_all_grasps",
  "response": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_grasps

gibt alle definierten Greifpunkte mit den angegebenen IDs (`grasp_ids`) zurück, die zu den Templates mit den angegebenen `template_ids` gehören.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/get_
↪grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_grasps
```

Request

Wenn keine `grasp_ids` angegeben werden, werden alle Greifpunkte zu den angegebenen `template_ids` zurückgeliefert. Wenn keine `template_ids` angegeben werden, werden alle Greifpunkte mit den geforderten `grasp_ids` zurückgeliefert. Wenn gar keine IDs angegeben werden, werden alle gespeicherten Greifpunkte zurückgeliefert.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_grasps",
  "response": {
    "grasps": [
      {
        "gripper_id": "string",
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "priority": "int8",
        "replication": {
          "max_x_deg": "float64",
          "min_x_deg": "float64",
          "origin": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "step_x_deg": "float64"
        },
        "stroke_per_finger_approach_mm": "float64",
        "stroke_per_finger_grasp_mm": "float64",
        "template_id": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


delete_grasps

löscht alle Greifpunkte mit den angegebenen `grasp_ids`, die zu den Templates mit den angegebenen `template_ids` gehören.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/delete_
↪grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/delete_grasps
```

Request

Wenn keine `grasp_ids` angegeben werden, werden alle Greifpunkte gelöscht, die zu den Templates mit den angegebenen `template_ids` gehören. Die Liste `template_ids` darf nicht

leer sein.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_grasps",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_symmetric_grasps

gibt alle Greifpunkte zurück, die symmetrisch zum angegebenen Greifpunkt sind.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/get_
↪symmetric_grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_symmetric_grasps
```

Request

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 6.3.6.4) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "grasp": {
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "replication": {
        "max_x_deg": "float64",
        "min_x_deg": "float64",
        "origin": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "step_x_deg": "float64"
      },
      "template_id": "string"
    }
  }
}
```

Response

Der erste Greifpunkt in der Rückgabeliste ist derselbe, der dem Service übergeben wurde. Wenn das Template keine exakte Symmetrie hat, wird nur der übergebene Greifpunkt zurückgeliefert. Wenn das Template eine kontinuierliche Symmetrie hat (z.B. ein zylindrisches Objekt), werden nur 12 gleichverteilte Greifpunkte zurückgeliefert.

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 6.3.6.4) beschrieben.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_symmetric_grasps",
  "response": {
    "grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "replication": {
          "max_x_deg": "float64",
          "min_x_deg": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "origin": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "step_x_deg": "float64"
        },
        "template_id": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


6.3.6.13 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind.

Tab. 6.40: Rückgabecodes und Warnungen der Services des SilhouetteMatch und SilhouetteMatchAI Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-3	Ein interner Timeout ist aufgetreten, beispielsweise während der Objekterkennung.
-4	Die maximal erlaubte Zeitspanne für die interne Akquise der Bilddaten wurde überschritten.
-7	Daten konnten nicht in den persistenten Speicher geschrieben oder vom persistenten Speicher gelesen werden.
-8	Das Modul befindet sich in einem Zustand, in welchem dieser Service nicht aufgerufen werden kann. Beispielsweise kann `detect_object` nicht aufgerufen werden, solange keine Kalibrierung der Basisebene durchgeführt wurde.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an ROIs oder Templates überschritten wurde.
-100	Ein interner Fehler ist aufgetreten.
-101	Die Erkennung der Basisebene schlug fehl.
-102	Die Hand-Auge-Kalibrierung hat sich seit der letzten Kalibrierung der Basisebene verändert.
-104	Die Verkippung zwischen der Basisebene und der Sichtachse der Kamera überschreitet das 10-Grad-Limit.
10	Die maximale Anzahl an ROIs oder Templates wurde erreicht.
11	Ein bestehendes Element wurde überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
101	Keiner der Greifpunkte ist erreichbar.
102	Der detektierte Load Carrier ist leer.
103	Alle berechneten Greifpunkte sind in Kollision.
107	Die Basisebene wurde nicht zur aktuellen Kamerapose transformiert, z.B. weil keine Roboterpose während der Kalibrierung der Basisebene angegeben wurde.
108	Das Template ist überholt.
109	Die Ebene für die Objekterkennung passt nicht zum Load Carrier, z.B. liegen die Objekte unterhalb des Load Carrier Bodens.
111	Die Pose des Detektionsergebnisses konnte nicht mit der Punktwolke verfeinert werden, da die Außenkontur des Templates nicht geschlossen ist.
113	Kein Greifer für die Kollisionsprüfung gefunden.
114	Kollisionsprüfung während Entnahme wurde nicht durchgeführt, z.B. weil kein Load Carrier oder kein Greif-Offset angegeben wurden.
151	Das Objekt-Template hat eine kontinuierliche Symmetrie.
999	Zusätzliche Hinweise für die Anwendungsentwicklung

6.3.6.14 Template API

Für den Upload, Download, das Auflisten und Löschen von Templates werden spezielle REST-API-Endpunkte zur Verfügung gestellt. Templates können auch über die Web GUI hoch- und runtergeladen werden. Die Templates beinhalten die Greifpunkte, falls Greifpunkte konfiguriert wurden. Bis zu 50 Templates können gleichzeitig auf dem *rc_reason_stack* gespeichert werden.

`GET /templates/rc_silhouettematch`
listet alle rc_silhouettematch-Templates auf.

Musteranfrage

```
GET /api/v2/templates/rc_silhouettematch HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "string"
  }
]
```

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Array der Templates)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Template* (Abschnitt 7.2.3)

GET /templates/rc_silhouettematch/{id}
ruft ein rc_silhouettematch-Template ab. Falls der angefragte Content-Typ application/octet-stream ist, wird das Template als Datei zurückgegeben.

Musteranfrage

```
GET /api/v2/templates/rc_silhouettematch/<id> HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **id** (*string*) – ID des Templates *(obligatorisch)*

Antwort-Header

- Content-Type – application/json application/ubjson application/octet-stream

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 404 Not Found – Modul oder Template wurden nicht gefunden.

Referenzierte Datenmodelle

- *Template* (Abschnitt 7.2.3)

PUT /templates/rc_silhouettematch/{id}
erstellt oder aktualisiert ein rc_silhouettematch-Template.

Musteranfrage

```
PUT /api/v2/templates/rc_silhouettematch/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **id** (*string*) – ID des Templates *(obligatorisch)*

Formularparameter

- **file** – Template-Datei oder DXF-Datei *(obligatorisch)*
- **object_height** – Objekthöhe in Metern, benötigt bei DXF-Upload *(optional)*
- **units** – Einheit für DXF Datei falls nicht in Datei enthalten (mögliche Werte: `mm`, `cm`, `m`, `in`, `ft`) *(optional)*

Anfrage-Header

- Accept – multipart/form-data application/json

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 400 Bad Request – Template ist ungültig oder die maximale Zahl an Templates wurde erreicht.
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das SilhouetteMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.
- 413 Request Entity Too Large – Template ist zu groß.

Referenzierte Datenmodelle

- *Template* (Abschnitt 7.2.3)

DELETE /templates/rc_silhouettematch/{id}

entfernt ein rc_silhouettematch-Template.

Musteranfrage

```
DELETE /api/v2/templates/rc_silhouettematch/<id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameter

- **id** (*string*) – ID des Templates *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das SilhouetteMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.

6.3.7 CADMatch

6.3.7.1 Einleitung

Das CADMatch Modul ist ein optionales Modul des *rc_reason_stack* und benötigt eine eigene *Lizenz* (Abschnitt 8.2), welche erworben werden muss.

Dieses Modul bietet eine gebrauchsfertige Lösung für die 3D-Objekterkennung anhand von CAD-Templates und liefert Greifpunkte für allgemeine Greifer. Die Objekte können sich in einer Kiste (Bin, Load Carrier) oder frei platziert im Erfassungsbereich der Kamera befinden.

Für jedes Objekt, das mit dem CADMatch-Modul erkannt werden soll, wird ein Template benötigt. Um Templates zu erhalten, setzen Sie sich bitte mit dem Roboception Support (*Kontakt*, Abschnitt 10) in Verbindung.

Bemerkung: Dieses Softwaremodul ist pipelinespezifisch. Änderungen seiner Einstellungen oder Parameter betreffen nur die zugehörige Kamerapipeline und haben keinen Einfluss auf die anderen Pipelines, die auf dem *rc_reason_stack* laufen.
Achtung: Die Objekt-Templates und ihre Greifpunkte und Posenvorgaben werden global gespeichert. Das Anlegen, Ändern oder Löschen eines Templates, seiner Greifpunkte oder Posenvorgaben betrifft alle Kamerapipelines.

Das CADMatch-Modul bietet darüber hinaus:

- eine intuitiv gestaltete Bedienoberfläche für Inbetriebnahme, Konfiguration und Test auf der *rc_reason_stack Web GUI* (Abschnitt 7.1)
- eine *REST-API-Schnittstelle* (Abschnitt 7.2) und eine *KUKA Ethernet KRL Schnittstelle* (Abschnitt 7.5)
- die Möglichkeit, sogenannte Regions of Interest (ROIs) zu definieren, um relevante Teilbereiche der Szene auszuwählen (siehe *RoiDB*, Abschnitt 6.5.2)
- eine integrierte Load Carrier Erkennung (siehe *LoadCarrier*, Abschnitt 6.3.2), um in Bin-Picking-Anwendungen („Griff in die Kiste“) Greifpunkte nur für Objekte in dem erkannten Load Carrier zu berechnen
- die Unterstützung von Load Carriern mit Abteilen, sodass Greifpunkte für Objekte nur in einem definierten Teilvolumen des Load Carriers berechnet werden
- die Option benutzerdefinierte Posenvorgaben zu nutzen.
- die Speicherung von bis zu 50 Templates
- die Definition von bis zu 100 Greifpunkten für jedes Template über eine interaktive Visualisierung in der Web GUI
- eine Kollisionsprüfung zwischen Greifer und Load Carrier, anderen erkannten Objekten, und/oder der Punktwolke
- eine Kollisionsprüfung zwischen dem Objekt im Greifer und den Wänden des Load Carriers während der Entnahme
- die Unterstützung von sowohl statisch montierten als auch robotergeführten Kameras. Optional kann es mit der *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1) kombiniert werden, um Greifposen in einem benutzerdefinierten externen Koordinatensystem zu liefern.
- Auswahl einer Strategie zum Sortieren der erkannten Objekte und zurückgelieferten Greifpunkte

- eine 3D Visualisierung des Detektionsergebnisses mit Greifpunkten und einer Greiferanimation in der Web GUI

6.3.7.2 Setzen von Greifpunkten

Das CADMatch-Modul erkennt 3D-Objekte in einer Szene basierend auf einem CAD-Template und liefert die Posen der Objektursprünge zurück. Um das CADMatch-Modul direkt in einer Roboteranwendung zu nutzen, können für jedes Template bis zu 100 Greifpunkte definiert werden. Ein Greifpunkt repräsentiert die gewünschte Position und Orientierung des Roboter-TCPs (Tool Center Point), mit der das Objekt gegriffen werden kann.

Weitere Details sind unter *Setzen von Greifpunkten* (Abschnitt 6.3.6.4) beschrieben.

Setzen von Greifpunkten in der Web GUI

Die *rc_reason_stack* Web GUI bietet eine interaktive und intuitive Möglichkeit, Greifpunkte für Objekt-Templates zu setzen. Im ersten Schritt muss das Objekt-Template auf den *rc_reason_stack* hochgeladen werden. Das kann über die Web GUI in einer beliebigen Kamerapipeline unter *Module* → *CAD-Match* erfolgen, indem im Abschnitt *Templates, Greifpunkte und Posenvorgaben* auf *+ Neues Template hinzufügen* geklickt wird, oder unter *Datenbank* → *Templates* im Abschnitt *CADMatch Templates, Greifpunkte und Posenvorgaben*. Wenn der Upload abgeschlossen ist, erscheint ein Fenster mit einer 3D-Visualisierung des Objekts, in dem Greifpunkte hinzugefügt oder existierende Greifpunkte bearbeitet werden können. Dasselbe Fenster erscheint, wenn ein vorhandenes Template bearbeitet wird.

Weitere Details werden in *Setzen von Greifpunkten in der Web GUI* (Abschnitt 6.3.6.4) beschrieben.

Setzen von Greifpunkten über die REST-API

Greifpunkte können über die *REST-API-Schnittstelle* (Abschnitt 7.2) mithilfe des `set_grasp` oder `set_all_grasps` Services gesetzt werden (siehe *Interne Services*, Abschnitt 6.3.7.11).

Weitere Details werden in *Setzen von Greifpunkten über die REST-API* (Abschnitt 6.3.6.4) beschrieben.

6.3.7.3 Setzen von Posenvorgaben

Das CADMatch Modul bietet die Möglichkeit, Posenvorgaben für die zu erkennenden Objekte zu definieren. Wenn eine Posenvorgabe gesetzt ist, nutzt die Objekterkennung diese Pose und führt nur eine Verfeinerung durch. Dadurch wird die Erkennung deutlich beschleunigt.:ngonly:‘Die Posenvorgaben werden verfeinert um die tatsächlichen Posen der Objekte zu bestimmen. Eine Posenvorgabe stellt die ungefähre Position und Orientierung des zu erkennenden Objekts dar. Die Pose kann im Kamera- oder im externen Koordinatensystem definiert werden, wenn eine Hand-Auge-Kalibrierung verfügbar ist.

Jede Posenvorgabe enthält eine `id`, die eindeutig über alle Posenvorgaben eines Objekt-Templates sein muss, die ID des Templates (`template_id`), zu dem die Posenvorgabe hinzugefügt wird, die Posenvorgabe (`pose`) und das Koordinatensystem (`pose_frame`) der Pose. Posenvorgaben können über die *REST-API-Schnittstelle* (Abschnitt 7.2), oder über die interaktive Visualisierung in der Web GUI definiert werden. Die Web GUI ermöglicht die interaktive Positionierung des Objekts in der aktuellen Punktwolke. Dies kann im Reiter „Posenvorgaben“ während des Editieren eines Templates geschehen.

Posenvorgaben können in Anwendungen genutzt werden, in denen die ungefähren Posen der Objekte im Voraus bekannt sind. Der *rc_reason_stack* kann bis zu 50 Posenvorgaben pro Template speichern.

6.3.7.4 Setzen der bevorzugten TCP-Orientierung

Das CADMatch-Modul berechnet die Erreichbarkeit von Greifpunkten basierend auf der bevorzugten Orientierung des TCPs. Die bevorzugte Orientierung kann über den Service

`set_preferred_orientation` oder über die *CADMatch*-Seite in der Web GUI gesetzt werden. Die bevorzugten Orientierung des TCPs wird genutzt, um Greifpunkte zu verwerfen, die der Greifer nicht erreichen kann, und kann auch zur Sortierung der Greifpunkte genutzt werden.

Die bevorzugte TCP-Orientierung kann im Kamerakoordinatensystem oder im externen Koordinatensystem gesetzt werden, wenn eine Hand-Auge-Kalibrierung verfügbar ist. Wenn die bevorzugte TCP-Orientierung im externen Koordinatensystem definiert ist, und der Sensor am Roboter montiert ist, muss bei jedem Aufruf der Objekterkennung die aktuelle Roboterpose angegeben werden. Wenn keine bevorzugte TCP-Orientierung gesetzt wird, wird die Orientierung der linken Kamera (siehe Coordinate frames im rc_visard Handbuch) als die bevorzugte TCP-Orientierung genutzt.

6.3.7.5 Setzen der Sortierstrategie

Die vom `detect_object` und `detect_object_extended` Service zurückgelieferten Matches und Greifpunkte werden gemäß einer Sortierstrategie sortiert, die vom Nutzer gewählt werden kann. Folgende Sortierstrategien sind verfügbar und können über die *Web GUI* (Abschnitt 7.1) oder über den `set_sorting_strategies` Service gesetzt werden:

- `gravity`: die entlang der Gravitationsrichtung am höchsten gelegenen Matches und Greifpunkte werden zuerst zurückgeliefert.
- `match_score`: Matches mit dem höchsten Match Score und Greifpunkte auf Objekten mit dem höchsten Match Score werden zuerst zurückgeliefert.
- `preferred_orientation`: Matches und Greifpunkte mit der geringsten Rotationsänderung einer gewählten Achse (`axis`), oder aller Achsen, wenn `axis` leer ist, bezogen auf die bevorzugte TCP-Orientierung werden zuerst zurückgeliefert.
- `direction`: Matches und Greifpunkte mit dem kleinsten Abstand entlang der gesetzten Sortierrichtung `vector` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.
- `distance_to_point`: Matches und Greifpunkte mit dem kleinsten oder größten (falls `farthest_first` auf true gesetzt ist) Abstand von einem gesetzten Sortierpunkt `point` im angegebenen Referenzkoordinatensystem `pose_frame` werden zuerst zurückgeliefert.

Wenn keine Sortierstrategie gesetzt ist, oder die Standard-Sortierstrategie in der Web GUI ausgewählt ist, geschieht die Sortierung der Greifpunkte basierend auf einer Kombination von `match_score` und dem kleinsten Abstand entlang der z-Achse der bevorzugten TCP-Orientierung von der Kamera.

6.3.7.6 Objekterkennung

Das CADMatch-Modul benötigt ein Objekt-Template für die Objekterkennung. Dieses Template enthält Informationen über die dreidimensionale Form des Objekts und markante Kanten, die im Kamerabild sichtbar sein können. CADMatch unterstützt auch partielle Objekt-Templates, die nur einen bestimmten Teil des Objekts beinhalten, der gut erkannt werden kann, zum Beispiel im Fall von Verdeckungen. Weiterhin gibt es Templates, die eine Posenvorgabe zur Erkennung benötigen, die dann nur mit Hilfe die Bilddaten verfeinert wird.

Die Objekterkennung ist ein zweistufiger Prozess bestehend aus einem initialen Schätzungsschritt und einem Posenverfeinerungsschritt. Als erstes wird die initiale Pose des Objekts anhand der Erscheinung des Objekts im Kamerabild berechnet. Als zweiter Schritt wird die geschätzte Pose anhand der 3D-Punktwolke und der Kanten im Kamerabild verfeinert. Damit das funktionieren kann, müssen die zu detektierenden Objekte im linken und rechten Kamerabild sichtbar sein. Wenn Posenvorgaben gesetzt wurden, findet nur der zweite Verfeinerungsschritt statt, was die Laufzeit deutlich verringert.

Um eine Objekterkennung durchzuführen, können die folgenden Serviceargumente an das CADMatch-Modul übergeben werden:

- die ID des Objekt-Templates, welches in der Szene erkannt werden soll

- das Koordinatensystem, in dem die Posen der detektierten Objekte zurückgegeben werden sollen (siehe *Hand-Auge-Kalibrierung*, Abschnitt 6.3.7.7)

Optional können auch folgende Serviceargumente an das CADMatch-Modul übergeben werden:

- die IDs der Posenvorgaben, die ungefähr den Posen der zu erkennenden Objekten entsprechen. Falls ein Template verwendet wird, das eine Posenvorgabe benötigt, müssen eine oder mehrere Posenvorgaben angegeben werden.
- die ID des Load Carriers, der die zu detektierenden Objekte enthält
- ein Unterabteil (`load_carrier_compartment`) innerhalb eines Load Carriers, in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteile*, Abschnitt 6.5.1.3).
- die ID der Region of Interest, innerhalb der nach dem Load Carrier gesucht wird, oder – falls kein Load Carrier angegeben ist – die Region of Interest, innerhalb der Objekte erkannt werden sollen
- die aktuelle Roboterpose, wenn die Kamera am Roboter montiert ist und als Koordinatensystem `external` gewählt wurde, oder die bevorzugte TCP-Orientierung im externen Koordinatensystem angegeben ist, oder die gewählte Region of Interest im externen Koordinatensystem definiert ist
- Informationen für die Kollisionsprüfung: Die ID des Greifers, um die Kollisionsprüfung zu aktivieren, und optional ein Greif-Offset, der die Vorgreifposition definiert. Details zur Kollisionsprüfung sind in *CollisionCheck* (Abschnitt 6.3.7.7) gegeben.
- Datenaufnahme-Modus: Der Nutzer kann auswählen, ob ein neuer Bilddatensatz für die Erkennung aufgenommen werden soll (Standardwert), oder ob die Detektion mit den zuletzt verwendeten Bilddaten durchgeführt soll.Dies spart die Bildaufnahmezeit, z.B. für den Fall, dass verschiedene Templates im selben Bild erkannt werden sollen.

In der Web GUI kann die Objekterkennung in Bereich *Ausprobieren* auf der *CADMatch*-Seite getestet werden.

Die erkannten Objekte werden in einer Liste von `matches` zurückgeliefert, die entsprechend der gewählten Sortierstrategie sortiert ist. Jedes erkannte Objekt enthält eine `uuid` (Universally Unique Identifier) und den Zeitstempel (`timestamp`) des ältesten Bildes, das zur Erkennung benutzt wurde. Die Pose (`pose`) eines erkannten Objekts entspricht der Pose des Ursprungs des Koordinatensystems des Objekt-Templates, das zur Detektion verwendet wurde. Weiterhin wird ein Matching-Score zurückgegeben, der die Qualität der Erkennung angibt.

Wenn das ausgewählte Template auch Greifpunkte hat (siehe *Setzen von Greifpunkten*, Abschnitt 6.3.7.2), dann wird zusätzlich zu den erkannten Objekten auch eine Liste von Greifpunkten (`grasps`) für alle erkannten Objekte zurückgegeben. Diese Liste ist gemäß der gewählten Sortierstrategie sortiert (siehe *Setzen der Sortierstrategie*, Abschnitt 6.3.7.5). Die Posen der Greifpunkte sind im gewünschten Koordinatensystem angegeben. Die erkannten Objekte und die Greifpunkte können über ihre UUIDs einander zugeordnet werden.

Für Objekte mit einer diskreten Symmetrie (z.B. prismatische Objekte), werden alle kollisionsfreien Symmetrien jedes Greifpunkts, die entsprechend der gesetzten bevorzugten TCP-Orientierung erreichbar sind, zurückgeliefert, und gemäß der gewählten Sortierstrategie sortiert.

Bei Objekten mit einer kontinuierlichen Symmetrie (z.B. zylindrische Objekte), werden alle Symmetrien eines Greifpunkts auf Erreichbarkeit und Kollisionsfreiheit geprüft, und anschließend nur der jeweilige beste gemäß der gewählten Sortierstrategie zurückgeliefert.

Die zurückgegebenen Matches werden im Ergebnisbild auf der CADMatch-Seite der Web GUI mit grünen Kanten dargestellt. Matches, die von anderen Objekten oder Szenenteilen überlappt werden (wenn `max_object_overlap` kleiner als 1 ist), werden im Ergebnisbild mit roten Rändern dargestellt, und der überlappende Bereich ist durch rote Streifen markiert. Zusätzlich werden Matches, die aufgrund eines niedrigen Scores, aufgrund von Überlappungen oder der maximalen Anzahl von Matches herausgefiltert wurden, im Bild *Verworfene Matches* dargestellt.

Bemerkung: Der erste Aufruf der Objekterkennung mit einem neuen Objekt-Template dauert etwas länger als die nachfolgenden Aufrufe, weil das Template erst in das CADMatch-Modul geladen werden muss. Um das zu vermeiden, kann der `warmup_template` Service genutzt werden, der das Template lädt damit es bereit ist, wenn die erste Detektion getriggert wird.

6.3.7.7 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_reason_stack* laufenden Module liefern Daten für das CADMatch-Modul oder haben Einfluss auf die Datenverarbeitung.

Bemerkung: Jede Konfigurationsänderung dieser Module kann direkte Auswirkungen auf die Qualität oder das Leistungsverhalten des CADMatch-Moduls haben.

Kamera- und Tiefendaten

Folgende Daten werden vom CADMatch-Modul verarbeitet:

- die rektifizierten Bilder des *Kamera Modul* (`rc_camera`, Abschnitt 6.1)
- die Disparitäts-, Konfidenz- und Fehlerbilder des *Stereo-Matching Modul* (`rc_stereomatching`, Abschnitt 6.2.2), falls eine Stereokamera verwendet wird. Der Parameter *Qualität* (`quality`) des Stereo-Matching-Moduls muss auf `Medium` oder höher gesetzt werden (siehe *Parameter*, Abschnitt 6.2.2.1). Die Einstellung `Full` oder `High` wird für CADMatch empfohlen.
- die Disparitäts-, Konfidenz- und Fehlerbilder der *Orbbec Modul* (`rc_orbbec`, Abschnitt 6.2.4), falls eine *Orbbec* Kamera verwendet wird
- die Disparitäts-, Konfidenz- und Fehlerbilder der *Zivid Modul* (`rc_zivid`, Abschnitt 6.2.3), falls eine *zivid* Kamera verwendet wird

Für alle genutzten Bilder ist garantiert, dass diese nach dem Auslösen des Services aufgenommen wurden.

IOControl und Projektor-Kontrolle

Für den Anwendungsfall, dass der *rc_reason_stack* zusammen mit einem externen Musterprojektor und dem Modul für *IOControl und Projektor-Kontrolle* (`rc_iocontrol`, Abschnitt 6.4.4) betrieben wird, wird empfohlen, den Projektor an GPIO Out 1 anzuschließen und den Aufnahmemodus des Stereokamera-Moduls auf `SingleFrameOut1` zu setzen (siehe *Stereomatching-Parameter*, Abschnitt 6.2.2.1), damit bei jedem Aufnahme-Trigger ein Bild mit und ohne Projektormuster aufgenommen wird.

Alternativ kann der verwendete digitale Ausgang in den Betriebsmodus `ExposureAlternateActive` geschaltet werden (siehe *Beschreibung der Laufzeitparameter*, Abschnitt 6.4.4.1).

In beiden Fällen sollte die Belichtungszeitregelung (`exp_auto_mode`) auf `AdaptiveOut1` gesetzt werden, um die Belichtung beider Bilder zu optimieren.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann das CADMatch-Modul automatisch Posen im Roboterkoordinatensystem ausgeben. Für die *Services* (Abschnitt 6.3.7.10) kann das Koordinatensystem der berechneten Posen mit dem Argument `pose_frame` spezifiziert werden.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Alle Posen sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Alle Posen sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das CADMatch-Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1). Für den Fall einer robotergeführten Kamera ist vom Nutzer zusätzlich die jeweils aktuelle Roboterpose `robot_pose` anzugeben.

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

Für den Fall einer robotergeführten Kamera ist es abhängig von `pose_frame`, der bevorzugten TCP-Orientierung und der Sortierrichtung bzw. des Sortierpunktes nötig, zusätzlich die aktuelle Roboterpose (`robot_pose`) zur Verfügung zu stellen:

- Wenn `external` als `pose_frame` ausgewählt ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die bevorzugte TCP-Orientierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn die Sortierrichtung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- Wenn der Sortierpunkt für die Abstandssortierung in `external` definiert ist, ist die Angabe der Roboterpose obligatorisch.
- In allen anderen Fällen ist die Angabe der Roboterpose optional.

LoadCarrier

Das CADMatch Modul nutzt die Funktionalität zur Load Carrier Erkennung aus dem *LoadCarrier* Modul (`rc_load_carrier`, Abschnitt 6.3.2) mit den Laufzeitparametern, die für dieses Modul festgelegt wurden. Wenn sich jedoch mehrere Load Carrier in der Szene befinden, die zu der angegebenen Load Carrier ID passen, wird nur einer davon zurückgeliefert. In diesem Fall sollte eine 3D Region of Interest gesetzt werden, um sicherzustellen, dass immer derselbe Load Carrier für das CADMatch Modul verwendet wird.

CollisionCheck

Die Kollisionsprüfung kann für die Greifpunktberechnung des CADMatch-Moduls aktiviert werden, indem das `collision_detection` Argument an den `detect_object` oder `detect_object_extended` Service übergeben wird. Es enthält die ID des benutzten Greifers und optional einen Greif-Offset. Der Greifer muss im GripperDB Modul definiert werden (siehe *Erstellen eines Greifers*, Abschnitt 6.5.3.2) und Details über die Kollisionsprüfung werden in *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 6.4.2.2) gegeben.

Alternativ können Greifpunkten individuell Greifer IDs zugewiesen werden, und die Kollisionsprüfung kann für alle Greifpunkte mit einer Greifer ID über den Laufzeitparameter `check_collisions` eingeschaltet werden.

Wenn das ausgewählte CADMatch Template eine Kollisionsgeometrie enthält und der Laufzeitparameter `check_collisions_with_matches` auf true gesetzt ist, werden auch Kollisionen zwischen dem Greifer

und den anderen detektierten Objekten (nicht limitiert auf die Anzahl `max_matches`) geprüft. Dabei ist das Objekt, auf dem sich der jeweilige Greifpunkt befindet, von der Prüfung ausgenommen.

Wenn der Laufzeitparameter `check_collisions_with_point_cloud` auf true gesetzt ist, werden auch Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Wenn diese Funktionalität in Kombination mit Sauggreifern genutzt wird, muss sichergestellt werden, dass sich der TCP außerhalb der Greifergeometrie befindet, oder dass die Greifpunkte oberhalb der Objektoberfläche definiert sind. Andernfalls wird für jeden Greifpunkt eine Kollision zwischen Greifer und Punktwolke erkannt.

Wenn der Laufzeitparameter `check_collisions_during_retraction` auf true gesetzt ist, und ein Load Carrier sowie ein Greif-Offset angegeben wurden, wird jeder Greifpunkt auf Kollisionen zwischen dem Objekt im Greifer und den Wänden des Load Carriers während der Entnahme geprüft. Die Prüfung findet auf der gesamten linearen Trajektorie von der Greifposition bis zurück zur Vorgreifposition statt.

Wenn die Kollisionsprüfung aktiviert ist, werden nur Greifpunkte zurückgeliefert, die kollisionsfrei sind, oder die nicht auf Kollisionen geprüft werden konnten (z.B. weil kein Greifer angegeben wurde). In der Ergebnis-Visualisierung oben auf der *CADMatch*-Seite der Web GUI werden kollisionsfreie Greifpunkte grün dargestellt, ungeprüfte Greifpunkte gelb und kollidierende Greifpunkte rot. Die erkannten Objekte, die bei der Kollisionsprüfung betrachtet werden, werden mit roten Kanten visualisiert.

Die Laufzeitparameter des CollisionCheck-Moduls beeinflussen die Kollisionserkennung wie in *CollisionCheck-Parameter* (Abschnitt 6.4.2.3) beschrieben.

6.3.7.8 Parameter

Das CADMatch-Modul wird in der REST-API als `rc_cadmatch` bezeichnet und in der *Web GUI* (Abschnitt 7.1) in der gewünschten Pipeline unter *Module* → *CADMatch* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.41: Laufzeitparameter des rc_cadmatch-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
check_collisions	bool	false	true	false	Gibt an, ob Kollisionen geprüft werden sollen, wenn ein Greifer für einen Greifpunkt definiert wurde
check_collisions_during_retraction	bool	false	true	false	Gibt an, ob Kollisionen zwischen dem Objekt im Greifer und dem Load Carrier während der Entnahme geprüft werden
check_collisions_with_matches	bool	false	true	true	Gibt an, ob Kollisionen zwischen Greifer und anderen Matches geprüft werden
check_collisions_with_point_cloud	bool	false	true	false	Gibt an, ob Kollisionen zwischen Greifer und der Punktwolke geprüft werden
edge_max_distance	float64	0.5	5.0	2.0	Der maximale tolerierte Abstand zwischen den Templatekanten und den detektierten Kanten im Bild in Pixeln
edge_sensitivity	float64	0.05	1.0	0.5	Empfindlichkeit des Kantendetektors
grasp_filter_orientation_threshold	float64	0.0	180.0	45.0	Maximal erlaubte Orientierungsabweichung zwischen Greifpunkt und bevorzugter TCP-Orientierung in Grad
max_matches	int32	1	30	10	Maximale Anzahl von Matches
max_object_overlap	float64	0.0	1.0	1.0	Maximaler Anteil des Objekts, der von etwas anderem überlappt sein darf
min_score	float64	0.05	1.0	0.3	Minimaler Score für Matches
only_highest_priority_grasps	bool	false	true	false	Gibt an, ob ausschließlich Greifpunkte der höchsten Priorität zurückgegeben werden sollen.
prior_selection_mode	string	-	-	MatchSorting	Methode zur Auswahl von Priors für die Posenverfeinerung [MatchSorting, PriorAccessibility]

Beschreibung der Laufzeitparameter

Die Laufzeitparameter werden zeilenweise auf der *CADMatch*-Seite in der Web GUI dargestellt. Im folgenden wird der Name des Parameters in der Web GUI in Klammern hinter dem eigentlichen Parameternamen angegeben. Die Parameter sind in derselben Reihenfolge wie in der Web GUI aufgelistet:

max_matches (*Maximale Matches*)

ist die maximale Anzahl der zu erkennenden Objekte.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?max_matches=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?max_matches=<value>
```

`min_score` (*Minimaler Score*)

ist der minimale Score für die Erkennung nach der Posenverfeinerung. Umso höher dieser Wert ist, umso besser müssen die 2D-Kanten und die 3D-Punktwolke mit dem angegebenen Template übereinstimmen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?min_score=
↪<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?min_score=<value>
```

`edge_sensitivity` (*Kantenempfindlichkeit*)

ist die Empfindlichkeit des Kantendetektors. Umso höher dieser Wert ist, umso mehr Kanten werden für die Posenverfeinerung genutzt.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?edge_
↪sensitivity=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?edge_sensitivity=<value>
```

`edge_max_distance` (*Maximale Kantendistanz*)

ist die maximal erlaubte Distanz in Pixeln zwischen den Templatekanten und den detektierten Kanten im Bild während der Posenverfeinerung.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?edge_max_
↪distance=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?edge_max_distance=<value>
```

`grasp_filter_orientation_threshold` (*Maximale Abweichung*)

ist die maximale Abweichung der TCP-z-Achse am Greifpunkt von der z-Achse der bevorzugten TCP-Orientierung in Grad. Es werden nur Greifpunkte zurückgeliefert, deren Orientierungsabweichung kleiner als der angegebene Wert ist. Falls der Wert auf Null gesetzt wird, sind alle Abweichungen valide.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?grasp_filter_
↪orientation_threshold=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?grasp_filter_orientation_
↪threshold=<value>
```

`prior_selection_mode` (*Prior-Auswahlmodus*)

bestimmt die Methode zur Auswahl der erkannten Priors (ursprüngliche Posenschätzungen) für die Posenverfeinerung. Verfügbare Optionen sind `MatchSorting` und `PriorAccessibility`. Bei `MatchSorting` („Matchsortierung“) werden die Priors entsprechend der gesetzten Sortierstrategie ausgewählt. Dies ist der Standardmodus. Bei `PriorAccessibility` („Prior-Erreichbarkeit“) werden die Priors entsprechend ihrer Erreichbarkeit für das Greifen ausgewählt. Dieser Modus sollte für chaotische Szenen mit vielen überlappenden Objekten verwendet werden, z.B. beim Bin Picking.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?prior_
↪selection_mode=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?prior_selection_mode=<value>
```

`max_object_overlap` (*Maximale Objektüberlappung*)

Dieser Parameter bestimmt den maximalen Anteil eines Matches, der von anderen Objekten oder Szenenteilen überlappt sein darf, bezogen auf die Sichtachse der Kamera. Matches mit größeren Überlappungswerten werden gefiltert. Ein Wert von 1 schaltet den Überlappungscheck aus. Dieser Parameter kann genutzt werden, um nur Greifpunkte auf Objekten zu erhalten, die nicht von anderen überlappt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?max_object_
↪overlap=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?max_object_overlap=<value>
```

`only_highest_priority_grasps` (*Nur Greifpunkte höchster Priorität*)

Wenn dieser Parameter auf *true* gesetzt ist, werden ausschließlich Greifpunkte der höchsten Priorität zurückgegeben. Sofern die Kollisionsprüfung aktiviert ist, werden ausschließlich kollisionsfreie Greifpunkt der höchstmöglichen Priorität zurückgegeben. Dadurch kann Rechenzeit gespart und die Anzahl der applikationsseitig zu verarbeitenden Greifpunkte reduziert werden.

Ohne Kollisionsprüfung werden ausschließlich Greifpunkte der höchsten Priorität zurückgegeben.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?only_highest_
↪priority_grasps=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?only_highest_priority_grasps=
↪<value>
```

`check_collisions` (*Kollisionsprüfung*)

Wenn diese Option aktiv ist, wird die Kollisionsprüfung für alle Greifpunkte durchgeführt, denen eine Greifer ID zugewiesen wurde, auch wenn kein Standardgreifer im `detect_object` oder `detect_object_extended` Service gesetzt wurde. Wenn ein Load Carrier verwendet wird, wird die Kollisionsprüfung immer zwischen dem Greifer und dem Load Carrier durchgeführt. Kollisionen mit der Punktwolke oder anderen Matches werden nur geprüft, wenn die zugehörigen Laufzeitparameter aktiv sind.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?check_
↪collisions=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?check_collisions=<value>
```

`check_collisions_with_matches` (*Kollisionsprüfung mit Matches*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` oder `detect_object_extended` Service oder durch Setzen des Parameters `check_collisions` aktiviert ist. Wenn `check_collisions_with_matches` auf true gesetzt ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und den anderen Matches (nicht begrenzt auf die Anzahl `max_matches`) geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit anderen Matches wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?check_
↪collisions_with_matches=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?check_collisions_with_matches=
↪<value>
```

`check_collisions_with_point_cloud` (*Kollisionsprüfung mit Punktwolke*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` oder `detect_object_extended` Service oder durch Setzen des Parameters `check_collisions` aktiviert ist. Wenn `check_collisions_with_point_cloud` auf true gesetzt ist, werden alle Greifpunkte auf Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Nur Greifpunkte, bei denen der Greifer nicht in Kollision mit dieser Punktwolke wäre, werden zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?check_
↪collisions_with_point_cloud=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?check_collisions_with_point_
↪cloud=<value>
```

`check_collisions_during_retraction` (*Kollisionsprüfung während Entnahme*)

Dieser Parameter wird nur beachtet, wenn die Kollisionsprüfung durch Übergabe eines Greifers an den `detect_object` oder `detect_object_extended` Service oder durch Setzen des Parameters `check_collisions` aktiviert ist. Wenn `check_collisions_during_retraction` auf true gesetzt ist und ein Load Carrier sowie ein Greif-Offset angegeben wurden, wird jeder Greifpunkt auf Kollisionen zwischen dem Objekt im Greifer und den Wänden des Load Carriers während der Entnahme geprüft. Die Prüfung findet auf der gesamten linearen Trajektorie von der Greifposition bis zurück zur Vorgreifposition statt. Es werden nur kollisionsfreie Greifpunkte zurückgeliefert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/parameters?check_
↪collisions_during_retraction=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?check_collisions_during_
↪retraction=<value>
```

6.3.7.9 Statuswerte

Das CADMatch-Modul meldet folgende Statuswerte.

Tab. 6.42: Statuswerte des `rc_cadmatch`-Moduls

Name	Beschreibung
`data_acquisition_time`	Zeit in Sekunden, für die beim letzten Aufruf auf Bilddaten gewartet werden musste
`last_timestamp_processed`	Zeitstempel des letzten verarbeiteten Bilddatensatzes
`last_request_timestamp`	Zeitstempel der letzten Detektionsanfrage
`load_carrier_detection_time`	Berechnungszeit für die letzte Load Carrier Detektion in Sekunden
`object_detection_time`	Berechnungszeit für die letzte Objekterkennung in Sekunden
`processing_time`	Berechnungszeit für die letzte Detektion (einschließlich Load Carrier Detektion) in Sekunden
`state`	Aktueller Zustand des CADMatch-Moduls

Folgende `state`-Werte werden gemeldet.

Tab. 6.43: Mögliche Werte für den Zustand des CADMatch-Moduls

Zustand	Beschreibung
IDLE	Das Modul ist inaktiv.
RUNNING	Das Modul wurde gestartet und ist bereit, Load Carrier und Objekte zu erkennen.
FATAL	Ein schwerwiegender Fehler ist aufgetreten.

6.3.7.10 Services

Die angebotenen Services von `rc_cadmatch` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der *rc_reason_stack* *Web GUI* (Abschnitt 7.1) ausprobiert und getestet werden.

Das CADMatch-Modul stellt folgende Services zur Verfügung.

`detect_object`

führt eine Objekterkennung basierend auf einem Template durch, wie in *Objekterkennung* (Abschnitt 6.3.7.6) beschrieben.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/detect_object
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/detect_object
```

Request

Obligatorische Serviceargumente:

`pose_frame`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.7.7).

`template_id`: ID des Templates, welches erkannt werden soll.

Möglicherweise benötigte Serviceargumente:

`robot_pose`: siehe *Hand-Auge-Kalibrierung* (Abschnitt 6.3.7.7).

`pose_prior_ids`: IDs der Posenvorgaben für die zu erkennenden Objekte. Falls das ausgewählte Template eine Posenvorgabe für die Erkennung benötigt, dann muss dieses Argument angegeben werden.

Optionale Serviceargumente:

`load_carrier_id`: ID des Load Carriers, welcher die zu erkennenden Objekte enthält.

`load_carrier_compartment`: Teilvolumen (Fach oder Abteil) in einem zu detektierenden Load Carrier (Behälter), in dem Objekte erkannt werden sollen (siehe *Load Carrier Abteile*, Abschnitt 6.5.1.3).

`region_of_interest_id`: Falls `load_carrier_id` gesetzt ist, die ID der 3D Region of Interest, innerhalb welcher nach dem Load Carrier gesucht wird. Andernfalls die ID der 3D Region of Interest, in der nach Objekten gesucht wird.

`collision_detection`: siehe *Integrierte Kollisionsprüfung in anderen Modulen* (Abschnitt 6.4.2.2)

`data_acquisition_mode`: Falls der Aufnahmemodus auf `CAPTURE_NEW` (Standardwert) gesetzt ist, wird ein neuer Bild-Datensatz für die Detektion aufgenommen. Falls der Modus auf `USE_LAST` gesetzt wird, wird der Datensatz der vorherigen Detektion erneut verwendet.“

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "data_acquisition_mode": "string",
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "pose_prior_ids": [
      "string"
    ],
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "template_id": "string"
  }
}
```


Response

`grasps`: Liste von Greifpunkten auf den erkannten Objekten. Die Greifpunkte sind gemäß der gewählten Sortierstrategie sortiert. Die `match_uuid` gibt eine Referenz auf das detektierte Objekt in `matches` an, zu dem dieser Greifpunkt gehört. Die Liste der Greifpunkte wird auf die 100 besten Greifpunkte gekürzt, falls mehr erreichbare Greifpunkte gefunden werden. Jeder Greifpunkt enthält ein Flag `collision_checked` und das Feld `gripper_id` (siehe *Integrierte Kollisionsprüfung in anderen Modulen* Abschnitt 6.4.2.2).

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`matches`: Liste der erkannten Objekte für das angegebene Template, sortiert gemäß der gewählten Sortierstrategie. Der `score` gibt an, wie gut das Objekt mit dem Template übereinstimmt. Die `grasp_uuids` geben die Greifpunkte in der `grasps`-Liste an, die auf diesem Objekt erreichbar sind.

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "detect_object",
  "response": {
    "grasps": [
      {
        "collision_checked": "bool",
        "gripper_id": "string",
        "id": "string",
        "match_uuid": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "priority": "int8",
        "stroke_per_finger_approach_mm": "float64",
        "stroke_per_finger_grasp_mm": "float64",
        "timestamp": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "nsec": "int32",
        "sec": "int32"
      },
      "uuid": "string"
    }
  ],
  "load_carriers": [
    {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "overfilled": "bool",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "rim_ledge": {
        "x": "float64",
        "y": "float64"
      },
      "rim_step_height": "float64",
      "rim_thickness": {
        "x": "float64",
        "y": "float64"
      },
      "type": "string"
    }
  ],
  "matches": [
    {
      "grasp_uuids": [
        "string"
      ],
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "score": "float32",
        "template_id": "string",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "uuid": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


detect_object_extended

führt eine Objekterkennung basierend auf einem Template durch. Dieser Service verhält sich analog zu `detect_object`, gibt aber die Matchinformationen für jeden Greifpunkt direkt zurück, anstatt sie in einer separaten Liste zu speichern. Dies ermöglicht ein einfacheres Parsen, wenn z.B. die Matchposen für jeden Greifpunkt benötigt werden, um das Objekt platziert abzulegen.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/detect_object_
↪extended
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/detect_object_extended
```

Request

Siehe `detect_object` Service.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    },
    "data_acquisition_mode": "string",
    "load_carrier_compartment": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "pose_prior_ids": [
      "string"
    ],
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "template_id": "string"
  }
}
```


Response

`grasps`: Liste von Greifpunkten auf den erkannten Objekten. Die Greifpunkte sind gemäß der gewählten Sortierstrategie sortiert. Jeder Greifpunkt enthält ein Feld `match` mit Informationen des detektierten Objekts, z.B. seiner Pose. Die Liste der Greifpunkte wird auf die 100 besten Greifpunkte gekürzt, falls mehr erreichbare Greifpunkte gefunden werden. Jeder Greifpunkt enthält ein Flag `collision_checked` und das Feld `gripper_id` (siehe *Integrierte Kollisionsprüfung in anderen Modulen* Abschnitt 6.4.2.2).

`load_carriers`: Liste der erkannten Load Carrier (Behälter).

`timestamp`: Zeitstempel des Bildes, auf dem die Erkennung durchgeführt wurde.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "detect_object_extended",
  "response": {
    "grasps": [
      {
        "collision_checked": "bool",
        "gripper_id": "string",
        "id": "string",
        "match": {
          "pose": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "pose_frame": "string",
          "score": "float32",
          "template_id": "string",
          "uuid": "string"
        },
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "priority": "int8",
        "stroke_per_finger_approach_mm": "float64",
        "stroke_per_finger_grasp_mm": "float64",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "uuid": "string"
      }
    ],
    "load_carriers": [
      {
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "y": "float64",
          "z": "float64"
        },
        "overfilled": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```


set_preferred_orientation

speichert die bevorzugte TCP-Orientierung zum Berechnen der Erreichbarkeit der Greifpunkte, die zur Filterung und optional zur Sortierung der vom `detect_object` und `detect_object_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 6.3.7.4).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/set_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_preferred_orientation
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_preferred_orientation",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


get_preferred_orientation

gibt die bevorzugte TCP-Orientierung zurück, die für die Filterung und optional für die Sortierung der vom `detect_object` und `detect_object_extended` Service zurückgelieferten Greifpunkte verwendet wird (siehe *Setzen der bevorzugten TCP-Orientierung*, Abschnitt 6.3.7.4).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/get_preferred_
↪orientation
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_preferred_orientation
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_preferred_orientation",
  "response": {
    "orientation": {
      "w": "float64",
      "x": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

set_sorting_strategies

speichert die gewählte Strategie zur Sortierung der Greifpunkte und erkannten Objekte, die vom `detect_object` und `detect_object_extended` Service zurückgeliefert werden (siehe *Objekterkennung*, Abschnitt 6.3.7.6).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/set_sorting_
↪strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_sorting_strategies
```

Request

Nur eine Sortierstrategie darf einen Gewichtswert `weight` größer als 0 haben. Wenn alle Werte für `weight` auf 0 gesetzt sind, wird die Standardsortierstrategie verwendet.

Wenn der Wert `weight` für `direction` gesetzt ist, muss `vector` den Richtungsvektor enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Wenn der Wert `weight` für `distance_to_point` gesetzt ist, muss `point` den Sortierpunkt enthalten und `pose_frame` auf `camera` oder `external` gesetzt sein.

Wenn der Wert `weight` für `preferred_orientation` gesetzt ist, kann `axis` auf `x`, `y` oder `z` gesetzt werden, um nur Rotationsunterschiede zwischen diesen Achsen zu berücksichtigen. Wenn `axis` nicht gesetzt wird, wird die volle Rotationsdifferenz zur Sortierung verwendet.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "gravity": {
      "weight": "float64"
    },
    "match_score": {
      "weight": "float64"
    },
    "preferred_orientation": {
      "axis": "string",
      "weight": "float64"
    }
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_sorting_strategies",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


get_sorting_strategies

gibt die gewählte Sortierstrategie zurück, die zur Sortierung der vom `detect_object` und `detect_object_extended` Service zurückgelieferten Objekte und Greifpunkte verwendet wird (siehe *Objekterkennung*, Abschnitt 6.3.7.6).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/get_sorting_
↪strategies
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_sorting_strategies
```

Request

Dieser Service hat keine Argumente.

Response

Wenn alle Werte für `weight` 0 sind, wird die Standardsortierstrategie verwendet.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_sorting_strategies",
  "response": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
    "distance_to_point": {
      "farthest_first": "bool",
      "point": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose_frame": "string",
      "weight": "float64"
    },
    "gravity": {
      "weight": "float64"
    },
    "match_score": {
      "weight": "float64"
    },
    "preferred_orientation": {
      "axis": "string",
      "weight": "float64"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


warmup_template

Lädt ein Template, damit es bei der Detektion schon bereit ist. Ohne diesen Service dauert die erste Detektion mit einem neuen Template länger als die folgenden, da dann das Template bei der ersten Detektion erst geladen werden muss.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/warmup_template
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/warmup_template
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "template_id": "string"
  }
}
```

Die `template_id` ist die ID des Templates, welches in das CADMatch-Modul geladen werden soll.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "warmup_template",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

start

versetzt das CADMatch-Modul in den Zustand `RUNNING`.“

Details

Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `RUNNING` unterscheidenden Zustand zurück.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/start
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/start
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "start",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

stoppt das Modul und versetzt es in den Zustand `IDLE`.

Details

Es kann vorkommen, dass der Zustandsübergang noch nicht vollständig abgeschlossen ist, wenn die Serviceantwort generiert wird. In diesem Fall liefert diese den entsprechenden, sich von `IDLE` unterscheidenden Zustand zurück.

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/stop
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/stop
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

trigger_dump

speichert die Detektion auf dem angeschlossenen USB Speicher, die dem übergebenen Zeitstempel entspricht, oder die letzte, falls kein Zeitstempel angegeben wurde.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/trigger_dump
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/trigger_dump
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "comment": "string",
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "trigger_dump",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter und die bevorzugte TCP-Orientierung sowie die Sortierstrategie dieses Moduls wieder her und wendet sie an („factory reset“). Dies betrifft nicht die konfigurierten Templates.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.3.7.11 Interne Services

Die folgenden Services für die Konfiguration von Greifpunkten und Posenvorgaben können sich in Zukunft ohne weitere Ankündigung ändern. Es wird empfohlen, das Setzen, Abrufen und Löschen von Greifpunkten und Posenvorgaben über die Web GUI vorzunehmen.

Bemerkung: Das Konfigurieren von Greifpunkten und Posenvorgaben ist global für alle Templates auf dem *rc_reason_stack* und hat Einfluss auf alle Kamerapipelines.

set_grasp

speichert einen Greifpunkt für das angegebene Template auf dem *rc_reason_stack*. Alle Greifpunkte sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/set_grasp
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_grasp
```

Request

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 6.3.7.2) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "grasp": {
      "gripper_id": "string",
      "id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "priority": "int8",
      "replication": {
        "max_x_deg": "float64",
        "min_x_deg": "float64",
        "origin": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "step_x_deg": "float64"
      },
      "stroke_per_finger_approach_mm": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      "stroke_per_finger_grasp_mm": "float64",
      "template_id": "string"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_grasp",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

set_all_grasps

Ersetzt die gesamte Liste von Greifpunkten auf dem *rc_reason_stack* für das angegebene Template.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/set_all_grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_all_grasps
```

Request

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 6.3.7.2) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "grasps": [
      {
        "gripper_id": "string",
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        }
      },
      "priority": "int8",
      "replication": {
        "max_x_deg": "float64",
        "min_x_deg": "float64",
        "origin": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "step_x_deg": "float64"
      },
      "stroke_per_finger_approach_mm": "float64",
      "stroke_per_finger_grasp_mm": "float64",
      "template_id": "string"
    }
  ],
  "template_id": "string"
 }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_all_grasps",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


get_grasps

gibt alle definierten Greifpunkte mit den angegebenen IDs (`grasp_ids`) zurück, die zu den Templates mit den angegebenen `template_ids` gehören.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/get_grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_grasps
```

Request

Wenn keine `grasp_ids` angegeben werden, werden alle Greifpunkte zu den angegebenen `template_ids` zurückgeliefert. Wenn keine `template_ids` angegeben werden, werden alle Greifpunkte mit den geforderten `grasp_ids` zurückgeliefert. Wenn gar keine IDs angegeben werden, werden alle gespeicherten Greifpunkte zurückgeliefert.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_grasps",
  "response": {
    "grasps": [
      {
        "gripper_id": "string",
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "priority": "int8",
        "replication": {
          "max_x_deg": "float64",
          "min_x_deg": "float64",
          "origin": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "step_x_deg": "float64"
        },
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "stroke_per_finger_approach_mm": "float64",
        "stroke_per_finger_grasp_mm": "float64",
        "template_id": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


delete_grasps

löscht alle Greifpunkte mit den angegebenen `grasp_ids`, die zu den Templates mit den angegebenen `template_ids` gehören.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/delete_grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/delete_grasps
```

Request

Wenn keine `grasp_ids` angegeben werden, werden alle Greifpunkte gelöscht, die zu den Templates mit den angegebenen `template_ids` gehören. Die Liste `template_ids` darf nicht leer sein.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "delete_grasps",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_symmetric_grasps

gibt alle Greifpunkte zurück, die symmetrisch zum angegebenen Greifpunkt sind.“

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/get_symmetric_
↪grasps
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_symmetric_grasps
```

Request

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 6.3.7.2) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "grasp": {
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "replication": {
        "max_x_deg": "float64",
        "min_x_deg": "float64",
        "origin": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "step_x_deg": "float64"
      },
      "template_id": "string"
    }
  }
}
```

Response

Der erste Greifpunkt in der Rückgabeliste ist derselbe, der dem Service übergeben wurde. Wenn das Template keine exakte Symmetrie hat, wird nur der übergebene Greifpunkt zurückgeliefert. Wenn das Template eine kontinuierliche Symmetrie hat (z.B. ein zylindrisches Objekt), werden nur 12 gleichverteilte Greifpunkte zurückgeliefert.

Die Definition des Typs `grasp` wird in *Setzen von Greifpunkten* (Abschnitt 6.3.7.2) beschrieben.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_symmetric_grasps",
  "response": {
    "grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "replication": {
          "max_x_deg": "float64",
          "min_x_deg": "float64",
          "origin": {
            "orientation": {
              "w": "float64",
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "step_x_deg": "float64"
        },
        "template_id": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


set_pose_prior

speichert eine Posenvorgabe für das angegebene Template auf dem *rc_reason_stack*. Alle Posenvorgaben sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/set_pose_prior
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_pose_prior
```

Request

Die Definition des Typs `pose_prior` wird in *Setzen von Posenvorgaben* (Abschnitt 6.3.7.3) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "pose_prior": {
      "id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "template_id": "string"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_pose_prior",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

set_all_pose_priors

Ersetzt die gesamte Liste von Posenvorgaben auf dem *rc_reason_stack* für das angegebene Template.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/set_all_pose_
↪priors
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_all_pose_priors
```

Request

Die Definition des Typs `pose_prior` wird in *Setzen von Posenvorgaben* (Abschnitt 6.3.7.3) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "pose_priors": [
      {
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "template_id": "string"
      }
    ],
    "template_id": "string"
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_all_pose_priors",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`get_pose_priors`

gibt alle definierten Posenvorgaben mit den angegebenen IDs (`pose_prior_ids`) zurück, die zu den Templates mit den angegebenen `template_ids` gehören.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/get_pose_priors
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_pose_priors
```

Request

Wenn keine `pose_prior_ids` angegeben werden, werden alle Posenvorgaben zu den angegebenen `template_ids` zurückgeliefert. Wenn keine `template_ids` angegeben werden, werden alle Posenvorgaben mit den geforderten `pose_prior_ids` zurückgeliefert. Wenn gar keine IDs angegeben werden, werden alle gespeicherten Posenvorgaben zurückgeliefert.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "pose_prior_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_pose_priors",
  "response": {
    "pose_priors": [
      {
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "template_id": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

delete_pose_priors

löscht alle Posenvorgaben mit den angegebenen `pose_prior_ids`, die zu den Templates mit den angegebenen `template_ids` gehören.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/delete_pose_
↪priors
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/delete_pose_priors
```

Request

Wenn keine `pose_prior_ids` angegeben werden, werden alle Posenvorgaben gelöscht, die zu den Templates mit den angegebenen `template_ids` gehören. Die Liste `template_ids` darf nicht leer sein.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "pose_prior_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_pose_priors",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.3.7.12 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabecodes auf:

Tab. 6.44: Rückgabecodes der Services des CADMatch-Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-2	Ein interner Fehler ist aufgetreten.
-3	Ein interner Timeout ist aufgetreten, beispielsweise während der Objekterkennung.
-4	Die maximal erlaubte Zeitspanne für die interne Akquise der Bilddaten wurde überschritten.
-8	Das Modul befindet sich in einem Zustand, in welchem dieser Service nicht aufgerufen werden kann. Beispielsweise muss die Stereo-Matching Qualität `quality` mindestens `Medium` sein.
-9	Ungültige Lizenz
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern oder ROIs überschritten wurde.
-11	Sensor nicht verbunden, nicht unterstützt oder nicht bereit.
-12	Ressource ausgelastet, z.B. wenn `trigger_dump` zu häufig aufgerufen wird
10	Die maximal speicherbare Anzahl an Load Carriern oder ROIs wurde erreicht.
11	Existierende Daten wurden überschrieben.
100	Die angefragten Load Carrier wurden in der Szene nicht gefunden.
101	Keiner der gefundenen Greifpunkte ist erreichbar.
102	Der erkannte Load Carrier ist leer.
103	Alle berechneten Greifpunkte sind in Kollision.
106	Die Liste der zurückgelieferten Greifpunkte wurde auf die besten 100 Greifpunkte gekürzt.
110	Hinweise für die Einrichtung der Anwendung, z.B. Reduzieren des Abstands von der Kamera, Setzen einer Region of Interest.
113	Kein Greifer für die Kollisionsprüfung gefunden.
114	Kollisionsprüfung während Entnahme wurde nicht durchgeführt, z.B. weil kein Load Carrier oder kein Greif-Offset angegeben wurden.
151	Das Objekt-Template hat eine kontinuierliche Symmetrie.
152	Die Objekte liegen außerhalb der angegebenen Region of Interest, außerhalb des Load Carriers oder außerhalb des Bildes.
153	Es konnten keine Kanten im Kamerabild erkannt werden. Überprüfen Sie die Kantenempfindlichkeit.
999	Zusätzliche Hinweise für die Anwendungsentwicklung

6.3.7.13 Template API

Für den Upload, Download, das Auflisten und Löschen von Templates werden spezielle REST-API-Endpunkte zur Verfügung gestellt. Templates können auch über die Web GUI hoch- und runtergeladen werden. Die Templates beinhalten die Greifpunkte und Posenvorgaben, falls Greifpunkte oder Posenvorgaben konfiguriert wurden. Bis zu 50 Templates können gleichzeitig auf dem *rc_reason_stack* gespeichert werden.

`GET /templates/rc_cadmatch`
listet alle rc_cadmatch-Templates auf.

Musteranfrage

```
GET /api/v2/templates/rc_cadmatch HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  {
    "id": "string"
  }
]
```

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Array der Templates)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Template* (Abschnitt 7.2.3)

`GET /templates/rc_cadmatch/{id}`
ruft ein rc_cadmatch-Template ab. Falls der angefragte Content-Typ application/octet-stream ist, wird das Template als Datei zurückgegeben.

Musteranfrage

```
GET /api/v2/templates/rc_cadmatch/<id> HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **`id`** (*`string`*) – ID des Templates *(obligatorisch)*

Antwort-Header

- Content-Type – application/json application/ubjson application/octet-stream

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 404 Not Found – Modul oder Template wurden nicht gefunden.

Referenzierte Datenmodelle

- *Template* (Abschnitt 7.2.3)

`PUT /templates/rc_cadmatch/{id}`
erstellt oder aktualisiert ein rc_cadmatch-Template.

Musteranfrage

```
PUT /api/v2/templates/rc_cadmatch/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **id** (*string*) – ID des Templates *(obligatorisch)*

Formularparameter

- **file** – Template-Datei *(obligatorisch)*

Anfrage-Header

- Accept – multipart/form-data application/json

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Template)*
- 400 Bad Request – Template ist ungültig oder die maximale Zahl an Templates wurde erreicht.
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.
- 413 Request Entity Too Large – Template ist zu groß.

Referenzierte Datenmodelle

- *Template* (Abschnitt 7.2.3)

DELETE /templates/rc_cadmatch/{id}

entfernt ein rc_cadmatch-Template.

Musteranfrage

```
DELETE /api/v2/templates/rc_cadmatch/<id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameter

- **id** (*string*) – ID des Templates *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Modul oder Template wurden nicht gefunden.

6.4 Konfigurationsmodule

Der *rc_reason_stack* bietet mehrere verschiedene Konfigurationsmodule, welche es dem Nutzer ermöglichen, den *rc_reason_stack* für spezielle Anwendungen zu konfigurieren.

Die Konfigurationsmodule sind:

- ***Hand-Auge-Kalibrierung*** **(`rc_hand_eye_calibration`, Abschnitt 6.4.1)** ermöglicht dem Benutzer, die Kamera entweder über die Web GUI oder die REST-API zu einem Roboter zu kalibrieren.
- ***CollisionCheck*** **(`rc_collision_check`, Abschnitt 6.4.2)** bietet eine einfache Möglichkeit zu prüfen, ob ein Greifer in Kollision ist.
- ***Kamerakalibrierung*** **(`rc_stereocalib`, Abschnitt 6.4.3)** ermöglicht die Überprüfung und Durchführung der Kamerakalibrierung über die *Web GUI* (Abschnitt 7.1).
- ***IOControl und Projektor-Kontrolle*** **(`rc_iocontrol`, Abschnitt 6.4.4)** bietet die Kontrolle über die Ein- und Ausgänge der Kamera mit speziellen Betriebsarten zur Kontrolle eines externen Musterprojektors.

Diese Softwaremodule sind pipelinespezifisch, was heißt, dass sie innerhalb jeder Kamerapipeline laufen. Änderungen ihrer Einstellungen oder Parameter gelten nur für die zugehörige Pipeline und haben keinen Einfluss auf andere Kamerapipelines auf dem *rc_reason_stack*.

6.4.1 Hand-Auge-Kalibrierung

Für Anwendungen, bei denen die Kamera in eines oder mehrere Robotersysteme integriert wird, muss sie zum jeweiligen Roboter-Koordinatensystem kalibriert werden. Zu diesem Zweck wird der *rc_reason_stack* mit einer internen Kalibrierroutine, dem Modul zur *Hand-Auge-Kalibrierung*, ausgeliefert. Dieses Modul ist ein Basismodul, welches auf jedem *rc_reason_stack* verfügbar ist.

Bemerkung: Dieses Softwaremodul ist pipelinespezifisch. Änderungen seiner Einstellungen oder Parameter betreffen nur die zugehörige Kamerapipeline und haben keinen Einfluss auf die anderen Pipelines, die auf dem *rc_reason_stack* laufen.

Bemerkung: Für die Hand-Auge-Kalibrierung ist es völlig unerheblich, in Bezug auf welches benutzerdefinierte Roboter-Koordinatensystem die Kamera kalibriert wird. Hierbei kann es sich um einen Endeffektor des Roboters (z.B. Flansch oder Tool Center Point (Werkzeugmittelpunkt)) oder um einen beliebigen anderen Punkt in der Roboterstruktur handeln. Einzige Voraussetzung für die Hand-Auge-Kalibrierung ist, dass die Pose (d.h. Positions- und Rotationswerte) dieses Roboter-Koordinatensystems in Bezug auf ein benutzerdefiniertes externes Koordinatensystem (z.B. Welt oder Roboter-Montagepunkt) direkt von der Robotersteuerung erfasst und an das Kalibriermodul übertragen werden kann.

Die *Kalibrierroutine* (Abschnitt 6.4.1.3) ist ein benutzerfreundliches mehrstufiges Verfahren, für das mit einem Kalibriermuster gearbeitet wird. Entsprechende Kalibriermuster können von Roboception bezogen werden.

6.4.1.1 Kalibrierschnittstellen

Für die Durchführung der Hand-Auge-Kalibrierung stehen die folgenden beiden Schnittstellen zur Verfügung:

1. Alle Services und Parameter dieses Moduls, die für eine **programmgesteuerte** Durchführung der Hand-Auge-Kalibrierung benötigt werden, sind in der *REST-API-Schnittstelle* (Abschnitt 7.2) des *rc_reason_stack* enthalten. Der REST-API-Name dieses Moduls lautet `rc_hand_eye_calibration` und seine Services werden in *Services* (Abschnitt 6.4.1.5) erläutert.

Bemerkung: Für den beschriebenen Ansatz wird eine Netzwerkverbindung zwischen dem *rc_reason_stack* und der Robotersteuerung benötigt, damit die Steuerung die Roboterposen an das Kalibriermodul des *rc_reason_stack* übertragen kann.

2. Für Anwendungsfälle, bei denen sich die Roboterposen nicht programmgesteuert an das Modul zur Hand-Auge-Kalibrierung des *rc_reason_stack* übertragen lassen, sieht die Seite *Hand-Auge-Kalibrierung* unter dem Menüpunkt *Konfiguration* in der gewünschten Pipeline der *Web GUI* (Abschnitt 7.1) einen geführten Prozess vor, mit dem sich die Kalibrierroutine **manuell** durchführen lässt.

   Bemerkung: Während der Kalibrierung muss der Benutzer die Roboterposen, auf die über das jeweilige Teach-in- oder Handheld-Gerät zugegriffen werden muss, manuell in die Web GUI eingeben.

6.4.1.2 Kameramontage

Wie in Abb. 6.15 und Abb. 6.17 dargestellt, ist für die Montage der Kamera zwischen zwei unterschiedlichen Anwendungsfällen zu unterscheiden:

a. Die Kamera wird **am Roboter montiert**, d.h. sie ist mechanisch mit einem Roboterpunkt (d.h. Flansch oder flanschmontiertes Werkzeug) verbunden und bewegt sich demnach mit dem Roboter.

b. Die Kamera ist nicht am Roboter montiert, sondern an einem Tisch oder anderen Ort in der Nähe des Roboters befestigt und verbleibt daher verglichen mit dem Roboter in einer **statischen** Position.

Die allgemeine *Kalibrierroutine* (Abschnitt 6.4.1.3) ist in beiden Anwendungsfällen sehr ähnlich. Sie unterscheidet sich jedoch hinsichtlich der semantischen Auslegung der Ausgabedaten, d.h. der erhaltenen Kalibriertransformation, und hinsichtlich der Befestigung des Kalibriermusters.

Kalibrierung einer robotergeführten Kamera Soll eine robotergeführte Kamera zum Roboter kalibriert werden, so muss das Kalibriermuster in einer statischen Position zum Roboter, z.B. auf einem Tisch oder festen Sockel, befestigt werden (siehe Abb. 6.15).

Warnung: Es ist äußerst wichtig, dass sich das Kalibriermuster in Schritt 2 der *Kalibrierroutine* (Abschnitt 6.4.1.3) nicht bewegt. Daher wird dringend empfohlen, das Muster in seiner Position sicher zu fixieren, um unbeabsichtigte Bewegungen, wie sie durch Vibrationen, Kabelbewegungen oder Ähnliches ausgelöst werden, zu verhindern.

Das Ergebnis der Kalibrierung (Schritt 3 der *Kalibrierroutine*, Abschnitt 6.4.1.3) ist eine Pose $\mathbf{T}_{\text{camera}}^{\text{robot}}$, die die (zuvor unbekannte) relative Transformation zwischen dem *Kamera*-Koordinatensystem und dem benutzerdefinierten *Roboter*-Koordinatensystem beschreibt, sodass Folgendes gilt:

$$\mathbf{p}_{\text{robot}} = \mathbf{R}_{\text{camera}}^{\text{robot}} \cdot \mathbf{p}_{\text{camera}} + \mathbf{t}_{\text{camera}}^{\text{robot}} , \tag{6.1}$$

wobei $\mathbf{p}_{\text{robot}} = (x, y, z)^T$ ein 3D-Punkt ist, dessen Koordinaten im *Roboter*-Koordinatensystem angegeben werden, $\mathbf{p}_{\text{camera}}$ denselben Punkt im *Kamera*-Koordinatensystem darstellt, und $\mathbf{R}_{\text{camera}}^{\text{robot}}$ sowie $\mathbf{t}_{\text{camera}}^{\text{robot}}$ die 3×3 Drehmatrix und den 3×1 Translationsvektor für eine Pose $\mathbf{T}_{\text{camera}}^{\text{robot}}$ angeben. In der Praxis wird die Rotation für das Kalibrierergebnis und die Roboterposen als Eulerwinkel oder Quaternion anstatt einer Rotationsmatrix definiert (siehe *Formate für Posendaten*, Abschnitt 11.1).

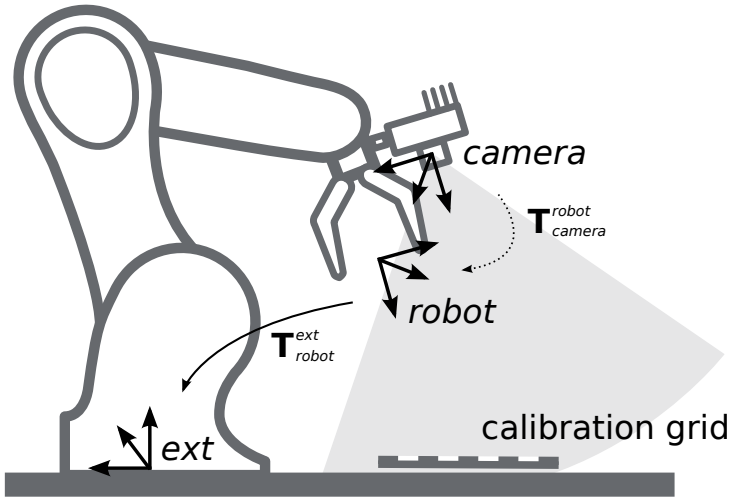


Abb. 6.15: Wichtige Koordinatensysteme und Transformationen für die Kalibrierung einer robotergeführten Kamera: Sie wird mit einer festen relativen Position zu einem benutzerdefinierten *Roboter*-Koordinatensystem (z.B. Flansch oder Werkzeugmittelpunkt) montiert. Es ist wichtig, dass die Pose $\mathbf{T}^{ext}_{robot}$ des *Roboter*-Koordinatensystems in Bezug auf ein benutzerdefiniertes externes Referenzkoordinatensystem (*ext*) während der Kalibrierroutine gemessen werden kann. Das Ergebnis des Kalibriervorgangs ist die gewünschte Kalibriertransformation $\mathbf{T}^{robot}_{camera}$, d.h. die Pose des *Kamera*-Koordinatensystems im benutzerdefinierten *Roboter*-Koordinatensystem.

Zusätzliche Benutzereingaben werden benötigt, falls die Bewegung des Roboters so beschränkt ist, dass der Tool Center Point (TCP) nur um eine Achse rotieren kann. Das ist üblicherweise für Roboter mit vier Freiheitsgraden (4DOF) der Fall, welche häufig zur Palettierung eingesetzt werden. In diesem Fall muss der Benutzer angeben, welche Achse des Roboterkoordinatensystems der Rotationsachse des TCP entspricht. Außerdem muss der vorzeichenbehaftete Offset vom TCP zum Kamerakoordinatensystem entlang der TCP-Rotationsachse angegeben werden. Abb. 6.16 zeigt die Situation.

Für den *rc_visard* und *rc_visard NG* befindet sich der Ursprung des Kamerakoordinatensystems im optischen Zentrum der linken Kamera. Die ungefähre Position wird im Abschnitt Coordinate frames im rc_visard Handbuch angegeben.

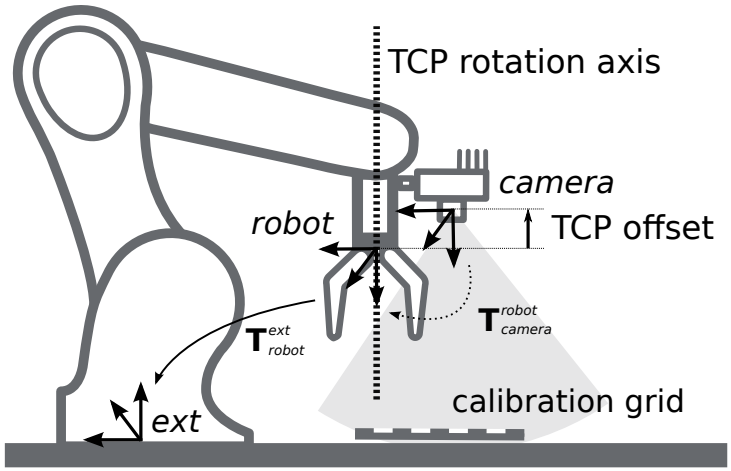


Abb. 6.16: Im Fall eines 4DOF-Roboters müssen die TCP-Rotationsachse und der Offset vom TCP zum Kamerakoordinatensystem entlang der TCP-Rotationsachse angegeben werden. Im dargestellten Fall ist der Offset negativ.

Kalibrierung einer statisch montierten Kamera In Anwendungsfällen, bei denen die Kamera statisch verglichen zum Roboter montiert wird, muss das Kalibriermuster, wie im Beispiel in Abb.

6.17 und Abb. 6.18 angegeben, angebracht werden.

Bemerkung: Für das Modul zur Hand-Auge-Kalibrierung spielt es keine Rolle, wie das Kalibriermuster in Bezug auf das benutzerdefinierte *Roboter*-Koordinatensystem genau angebracht und positioniert wird. Das bedeutet, dass die relative Positionierung des Kalibriermusters zu diesem Koordinatensystem weder bekannt sein muss, noch für die Kalibrierroutine relevant ist (siehe in Abb. 6.18).

Warnung: Es ist äußerst wichtig, das Kalibriermuster sicher am Roboter anzubringen, damit sich seine relative Position in Bezug auf das in Schritt 2 der *Kalibrierroutine* (Abschnitt 6.4.1.3) vom Benutzer definierte *Roboter*-Koordinatensystem nicht verändert.

In diesem Anwendungsfall ist das Ergebnis der Kalibrierung (Schritt 3 der *Kalibrierroutine*, Abschnitt 6.4.1.3) die Pose $\mathbf{T}^{\text{ext}}_{\text{camera}}$, die die (zuvor unbekannte) relative Transformation zwischen dem *Kamera*-Koordinatensystem und dem benutzerdefinierten *Roboter*-Koordinatensystem beschreibt, sodass Folgendes gilt:

$$\mathbf{p}_{\text{ext}} = \mathbf{R}^{\text{ext}}_{\text{camera}} \cdot \mathbf{p}_{\text{camera}} + \mathbf{t}^{\text{ext}}_{\text{camera}} \,, \tag{6.2}$$

wobei $\mathbf{p}_{\text{ext}} = (x, y, z)^T$ ein 3D-Punkt im externen Referenzkoordinatensystem *ext*, $\mathbf{p}_{\text{camera}}$ derselbe Punkt im Kamerakoordinatensystem *camera* und $\mathbf{R}^{\text{ext}}_{\text{camera}}$ sowie $\mathbf{t}^{\text{ext}}_{\text{camera}}$ die 3×3 Rotationsmatrix und 3×1 Translationsvektor der Pose $\mathbf{T}^{\text{ext}}_{\text{camera}}$ sind. In der Praxis wird die Rotation für das Kalibrierergebnis und die Roboterposen als Eulerwinkel oder Quaternion anstatt einer Rotationsmatrix definiert (siehe *Formate für Posendaten*, Abschnitt 11.1).

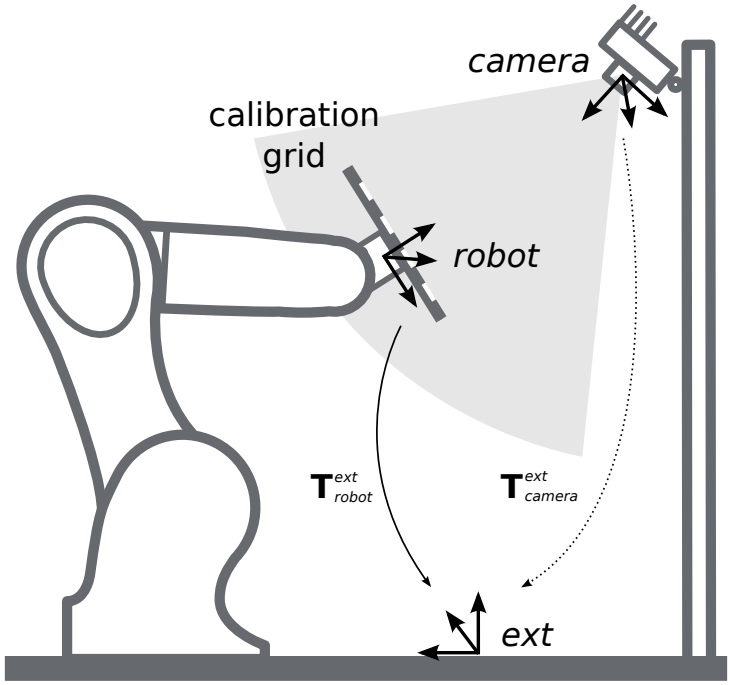


Abb. 6.17: Wichtige Koordinatensysteme und Transformationen für die Kalibrierung einer statisch montierten Kamera: Sie wird mit einer festen Position relativ zu einem benutzerdefinierten externen Referenzkoordinatensystem *ext* (z.B. Weltkoordinatensystem oder Roboter-Montagepunkt) montiert. Es ist wichtig, dass die Pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ des benutzerdefinierten *Roboter*-Koordinatensystems in Bezug auf dieses Koordinatensystem während der Kalibrierroutine gemessen werden kann. Das Ergebnis des Kalibrierprozesses ist die gewünschte Kalibriertransformation $\mathbf{T}^{\text{ext}}_{\text{camera}}$, d.h. die Pose des *Kamera*-Koordinatensystems im benutzerdefinierten externen Koordinatensystem *ext*.

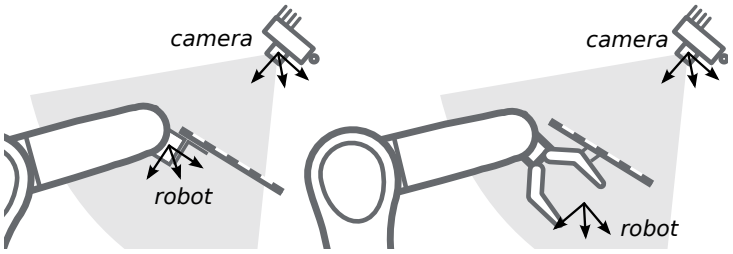


Abb. 6.18: Alternative Montageoptionen für die Befestigung des Kalibriermusters am Roboter

Zusätzliche Benutzereingaben werden benötigt, falls die Bewegung des Roboters so beschränkt ist, dass der Tool Center Point (TCP) nur um eine Achse rotieren kann. Das ist üblicherweise für Roboter mit vier Freiheitsgraden (4DOF) der Fall, welche häufig zur Palettierung eingesetzt werden. In diesem Fall muss der Benutzer angeben, welche Achse des Roboterkoordinatensystems der Rotationsachse des TCP entspricht. Außerdem muss der vorzeichenbehaftete Offset vom TCP zur sichtbaren Oberfläche des Kalibriermusters entlang der TCP-Rotationsachse angegeben werden. Das Kalibriermuster muss so angebracht werden, dass die TCP-Rotationsachse orthogonal zum Kalibriermuster verläuft. Abb. 6.19 zeigt die Situation.

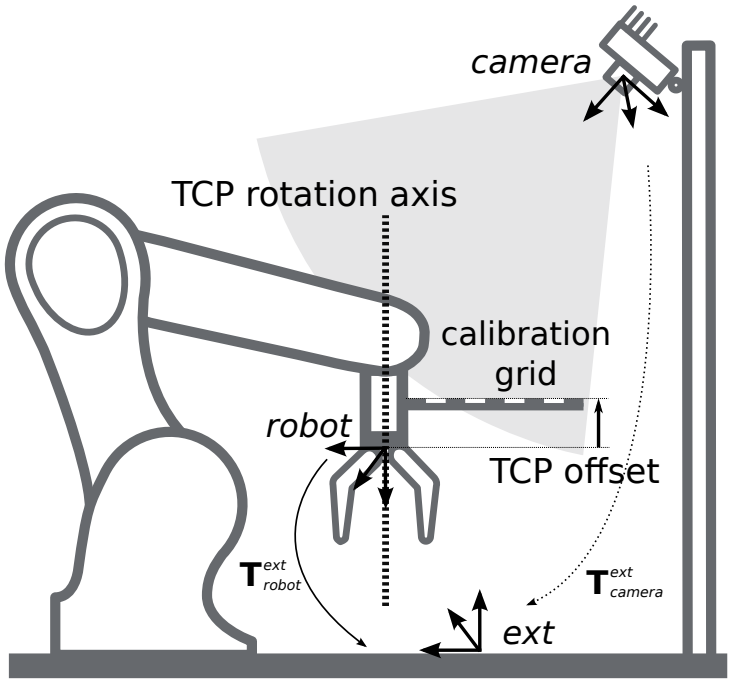


Abb. 6.19: Im Fall eines 4DOF-Roboters müssen die TCP-Rotationsachse und der Offset vom TCP zur sichtbaren Oberfläche des Kalibriermusters entlang der TCP-Rotationsachse angegeben werden. Im dargestellten Fall ist der Offset negativ.

6.4.1.3 Kalibrierroutine

Die Hand-Auge-Kalibrierung kann manuell über die *Web GUI* (Abschnitt 7.1) oder programmgesteuert über die *REST-API-Schnittstelle* (Abschnitt 7.2) durchgeführt werden. Die allgemeine Vorgehensweise wird beschrieben anhand der Schritte in der Web GUI in der gewünschten Pipeline unter *Konfiguration → Hand-Auge-Kalibrierung*. Verweise auf die zugehörigen REST-API Aufrufe werden an den entsprechenden Stellen bereitgestellt.

Schritt 1: Hand-Auge-Kalibrierstatus

Die Startseite des Assistenten für die Hand-Auge-Kalibrierung zeigt den aktuellen Status der Hand-Auge-Kalibrierung. Wenn eine Hand-Auge-Kalibrierung auf dem *rc_reason_stack* gespeichert ist, wird die Kalibriertransformation hier angezeigt (siehe Abb. 6.20).

Abb. 6.20: Aktueller Status der Hand-Auge-Kalibrierung falls eine Hand-Auge-Kalibrierung gespeichert ist

Um den Status der Hand-Auge-Kalibrierung programmgesteuert abzufragen bietet die REST-API den Service `get_calibration` (siehe *Services*, Abschnitt 6.4.1.5). Eine vorhandene Hand-Auge-Kalibrierung kann über *Kalibrierung entfernen* oder den REST-API Service `remove_calibration` (siehe *Services*, Abschnitt 6.4.1.5) gelöscht werden.

Durch Klick auf *Kalibrierung durchführen* wird eine neue Hand-Auge-Kalibrierung gestartet.

Schritt 2: Testen der Mustererkennung

Um gute Kalibrierergebnisse zu erzielen müssen die Bilder gut belichtet sein, damit das Kalibriermuster genau und verlässlich erkannt werden kann. In diesem Schritt kann die Erkennung des Kalibriermusters getestet werden und die Kameraeinstellungen können angepasst werden, falls nötig. Falls Teile des Kalibriermusters überbelichtet sind, werden die zugehörigen Quadrate rot hervorgehoben. Die erfolgreiche Erkennung des Kalibriermusters wird durch grüne Häkchen auf jedem Quadrat und einen dicken grünen Rahmen um das Kalibriermuster visualisiert, wie in Abb. 6.21 dargestellt ist.

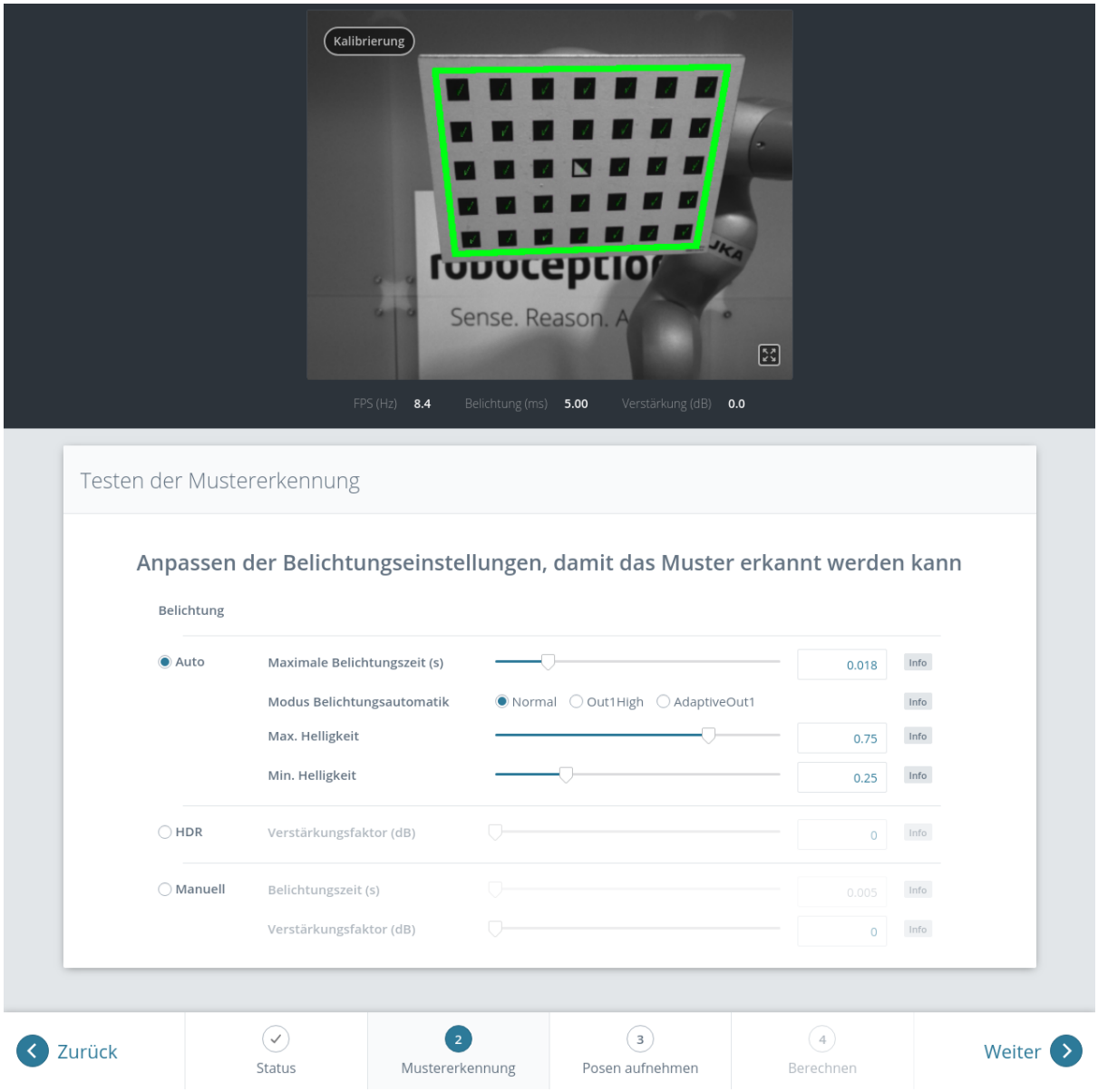


Abb. 6.21: Testen der Mustererkennung

Schritt 3: Posen aufnehmen

In diesem Schritt werden Bilder des Kalibriermusters an verschiedenen Roboterposen aufgenommen. Dabei ist sicherzustellen, dass das Kalibriermuster bei allen Posen im linken Kamerabild vollständig sichtbar ist. Zudem müssen die Roboterpositionen sorgsam ausgewählt werden, damit das Kalibriermuster aus unterschiedlichen Perspektiven aufgenommen wird. Abb. 6.22 zeigt eine schematische Darstellung der empfohlenen acht Ansichten.

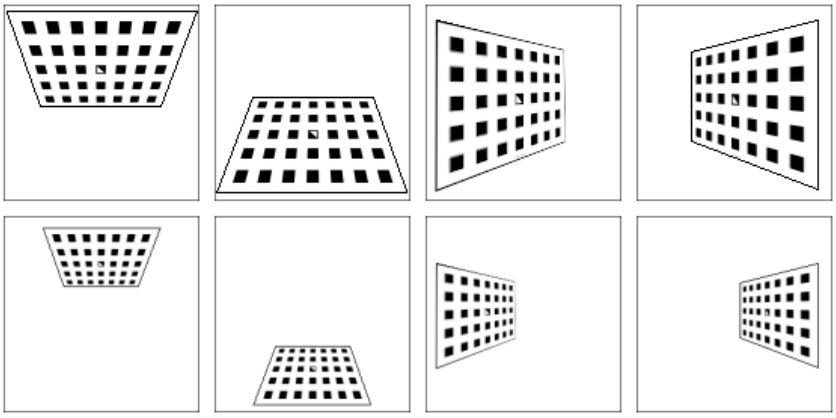


Abb. 6.22: Empfohlene Ansichten des Kalibriermusters während des Kalibriervorgangs. Im Fall von 4DOF-Robotern müssen andere Ansichten gewählt werden, welche so unterschiedlich wie möglich sein sollten.

Warnung: Die Kalibrierqualität, d.h. die Genauigkeit des berechneten Kalibrierergebnisses, hängt von den Ansichten des Kalibriermusters ab. Je vielfältiger die Perspektiven sind, desto besser gelingt die Kalibrierung. Werden sehr ähnliche Ansichten ausgewählt, d.h. wird die Pose des Roboters vor der Aufnahme einer neuen Kalibrierpose nur leicht variiert, kann dies zu einer ungenauen Schätzung der gewünschten Kalibriertransformation führen.

Nachdem der Roboter die jeweilige Kalibrierposition erreicht hat, muss die entsprechende Pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ des benutzerdefinierten *Roboter*-Koordinatensystems im benutzerdefinierten externen Referenzkoordinatensystem *ext* an das Modul zur Hand-Auge-Kalibrierung übertragen werden. Hierfür bietet das Softwaremodul verschiedene *Slots*, in denen die gemeldeten Posen mit den zugehörigen Bildern der linken Kamera hinterlegt werden können. Alle gefüllten Slots werden dann verwendet, um die gewünschte Kalibriertransformation zwischen dem *Kamera*-Koordinatensystem und dem benutzerdefinierten *Roboter*-Koordinatensystem (bei robotergeführten Kameras) bzw. dem benutzerdefinierten externen Referenzkoordinatensystem *ext* (bei statisch montierten Kameras) zu berechnen.

In der Web GUI kann der Nutzer zwischen vielen verschiedenen Formaten für die Kalibrierposen wählen (siehe *Formate für Posendaten*, Abschnitt 11.1). Wird die Kalibrierung über die REST-API vorgenommen, dann werden die Kalibrierdaten immer im Format *XYZ+Quaternion* angegeben. Die Web GUI bietet acht Slots (*Nahaufnahme 1*, *Nahaufnahme 2*, usw.), in die der Benutzer die Posen manuell eintragen kann. Neben jedem Slot wird eine Empfehlung für die Ansicht des Kalibriermusters angezeigt. Der Roboter sollte für jeden Slot so bewegt werden, dass die empfohlene Ansicht erreicht wird.

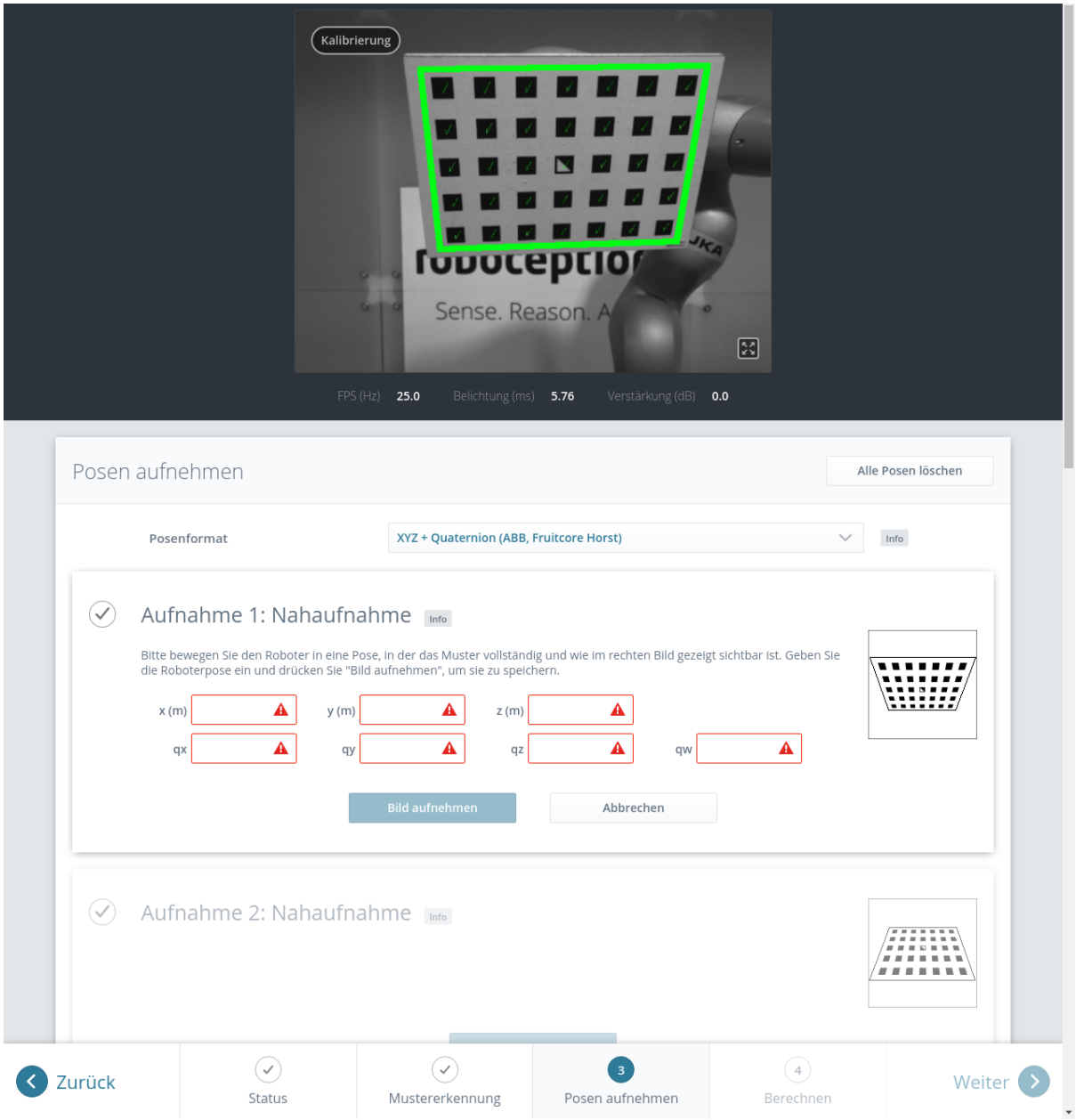


Abb. 6.23: Setzen der ersten Kalibrierpose für die Hand-Auge-Kalibrierung bei einer statisch montierten Kamera

Nach Klick auf *Pose setzen* kann die Pose des benutzerdefinierten *Roboter*-Koordinatensystems manuell in die entsprechenden Textfelder eingegeben werden. Durch *Bild aufnehmen* werden die Pose und das aktuelle Kamerabild im jeweiligen Slot gespeichert.

Um diese Posen programmgesteuert zu übertragen, bietet die REST-API den Service `set_pose` (siehe *Services*, Abschnitt 6.4.1.5).

Bemerkung: Der Zugriff auf die Posendaten des Roboters hängt vom Modell des Roboters und seinem Hersteller ab. Möglicherweise lassen sie sich über ein im Lieferumfang des Roboters enthaltenes Teach-in- oder Handheld-Gerät ablesen.

Warnung: Es ist wichtig darauf zu achten, dass genaue und korrekte Werte eingegeben werden. Selbst kleinste Ungenauigkeiten oder Tippfehler können dazu führen, dass die Kalibrierung fehl-

schlägt.

Die Web GUI zeigt die aktuell gespeicherten Kalibrierposen (nur mit den Slot-Nummern 0-7) und die zugehörigen Kamerabilder an und ermöglicht auch das Löschen von einzelnen Posen über *Pose löschen*, oder das Löschen aller gesetzten Posen über *Alle Posen löschen*. In der REST-API können die aktuell gespeicherten Kalibrierposen über `get_poses` abgefragt und über `delete_poses` oder `reset_calibration` einzeln bzw. komplett gelöscht werden (siehe *Services*, Abschnitt 6.4.1.5).

Wenn mindestens vier Posen gesetzt wurden, gelangt man über die Schaltfläche *Weiter* zur Berechnung des Kalibrierergebnisses.

Bemerkung: Um die Transformation für die Hand-Auge-Kalibrierung erfolgreich zu berechnen, müssen mindestens vier verschiedene Roboter-Kalibrierposen übertragen und in Slots hinterlegt werden. Um Kalibrierfehler zu verhindern, die durch ungenaue Messungen entstehen können, sind mindestens **acht Kalibrierposen empfohlen**.

Schritt 4: Kalibrierung berechnen

Bevor das Kalibrierergebnis berechnet werden kann, muss der Nutzer die korrekten Kalibrierparameter angeben. Diese beinhalten die exakten Abmessungen des Kalibriermusters und die Art der Sensormontage. Weiterhin kann die Kalibrierung von 4DOF-Robotern eingestellt werden. In diesem Fall müssen die Rotationsachse, sowie der Offset vom TCP zum Kamerakoordinatensystem (für Kameras am Roboter) oder zur Oberfläche des Kalibriermusters (für statische Kameras) angegeben werden. Für die REST-API sind die entsprechenden *Parameter* (Abschnitt 6.4.1.4) aufgelistet.

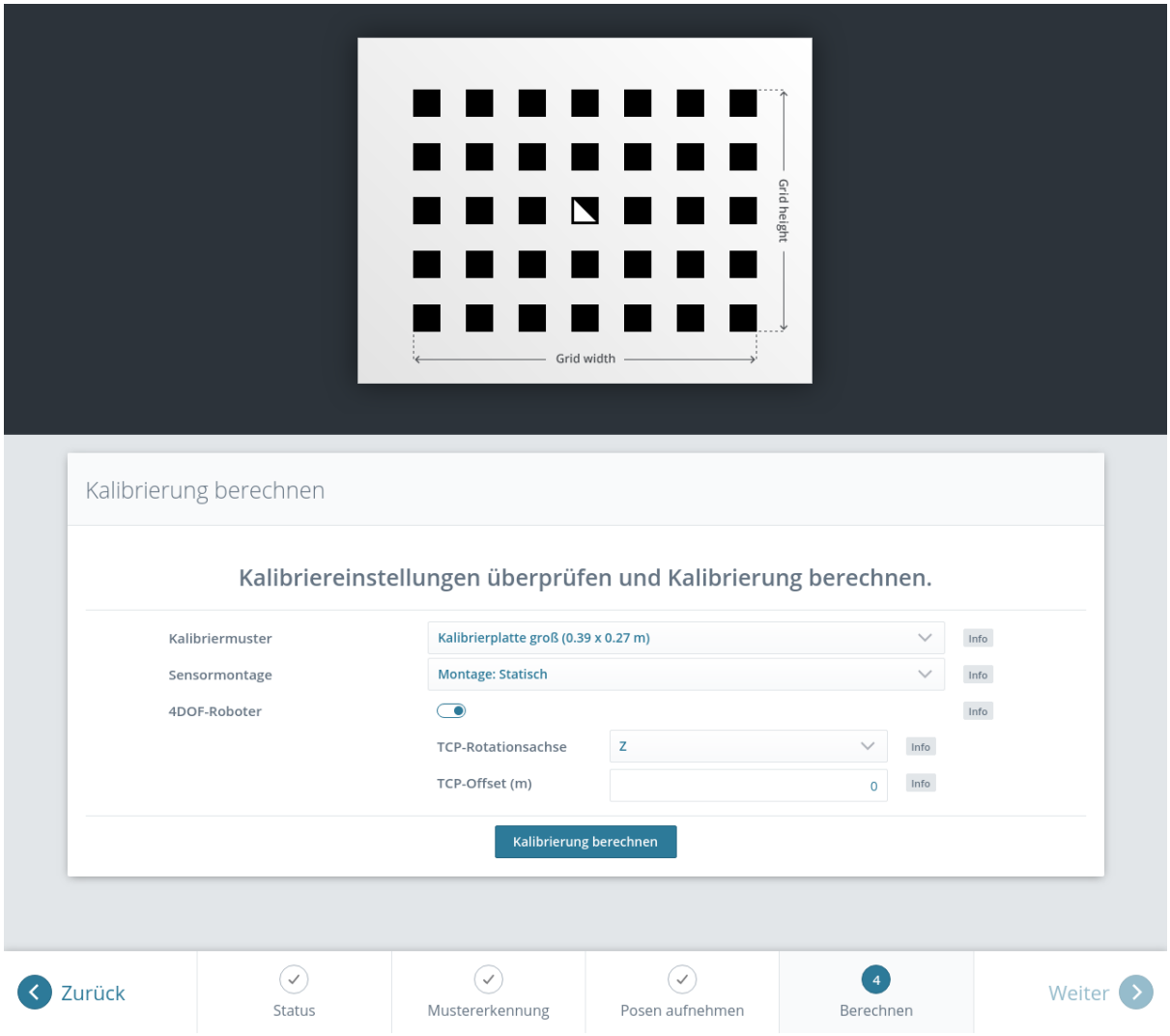


Abb. 6.24: Setzen der Parameter und Berechnen der Hand-Auge-Kalibrierung in der Web GUI des *rc_reason_stack*

Wenn die Parameter korrekt sind, kann durch *Kalibrierung berechnen* die gewünschte Kalibriertransformation aus den aufgenommenen Kalibrierposen und den zugehörigen Kamerabildern berechnet werden. Die REST-API bietet diese Funktion über den Service `calibrate` (siehe *Services*, Abschnitt 6.4.1.5).

Je nachdem, wie die Kamera montiert ist, wird dabei die Transformation (d.h. die Pose) zwischen dem *Kamera*-Koordinatensystem und entweder dem benutzerdefinierten *Roboter*-Koordinatensystem (bei robotergeführten Kameras) oder dem benutzerdefinierten externen Referenzkoordinatensystem *ext* (bei statisch montierten Kameras) berechnet und ausgegeben (siehe *Kameramontage*, Abschnitt 6.4.1.2).

Damit der Benutzer die Qualität der resultierenden Kalibriertransformation beurteilen kann, werden die translatorischen und rotatorischen Kalibrierfehler ausgegeben. Diese Werte werden aus der Varianz des Kalibrierergebnisses berechnet.

Wenn der Kalibrierfehler nicht akzeptabel ist, können die Kalibrierparameter geändert und das Ergebnis neu berechnet werden. Außerdem ist es möglich, zu Schritt 3 zurückzukehren, um mehr Posen aufzunehmen oder die vorhandenen Posen zu aktualisieren.

Durch Klicken auf *Kalibrierung speichern* oder über den REST-API Service `save_calibration` (siehe *Services*, Abschnitt 6.4.1.5) wird das Kalibrierergebnis gespeichert.

6.4.1.4 Parameter

Das Modul zur Hand-Auge-Kalibrierung wird in der REST-API als `rc_hand_eye_calibration` bezeichnet und in der *Web GUI* (Abschnitt 7.1) in der gewünschten Pipeline unter *Konfiguration* → *Hand-Auge Kalibrierung* dargestellt. Der Benutzer kann die Kalibrierparameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.45: Laufzeitparameter des `rc_hand_eye_calibration`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`grid_height`	float64	0.0	10.0	0.0	Höhe des Kalibriermusters in Metern
`grid_width`	float64	0.0	10.0	0.0	Breite des Kalibriermusters in Metern
`robot_mounted`	bool	false	true	true	Angabe, ob der *rc_visard* auf einem Roboter montiert ist
`tag_ids`	string	-	-	-	Optional, kommaseparierte Liste der AprilTag IDs, die mit kalibriert werden
`tcp_offset`	float64	-10.0	10.0	0.0	Offset vom TCP entlang tcp_rotation_axis
`tcp_rotation_axis`	int32	-1	2	-1	-1 für aus, 0 für x, 1 für y, 2 für z

Beschreibung der Laufzeitparameter

Für die Beschreibungen der Parameter sind die in der Web GUI gewählten Namen der Parameter in Klammern angegeben.

`grid_width` (*Breite*)

Breite des Kalibriermusters in Metern. Die Breite sollte mit sehr hoher Genauigkeit, vorzugsweise im Submillimeterbereich, angegeben werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/parameters?
↪grid_width=<value>
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?grid_width=<value>
```

`grid_height` (*Höhe*)

Höhe des Kalibriermusters in Metern. Die Höhe sollte mit sehr hoher Genauigkeit, vorzugsweise im Submillimeterbereich, angegeben werden.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/parameters?
↪grid_height=<value>
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?grid_height=<value>
```

`robot_mounted` (*Sensormontage*)

Ist dieser Parameter auf *true* gesetzt, dann ist die Kamera an einem Roboter montiert. Ist er auf *false* gesetzt, ist sie statisch montiert und das Kalibriermuster ist am Roboter angebracht.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/parameters?
↪robot_mounted=<value>
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?robot_mounted=<value>
```

`tcp_offset` (*TCP-Offset*)

Der vorzeichenbehaftete Offset vom TCP zum Kamerakoordinatensystem (für Kameras auf dem Roboter) oder der sichtbaren Oberfläche des Kalibriermusters (für statische Kameras) entlang der TCP-Rotationsachse in Metern. Dies wird benötigt, falls die Roboterbewegung eingeschränkt ist, sodass der TCP nur um eine Achse gedreht werden kann (z.B. bei 4DOF-Robotern).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/parameters?
↪tcp_offset=<value>
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?tcp_offset=<value>
```

`tcp_rotation_axis` (*TCP-Rotationsachse*)

Die Achse des Roboterkoordinatensystems, um die der Roboter seinen TCP drehen kann. 0 für X-, 1 für Y- und 2 für Z-Achse. Dies wird benötigt falls, die Roboterbewegung eingeschränkt ist, sodass der TCP nur um eine Achse gedreht werden kann (z.B. bei 4DOF-Robotern). -1 bedeutet, dass der Roboter seinen TCP um zwei unabhängige Achsen drehen kann. `tcp_offset` wird in diesem Fall ignoriert.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/parameters?
↪tcp_rotation_axis=<value>
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/parameters?tcp_rotation_axis=
↪<value>
```

6.4.1.5 Services

Auf die Services, die die REST-API für die programmgesteuerte Durchführung der Hand-Auge-Kalibrierung und für die Wiederherstellung der Modulparameter bietet, wird im Folgenden näher eingegangen.

`get_calibration`

Hiermit wird die derzeit auf dem *rc_reason_stack* gespeicherte Hand-Auge-Kalibrierung abgerufen.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/services/
↪get_calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/get_calibration
```

Request

Dieser Service hat keine Argumente.

Response

Das Feld `error` gibt den Kalibrierfehler in Pixeln an, der aus dem translatorischen Fehler `translation_error_meter` und dem rotatorischen Fehler `rotation_error_degree` berechnet wird. Dieser Wert wird nur aus Kompatibilitätsgründen mit älteren Versionen angegeben. Die translatorischen und rotatorischen Fehler sollten bevorzugt werden.

Tab. 6.46: Rückgabewerte des `get_calibration`-Services

status	success	Beschreibung
0	true	eine gültige Kalibrierung wurde zurückgegeben
2	false	die Kalibrierung ist nicht verfügbar

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_calibration",
  "response": {
    "error": "float64",
    "message": "string",
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "robot_mounted": "bool",
    "rotation_error_degree": "float64",
    "status": "int32",
    "success": "bool",
    "tags": [
      {
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "size": "float64"
      }
    ],
    "translation_error_meter": "float64"
  }
}
```


remove_calibration

Dieser Service löscht die persistente Hand-Auge-Kalibrierung auf dem *rc_reason_stack*. Nach diesem Aufruf gibt der `get_calibration` Service zurück, dass keine Hand-Auge-Kalibrierung vorliegt. Dieser Service löscht ebenfalls alle gespeicherten Kalibrierposen und die zugehörigen Kamerabilder.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/services/
↪remove_calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/remove_calibration
```

Request

Dieser Service hat keine Argumente.

Response

Tab. 6.47: Rückgabewerte des `get_calibration`-Services

status	success	Beschreibung
0	true	persistente Kalibrierung gelöscht, Gerät nicht mehr kalibriert
1	true	keine persistente Kalibrierung gefunden, Gerät nicht kalibriert
2	false	die Kalibrierung konnte nicht gelöscht werden

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "remove_calibration",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

set_pose

Dieser Service setzt die Roboterpose als Kalibrierpose für die Hand-Auge-Kalibrierroutine und nimmt das aktuelle Bild des Kalibriermusters auf.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/services/
↪set_pose
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/set_pose
```

Request

Das `slot`-Argument wird verwendet, um den verschiedenen Kalibrierpositionen eindeutige Ziffern im Wertebereich von 0-15 zuzuordnen. Wann immer der Service `set_pose` aufgerufen wird, wird ein Kamerabild aufgezeichnet. Dieser Service schlägt fehl, wenn das Kalibriermuster im aktuellen Bild nicht erkannt werden kann.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "slot": "uint32"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

Response

Tab. 6.48: Rückgabewerte des `set_pose`-Services

status	success	Beschreibung
1	true	Pose erfolgreich gespeichert
3	true	Pose erfolgreich gespeichert. Es wurden genügend Posen für die Kalibrierung gespeichert, d.h. die Kalibrierung kann durchgeführt werden
4	false	das Kalibriermuster wurde nicht erkannt, z.B. weil es im Kamerabild nicht vollständig sichtbar ist
8	false	keine Bilddaten verfügbar
12	false	die angegebenen Orientierungswerte sind ungültig
13	false	ungültige Slot-Nummer

Das Feld `overexposed` gibt an, ob Teile des Kalibriermusters bei diesem Bild überbelichtet sind.

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_pose",
  "response": {
    "message": "string",
    "overexposed": "bool",
    "status": "int32",
    "success": "bool"
  }
}
```

get_poses

Dieser Service gibt die aktuell gespeicherten Kalibrierposen für die Hand-Auge-Kalibrierung zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/services/
↪get_poses
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/get_poses
```

Request

Dieser Service hat keine Argumente.

Response

Tab. 6.49: Rückgabewerte des `get_poses`-Services

status	success	Beschreibung
0	true	gespeicherte Posen werden zurückgeliefert
1	true	keine Kalibrierposen verfügbar

Das Feld `overexposed` gibt an, ob Teile des Kalibriermusters bei diesem Bild überbelichtet sind.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_poses",
  "response": {
    "message": "string",
    "poses": [
      {
        "overexposed": "bool",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "slot": "uint32",
        "tag_ids": [
          "string"
        ]
      }
    ],
    "status": "int32",
    "success": "bool"
  }
}
```


delete_poses

Dieser Service löscht die Kalibrierposen und die zugehörigen Bilder mit den angegebenen Nummern in `slots`.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/services/
↪delete_poses
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/delete_poses
```

Request

Das Argument `slots` gibt die Ziffern der Kalibrierposen an, die gelöscht werden sollen. Wenn `slots` leer ist, werden keine Kalibrierposen gelöscht.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "slots": [
      "uint32"
    ]
  }
}
```

Response

Tab. 6.50: Rückgabewerte des `delete_poses`-Services

status	success	Beschreibung
0	true	Posen erfolgreich gelöscht
1	true	Keine Slots angegeben

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_poses",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

reset_calibration

Hiermit werden alle zuvor aufgenommenen Posen mitsamt der zugehörigen Bilder gelöscht. Das letzte hinterlegte Kalibrierergebnis wird neu geladen. Dieser Service kann verwendet werden, um die Hand-Auge-Kalibrierung (neu) zu starten.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/services/
↪reset_calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/reset_calibration
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_calibration",
  "response": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

calibrate

Dieser Service dient dazu, das Ergebnis der Hand-Auge-Kalibrierung auf Grundlage der über den Service `set_pose` konfigurierten Roboterposen zu berechnen und auszugeben.

Details

Damit die Kalibrierung für andere Module mit `get_calibration` verfügbar ist und persistent gespeichert wird, muss `save_calibration` aufgerufen werden.

Bemerkung: Zur Berechnung der Transformation der Hand-Auge-Kalibrierung werden mindestens vier Roboterposen benötigt (siehe `set_pose`). Empfohlen wird jedoch die Verwendung von acht Kalibrierposen.

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/services/
↪calibrate
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/calibrate
```

Request

Dieser Service hat keine Argumente.

Response

Das Feld `error` gibt den Kalibrierfehler in Pixeln an, der aus dem translatorischen Fehler `translation_error_meter` und dem rotatorischen Fehler `rotation_error_degree` berechnet wird. Dieser Wert wird nur aus Kompatibilitätsgründen mit älteren Versionen angegeben. Die translatorischen und rotatorischen Fehler sollten bevorzugt werden.

Tab. 6.51: Rückgabewerte des `calibrate`-Services

status	success	Beschreibung
0	true	Kalibrierung erfolgreich, das Kalibrierergebnis wurde zurückgegeben.
1	false	Nicht genügend Posen gespeichert, um die Kalibrierung durchzuführen
2	false	Das berechnete Ergebnis ist ungültig, bitte prüfen Sie die Eingabewerte.
3	false	Die angegebenen Abmessungen des Kalibriermusters sind ungültig.
4	false	Ungenügende Rotation, `tcp_offset` and `tcp_rotation_axis` müssen angegeben werden
5	false	Genügend Rotation verfügbar, `tcp_rotation_axis` muss auf -1 gesetzt werden
6	false	Die Posen sind nicht unterschiedlich genug.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "calibrate",
  "response": {
    "error": "float64",
    "message": "string",
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "robot_mounted": "bool",
    "rotation_error_degree": "float64",
    "status": "int32",
    "success": "bool",
    "tags": [
      {
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "size": "float64"
      }
    ],
    "translation_error_meter": "float64"
  }
}
```


save_calibration

Hiermit wird das Ergebnis der Hand-Auge-Kalibrierung persistent auf dem *rc_reason_stack* gespeichert und das vorherige Ergebnis überschrieben. Das gespeicherte Ergebnis lässt sich jederzeit über den Service `get_calibration` abrufen. Dieser Service löscht ebenfalls alle gespeicherten Kalibrierposen und die zugehörigen Kamerabilder.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/services/
↪save_calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/save_calibration
```

Request

Dieser Service hat keine Argumente.

Response

Tab. 6.52: Rückgabewerte des `save_calibration`-Services

status	success	Beschreibung
0	true	die Kalibrierung wurde erfolgreich gespeichert
1	false	die Kalibrierung konnte nicht im Dateisystem gespeichert werden
2	false	die Kalibrierung ist nicht verfügbar

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "save_calibration",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

set_calibration

Hiermit wird die übergebene Transformation als Hand-Auge-Kalibrierung gesetzt.

Details

Die Kalibrierung wird im gleichen Format erwartet, in dem sie beim `calibrate` und `get_calibration` Aufruf zurückgegeben wird. Die gegebene Kalibrierung wird auch persistent gespeichert, indem intern `save_calibration` aufgerufen wird.

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/services/
↪set_calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/set_calibration
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "robot_mounted": "bool",
    "tags": [
      {
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "size": "float64"
      }
    ]
  }
}
```


Response

Tab. 6.53: Rückgabewerte des `set_calibration`-Services

status	success	Beschreibung
0	true	Setzen der Kalibrierung war erfolgreich
12	false	die angegebenen Orientierungswerte sind ungültig

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_calibration",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```


reset_defaults

Hiermit werden die Werkseinstellungen der Parameter dieses Moduls wieder hergestellt und angewandt („factory reset“). Dies hat keine Auswirkungen auf das Kalibrierergebnis oder auf die während der Kalibrierung gefüllten `Slots`. Es werden lediglich Parameter, wie die Maße des Kalibriermusters oder die Montageart des Sensors, zurückgesetzt.

Details

Dieser Service kann wie folgt aufgerufen werden.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_hand_eye_calibration/services/
↪reset_defaults
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_hand_eye_calibration/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.4.2 CollisionCheck

6.4.2.1 Einleitung

Das CollisionCheck Modul ist ein optionales Modul, welches intern auf dem *rc_reason_stack* läuft, und ist freigeschaltet, sobald eine gültige Lizenz für eines der Module *ItemPick und ItemPickAI* (Abschnitt 6.3.4) und *BoxPick* (Abschnitt 6.3.5) oder *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6) vorhanden ist. Andernfalls benötigt dieses Modul eine separate *Lizenz* (Abschnitt 8.2).

Das Modul ermöglicht die Kollisionsprüfung zwischen dem Greifer und dem Load Carrier, der Punktwolke (nur in Kombination mit *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6)),‘ oder anderen detektierten Objekten (nur in Kombination mit *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6)). Es ist in die Module *ItemPick und ItemPickAI* (Abschnitt 6.3.4) und *BoxPick* (Abschnitt 6.3.5) und *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6) integriert, kann aber auch als eigenständiges Modul genutzt werden. Die Greifermodelle für die Kollisionsprüfung müssen über das *GripperDB* (Abschnitt 6.5.3) Modul definiert werden.

Warnung: Es werden nur Kollisionen zwischen dem Load Carrier und dem Greifer geprüft, aber nicht Kollisionen mit dem Roboter, dem Flansch oder anderen Objekten. Nur wenn `check_collisions_with_point_cloud` im entsprechenden Modul aktiviert ist, werden auch Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Nur in Kombination mit *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6), und nur wenn das gewählte Template eine Kollisionsgeometrie enthält und `check_collisions_with_matches` im entsprechenden Modul aktiviert ist, werden auch Kollisionen zwischen dem Greifer und den anderen *detektierten* Objekten geprüft. Kollisionen mit Objekten, die nicht detektiert werden können, werden nicht geprüft.

Bemerkung: Dieses Softwaremodul ist pipelinespezifisch. Änderungen seiner Einstellungen oder Parameter betreffen nur die zugehörige Kamerapipeline und haben keinen Einfluss auf die anderen Pipelines, die auf dem *rc_reason_stack* laufen.

Tab. 6.54: Spezifikationen des CollisionCheck-Moduls

Kollisionsprüfung mit	detektierter Load Carrier, detektierte Objekte (nur *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6)), Basisebene (nur *SilhouetteMatch und SilhouetteMatchAI*, Abschnitt 6.3.6), Punktwolke (nur *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6))
Kollisionsprüfung verfügbar in	*ItemPick und ItemPickAI* (Abschnitt 6.3.4) und *BoxPick* (Abschnitt 6.3.5), *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6)

6.4.2.2 Kollisionsprüfung

Stand-Alone Kollisionsprüfung

Der Service `check_collisions` triggert die Kollisionsprüfung zwischen dem angegebenen Greifer und dem angegebenen Load Carrier für jeden der übergebenen Greifpunkte. Eine Kollisionsprüfung mit anderen Objekten oder der Punktwolke ist nicht möglich. Das CollisionCheck-Modul überprüft, ob sich der Greifer in Kollision mit mindestens einem Load Carrier befindet, wenn sich der TCP an der Greifposition befindet. Es können mehrere Load Carrier gleichzeitig getestet werden. Der Griff wird als Kollision markiert, wenn es mit mindestens einem der definierten Load Carriern zu einer Kollision kommen würde.

Das Argument `pre_grasp_offset` (Greif-Offset) kann für eine erweiterte Kollisionsprüfung genutzt werden. Der Greif-Offset P_{off} ist der Offset vom Greifpunkt P_{grasp} zur Vorgreifposition P_{pre} im Koordinatensystem des Greifpunkts (siehe Abb. 6.25). Wenn der Greif-Offset angegeben wird, werden Greifpunkte auch dann als Kollisionen erkannt, wenn der Greifer an einem beliebigen Punkt während der linearen Bewegung zwischen Vorgreifposition und Greifposition in Kollision geraten würde.

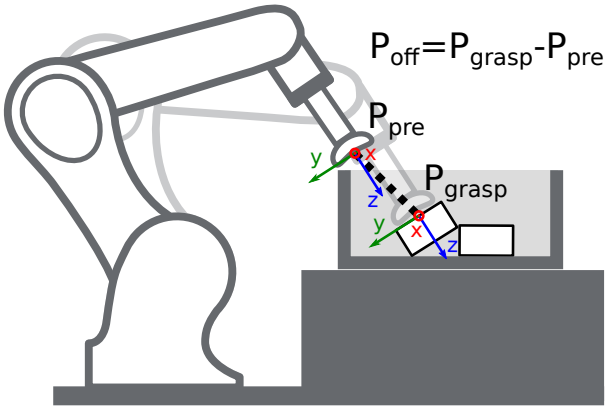


Abb. 6.25: Darstellung des Greif-Offsets für die Kollisionsprüfung. Im dargestellten Fall sind sowohl die Vorgreifposition als auch die Greifposition kollisionsfrei, aber die Trajektorie zwischen diesen Punkten hätte eine Kollision mit dem Load Carrier. Deswegen wird dieser Greifpunkt als Kollision erkannt.

Integrierte Kollisionsprüfung in anderen Modulen

Die Kollisionsprüfung ist in die Services der folgenden Softwaremodule integriert:

- *ItemPick und ItemPickAI* (Abschnitt 6.3.4): `compute_grasps` (siehe *compute_grasps*, Abschnitt 6.3.4.7
- *BoxPick* (Abschnitt 6.3.5): `compute_grasps` (siehe *compute_grasps*, Abschnitt 6.3.5.8
- *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6): `detect_object` (siehe *detect_object*, Abschnitt 6.3.6.11)

- *CADMatch* (Abschnitt 6.3.7): `detect_object` (siehe *detect_object*, Abschnitt 6.3.7.10)

Jedem dieser Services kann ein `collision_detection`-Argument übergeben werden, das aus der ID des Standardgreifers (`gripper_id`) und aus dem Greif-Offset (`pre_grasp_offset`, siehe *Stand-Alone Kollisionsprüfung*, Abschnitt 6.4.2.2) besteht. Der Standardgreifer, der durch das `gripper_id` Argument übergeben wird, wird nur für Greifpunkte verwendet, denen keine individuelle Greifer-ID zugewiesen wurde. Wenn das `collision_detection` Argument übergeben wird, liefern diese Services nur die Greifpunkte zurück, an denen der Greifer nicht in Kollision mit dem erkannten Load Carrier ist, oder für die keine Kollisionsprüfung durchgeführt werden konnte. Wenn eine Load Carrier ID angegeben wurde, wird die Kollisionsprüfung immer mit dem Load Carrier durchgeführt. Zusätzliche Funktionen für die Kollisionsprüfung können abhängig vom Modul aktiviert werden.

Nur in *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6), und nur wenn das gewählte Template eine Kollisionsgeometrie enthält und `check_collisions_with_matches` im entsprechenden Modul aktiviert ist, werden auch Greifpunkte, bei denen der Greifer in Kollision mit anderen *detektierten* Objekten wäre, herausgefiltert. Dabei ist das Objekt, auf dem sich der jeweilige Greifpunkt befindet, von der Prüfung ausgenommen.

Wenn ein Greifer für einen Greifpunkt in einem Template von *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6) definiert ist, wird dieser Greifer und nicht der im `collision_detection` Argument des `detect_object` Services angegebene Greifer zur Kollisionsprüfung dieses Greifpunkts verwendet (siehe *Setzen von Greifpunkten*, Abschnitt 6.3.6.4). Die vom `detect_object` Service zurückgelieferten Greifpunkte enthalten ein Flag `collision_checked`, das angibt ob der jeweilige Greifpunkt auf Kollisionen geprüft wurde, und das Feld `gripper_id`. Wenn `collision_checked` true ist, enthält das Feld `gripper_id` die ID des Greifers, der für die Kollisionsprüfung dieses Greifpunkts verwendet wurde. Dies ist die ID des Greifers, der für den jeweiligen Greifpunkt definiert wurde, oder, falls kein Greifer definiert wurde, die ID des Greifers die im `collision_detection` Argument des Serviceaufrufs angegeben wurde. Wenn `collision_checked` false ist, enthält `gripper_id` die ID des Greifers, die für den Greifpunkt definiert wurde.

In *SilhouetteMatch und SilhouetteMatchAI*, Abschnitt 6.3.6 werden Kollisionen zwischen dem Greifer und der Basisebene geprüft, wenn der Parameter `check_collisions_with_base_plane` in SilhouetteMatch aktiviert ist.

Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke können geprüft werden, wenn der Parameter `check_collisions_with_point_cloud` im jeweiligen Modul aktiviert ist.

Warnung: Es werden nur Kollisionen zwischen dem Load Carrier und dem Greifer geprüft, aber nicht Kollisionen mit dem Roboter, dem Flansch oder anderen Objekten. Nur wenn `check_collisions_with_point_cloud` im entsprechenden Modul aktiviert ist, werden auch Kollisionen zwischen dem Greifer und einer wasserdichten Version der Punktwolke geprüft. Nur in Kombination mit *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6), und nur wenn das gewählte Template eine Kollisionsgeometrie enthält und `check_collisions_with_matches` im entsprechenden Modul aktiviert ist, werden auch Kollisionen zwischen dem Greifer und den anderen *detektierten* Objekten geprüft. Kollisionen mit Objekten, die nicht detektiert werden können, werden nicht geprüft.

Nur in Kombination mit *CADMatch*, Abschnitt 6.3.7 und nur wenn `check_collisions_during_retraction` in CADMatch aktiviert ist, ein Load Carrier und ein Greif-Offset angegeben werden, werden Kollisionen zwischen dem Objekt im Greifer und den Wänden des Load Carriers auf der linearen Trajektorie zwischen Greifpunkt und Vorgreifposition geprüft.

Die Kollisionsprüfung wird von Laufzeitparametern beeinflusst, die weiter unten aufgeführt und beschrieben werden.

6.4.2.3 Parameter

Das CollisionCheck-Modul wird in der REST-API als `rc_collision_check` bezeichnet und in der *Web GUI* (Abschnitt 7.1) in der gewünschten Pipeline unter *Konfiguration* → *CollisionCheck* dargestellt. Der Benutzer kann die Parameter entweder dort oder über die *REST-API-Schnittstelle* (Abschnitt 7.2) ändern.

Übersicht der Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.55: Applikationsspezifische Laufzeitparameter des `rc_collision_check` Moduls

Name	Typ	Min	Max	Default	Beschreibung
`check_bottom`	bool	false	true	true	Aktiviert die Kollisionsprüfung mit dem Boden des Load Carriers.
`check_flange`	bool	false	false	true	Bestimmt, ob ein Greifpunkt als Kollision erkannt wird, sobald der Flansch innerhalb des Load Carriers ist.
`collision_dist`	float64	0.0	0.1	0.01	Minimaler Abstand in Metern zwischen einem Greiferelement und dem Load Carrier und/oder der Basisebene (nur SilhouetteMatch) für einen kollisionsfreien Griff.
`pointcloud_watertight`	bool	false	true	true	Ob ein wasserdichtes Disparitätsbild für die Kollisionsprüfung mit der Punktwolke verwendet werden soll

Beschreibung der Laufzeitparameter

Jeder Laufzeitparameter ist durch eine eigene Zeile in der Web GUI im Abschnitt *Einstellungen* in der gewünschten Pipeline unter *Konfiguration* → *CollisionCheck* repräsentiert. Der Web GUI-Name des Parameters ist in Klammern hinter dem Namen des Parameters angegeben:

`collision_dist` (*Sicherheitsabstand*)

Minimaler Abstand in Metern zwischen einem Greiferelement und dem Load Carrier und/oder der Basisebene (nur SilhouetteMatch) für einen kollisionsfreien Griff.

Bemerkung: Der Sicherheitsabstand wird nicht für die Kollisionsprüfung zwischen dem Greifer und der Punktwolke, oder dem Greifer und anderen detektierten Objekten angewendet. Er wird auch nicht verwendet um zu prüfen, ob sich der Flansch innerhalb des Load Carriers befindet (`check_flange`).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_collision_check/parameters?
↪collision_dist=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?collision_dist=<value>
```

`check_flange` (*Flansch-Check*)

Ermöglicht einen Sicherheitscheck mit dem Flansch, wie in *Flanschradius* (Abschnitt 6.5.3.2) beschrieben. Wenn dieser Parameter gesetzt ist, gelten alle Griffe, bei denen der Flansch innerhalb des Load Carriers wäre, als Kollisionen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_collision_check/parameters?check_
↪flange=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?check_flange=<value>
```

`check_bottom` (*Boden-Check*)

Wenn dieser Check aktiviert ist, werden Kollisionen nicht nur mit den Load Carrier Wänden, sondern auch mit dem Boden geprüft. Falls der TCP innerhalb der Kollisionsgeometrie (z.B. innerhalb des Sauggreifers) liegt, ist es möglicherweise nötig, diesen Check zu deaktivieren.

Der Load Carrier Boden wird immer ausgenommen von der Kollisionsprüfung zwischen dem Objekt im Greifer und dem Load Carrier während der Entnahme in Kombination mit *ItemPick und ItemPickAI* (Abschnitt 6.3.4) und *BoxPick* (Abschnitt 6.3.5), wenn `check_collisions_during_retraction` aktiv ist.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_collision_check/parameters?check_
↪bottom=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?check_bottom=<value>
```

`pointcloud_watertight` (*Wasserdichte Punktwolke*)

Wenn diese Option aktiv ist, wird die Punktwolke für die Kollisionsprüfung wasserdicht gemacht. In einer wasserdichten Punktwolke werden Lücken im Disparitätsbild durch gültige Messungen benachbarter Pixel interpoliert, sodass die resultierende Punktwolke keine Löcher mehr aufweist. Dies führt zu konservativen Ergebnissen bei der Kollisionsprüfung.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_collision_check/parameters?
↪pointcloud_watertight=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?pointcloud_watertight=
↪<value>
```

6.4.2.4 Statuswerte

Statuswerte des `rc_collision_check`-Moduls:

Tab. 6.56: Statuswerte des `rc_collision_check`-Moduls

Name	Beschreibung
`last_evaluated_grasps`	Anzahl der ausgewerteten Griffe
`last_collision_free_grasps`	Anzahl der kollisionsfreien Griffe
`collision_check_time`	Laufzeit der Kollisionsprüfung

6.4.2.5 Services

Die angebotenen Services von `rc_collision_check` können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der *rc_reason_stack Web GUI* (Abschnitt 7.1) ausprobiert und getestet werden.

Das CollisionCheck-Modul stellt folgende Services zur Verfügung.

`reset_defaults`

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_collision_check/services/reset_
↪defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`check_collisions` (veraltet)

löst eine Kollisionsprüfung zwischen dem Greifer und einem Load Carrier aus.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_collision_check/services/check_
↪collisions
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/check_collisions
```

Request

Obligatorische Serviceargumente:

`grasps`: Liste von Griffen, die überprüft werden sollen.

`load_carriers`: Liste von Load Carriern, die auf Kollisionen überprüft werden sollen. Die Felder der Load Carrier Definition sind in *Erkennung von Load Carriern* (Abschnitt 6.3.2.2) beschrieben. Die Griffe und die Load Carrier Positionen müssen im selben Koordinatensystem angegeben werden.

`gripper_id`: Die ID des Greifers, der in der Kollisionsprüfung verwendet werden soll. Der Greifer muss zuvor konfiguriert worden sein.

Optionale Serviceargumente:

`pre_grasp_offset`: Der Greif-Offset in Metern vom Greifpunkt zur Vorgreifposition. Wird ein Greif-Offset angegeben, dann wird die Kollisionsprüfung auf der gesamten linearen Trajektorie von der Vorgreifposition bis zur Greifposition durchgeführt.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "uuid": "string"
      }
    ],
    "gripper_id": "string",
    "load_carriers": [
      {
        "id": "string",
        "inner_dimensions": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        }
      }
    ],
    "pre_grasp_offset": {
      "x": "float64",
      "y": "float64",
      "z": "float64"
    }
  }
}
```


Response

`colliding_grasps`: Liste von Griffen, die in Kollision mit einem oder mehreren Load Carriern sind.

`collision_free_grasps`: Liste von kollisionsfreien Griffen.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "check_collisions",
  "response": {
    "colliding_grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "uuid": "string"
      }
    ],
    "collision_free_grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "uuid": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


`set_gripper` (veraltet)

konfiguriert und speichert einen Greifer auf dem *rc_reason_stack*.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *set_gripper* (Abschnitt 6.5.3.3) in `rc_gripper_db`.

API Version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/set_gripper
```

Die Definitionen von Request und Response sind dieselben wie in *set_gripper* (Abschnitt 6.5.3.3) in `rc_gripper_db` beschrieben.

`get_grippers` (veraltet)

gibt die mit `gripper_ids` spezifizierten und gespeicherten Greifer zurück.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *get_grippers* (Abschnitt 6.5.3.3) in `rc_gripper_db`.

API Version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/get_grippers
```

Die Definitionen von Request und Response sind dieselben wie in *get_grippers* (Abschnitt 6.5.3.3) in `rc_gripper_db` beschrieben.

`delete_grippers` (veraltet)

löscht die mit `gripper_ids` spezifizierten, gespeicherten Greifer.

API Version 2

Dieser Service ist in API Version 2 nicht verfügbar. Nutzen Sie stattdessen *delete_grippers* (Abschnitt 6.5.3.3) in `rc_gripper_db`.

API Version 1 (veraltet)

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/delete_grippers
```

Die Definitionen von Request und Response sind dieselben wie in *delete_grippers* (Abschnitt 6.5.3.3) in `rc_gripper_db` beschrieben.

6.4.2.6 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabecodes auf:

Tab. 6.57: Fehlercodes des CollisionCheck-Services

Code	Beschreibung
0	Erfolgreich
-1	Ein ungültiges Argument wurde übergeben.
-7	Daten konnten nicht in den persistenten Speicher geschrieben oder vom persistenten Speicher gelesen werden.
-9	Lizenz für CollisionCheck ist nicht verfügbar.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Greifern überschritten wurde.
10	Die maximal speicherbare Anzahl an Greifern wurde erreicht.
11	Bestehender Greifer wurde überschrieben.

6.4.3 Kamerakalibrierung

Das Kamerakalibrierungsmodul ist ein Basismodul, welches auf jedem *rc_reason_stack* verfügbar ist.

Bemerkung: Dieses Softwaremodul ist pipelinespezifisch. Änderungen seiner Einstellungen oder Parameter betreffen nur die zugehörige Kamerapipeline und haben keinen Einfluss auf die anderen Pipelines, die auf dem *rc_reason_stack* laufen.

Um die Kamera als Messinstrument zu verwenden, müssen die Kameraparameter, wie die Brennweite, die Objektivverzeichnung und die Lage der Kameras zueinander, genau bekannt sein. Diese Parameter werden durch Kalibrierung bestimmt und für die Rektifizierung der Bilder, die Grundlage für alle anderen Bildverarbeitungsmodule ist, verwendet (siehe *Rektifizierung*, Abschnitt 6.1.1).

Mit dem Kamerakalibrierungsmodul lassen sich die Kalibrierungsüberprüfung und Kalibrierung vornehmen.

6.4.3.1 Kalibriervorgang

Die Kamerakalibrierung kann über die *Web GUI* (Abschnitt 7.1) in der gewünschten Pipeline unter *Konfiguration* → *Kamera Kalibrierung* vorgenommen werden. Diese Seite bietet einen Assistenten, der den Benutzer durch den Kalibriervorgang führt.

Während der Kalibrierung muss das Kalibriermuster in verschiedenen Posen erkannt werden. Dabei müssen alle schwarzen Quadrate des Musters müssen komplett in beiden Kameras sichtbar sein und dürfen nicht verdeckt werden. Jedes korrekt erkannte Quadrat wird mit einem grünen Haken belegt. Das Muster kann nur dann korrekt erkannt werden, wenn alle schwarzen Quadrate erkannt werden. Werden einige der Quadrate nicht oder nur für kurze Zeit erkannt, so kann dies an schlechten Lichtverhältnissen oder einem beschädigten Kalibriermuster liegen. Quadrate, die in überbelichteten Bereichen des Kalibriermusters liegen, werden rot hervorgehoben. In diesem Fall müssen die Beleuchtung oder die Belichtungseinstellungen angepasst werden. Ein dicker grüner Rahmen um das Kalibriermuster zeigt an, dass das Muster korrekt in beiden Kamerabildern erkannt wurde.

Kalibriereinstellungen

Die Qualität der Kamerakalibrierung hängt stark von der Qualität des Kalibriermusters ab. Kalibriermuster können von Roboception bezogen werden.

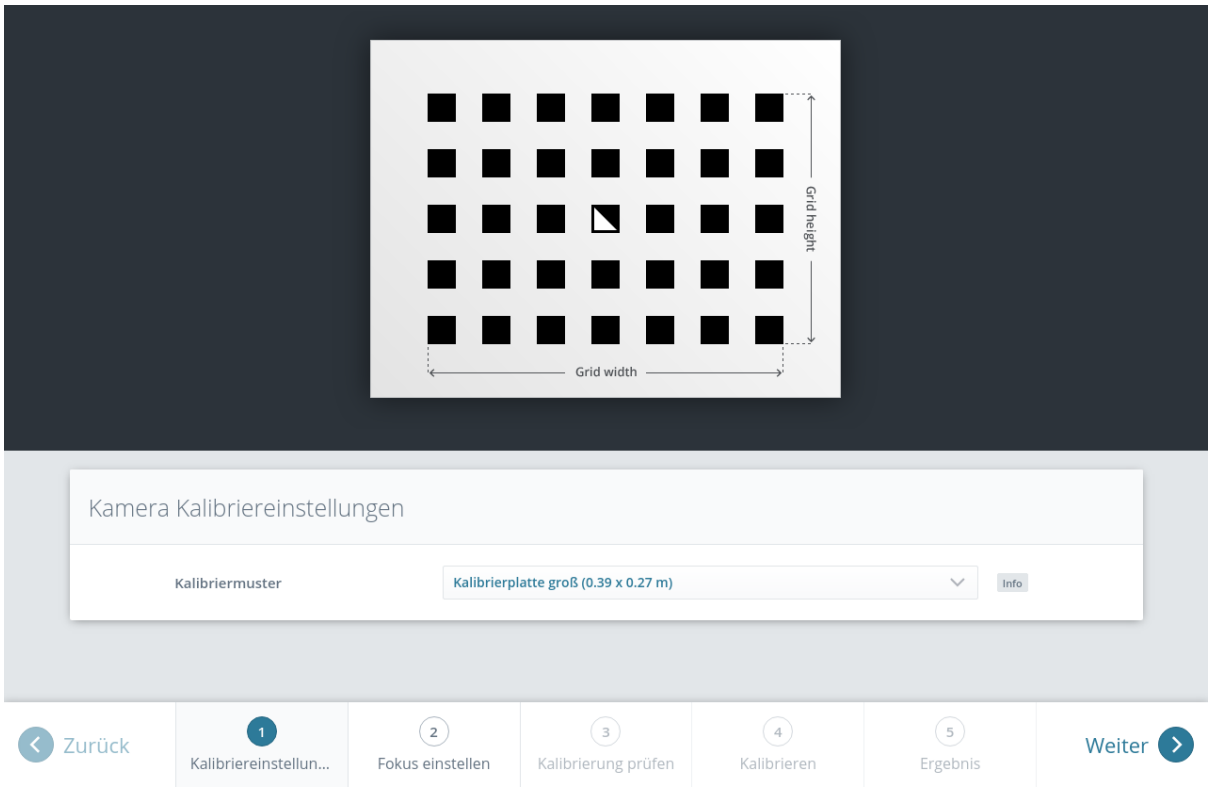


Abb. 6.26: Kalibriereinstellungen

Im ersten Schritt muss das verwendete Kalibriermuster angegeben werden. Mit Klick auf *Weiter* gelangt der Benutzer zum nächsten Schritt.

Fokus einstellen

Bemerkung: Dieser Schritt entfällt bei *rc_visard* Pipelines.

In diesem Schritt kann der Fokus der Kameras eingestellt werden. Dazu muss das Kalibriermuster so gehalten werden, dass es gleichzeitig in beiden Kameras sichtbar ist. Nachdem das Kalibriermuster erkannt wurde, erscheint an den rechten Bildrändern jeweils ein grüner Balken, der die Unschärfe des Bildes angibt. Der Fokus jeder Kamera sollte so eingestellt werden, dass dieser Balken für jedes Bild minimal wird.

Bemerkung: Während der Kalibrierung eines *rc_viscore* werden die Belichtungseinstellungen der Kamera temporär so verändert, dass die Kalibrierung einfacher möglich ist. Diese Belichtungswerte können weiterhin geändert werden, und sie werden zurückgesetzt, sobald die Kalibrierung abgeschlossen oder abgebrochen wird.

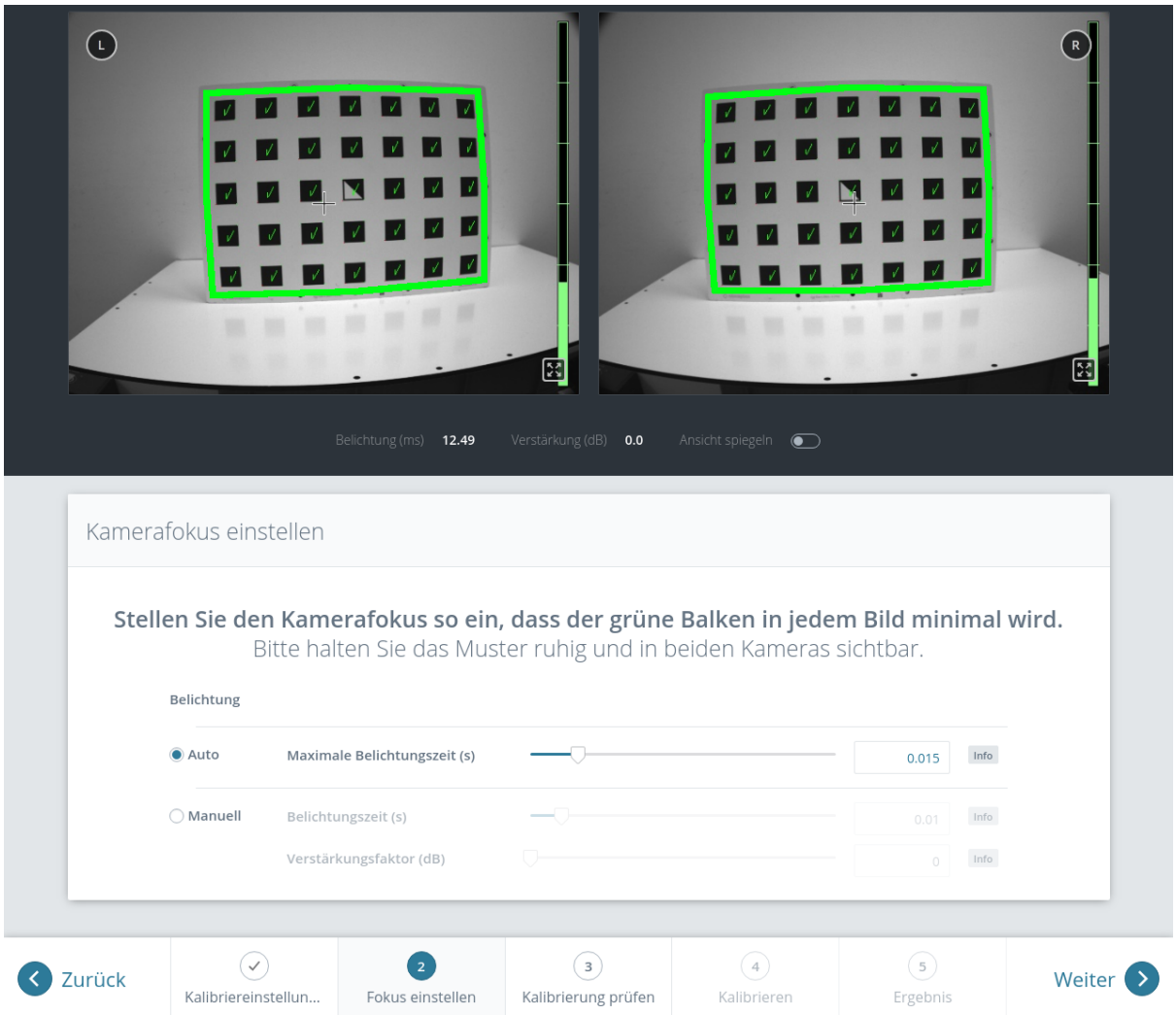


Abb. 6.27: Fokuseinstellung jeder Kamera

Kalibrierung prüfen

In diesem Schritt kann die aktuelle Kalibrierung überprüft werden. Um diese Prüfung vorzunehmen, muss das Muster so gehalten werden, dass es sich gleichzeitig im Sichtfeld beider Kameras befindet. Nachdem das Muster vollständig erkannt wurde, wird der Kalibrierfehler automatisch berechnet und das Ergebnis auf dem Bildschirm angegeben.

Abb. 6.28: Überprüfung der Kalibrierung

Bemerkung: Um einen aussagekräftigen Kalibrierfehler berechnen zu können, muss das Muster so nah wie möglich an die Kameras gehalten werden. Bedeckt das Muster lediglich einen kleinen Bereich der Kamerabilder, ist der Kalibrierfehler grundsätzlich geringer als wenn das Muster das gesamte Bild ausfüllt. Aus diesem Grund werden zusätzlich zum Kalibrierfehler an der aktuellen Position des Kalibriermusters auch der minimale und maximale Fehler während der Überprüfung der Kalibrierung angezeigt.

Der typische Kalibrierfehler beläuft sich auf unter 0,2 Pixel. Liegt der Fehler in diesem Bereich, kann der Kalibriervorgang übersprungen werden. Ist der errechnete Kalibrierfehler jedoch größer, sollte eine Neukalibrierung vorgenommen werden, um sicherzustellen, dass der Sensor volle Leistung erbringt. Mit Klick auf *Weiter* gelangt der Benutzer zum nächsten Schritt.

Warnung: Große Kalibrierfehler können durch falsch kalibrierte Kameras, ein unpräzises Kalibriermuster oder eine falsch eingetragene Musterbreite oder Musterhöhe verursacht werden. Bei der Verwendung eines benutzerdefinierten Kalibriermusters muss sichergestellt werden, dass das Muster präzise und die angegebenen Breiten- und Höhendaten korrekt sind. Anderenfalls kann die manuelle Kalibrierung sogar dazu führen, dass die Kameras dekalibriert werden!

Kalibrieren

Bevor die Kalibrierung vorgenommen wird, sollte die Belichtungszeit der Kamera richtig eingestellt werden. Um ein gutes Kalibrierergebnis zu erzielen, sollten die Bilder gut belichtet und Bewegungsunschärfe vermieden werden. Die maximale Belichtungszeit im automatischen Modus sollte so klein wie möglich sein, aber dennoch eine gute Belichtung ermöglichen. Die aktuelle Belichtungszeit wird, wie in Abb. 6.30 gezeigt, unter den Kamerabildern angegeben.

Für eine vollständige Kalibrierung müssen zunächst beide Kameras einzeln intrinsisch kalibriert werden (Monokalibrierung). Anschließend wird durch die Stereokalibrierung die Ausrichtung der beiden Kameras zueinander bestimmt. In den meisten Fällen wird die intrinsische Kalibrierung der beiden Kameras nicht beeinträchtigt. Daher wird die Monokalibrierung standardmäßig bei einer Neukalibrierung übersprungen, kann aber durch Klick auf *Monokalibrierung durchführen* durchgeführt werden. Dies sollte nur geschehen, wenn das Ergebnis der Stereokalibrierung nicht zufriedenstellend ist.

Stereokalibrierung

Bei der Stereokalibrierung wird die relative Rotation und Translation der Kameras zueinander ermittelt.

Die Kamerabilder können auch gespiegelt angezeigt werden, um die korrekte Ausrichtung des Kalibriermusters zu vereinfachen.

Als erstes muss das Kalibriermuster möglichst ruhig und so nah wie möglich an die Kamera gehalten werden. Es muss vollständig in beiden Bildern sichtbar sein und die Kameras sollten senkrecht auf das Kalibriermuster gerichtet sein. Wenn das Kalibriermuster nicht senkrecht zur Sichtachse der Kameras ausgerichtet ist, erscheinen kleine grüne Pfeile auf dem Kamerabild, die auf die erwarteten Positionen der Ecken des Kalibriermusters zeigen (siehe Abb. 6.29).

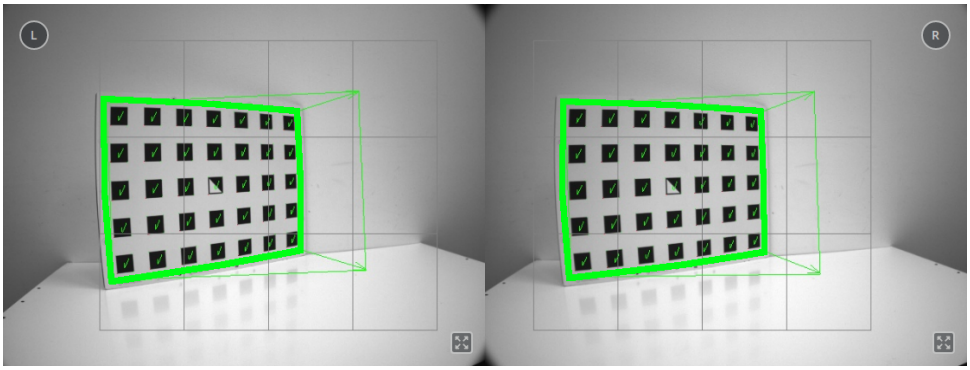


Abb. 6.29: Pfeile weisen darauf hin, wenn das Muster während der Stereokalibrierung nicht senkrecht zur Blickrichtung der Kamera gehalten wird.

Das Muster muss für die Erkennung sehr ruhig gehalten werden. Wenn Bewegungsunschärfe auftritt, wird das Muster nicht erkannt. Alle Zellen, die im Kamerabild dargestellt sind, müssen vom Kalibriermuster abgedeckt werden. Dies wird durch eine grüne Füllung der erfassten Zellen dargestellt (siehe Abb. 6.30).

In Abhängigkeit von der Kamera muss das Kalibriermuster möglicherweise an verschiedene Position gehalten werden, bis alle Zellen erfasst und grün hinterlegt sind.

Abb. 6.30: Stereokalibrierung: Das Muster sollte so nah wie möglich gehalten werden, um alle dargestellten Zellen zu füllen.

Bemerkung: Falls alle Häkchen auf dem Kalibriermuster verschwinden, liegt dies daran, dass die Kamerablickrichtung nicht senkrecht zum Muster steht, oder das Muster zu weit von der Kamera entfernt ist.

Sobald alle Zellen erfasst und gefüllt sind, verschwinden sie und eine einzelne entfernte Zelle wird angezeigt. Nun muss das Kalibriermuster so weit entfernt wie möglich gehalten werden, damit die kleine Zelle erfasst wird. Pfeile zeigen an, falls das Muster noch zu nah an der Kamera ist. Wenn das Kalibriermuster erfolgreich detektiert wurde, wird die Zelle grün und das Kalibrierergebnis kann berechnet werden (siehe Abb. 6.31).

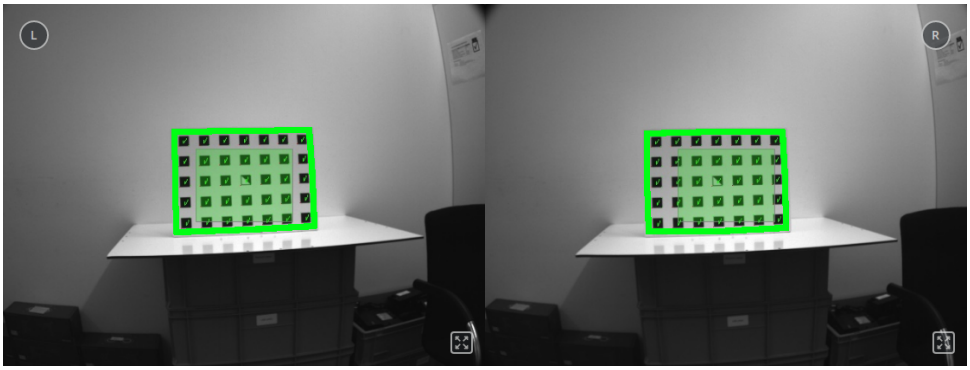


Abb. 6.31: Positionierung des Musters so entfernt wie möglich während der Stereokalibrierung

Führt die Stereokalibrierung nicht zu einem akzeptablen Kalibrierfehler, sollte die Kalibrierung erneut vorgenommen werden, jedoch mit Monokalibrierung (siehe nächster Abschnitt *Monokalibrierung*).

Monokalibrierung

Monokalibrierung ist die intrinsische Kalibrierung jeder einzelnen Kamera. Da die intrinsische Kalibrierung in der Regel nicht beeinträchtigt wird, sollte die Monokalibrierung nur durchgeführt werden, wenn das Ergebnis der Stereokalibrierung nicht zufriedenstellend ist.

Durch Klicken auf *Monokalibrierung durchführen* im Reiter *Kalibrieren* kann die Monokalibrierung gestartet werden.

Zur Kalibrierung muss das Kalibriermuster in verschiedenen Ausrichtungen vor die Kamera gehalten werden. Die Pfeile, die von den Ecken des Musters bis zu den grünen Bildschirmbereichen führen, geben an, dass alle Musterecken innerhalb der grünen Rechtecke platziert werden müssen. Diese grünen Rechtecke sind sensible Bereiche. Mit dem Schieberegler *Größe der sensiblen Bereiche* lässt sich die Größe der Rechtecke einstellen, um die Kalibrierung zu vereinfachen. Es ist jedoch zu bedenken, dass die Größe nicht zu stark erhöht werden darf, da dies auf Kosten der Kalibriergenauigkeit gehen kann.

Häufig wird der Fehler begangen, das Muster bei der Kalibrierung falsch herum zu halten. Dieser Fehler lässt sich leicht erkennen, da sich die von den Musterecken zu den grünen Rechtecken verlaufenden Linien in diesem Fall kreuzen (siehe Abb. 6.32).

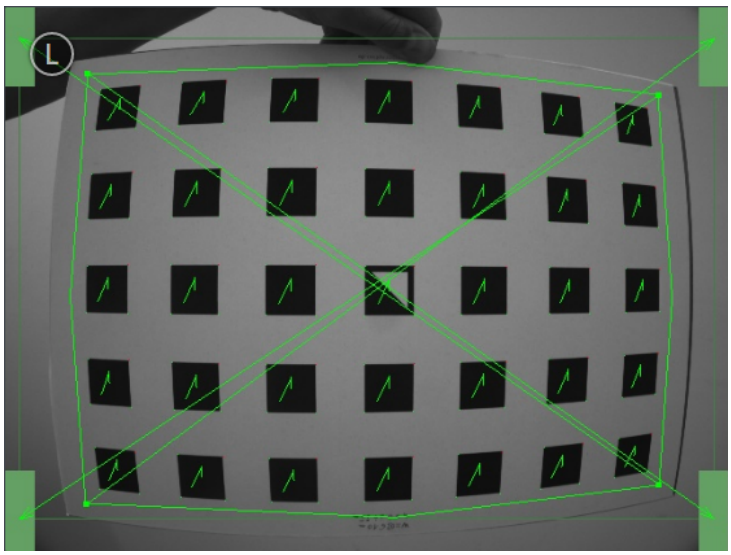


Abb. 6.32: Wird das Kalibriermuster falsch herum gehalten, kreuzen sich die grünen Linien.

Bemerkung: Die Kalibrierung mag umständlich erscheinen, da das Muster hierfür in bestimmten vordefinierten Stellungen gehalten werden muss. Dieses Vorgehen ist jedoch notwendig um ein qualitativ hochwertiges Kalibrierergebnis zu erreichen.

Für den Prozess der Monokalibrierung ist das Kalibriermuster für beide Kameras in den in Abb. 6.33 angegebenen Stellungen zu halten.

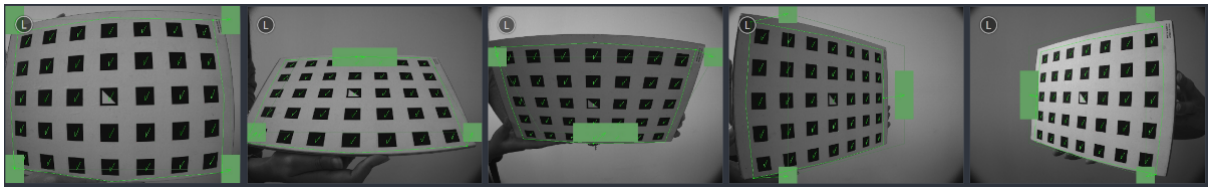


Abb. 6.33: Musterposen für die Monokalibrierung

Nachdem die Ecken oder Seiten des Kalibriermusters auf die sensiblen Bereiche ausgerichtet wurden, zeigt der Kalibriervorgang automatisch die nächste Stellung an. Sobald der Prozess für die linke Kamera abgeschlossen ist, ist er ebenso für die rechte Kamera zu wiederholen.

Anschließend folgen sind die Schritte im vorherigen Abschnitt *Stereokalibrierung* zu befolgen.

Kalibrierergebnis speichern

Mit Klick auf die Schaltfläche *Kalibrierung berechnen* wird der Kalibriervorgang beendet und das Endergebnis angezeigt. Der eingeblendete Wert ist der mittlere Reprojektionsfehler aller Kalibrierpunkte. Er ist in Pixeln angegeben und beläuft sich typischerweise auf einen Wert von unter 0,2.

Mit Klick auf *Kalibrierung speichern* wird das Kalibrierergebnis übernommen und auf dem Gerät gespeichert.

Bemerkung: Das eingeblendete Ergebnis ist der nach der Kalibrierung bestehende Mindestfehler. Der reale Fehler liegt auf keinen Fall darunter, könnte theoretisch jedoch höher sein. Dies gilt für jeden Algorithmus zur Kamerakalibrierung und ist der Grund dafür, warum das Kalibriermuster in verschiedenen Positionen vor den Sensor zu halten ist. So ist sichergestellt, dass der reale Kalibrierfehler den errechneten Fehler nicht signifikant überschreitet.

Warnung: War vor der Durchführung der Kamerakalibrierung eine Hand-Auge-Kalibrierung auf dem *rc_reason_stack* gespeichert, so sind die Werte der Hand-Auge-Kalibrierung möglicherweise ungültig geworden. Daher ist das Hand-Auge-Kalibrierverfahren zu wiederholen.

6.4.3.2 Parameter

Dieses Modul wird in der REST-API als `rc_stereocalib` bezeichnet.

Bemerkung: Die verfügbaren Parameter und die Statuswerte des Moduls zur Kamerakalibrierung sind nur für den internen Gebrauch bestimmt und können ohne vorherige Ankündigung Änderungen unterzogen werden. Die Kalibrierung sollte gemäß den vorstehenden Anweisungen und ausschließlich in der Web GUI vorgenommen werden.

6.4.3.3 Services

Bemerkung: Die verfügbaren Services des Moduls zur Kamerakalibrierung sind lediglich für den internen Gebrauch bestimmt und können ohne vorherige Ankündigung Änderungen unterzogen werden. Die Kalibrierung sollte gemäß den vorstehenden Anweisungen und ausschließlich in der Web GUI vorgenommen werden.

6.4.4 IOControl und Projektor-Kontrolle

Das IOControl Modul ist ein Basismodul, welches auf jedem *rc_reason_stack* läuft.

Das IOControl-Modul ermöglicht das Lesen der digitalen Eingänge und die Kontrolle der digitalen Ausgänge (GPIOs) der Kamera. Die Ausgänge können auf *aus* (LOW) oder *an* (HIGH) gesetzt werden. Sie können auch so konfiguriert werden, dass sie genau für die Belichtungszeit jedes Bildes, oder auch nur jedes zweiten Bildes, *an* sind.

Bemerkung: Dieses Softwaremodul ist pipelinespezifisch. Änderungen seiner Einstellungen oder Parameter betreffen nur die zugehörige Kamerapipeline und haben keinen Einfluss auf die anderen Pipelines, die auf dem *rc_reason_stack* laufen.

Das IOControl-Modul dient der Ansteuerung einer externen Lichtquelle oder eines Projektors, der an einen der GPIO-Ausgänge der Kamera angeschlossen wird, und der mit der Bildaufnahme synchronisiert ist. Für den Fall, dass ein Musterprojektor für die Verbesserung des Stereo-Matchings verwendet wird, ist das projizierte Muster auch in den Intensitätsbildern sichtbar. Das kann für Bildverarbeitungs-Anwendungen, die auf dem Intensitätsbild basieren (z.B. Kantendetektion), von Nachteil sein. Aus diesem Grund erlaubt das IOControl-Modul auch das Setzen der Ausgänge für nur jedes zweite Kamerabild. Somit sind auch Intensitätsbilder ohne projiziertes Muster verfügbar.

6.4.4.1 Parameter

Das IOControl-Modul wird in der REST-API als `rc_iocontrol` bezeichnet und in der *Web GUI* (Abschnitt 7.1) in der gewünschten Pipeline unter *Konfiguration* → *IOControl* dargestellt.

Der Benutzer kann die Parameter über die Web GUI oder die REST-API (*REST-API-Schnittstelle*, Abschnitt 7.2) ändern.

Übersicht über die Parameter

Dieses Softwaremodul bietet folgende Laufzeitparameter:

Tab. 6.58: Laufzeitparameter des `rc_iocontrol`-Moduls

Name	Typ	Min.	Max.	Default	Beschreibung
`out1_inverted`	bool	false	true	false	Out1 invertieren
`out1_mode`	string	-	-	Low	Out1 mode: [Low, High, ExposureActive, ExposureAlternateActive]
`out1_ratio`	float64	0.0	1.0	1.0	Anteil der Belichtungszeit für die Ausgang 1 (Out1) HIGH ist im Modus ExposureActive und ExposureAlternateActive.
`out2_inverted`	bool	false	true	false	Out2 invertieren
`out2_mode`	string	-	-	Low	Out2 mode: [Low, High, ExposureActive, ExposureAlternateActive]
`out2_ratio`	float64	0.0	1.0	1.0	Anteil der Belichtungszeit für die Ausgang 2 (Out2) HIGH ist im Modus ExposureActive und ExposureAlternateActive.

Beschreibung der Laufzeitparameter

`out1_mode` und `out2_mode` (*Ausgang 1 (Out1)* / *Projektor* und *Ausgang 2 (Out2)*)

Die Betriebsarten für GPIO-Ausgang 1 und GPIO-Ausgang 2 können individuell gesetzt werden:

`Low` schaltet den GPIO-Ausgang permanent *aus* (LOW). Das ist die Standardeinstellung.

`High` schaltet den GPIO-Ausgang permanent *an* (HIGH).

`ExposureActive` schaltet den GPIO-Ausgang für die Belichtungszeit jedes Bildes *an* (HIGH).

`ExposureAlternateActive` schaltet den GPIO-Ausgang für die Belichtungszeit jedes zweiten Bildes *an* (HIGH).

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_iocontrol/parameters?<out1_
↪mode|out2_mode>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/parameters?<out1_mode|out2_mode>=<value>
```

Abb. 6.34 zeigt, welche Bilder für das Stereo-Matching und die GigE Vision-Übertragung in der Betriebsart `ExposureActive` mit einer benutzerdefinierten Bildwiederholrate von 8 Hz benutzt werden.

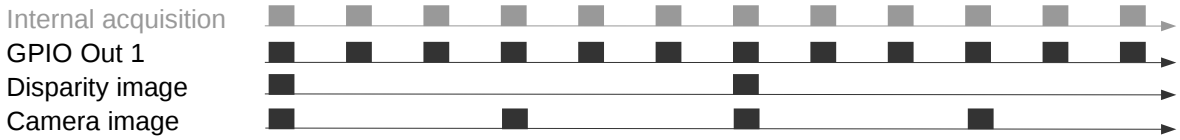


Abb. 6.34: Beispiel für die Nutzung der Betriebsart `ExposureActive` für GPIO-Ausgang 1 mit einer benutzerdefinierten Bildwiederholrate von 8 Hz. Die interne Bildaufnahme geschieht immer mit 25 Hz. GPIO-Ausgang 1 ist für die Dauer der Belichtungszeit jedes Bildes *an* (HIGH). Disparitätsbilder werden für Kamerabilder berechnet, die auch per GigE Vision in der benutzerdefinierten Bildwiederholrate versendet werden.

Die Betriebsart `ExposureAlternateActive` ist gedacht, um einen externen Musterprojektor anzusteuern, der am GPIO-Ausgang 1 der Kamera angeschlossen ist. In diesem Fall nutzt das *Stereo-Matching-Modul* (Abschnitt 6.2.2) nur Bilder, bei denen GPIO-Ausgang 1 *an* (HIGH) ist, d.h. der Projektor ist an. Die maximale Bildwiederholrate, welche für das Stereo-Matching genutzt wird, ist hierbei die halbe vom Benutzer konfigurierte Bildwiederholrate. Alle Module, die Intensitätsbilder benutzen, wie z.B. *TagDetect* (Abschnitt 6.3.3) und *ItemPick* (Abschnitt 6.3.4), benutzen die Intensitätsbilder, bei denen der GPIO-Ausgang 1 *aus* (LOW) ist, d.h. der Projektor ist aus. Abb. 6.35 zeigt ein Beispiel.

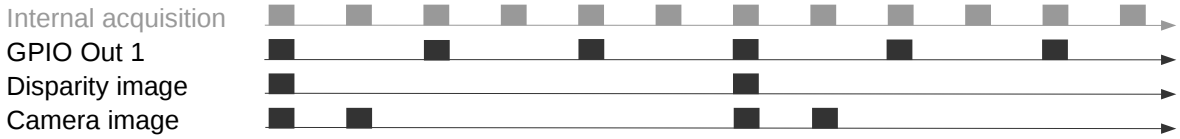


Abb. 6.35: Beispiel für die Nutzung der Betriebsart `ExposureAlternateActive` für GPIO-Ausgang 1 mit einer benutzerdefinierten Bildwiederholrate von 8 Hz. Die interne Bildaufnahme geschieht immer mit 25 Hz. GPIO-Ausgang 1 ist für die Dauer der Belichtungszeit jedes zweiten Bildes *an* (HIGH). Disparitätsbilder werden für Kamerabilder berechnet, bei denen GPIO-Ausgang 1 *an* (HIGH) ist, und die auch per GigE Vision in der benutzerdefinierten Bildwiederholrate versendet werden. In der Betriebsart `ExposureAlternateActive` werden Intensitätsbilder immer paarweise versendet: ein Bild mit GPIO-Ausgang 1 *an* (HIGH), für das ein Disparitätsbild verfügbar sein kann, und ein Bild mit GPIO-Ausgang 1 *aus* (LOW).

Bemerkung: In der Betriebsart `ExposureAlternateActive` gibt es zu einem Intensitätsbild mit angeschaltetem GPIO-Ausgang 1 (HIGH), d.h. mit projiziertem Muster, immer in 40 ms Abstand ein Intensitätsbild mit ausgeschaltetem GPIO-Ausgang 1 (LOW), d.h. ohne projiziertes Muster. Dies ist unabhängig von der benutzerdefinierten Bildwiederholrate und sollte in dieser speziellen Betriebsart für die Synchronisierung von Disparitäts- und projektionsfreien Kamerabildern berücksichtigt werden.

`out1_ratio` und `out2_ratio` (*High-Anteil Ausgang 1* und *High-Anteil Ausgang 2)

Die High-Anteile für die GPIOs Out 1 und Out 2 bestimmen, wie viel der Bildbelichtungszeit der entsprechende Ausgabe-GPIO HIGH sein soll, wenn ExposureActive oder ExposureAlternateActive verwendet werden. Wenn der Anteil auf 1 eingestellt ist, ist der Ausgang für die gesamte Belichtungszeit HIGH. Falls ein Projektor an den GPIO out1 angeschlossen ist, führt ein niedrigerer High-Anteil an Ausgang 1 (`out1_ratio`) zu dunkleren Projektionen.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_iocontrol/parameters?<out1_
↪ratio|out2_ratio>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/parameters?<out1_ratio|out2_ratio>=<value>
```

`out1_inverted` und `out2_inverted` (*Ausgang 1 invertieren* und *Ausgang 2 invertieren)

Die Parameter *Ausgang 1 invertieren* (`out1_inverted`) und *Ausgang 2 invertieren* (`out2_inverted`) legen fest, ob der zugehörige Ausgang invertiert werden soll.

Über die REST-API kann dieser Parameter wie folgt gesetzt werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_iocontrol/parameters?<out1_
↪inverted|out2_inverted>=<value>
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/parameters?<out1_inverted|out2_inverted>=
↪<value>
```

6.4.4.2 Services

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von 0 quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind.

Dieses Softwaremodul bietet folgende Services.

`get_io_values`

Mit diesem Aufruf kann der aktuelle Zustand der Ein- und Ausgänge (GPIOs) der Kamera abgefragt werden.

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_iocontrol/services/get_io_values
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/services/get_io_values
```

Request

Dieser Service hat keine Argumente.

Response

Das Feld `timestamp` ist der Zeitpunkt der Messung.

`input_mask` und `output_mask` sind Bitmasken, die definieren, welche Bits für die Werte der Eingänge bzw. Ausgänge verwendet werden.

`values` beinhaltet die Werte der Bits, die zu den in den Bitmasken *input_mask‘* und `output_mask` definierten Eingängen und Ausgängen gehören.

Das Feld `return_code` enthält mögliche Warnungen oder Fehlercodes und Nachrichten. Mögliche Werte für `return_code` sind in der Tabelle unten angegeben.

Code	Beschreibung
0	Erfolgreich
-2	Interner Fehler
-9	Lizenz für *IOControl* ist nicht verfügbar

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_io_values",
  "response": {
    "input_mask": "uint32",
    "inverter_mask": "uint32",
    "output_mask": "uint32",
    "ratio_mask": "uint32",
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    },
    "values": "uint32"
  }
}
```

reset_defaults

stellt die Werkseinstellungen der Parameter dieses Moduls wieder her und wendet sie an („factory reset“).

Details

Dieser Service kann wie folgt aufgerufen werden.

API Version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_iocontrol/services/reset_defaults
```

API Version 1 (veraltet)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/services/reset_defaults
```

Request

Dieser Service hat keine Argumente.

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "reset_defaults",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

6.5 Datenbankmodule

Der *rc_reason_stack* stellt mehrere Datenbankmodule zur Verfügung, die das Konfigurieren von globalen Daten ermöglichen, die in vielen Detektionsmodulen benötigt werden, zum Beispiel Load Carrier und Regions of Interest. Über die *REST-API-Schnittstelle* (Abschnitt 7.2) sind die Datenbankmodule nur in API Version 2 verfügbar.

Die Datenbankmodule sind:

- ***LoadCarrierDB*** **(`rc_load_carrier_db`, Abschnitt 6.5.1)** ermöglicht das Erstellen, Abfragen und Löschen von Load Carriern.
- ***RoiDB*** **(`rc_roi_db` , Abschnitt 6.5.2)** ermöglicht das Erstellen, Abfragen und Löschen von 2D und 3d Regions of Interest.
- ***GripperDB*** **(`rc_gripper_db`, Abschnitt 6.5.3)** ermöglicht das Erstellen, Abfragen und Löschen von Greifern für die Kollisionsprüfung.

Diese Softwaremodule sind global auf dem *rc_reason_stack*, was bedeutet, dass sie außerhalb der Kamerapipelines laufen. Änderungen ihrer Einstellungen oder Parameter betreffen alle Pipelines auf dem *rc_reason_stack*.

6.5.1 LoadCarrierDB

6.5.1.1 Einleitung

Das LoadCarrierDB Modul (Load Carrier Datenbank Modul) ermöglicht die globale Definition von Load Carriern (Behältern), die dann in vielen Detektionsmodulen genutzt werden können. Die definierten Load Carrier Modelle sind in allen Modulen auf dem *rc_reason_stack* verfügbar, die Load Carrier unterstützen.

Bemerkung: Dieses Softwaremodul läuft global auf dem *rc_reason_stack*. Änderungen seiner Einstellungen oder Parameter betreffen alle Kamerapipelines, die auf dem *rc_reason_stack* laufen.

Das LoadCarrierDB Modul ist ein Basismodul, welches auf jedem *rc_reason_stack* verfügbar ist.

Tab. 6.59: Spezifikationen des LoadCarrierDB Moduls

Unterstützte Load Carrier Typen	4-seitig oder 3-seitig
Mögliche Rand-Arten	durchgängig, abgestuft oder vorspringend
Min. Load Carrier Abmessungen	0.1 m x 0.1 m x 0.05 m
Max. Load Carrier Abmessungen	5 m x 5 m x 5 m
Max. Anzahl von Load Carriern	50
Load Carrier verfügbar in	*ItemPick und ItemPickAI* (Abschnitt 6.3.4) und *BoxPick* (Abschnitt 6.3.5) und *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6)
Mögliche Posen-Arten	keine Pose, Orientierungsprior, exakte Pose
Mögliche Referenzkoordinatensysteme	camera, external

6.5.1.2 Load Carrier Definition

Ein sogenannter Load Carrier ist ein Behälter mit vier Wänden, einem Boden und einem rechteckigen Rand, der Objekte enthalten kann. Er kann genutzt werden, um das Volumen, in dem nach Objekten oder Greifpunkten gesucht wird, zu begrenzen.

Seine Geometrie ist durch die inneren und äußeren Abmessungen (`inner_dimensions` und `outer_dimensions`) definiert. Die maximalen `outer_dimensions` betragen 5.0 m in allen Dimensionen.

Der Ursprung des Load Carrier Koordinatensystems liegt im Zentrum des durch die *Außenmaße* definierten Quaders. Dabei ist die z-Achse senkrecht zum Boden des Load Carriers und zeigt aus dem Load Carrier heraus (siehe Abb. 6.36).

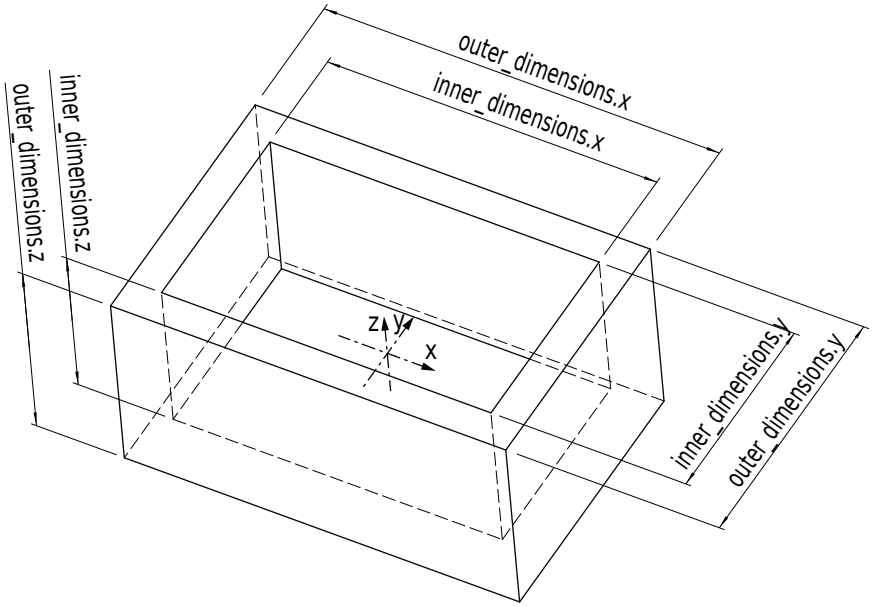


Abb. 6.36: Load Carrier mit Koordinatensystem und inneren und äußeren Abmessungen

Bemerkung: Die Innen- und Außenmaße eines Load Carriers sind typischerweise in den Angaben des jeweiligen Herstellers spezifiziert, und können im Produktblatt oder auf der Produktseite nachgeschlagen werden.

Das Innenvolumen eines Load Carriers ist durch seine Innenmaße definiert, aber enthält zusätzlich einen Bereich von 10 cm oberhalb des Load Carriers, damit Objekte, die aus dem Load Carrier herausragen, auch für die Detektion oder Greifpunktberechnung berücksichtigt werden. Weiterhin wird vom Innenvolumen in jeder Richtung ein zusätzlicher Sicherheitsabstand `crop_distance` abgezogen, welcher als Laufzeitparameter im LoadCarrier Modul konfiguriert werden kann (siehe *Parameter*, Abschnitt 6.3.2.5). Abb. 6.37 zeigt das Innenvolumen eines Load Carriers. Nur Punkte, die sich innerhalb dieses Volumens befinden, werden für Detektionen berücksichtigt.

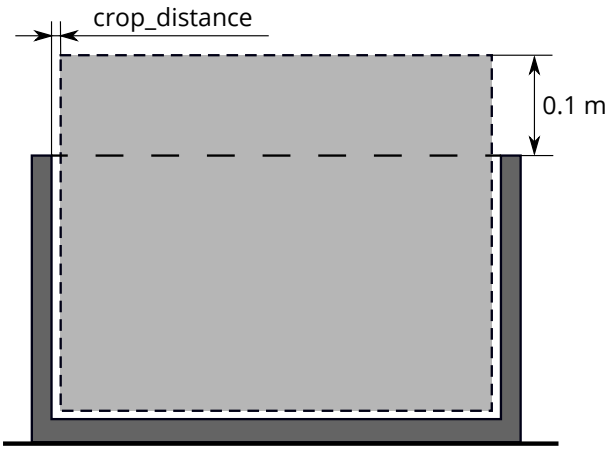


Abb. 6.37: Darstellung des Innenvolumens eines Load Carriers. Nur Punkte, die sich innerhalb dieses Volumens befinden, werden für Detektionen berücksichtigt.

Da die Erkennung von Load Carriern auf der Erkennung des Load Carrier Rands basiert, muss die Geometrie des Randes angegeben werden, wenn sie nicht aus der Differenz zwischen Außen- und Innenmaßen bestimmt werden kann. Dazu kann die Randstärke `rim_thickness` explizit gesetzt werden. Die Randstärke gibt die Breite des äußeren Rands in x- und y-Richtung an. Wenn eine Randstärke gesetzt ist, kann optional auch die Randstufenhöhe `rim_step_height` angegeben werden. Die Randstufenhöhe gibt die Höhe der Stufe zwischen dem äußeren und dem inneren Teil des Load Carrier Rands an. Wenn die Stufenhöhe angegeben wird, wird sie auch bei der Kollisionsprüfung berücksichtigt (siehe *CollisionCheck*, Abschnitt 6.4.2). Beispiele abgestufter Load Carrier sind in Abb. 6.38 A, B gezeigt. Zusätzlich zur Randstärke und Randstufenhöhe kann der Randvorsprung `rim_ledge` angegeben werden, um Load Carrier zu definieren, deren innerer Rand in den Load Carrier Innenraum hineinragt, wie zum Beispiel bei Gitterboxen. Der Randvorsprung `rim_ledge` gibt die Randstärke des inneren Teils des Randes in die x- und y-Richtung an. Ein Beispiel eines Load Carriers mit vorspringendem Rand ist in Abb. 6.38 C gezeigt.

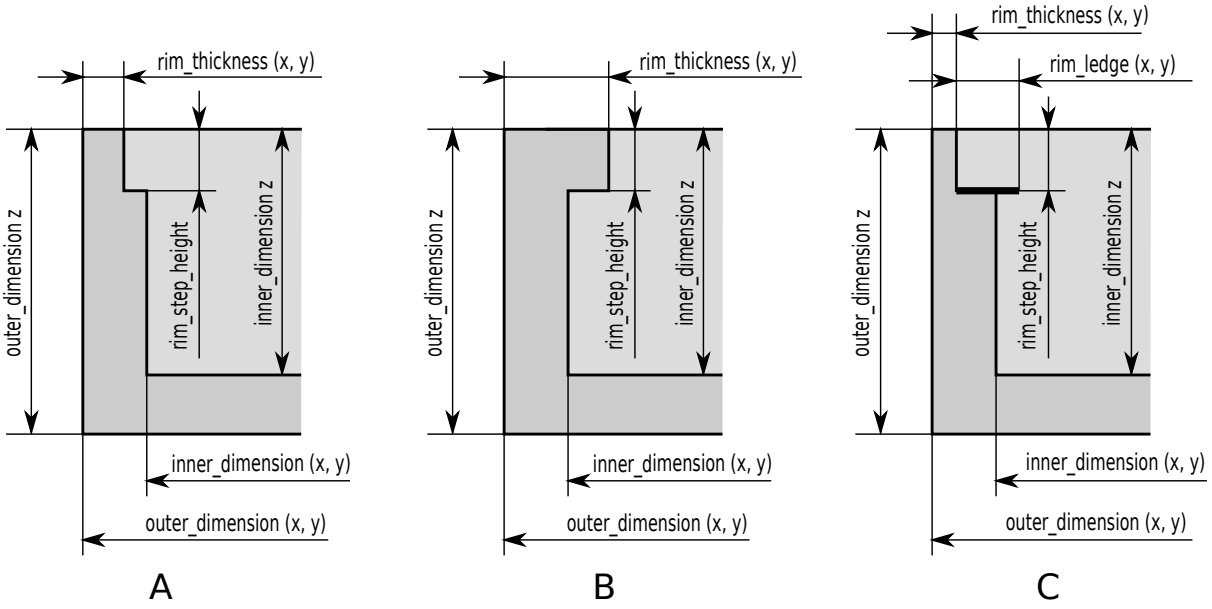


Abb. 6.38: Beispiele von Load Carriern mit abgestuftem (A, B) und vorspringendem Rand (C)

Die unterschiedlichen Randtypen können für gewöhnliche 4-seitige und 3-seitige Load Carrier angewendet werden. Für einen 3-seitigen Load Carrier muss das Feld `type` auf `THREE_SIDED` gesetzt werden. Wenn der Typ `STANDARD` oder leer ist, wird ein 4-seitiger Load Carrier angenommen. Ein 3-seitiger Load

Carrier hat eine Seite, die niedriger ist als die anderen drei Seiten. Die Höhe dieser niedrigeren offenen Seite `height_open_side` wird vom äußeren Boden des Load Carriers gemessen. Die offene Seite liegt an der negativen y-Achse des Load Carrier Koordinatensystems. Beispiele der zwei unterschiedlichen Load Carrier Typen sind in Abb. 6.39 zu sehen. Die Höhe der offenen Seite wird nur während der Kollisionsprüfung berücksichtigt und ist nicht notwendig für die Erkennung des Load Carriers.

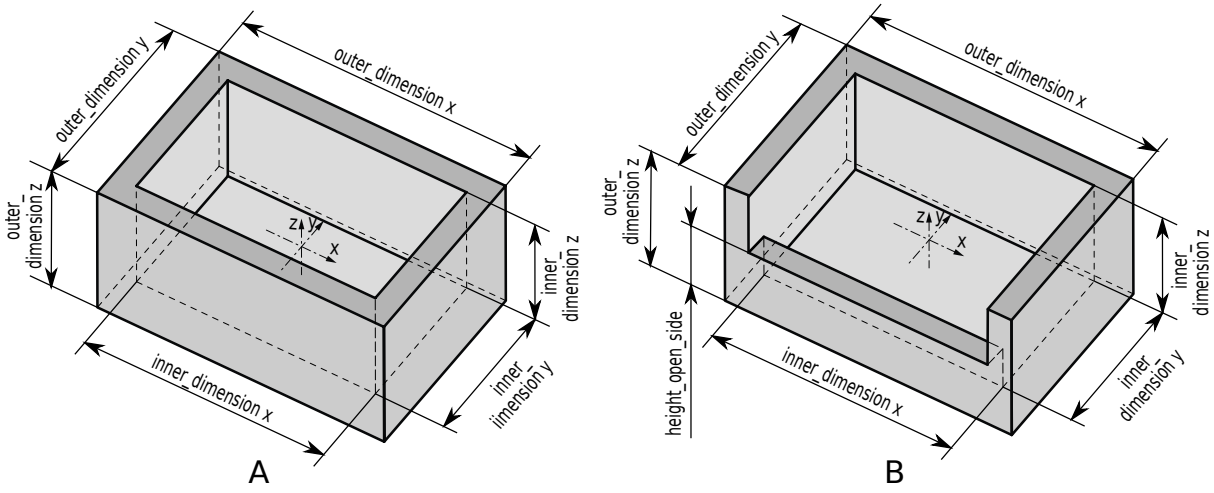


Abb. 6.39: Beispiele eines 4-seitigen (A) und 3-seitigen Load Carriers (B)

Für einen Load Carrier kann eine Pose bestehend aus `position` und `orientation` als Quaternion in einem Referenzkoordinatensystem angegeben werden. Basierend auf dem Posentyp `pose_type` wird diese Pose entweder als Vorgabe für die Load Carrier Orientierung (`pose_type` ist `ORIENTATION_PRIOR` oder leer) oder als exakte Pose (`pose_type` ist `EXACT_POSE`) verwendet.

Falls die angegebene Pose als Vorgabe (Prior) für die Orientierung dient, wird garantiert, dass die zurückgelieferte Pose des erkannten Load Carriers die minimale Rotation bezogen auf den gesetzten Prior hat. Dieser Posentyp ist nützlich für die Erkennung von geneigten Load Carriern, oder um Mehrdeutigkeiten in der x- und y-Richtung aufzulösen, die durch die Symmetrie des Load Carriers verursacht werden.

Falls der Posentyp auf `EXACT_POSE` gesetzt ist, wird keine Load Carrier Erkennung auf den Szenendaten durchgeführt, sondern die angegebene Pose wird so verwendet, als wäre der Load Carrier in dieser Pose in der Szene erkannt worden. Dieser Posentyp ist nützlich, wenn Load Carrier ihre Position nicht verändern und/oder schwer zu erkennen sind (z.B. weil ihr Rand zu schmal ist oder das Material zu stark reflektiert).

Der *rc_reason_stack* erlaubt das Speichern von bis zu 50 verschiedenen Load Carriern, von denen jeder mit einer `id` versehen ist. Die für eine spezifische Anwendung relevanten Load Carrier können mithilfe der *rc_reason_stack* Web GUI oder der *REST-API-Schnittstelle* (Abschnitt 7.2) konfiguriert werden.

Bemerkung: Die konfigurierten Load Carrier sind persistent auf dem *rc_reason_stack* gespeichert und auch nach Firmware-Updates und -Wiederherstellungen verfügbar.

6.5.1.3 Load Carrier Abteile

Bei einigen Detektionsmodulen kann ein Load Carrier Abteil (`load_carrier_compartment`) angegeben werden, um das Volumen für die Erkennung zu begrenzen, zum Beispiel in *ItemPick's compute_grasps Service* (siehe 6.3.4.7). Ein Load Carrier Abteil ist eine Box, deren Pose `pose` als Transformation vom Load Carrier Koordinatensystem in das Abteilkoordinatensystem, welches im Zentrum der durch das Abteil definierten Box liegt, angegeben wird (siehe Abb. 6.40). Das Load Carrier Abteil ist nicht Teil der Load Carrier Definition im LoadCarrierDB Modul, sondern muss für jeden Detektionsaufruf separat definiert werden.

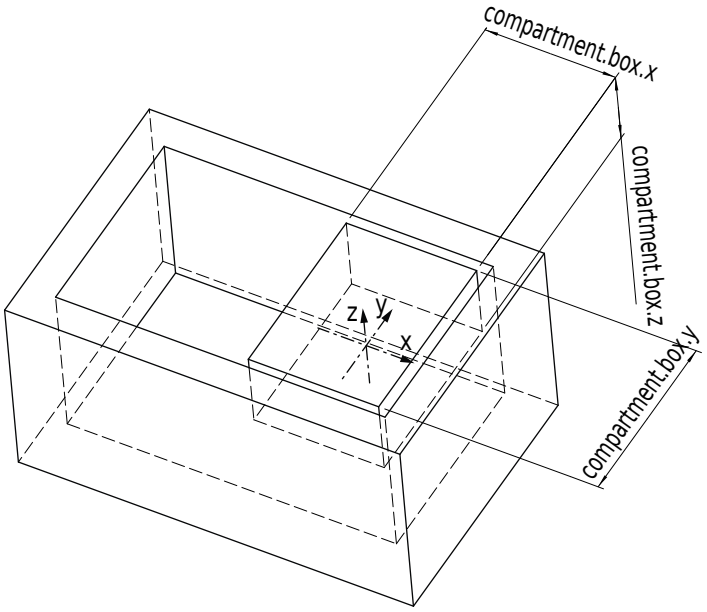


Abb. 6.40: Beispiel für ein Abteil innerhalb eines Load Carriers. Das gezeigte Koordinatensystem das Koordinatensystem des Load Carrier Abteils.

Als Volumen für die Detektion wird der Durchschnitt des Abteil-Volumens und des Load Carrier Innenraums verwendet. Wenn dieser Durchschnitt ebenfalls den Bereich von 10 cm oberhalb des Load Carriers enthalten soll, muss die Höhe der Box, die das Abteil definiert, entsprechend vergrößert werden.

6.5.1.4 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_reason_stack* laufenden Module liefern Daten für das LoadCarrierDB Modul oder haben Einfluss auf die Datenverarbeitung.

Hand-Auge-Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann die exakte Pose oder der Orientierungsprior im Roboterkoordinatensystem angegeben werden, indem das Argument `pose_frame` auf `external` gesetzt wird.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Die Load Carrier Pose oder der Orientierungsprior sind im Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Die Load Carrier Pose oder der Orientierungsprior sind im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1).

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

6.5.1.5 Services

Das LoadCarrierDB Modul wird in der REST-API als `rc_load_carrier_db` bezeichnet und in der *Web GUI* (Abschnitt 7.1) unter *Datenbank* → *Load Carrier* dargestellt. Die angebotenen Services des LoadCarrierDB Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der Web GUI ausprobiert und getestet werden.

Das LoadCarrierDB Modul stellt folgende Services zur Verfügung.

`set_load_carrier`

speichert einen Load Carrier auf dem *rc_reason_stack*. Alle Load Carrier sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_load_carrier_db/services/set_load_carrier
```

Request

Die Definition des Typs `load_carrier` wird in *Load Carrier Definition* (Abschnitt 6.5.1.2) beschrieben.

Das Feld `type` ist optional und akzeptiert `STANDARD` und `THREE_SIDED`.

Das Feld `pose_type` ist optional und akzeptiert `NO_POSE`, `EXACT_POSE` und `ORIENTATION_PRIOR`.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "load_carrier": {
      "height_open_side": "float64",
      "id": "string",
      "inner_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "outer_dimensions": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "pose_type": "string",
      "rim_ledge": {
        "x": "float64",
        "y": "float64"
      },
      "rim_step_height": "float64",
      "rim_thickness": {
        "x": "float64",
        "y": "float64"
      },
      "type": "string"
    }
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_load_carrier",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


get_load_carriers

gibt die mit `load_carrier_ids` spezifizierten, gespeicherten Load Carrier zurück. Wenn keine `load_carrier_ids` angegeben werden, werden alle gespeicherten Load Carrier zurückgeliefert.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_load_carrier_db/services/get_load_carriers
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "load_carrier_ids": [
      "string"
    ]
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_load_carriers",
  "response": {
    "load_carriers": [
      {
        "height_open_side": "float64",
        "id": "string",
        "inner_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "outer_dimensions": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "pose_type": "string",
        "rim_ledge": {
          "x": "float64",
          "y": "float64"
        },
        "rim_step_height": "float64",
        "rim_thickness": {
          "x": "float64",
          "y": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


delete_load_carriers

löscht die mit `load_carrier_ids` spezifizierten, gespeicherten Load Carrier. Alle zu löschenden Load Carrier müssen explizit in `load_carrier_ids` angegeben werden.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_load_carrier_db/services/delete_load_carriers
```

Request

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "load_carrier_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_load_carriers",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.5.1.6 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabecodes auf:

Tab. 6.60: Rückgabecodes der Services des LoadCarrierDB Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Load Carriern überschritten wurde.
10	Die maximal speicherbare Anzahl an Load Carriern wurde erreicht.
11	Mit dem Aufruf von `set_load_carrier` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.

6.5.2 RoiDB

6.5.2.1 Einleitung

Das RoiDB Modul (Region of Interest Datenbankmodul) ermöglicht die globale Definition von 2D und 3D Regions of Interest (ROIs), die dann in vielen Detektionsmodulen verwendet werden können. Die

definierten ROIs sind in allen Modulen auf dem *rc_reason_stack* verfügbar, die 2D oder 3D ROIs unterstützen.

Bemerkung: Dieses Softwaremodul läuft global auf dem *rc_reason_stack*. Änderungen seiner Einstellungen oder Parameter betreffen alle Kamerapipelines, die auf dem *rc_reason_stack* laufen.

Das RoiDB Modul ist ein Basismodul, welches auf jedem *rc_reason_stack* verfügbar ist.

3D ROIs können in den *CADMatch* (Abschnitt 6.3.7), *ItemPick und ItemPickAI* (Abschnitt 6.3.4) und *BoxPick* (Abschnitt 6.3.5) Modulen verwendet werden. 2D ROIs werden vom *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6) Modul und dem *LoadCarrier* (Abschnitt 6.3.2) Modul unterstützt.

Tab. 6.61: Spezifikationen des ROIDB Moduls

Unterstützte ROI Typen	2D, 3D
Unterstützte ROI Geometrien	2D ROI: Rechteck, 3D ROI: Box, Kugel
Max. Anzahl von ROIs	2D: 100, 3D: 100
ROIs verfügbar in	2D: *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6), *LoadCarrier* (Abschnitt 6.3.2), 3D: *CADMatch* (Abschnitt 6.3.7), *ItemPick und ItemPickAI* (Abschnitt 6.3.4) und *BoxPick* (Abschnitt 6.3.5)
Unterstützte Referenzkoordinatensysteme	camera, external

6.5.2.2 Region of Interest

Eine sogenannte Region of Interest (ROI) definiert ein abgegrenztes Raumvolumen (3D ROI, `region_of_interest`) oder eine rechteckige Region im linken Kamerabild (2D ROI, `region_of_interest_2d`), welche für eine spezifische Anwendung relevant sind.

Eine ROI kann das Volumen, in dem ein Load Carrier gesucht wird, einschränken, oder einen Bereich definieren, der nur die zu erkennenden oder zu greifenden Objekte enthält. Die Verarbeitungszeit kann sich deutlich verringern, wenn eine ROI genutzt wird.

Folgende Arten von 3D ROIs (`type`) werden unterstützt:

- `BOX`, für quaderförmige ROIs mit den Abmessungen `box.x`, `box.y`, `box.z`.
- `SPHERE`, für kugelförmige ROIs mit dem Radius `sphere.radius`.

Die Pose `pose` einer 3D ROI kann entweder relativ zum *Kamera*-Koordinatensystem `camera` oder mithilfe der Hand-Auge-Kalibrierung im *externen* Koordinatensystem `external` angegeben werden. Das externe Koordinatensystem steht nur zur Verfügung, wenn eine *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1) durchgeführt wurde. Wenn der Sensor am Roboter montiert ist, und die ROI im externen Koordinatensystem definiert ist, dann muss jedem Detektions-Service, der diese ROI benutzt, die aktuelle Roboterpose übergeben werden.

Eine 2D ROI ist als rechteckiger Teil des linken Kamerabilds definiert und kann sowohl über die *REST-API-Schnittstelle* (Abschnitt 7.2) als auch über die *rc_reason_stack Web GUI* (Abschnitt 7.1) auf der Seite *Regions of Interest* unter dem Menüpunkt *Datenbank* gesetzt werden. Die Web GUI bietet hierfür ein einfach zu benutzendes Werkzeug an. Jeder ROI muss ein eindeutiger Name zugewiesen werden, um diese später adressieren zu können.

In der REST-API ist eine 2D-ROI über folgende Werte spezifiziert:

- `id`: Eindeutiger Name der ROI
- `offset_x`, `offset_y`: Abstand in Pixeln von der oberen rechten Bildecke entlang der x- bzw. y-Achse
- `width`, `height`: Breite und Höhe in Pixeln

Der *rc_reason_stack* erlaubt das Speichern von bis zu 100 verschiedenen 3D ROIs und der gleichen Anzahl von 2D ROIs. Die Konfiguration von ROIs erfolgt in der Regel offline während der Einrichtung der gewünschten Anwendung. Die Konfiguration kann mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) des RoiDB Moduls vorgenommen werden, oder über die *rc_reason_stack Web GUI* (Abschnitt 7.1) auf der Seite *Regions of Interest* unter dem Menüpunkt *Datenbank*.

Bemerkung: Die erstellten ROIs sind persistent auf dem *rc_reason_stack* gespeichert und auch nach Firmware-Updates und -Wiederherstellungen verfügbar.

6.5.2.3 Wechselwirkung mit anderen Modulen

Die folgenden, intern auf dem *rc_reason_stack* laufenden Module liefern Daten für das RoiDB Modul oder haben Einfluss auf die Datenverarbeitung.

Hand-Auge Kalibrierung

Falls die Kamera zu einem Roboter kalibriert wurde, kann die Pose einer 3D ROI im Roboterkoordinatensystem angegeben werden, indem das Argument `pose_frame` auf `external` gesetzt wird.

Zwei verschiedene Werte für `pose_frame` können gewählt werden:

1. **Kamera-Koordinatensystem** (`camera`): Die Pose der 3D Region of Interest ist Kamera-Koordinatensystem angegeben und es ist kein zusätzliches Wissen über die Lage der Kamera in seiner Umgebung notwendig. Das bedeutet insbesondere, dass sich ROIs oder Load Carrier, welche in diesem Koordinatensystem angegeben sind, mit der Kamera bewegen. Es liegt daher in der Verantwortung des Anwenders, in solchen Fällen die entsprechenden Posen der Situation entsprechend zu aktualisieren (beispielsweise für den Anwendungsfall einer robotergeführten Kamera).
2. **Benutzerdefiniertes externes Koordinatensystem** (`external`): Die Pose der 3D Region of Interest ist im sogenannten externen Koordinatensystem angegeben, welches vom Nutzer während der Hand-Auge-Kalibrierung gewählt wurde. In diesem Fall bezieht das Modul alle notwendigen Informationen über die Kameramontage und die kalibrierte Hand-Auge-Transformation automatisch vom Modul *Hand-Auge-Kalibrierung* (Abschnitt 6.4.1).

Bemerkung: Wenn keine Hand-Auge-Kalibrierung durchgeführt wurde bzw. zur Verfügung steht, muss als Referenzkoordinatensystem `pose_frame` immer `camera` angegeben werden.

Zulässige Werte zur Angabe des Referenzkoordinatensystems sind `camera` und `external`. Andere Werte werden als ungültig zurückgewiesen.

6.5.2.4 Services

Das RoiDB Modul wird in der REST-API als `rc_roi_db` bezeichnet und in der *Web GUI* (Abschnitt 7.1) unter *Datenbank* → *Regions of Interest* dargestellt. Die angebotenen Services des RoiDB Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der Web GUI ausprobiert und getestet werden.

Das RoiDB Modul stellt folgende Services zur Verfügung.

set_region_of_interest

speichert eine 3D ROI auf dem *rc_reason_stack*. Alle ROIs sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/set_region_of_interest
```

Request

Die Definition des Typs `region_of_interest` wird in *Region of Interest* (Abschnitt 6.5.2.2) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "region_of_interest": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "sphere": {
        "radius": "float64"
      },
      "type": "string"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_region_of_interest",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`set_region_of_interest_2d`

speichert eine 2D ROI auf dem *rc_reason_stack*. Alle ROIs sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/set_region_of_interest_2d
```

Request

Die Definition des Typs `region_of_interest_2d` wird in *Region of Interest* (Abschnitt 6.5.2.2) beschrieben.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "region_of_interest_2d": {
      "height": "uint32",
      "id": "string",
      "offset_x": "uint32",
      "offset_y": "uint32",
      "width": "uint32"
    }
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "set_region_of_interest_2d",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

`get_regions_of_interest`

gibt die mit `region_of_interest_ids` spezifizierten, gespeicherten 3D ROIs zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/get_regions_of_interest
```

Request

Werden keine `region_of_interest_ids` angegeben, enthält die Serviceantwort alle gespeicherten ROIs.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "region_of_interest_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "get_regions_of_interest",
  "response": {
    "regions_of_interest": [
      {
        "box": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "sphere": {
          "radius": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


get_regions_of_interest_2d

gibt die mit `region_of_interest_2d_ids` spezifizierten, gespeicherten 2D ROIs zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/get_regions_of_interest_2d
```

Request

Werden keine `region_of_interest_2d_ids` angegeben, enthält die Serviceantwort alle gespeicherten ROIs.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "region_of_interest_2d_ids": [
      "string"
    ]
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_regions_of_interest_2d",
  "response": {
    "regions_of_interest": [
      {
        "height": "uint32",
        "id": "string",
        "offset_x": "uint32",
        "offset_y": "uint32",
        "width": "uint32"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

delete_regions_of_interest

löscht die mit `region_of_interest_ids` spezifizierten, gespeicherten 3D ROIs.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/delete_regions_of_interest
```

Request

Alle zu löschenden ROIs müssen explizit angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "region_of_interest_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_regions_of_interest",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
}
```

delete_regions_of_interest_2d

löscht die mit `region_of_interest_2d_ids` spezifizierten, gespeicherten 2D ROIs.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/delete_regions_of_interest_2d
```

Request

Alle zu löschenden ROIs müssen explizit angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "region_of_interest_2d_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "delete_regions_of_interest_2d",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.5.2.5 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabe-Codes auf:

Tab. 6.62: Rückgabe-Codes der Services des RoiDB Moduls

Code	Beschreibung
0	Erfolgreich
-1	Ungültige(s) Argument(e)
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an ROIs überschritten wurde.
10	Die maximal speicherbare Anzahl an ROIs wurde erreicht.
11	Mit dem Aufruf von `set_region_of_interest` oder `set_region_of_interest_2d` wurde ein bereits existierendes Objekt mit derselben `id` überschrieben.

6.5.3 GripperDB

6.5.3.1 Einleitung

Das GripperDB Modul ist ein optionales Modul, welches intern auf dem *rc_reason_stack* läuft, und ist freigeschaltet, sobald eine gültige Lizenz für eines der Module *ItemPick und ItemPickAI* (Abschnitt 6.3.4) und *BoxPick* (Abschnitt 6.3.5) oder *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6) vorhanden ist. Andernfalls benötigt dieses Modul eine separate *Lizenz* (Abschnitt 8.2).

Das Modul bietet Services zum Anlegen, Abfragen und Löschen von Greifern, die dann für die Kollisionsprüfung mit einem Load Carrier oder anderen erkannten Objekten (nur in Kombination mit *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6)) genutzt werden können. Die angelegten Greifer sind in allen Modulen auf dem *rc_reason_stack* verfügbar, die eine Kollisionsprüfung anbieten.

Bemerkung: Dieses Softwaremodul läuft global auf dem *rc_reason_stack*. Änderungen seiner Einstellungen oder Parameter betreffen alle Kamerapipelines, die auf dem *rc_reason_stack* laufen.

Tab. 6.63: Spezifikationen des GripperDB Moduls

Max. Anzahl Greifer	50
Mögliche Greiferelement-Geometrien	Box, Zylinder, CAD-Element
Max. Anzahl Elemente pro Greifer	15
Kollisionsprüfung verfügbar in	*ItemPick und ItemPickAI* (Abschnitt 6.3.4) und *BoxPick* (Abschnitt 6.3.5), *CADMatch* (Abschnitt 6.3.7) und *SilhouetteMatch und SilhouetteMatchAI* (Abschnitt 6.3.6)

6.5.3.2 Erstellen eines Greifers

Der Greifer ist eine Kollisionsgeometrie, die zur Prüfung auf Kollisionen zwischen dem geplanten Griff und dem Load Carrier verwendet wird. Der Greifer kann aus bis zu 15 miteinander verbundenen Elementen bestehen.

Es sind folgende Arten von Elementen möglich:

- Quader (`BOX`), mit den Abmessungen `box.x`, `box.y`, `box.z`.
- Zylinder (`CYLINDER`), mit dem Radius `cylinder.radius` und der Höhe `cylinder.height`.
- CAD-Element (`CAD`), mit der ID `cad.id` des gewählten CAD-Elements.

Jedem Greiferelement kann einer der folgenden Werte für `function_type` zugewiesen werden:

- `NONE`: Standardwert, entspricht einem leeren Eintrag. Dieses Element hat keine spezielle Funktion und wird bei der Kollisionsprüfung wie modelliert berücksichtigt.

- `FINGER`: Dieses Element ist ein beweglicher Finger oder eine Greiferbacke und besitzt neben seiner Standardpose `pose` eine Nullposition (`zero_pose`). Es kann sich linear von der Nullposition so weit in Richtung der Standardposition bewegen, wie der für jeden Griff definierte Hub (stroke) vorgibt.
- `SUCTION_CUP`: Dieses Element ist ein verformbarer Saugnapf und wird daher bei der Kollisionsprüfung ignoriert. Es dient lediglich der Visualisierung.

Weiterhin müssen für jeden Greifer der Flanschradius und der Tool Center Point (TCP) definiert werden.

Die Konfiguration des Greifers wird in der Regel während des Setups der Zielanwendung durchgeführt. Das kann über die *REST-API-Schnittstelle* (Abschnitt 7.2) oder die *rc_reason_stack Web GUI* (Abschnitt 7.1) geschehen.

Flanschradius

Es werden standardmäßig nur Kollisionen mit dem Greifer, nicht aber mit der Robotergeometrie geprüft. Um Kollisionen zwischen dem Load Carrier und dem Roboter zu vermeiden, kann über den Laufzeitparameter `check_flange` im CollisionCheck Modul (siehe *Übersicht der Parameter*, Abschnitt 6.4.2.3) ein zusätzlicher optionaler Test aktiviert werden. Dieser Test erkennt alle Griffe als Kollisionen, bei denen sich ein Teil des Roboterflanschs innerhalb des Load Carriers befinden würde (siehe Abb. 6.41). Der Test basiert auf der Greifergeometrie und dem Flanschradius.

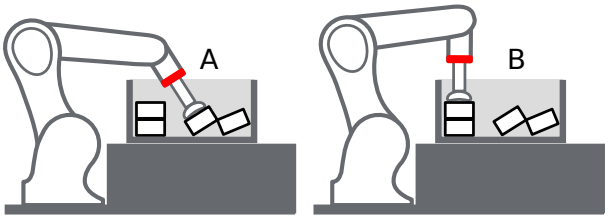


Abb. 6.41: Fall A: Der Griff wird nur als Kollision erkannt, wenn `check_flange` auf *true* gesetzt ist, denn der Flansch (rot) befindet sich im Load Carrier. Fall B: Der Griff ist in jedem Fall kollisionsfrei.

Hochladen von CAD-Greiferelementen

Ein Greifer kann aus Boxen, Zylindern und CAD-Elementen bestehen. Während Boxen und Zylinder während der Erstellung eines Greifers parametrisiert werden können, müssen CAD-Elemente im Vorfeld hochgeladen werden, um für die Greifererstellung verfügbar zu sein. Ein CAD-Element kann über die *REST-API-Schnittstelle* (Abschnitt 7.2) wie in Abschnitt *CAD-Greiferelement API* (Abschnitt 6.5.3.5) beschrieben, oder über die *rc_reason_stack Web GUI* (Abschnitt 7.1) hochgeladen werden. Unterstützte Dateiformate sind STEP (*.stp, *.step), STL (*.stl), OBJ (*.obj) und PLY (*.ply). Die maximal hochzuladende Dateigröße ist auf 30 MB begrenzt. Die Dateien werden intern in PLY konvertiert und, falls nötig, vereinfacht. Die CAD-Elemente können dann während der Greifererstellung über ihre ID referenziert werden.

Erstellen eines Greifers über die REST-API oder die Web GUI

Bei der Greifererstellung über die *REST-API-Schnittstelle* (Abschnitt 7.2) oder die *Web GUI* (Abschnitt 7.1) hat jedes Greifer-Element ein *Parent*-Element, das die Verbindung zwischen den Elementen definiert. Der Greifer wird immer vom Roboterflansch ausgehend in Richtung TCP aufgebaut, und mindestens ein Element muss den Parent ‚flange' (Flansch) haben. Die IDs der Elemente müssen eindeutig sein und dürfen nicht ‚tcp' oder ‚flange' sein. Die Pose des Elements muss im Koordinatensystem des *Parent*-Elements angegeben werden. Das Koordinatensystem eines Elements vom Typ `CYLINDER` oder `BOX` befindet sich genau in seinem geometrischen Mittelpunkt. Damit ein Element also genau unterhalb seines *Parent*-Elements platziert wird, muss seine Position aus der Höhe des *Parent*-Elements und seiner eigenen Höhe berechnet werden (siehe Abb. 6.42).

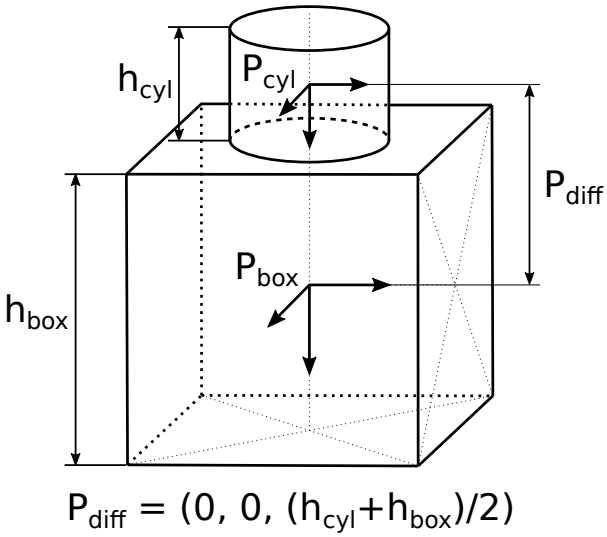


Abb. 6.42: Bezugskoordinatensysteme für das Erstellen von Greifern über die REST-API und die Web GUI

Im Falle eines CAD-Greiferelements wird der Ursprung durch die CAD-Daten bestimmt und befindet sich nicht notwendigerweise im Mittelpunkt der Bounding Box des Elements.

Es wird empfohlen Greifer über die Web GUI zu erstellen, da diese eine 3D Visualisierung der Greifergeometrie bietet und das automatische Anheften von Kind-Element an ihre Parent-Elemente ermöglicht, indem die entsprechende Option für dieses Element aktiviert wird. In diesem Fall bleiben Elemente an ihren Parent angeheftet, auch wenn sich ihre Größen ändern. Bei CAD-Greiferelementen wird die Bounding Box des Elements als Referenz verwendet. Das automatische Anheften ist nur möglich, wenn das Kind-Element in Bezug auf seinen Parent nicht um die x- oder y-Achse rotiert ist.

Das Bezugskoordinatensystem für das erste Element liegt immer im Mittelpunkt des Roboterflanschs, wobei die z-Achse nach unten gerichtet ist. Es können Greifer mit einer Baumstruktur erstellt werden, bei denen mehrere Elemente dasselbe *Parent*-Element haben, solange alle Elemente miteinander verbunden sind.

Berechnete TCP-Position

Nach dem Erstellen des Greifers mit dem Service `set_gripper` wird die TCP-Position im Flanschkoordinatensystem berechnet und als `tcp_pose_flange` zurückgegeben. Dieser Wert muss mit den tatsächlichen TCP-Koordinaten des Roboters übereinstimmen. Wenn ein Greifer über die Web GUI erstellt wird, wird die aktuelle TCP-Position zu jeder Zeit in der 3D-Visualisierung angezeigt.

Nicht-rotationssymmetrische Greifer erstellen

Bei Greifern, die nicht rotationssymmetrisch um die z-Achse sind, muss sichergestellt werden, dass der Greifer so montiert wird, dass seine Ausrichtung mit der im GripperDB-Modul gespeicherten Darstellung übereinstimmt.

6.5.3.3 Services

Das GripperDB Modul wird in der REST-API als `rc_gripper_db` bezeichnet und in der *Web GUI* (Abschnitt 7.1) unter *Datenbank* → *Greifer* dargestellt. Die angebotenen Services des GripperDB Moduls können mithilfe der *REST-API-Schnittstelle* (Abschnitt 7.2) oder der Web GUI ausprobiert und getestet werden.

Das GripperDB Modul stellt folgende Services zur Verfügung.

set_gripper

konfiguriert und speichert einen Greifer auf dem *rc_reason_stack*. Alle Greifer sind dauerhaft gespeichert, auch über Firmware-Updates und -Wiederherstellungen hinweg.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_gripper_db/services/set_gripper
```

Request

Obligatorische Serviceargumente:

`elements`: Liste von geometrischen Elementen, aus denen der Greifer besteht. Jedes Element muss den `type` ‚CYLINDER' oder ‚BOX' mit den zugehörigen Dimensionen im Feld `cylinder` bzw. `box`, oder den Typ ‚CAD' haben, wobei die entsprechende ID unter `id` im Feld `cad` angegeben werden muss. Die Pose jedes Elements muss im *Parent*-Koordinatensystem angegeben werden (siehe *Erstellen eines Greifers*, Abschnitt 6.5.3.2). Die `id` des Elements muss eindeutig sein und darf nicht ‚tcp' oder ‚flange' sein. Die `parent_id` ist die ID des *Parent*-Elements, welche entweder ‚flange' ist oder der ID eines anderen Elements entsprechen muss. Jedes Element kann einen `function_type` haben, der entweder `NONE`, `FINGER` oder `SUCTION_CUP` ist. Elemente vom Typ `FINGER` benötigen zusätzlich eine `zero_pose`, deren Orientierung mit der in der `pose` des Elements übereinstimmen muss. Elemente vom Typ `SUCTION_CUP` können keine Kind-Elemente haben.

`flange_radius`: Flanschradius der benutzt wird, falls der Parameter `check_flange` aktiviert ist.

`id`: Eindeutiger Name des Greifers.

`tcp_parent_id`: ID des Elements, auf dem der TCP definiert ist.

`tcp_pose_parent`: Die Pose des TCP im Koordinatensystem des Elements, das in `tcp_parent_id` angegeben ist.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "elements": [
      {
        "box": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "cad": {
          "id": "string"
        },
        "cylinder": {
          "height": "float64",
          "radius": "float64"
        },
        "id": "string",
        "parent_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "type": "string"
      }
    ],
    "flange_radius": "float64",
    "id": "string",
    "tcp_parent_id": "string",
    "tcp_pose_parent": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```


Response

`gripper`: Gibt den Greifer mit dem zusätzlichen Feld `tcp_pose_flange` zurück. Dieses Feld gibt die TCP-Koordinaten im Flanschkoordinatensystem an, um diese mit den Roboter-TCP-Koordinaten vergleichen zu können.

`return_code`: enthält mögliche Warnungen oder Fehlercodes und Nachrichten.

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "set_gripper",
  "response": {
    "gripper": {
      "elements": [
        {
          "box": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "cad": {
            "id": "string"
          },
          "cylinder": {
            "height": "float64",
            "radius": "float64"
          },
          "id": "string",
          "parent_id": "string",
          "pose": {
            "orientation": {
              "w": "float64",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "position": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            }
          },
          "type": "string"
        }
      ],
      "flange_radius": "float64",
      "id": "string",
      "tcp_parent_id": "string",
      "tcp_pose_flange": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "tcp_pose_parent": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "type": "string"
    },
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


get_grippers

gibt die mit `gripper_ids` spezifizierten und gespeicherten Greifer zurück.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_gripper_db/services/get_grippers
```

Request

Wenn keine `gripper_ids` angegeben werden, enthält die Serviceantwort alle gespeicherten Greifer.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:

```
{
  "args": {
    "gripper_ids": [
      "string"
    ]
  }
}
```

Response

Die Definition der *Response* mit jeweiligen Datentypen ist:

```
{
  "name": "get_grippers",
  "response": {
    "grippers": [
      {
        "elements": [
          {
            "box": {
              "x": "float64",
              "y": "float64",
              "z": "float64"
            },
            "cad": {
              "id": "string"
            },
            "cylinder": {
              "height": "float64",
              "radius": "float64"
            },
            "id": "string",
            "parent_id": "string",
            "pose": {
              "orientation": {
                "w": "float64",
                "x": "float64",
                "y": "float64",
                "z": "float64"
              },
              "position": {
                "x": "float64",
                "y": "float64",
                "z": "float64"
              }
            },
            "type": "string"
          }
        ],
        "flange_radius": "float64",
        "id": "string",
        "tcp_parent_id": "string",
        "tcp_pose_flange": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "tcp_pose_parent": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


delete_grippers

löscht die mit `gripper_ids` spezifizierten, gespeicherten Greifer.

Details

Dieser Service kann wie folgt aufgerufen werden.

```
PUT http://<host>/api/v2/nodes/rc_gripper_db/services/delete_grippers
```

Request

Alle zu löschenden Greifer müssen explizit angegeben werden.

Die Definition der *Request*-Argumente mit jeweiligen Datentypen ist:


```
{
  "args": {
    "gripper_ids": [
      "string"
    ]
  }
}
```


Response

Die Definition der *Response* mit jeweiligen Datentypen ist:


```
{
  "name": "delete_grippers",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


6.5.3.4 Rückgabecodes

Zusätzlich zur eigentlichen Serviceantwort gibt jeder Service einen sogenannten `return_code` bestehend aus einem Integer-Wert und einer optionalen Textnachricht zurück. Erfolgreiche Service-Anfragen werden mit einem Wert von `0` quittiert. Positive Werte bedeuten, dass die Service-Anfrage zwar erfolgreich bearbeitet wurde, aber zusätzliche Informationen zur Verfügung stehen. Negative Werte bedeuten, dass Fehler aufgetreten sind. Für den Fall, dass mehrere Rückgabewerte zutreffend wären, wird der kleinste zurückgegeben, und die entsprechenden Textnachrichten werden in `return_code.message` akkumuliert.

Die folgende Tabelle listet die möglichen Rückgabecodes auf:

Tab. 6.64: Rückgabecodes der GripperDB Services

Code	Beschreibung
0	Erfolgreich
-1	Ein ungültiges Argument wurde übergeben.
-7	Daten konnten nicht in den persistenten Speicher geschrieben oder vom persistenten Speicher gelesen werden.
-9	Lizenz für CollisionCheck ist nicht verfügbar.
-10	Das neue Element konnte nicht hinzugefügt werden, da die maximal speicherbare Anzahl an Greifern überschritten wurde.
10	Die maximal speicherbare Anzahl an Greifern wurde erreicht.
11	Bestehender Greifer wurde überschrieben.

6.5.3.5 CAD-Greiferelement API

Für den Upload, Download, das Auflisten und Löschen von CAD-Greiferelementen werden spezielle REST-API-Endpunkte zur Verfügung gestellt. CAD-Greiferelemente können auch über die Web GUI hoch- und runtergeladen werden. Bis zu 50 CAD-Greiferelemente können gleichzeitig auf dem *rc_reason_stack* gespeichert werden.

Die maximal hochzuladende Dateigröße ist auf 30 MB begrenzt.

`GET /cad/gripper_elements`
listet alle CAD-Greiferelemente auf.

Musteranfrage

```
GET /api/v2/cad/gripper_elements HTTP/1.1
```

Musterantwort


```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "string"
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  }
]
```

Antwort-Header

- Content-Type – application/json application/ubjson

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: Array von GripperElement)*
- 404 Not Found – Element nicht gefunden

Referenzierte Datenmodelle

- *GripperElement* (Abschnitt 7.2.3)

`GET /cad/gripper_elements/{id}`
ruft ein CAD-Greiferelement ab. Falls der angefragte Content-Typ application/octet-stream ist, wird das Element als Datei zurückgegeben.

Musteranfrage

```
GET /api/v2/cad/gripper_elements/<id> HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **`id`** (*`string`*) – ID des Elements *(obligatorisch)*

Antwort-Header

- Content-Type – application/json application/ubjson application/octet-stream

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: GripperElement)*
- 404 Not Found – Element nicht gefunden

Referenzierte Datenmodelle

- *GripperElement* (Abschnitt 7.2.3)

`PUT /cad/gripper_elements/{id}`
erstellt oder aktualisiert ein CAD-Greiferelement.

Musteranfrage

```
PUT /api/v2/cad/gripper_elements/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameter

- **id** (*string*) – ID des Elements *(obligatorisch)*

Formularparameter

- **file** – CAD-Datei *(obligatorisch)*

Anfrage-Header

- Accept – multipart/form-data application/json

Antwort-Header

- Content-Type – application/json application/ubjson

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabewert: GripperElement)*
- 400 Bad Request – CAD ist ungültig oder die maximale Zahl an Elementen wurde erreicht.
- 404 Not Found – Element nicht gefunden
- 413 Request Entity Too Large – Datei zu groß

Referenzierte Datenmodelle

- *GripperElement* (Abschnitt 7.2.3)

DELETE /cad/gripper_elements/{id}

entfernt ein CAD-Greiferelement.

Musteranfrage

```
DELETE /api/v2/cad/gripper_elements/<id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameter

- **id** (*string*) – ID des Elements *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuswerte

- 200 OK – Erfolgreiche Verarbeitung
- 404 Not Found – Element nicht gefunden

7 Schnittstellen

Es stehen die folgenden Schnittstellen zur Konfiguration und Datenübertragung des *rc_reason_stack* zur Verfügung:

- *Web GUI* (Abschnitt 7.1)

  Leicht zu bedienendes grafisches Interface zum Konfigurieren und Kalibrieren des *rc_reason_stack*, zum Anzeigen von Livebildern, Aufrufen von Services, Visualisieren von Ergebnissen, usw.

- *REST-API-Schnittstelle* (Abschnitt 7.2)

  Programmierschnittstelle zur Konfiguration des *rc_reason_stack*, zur Abfrage von Statusinformationen, zum Anfordern von Datenströmen, usw.

- *Generic Robot Interface* (Abschnitt 7.3)

  TCP-Socketkommunikationsschnittstelle zur Konfiguration des *rc_reason_stack* und für Serviceaufrufe.

- *OPC UA Interface* (Abschnitt 7.4)

  OPC UA Schnittstelle zur Konfiguration des *rc_reason_stack* und Ausführen von Service-Anfragen.

- *KUKA Ethernet KRL Schnittstelle* (Abschnitt 7.5)

  API zum Konfigurieren des *rc_reason_stack* und Ausführen von Service-Anfrage von KUKA KSS Robotern aus.

- *gRPC Bilddatenschnittstelle* (Abschnitt 7.6)

  Synchronisierte Bilddaten per gRPC.

7.1 Web GUI

Die Web GUI des *rc_reason_stack* dient dazu, das Gerät zu testen, zu kalibrieren und zu konfigurieren.

7.1.1 Zugriff auf die Web GUI

Auf die Web GUI des *rc_reason_stack* kann von jedem Webbrowser aus zugegriffen werden, z.B. Firefox, Google Chrome oder Microsoft Edge, indem man die IP-Adresse des Host-PCs mit dem Port 8080 aufruft:

```
http://<host-ip>:8080/
```

7.1.2 Kennenlernen der Web GUI

Die Dashboard-Seite der Web GUI enthält die wichtigsten Informationen über das Gerät und die laufenden Kamerapipelines.

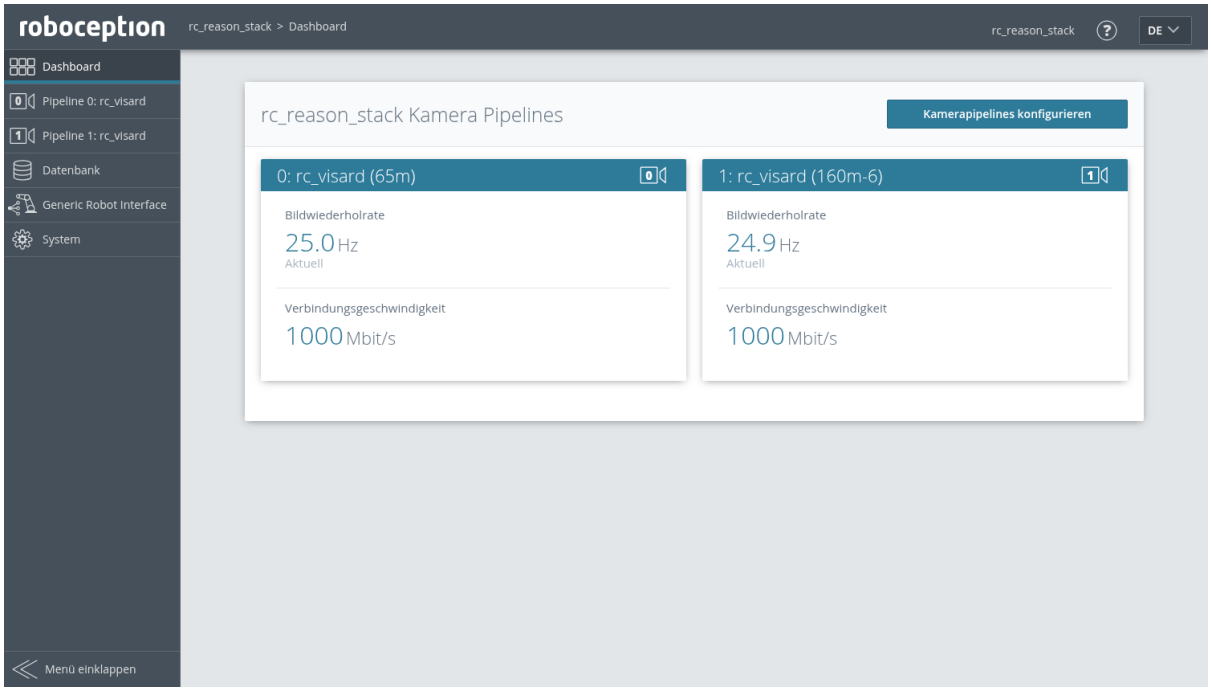


Abb. 7.1: Dashboard-Seite der Web GUI des *rc_reason_stack*

Über das Menü kann auf die einzelnen Seiten der Web GUI des *rc_reason_stack* zugegriffen werden:

Pipeline ermöglicht den Zugriff auf die zugehörige Kamerapipeline und ihre Kamera-, Detektions- und Konfigurationsmodule. Jede Pipeline hat eine Übersichtsseite mit den wichtigsten Informationen über die Kameraverbindung und die Softwaremodule, die in der Pipeline laufen.

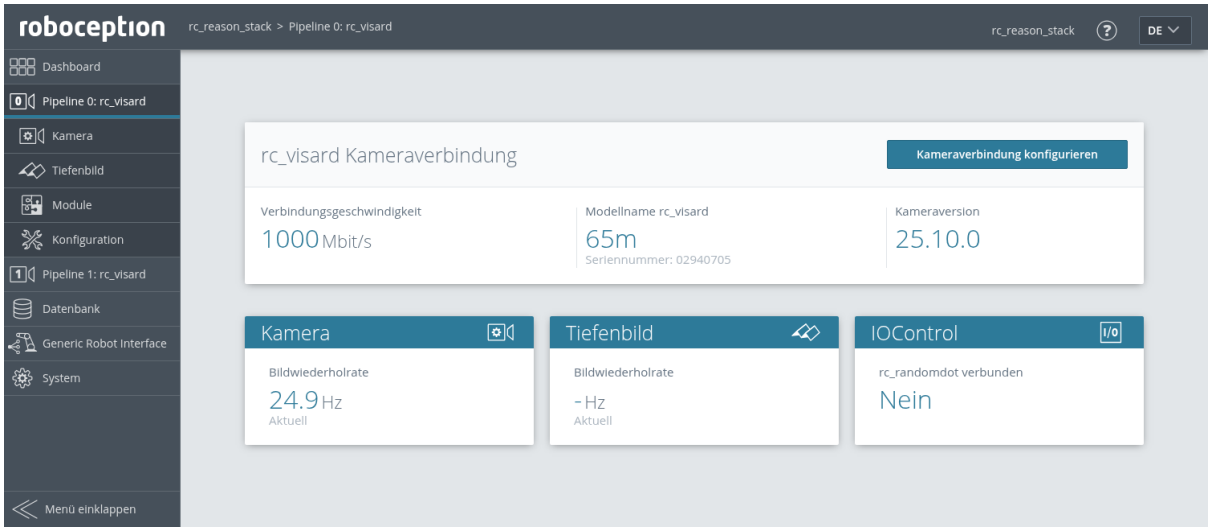


Abb. 7.2: Pipeline-Überblicksseite der Web GUI des *rc_reason_stack*

Jede Pipeline bietet ein Untermenü mit den einzelnen Seiten für die Module, die in der Pipeline laufen:

Kamera bietet einen Live-Stream der rektifizierten Bilder und ermöglicht das Einstellen der Kameraparameter. Für nähere Informationen siehe *Kamera Modul* (Abschnitt 6.1).

Tiefenbild bietet einen Live-Stream der rektifizierten Bilder der linken Kamera sowie Disparitäts- und Konfidenzbilder. Auf der Seite lassen sich verschiedene Einstellungen zur Berechnung und Filterung von Tiefenbildern vornehmen. Für nähere Informationen siehe *3D-Module* (Abschnitt 6.2).

Module ermöglicht den Zugriff auf die Detektionsmodule des *rc_reason_stack* (siehe *Detektions- und Messmodule*, Abschnitt 6.3).

Konfiguration ermöglicht den Zugriff auf die Konfigurationsmodule des *rc_reason_stack* (siehe *Konfigurationsmodule*, Abschnitt 6.4).

Die folgenden Module laufen außerhalb der Kamerapipelines und können über das Menü erreicht werden:

Datenbank ermöglicht den Zugriff auf die Datenbankmodule des *rc_reason_stack* (siehe *Datenbankmodule*, Abschnitt 6.5).

Generic Robot Interface zeigt die Jobs und Hand-Auge-Kalibrierkonfigurationen, die für das Generic Robot Interface definiert wurden.

System ermöglicht dem Nutzer den Zugriff auf allgemeine Systemeinstellungen, Informationen zur Software, den Log-Dateien, und Lizenzinformationen.

Bemerkung: Weitere Informationen zu den einzelnen Parametern der Web GUI lassen sich über die jeweils daneben angezeigte Schaltfläche *Info* aufrufen.

7.1.3 Web GUI Zugriffskontrolle

Die Web GUI bietet einen simplen Mechanismus das User Interface zu sperren um beiläufige und unbeabsichtigte Änderungen zu vermeiden.

Beim aktivieren der Web GUI Zugriffskontrolle über die *System* Seite muss ein Passwort gesetzt werden. Jetzt ist die Web GUI in einem gesperrten Zustand wie das Schloss Symbol in der Kopfleiste anzeigt. Alle Seiten, Kamerabilder, Parameter und Detektionen können wie gewohnt eingesehen werden, Änderungen sind aber nicht möglich.

Um die Web GUI temporär zu entsperren und Änderungen vorzunehmen, klicken Sie das Schloss Symbol und geben Sie das Passwort ein. Während das aktivieren und deaktivieren der Web GUI Zugriffskontrolle jeden betrifft der diesen *rc_reason_stack* nutzt, ist das Entsperren nur pro Browser gültig und wird durch das Symbol mit dem offenen Schloss angezeigt. Nach 10 Minutuen Inaktivität wird es automatisch wieder gesperrt.

Die Web GUI Zugriffskontrolle kann auf der *System* Seite wieder deaktiviert werden nachdem das aktuelle Passwort angegeben wurde.

Warnung: Dies ist keine Sicherheitsfunktion! Es sperrt nur die Web GUI und nicht die REST-API. Es ist dazu gedacht um unbeabsichtigte und beiläufige Änderungen, z.B. über einen angeschlossenen Bildschirm, zu verhindern.

Bemerkung: Im Fall eines vergessenen Passworts kann die Zugriffskontrolle über die REST-API mit *delete ui_lock* (Abschnitt 7.2.2.4) zurückgesetzt und deaktiviert werden.

7.1.4 Herunterladen von Kamerabildern

Die Web GUI bietet eine einfache Möglichkeit, einen Schnappschuss der aktuellen Szene als .tar.gz-Datei zu speichern. Dazu dient das Kamerasymbol unterhalb der Live-Streams auf der Seite *Kamera*.

Dieser Schnappschuss beinhaltet:

- die rektifizierten Kamerabilder in voller Auflösung als .png-Dateien,
- eine Kameraparameter-Datei mit Kameramatrix, Bildabmessungen, Belichtungszeit, Verstärkungsfaktor und Basisabstand der Kameras.
- die aktuellen IMU-Messungen als imu.csv-Datei, falls verfügbar,
- eine pipeline_status.json-Datei mit Informationen aller Module, die innerhalb der Kamerapipelines auf dem *rc_reason_stack* laufen,
- eine backup.json-Datei mit den Einstellungen des *rc_reason_stack* einschließlich konfigurierter Greifer, Load Carrier und Regions of Interest,
- eine system_info.json-Datei mit Systeminformationen des *rc_reason_stack*.

Die Dateinamen enthalten die Zeitstempel.

7.1.5 Herunterladen von Tiefenbildern und Punktwolken

Die Web GUI bietet eine einfache Möglichkeit, die Tiefendaten der aktuellen Szene als .tar.gz-Datei zu speichern. Dazu dient das Kamerasymbol unterhalb der Live-Streams auf der Seite *Tiefenbild*. Dieser Schnappschuss beinhaltet:

- die rektifizierten linken und rechten Kamerabilder in voller Auflösung als .png-Dateien,
- eine Parameterdatei für das linke Kamerabild mit Kameramatrix, Bildabmessungen, Belichtungszeit, Verstärkungsfaktor und Basisabstand der Kameras,
- die Disparitäts-, Fehler- und Konfidenzbilder in der Auflösung, die der aktuell eingestellten Qualität entspricht, als .png-Dateien,
- eine Parameterdatei zum Disparitätsbild mit Kameramatrix, Bildabmessungen, Belichtungszeit, Verstärkungsfaktor und Basisabstand der Kameras, sowie Informationen über die Disparitätswerte (ungültige Werte, Skalierung, Offset),
- die aktuellen IMU-Messungen als imu.csv-Datei, falls verfügbar,
- eine pipeline_status.json-Datei mit Informationen aller Module, die innerhalb der Kamerapipelines auf dem *rc_reason_stack* laufen,
- eine backup.json-Datei mit den Einstellungen des *rc_reason_stack* einschließlich konfigurierter Greifer, Load Carrier und Regions of Interest,
- eine system_info.json-Datei mit Systeminformationen des *rc_reason_stack*.

Die Dateinamen enthalten die Zeitstempel.

Durch Klick auf das Mesh-Symbol unterhalb der Live-Streams auf der Seite *Tiefenbild* kann man einen Schnappschuss herunterladen, der zusätzlich ein Mesh der Punktwolke in der aktuell eingestellten Auflösung (Qualität) als *.ply Datei enthält.

Bemerkung: Das Herunterladen der Tiefenbilder löst eine Bildaufnahme aus, in der gleichen Weise wie ein Klick auf den „Aufnehmen“-Button auf der *Tiefenbild*-Seite der Web GUI. Dies kann einen Einfluss auf laufende Anwendungen haben.

7.2 REST-API-Schnittstelle

Der *rc_reason_stack* bietet eine umfassende RESTful-Web-Schnittstelle (REST-API), auf die jeder HTTP-Client und jede HTTP-Bibliothek zugreifen kann. Während die meisten Parameter, Services und Funktionen auch über die benutzerfreundliche *Web GUI* (Abschnitt 7.1) zugänglich sind, dient die REST-API eher als Maschine-Maschine-Schnittstelle für folgende programmgesteuerte Aufgaben:

- Setzen und Abrufen der Laufzeitparameter der Softwaremodule, z.B. der Stereokamera oder von Bildverarbeitungsmodulen,
- Aufrufen von Services, z.B. zum Starten und Stoppen einzelner Softwaremodule, oder zum Nutzen spezieller Funktionen, wie der Hand-Auge-Kalibrierung,
- Abruf des aktuellen Systemstatus und des Status einzelner Softwaremodule, sowie
- Aktualisierung der Firmware des *rc_reason_stack* oder seiner Lizenz.

Bemerkung: In der REST-API des *rc_reason_stack* bezeichnet der Begriff *Node* ein Softwaremodul, das gewisse algorithmische Funktionen bündelt und eine ganzheitliche Benutzeroberfläche (Parameter, Services, aktueller Status) besitzt. Beispiele für solche Module sind das Stereo-Matching-Modul oder das Modul zur Hand-Auge-Kalibrierung.

7.2.1 Allgemeine Struktur der Programmierschnittstelle (API)

Der allgemeine **Einstiegspunkt** zur Programmierschnittstelle (API) des *rc_reason_stack* ist `http://<host>/api/` wobei `<host>` die IP-Adresse des Host-PC ist, auf dem der *rc_reason_stack* läuft, kombiniert mit dem Port 8080, d.h. `<host-ip>::8080`. Greift der Benutzer über einen Webbrowser auf diese Adresse zu, kann er die Programmierschnittstelle während der Laufzeit mithilfe der *Swagger UI* (Abschnitt 7.2.4) erkunden und testen.

Für die eigentlichen HTTP-Anfragen wird dem Einstiegspunkt der Programmierschnittstelle die **aktuelle Version der Schnittstelle als Postfix angehangen**, d.h. `http://<host>/api/v2`.

Alle Daten, die an die REST-API gesandt und von ihr empfangen werden, entsprechen dem JSON-Datenformat (JavaScript Object Notation). Die Programmierschnittstelle ist so gestaltet, dass der Benutzer die in *Verfügbare Ressourcen und Anfragen* (Abschnitt 7.2.2) aufgelisteten sogenannten **Ressourcen** über die folgenden HTTP-Anforderungen **anlegen, abrufen, ändern und löschen** kann.

Anfragetyp	Beschreibung
GET	Zugriff auf eine oder mehrere Ressourcen und Rückgabe des Ergebnisses im JSON-Format
PUT	Änderung einer Ressource und Rückgabe der modifizierten Ressource im JSON-Format
DELETE	Löschen einer Ressource
POST	Upload einer Datei (z.B. einer Lizenz oder eines Firmware-Images)

Je nach der Art der Anfrage und Datentyp können die **Argumente** für HTTP-Anfragen als Teil des **Pfads** (URI) zur Ressource, als **Abfrage**-Zeichenfolge, als **Formulardaten** oder im **Body** der Anfrage übertragen werden. Die folgenden Beispiele nutzen das Kommandozeilenprogramm *curl*, das für verschiedene Betriebssysteme verfügbar ist (siehe https://curl.haxx).se.

- Abruf des aktuellen Status eines Moduls, wobei sein Name im Pfad (URI) verschlüsselt ist

```
curl -X GET 'http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching'
```

- Abruf einiger Parameterwerte eines Moduls über eine Abfragezeichenfolge

```
curl -X GET 'http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters?
↪name=minconf&name=maxdepth'
```

- Setzen eines Modulparameters als JSON-formatierter Text im Body der Anfrage

```
curl -X PUT --header 'Content-Type: application/json' -d '[{"name": "mindepth", "value": 0.
↪1}]' 'http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters'
```

Zur Beantwortung solcher Anfragen greift die Programmierschnittstelle des *rc_reason_stack* auf übliche Rückgabecodes zurück:

Statuscode	Beschreibung
`200 OK`	Die Anfrage war erfolgreich. Die Ressource wird im JSON-Format zurückgegeben.
`400 Bad Request`	Ein für die API-Anfrage benötigtes Attribut oder Argument fehlt oder ist ungültig.
`404 Not Found`	Auf eine Ressource konnte nicht zugegriffen werden. Möglicherweise kann die ID einer Ressource nicht gefunden werden.
`403 Forbidden`	Der Zugriff ist (vorübergehend) verboten. Möglicherweise sind einige Parameter gesperrt, während eine GigE Vision-Anwendung verbunden ist.
`429 Too many requests`	Die Übertragungsrate ist aufgrund einer zu hohen Anfragefrequenz begrenzt.

Der folgende Eintrag zeigt eine Musterantwort auf eine erfolgreiche Anfrage, mit der Informationen zum `minconf`-Parameter des `rc_stereomatching`-Moduls angefordert werden:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 157

{
    "name": "minconf",
    "min": 0,
    "default": 0,
    "max": 1,
    "value": 0,
    "type": "float64",
    "description": "Minimum confidence"
}
```

Bemerkung: Das tatsächliche Verhalten, die zulässigen Anfragen und die speziellen Rückgabecodes hängen in hohem Maße von der gewählten Ressource, vom Kontext und von der Aktion ab. Siehe die *verfügbaren Ressourcen* (Abschnitt 7.2.2) des *rc_reason_stack* und einzelnen Parameter und Services jedes *Softwaremoduls* (Abschnitt 6).

7.2.2 Verfügbare Ressourcen und Anfragen

Die für die REST-API verfügbaren Ressourcen lassen sich in folgende Teilbereiche gliedern:

- **`/nodes`** Zugriff auf die globalen *Datenbankmodule* (Abschnitt 6.5) des *rc_reason_stack* mit ihren Laufzeitzuständen, Parametern und angebotenen Services, um Daten zu speichern, die in allen Kamerapipelines und mehreren Modulen genutzt werden, z.B. Load Carrier, Greifer und Regions of Interest.

- **`/pipelines/<number>/nodes`** Zugriff auf die 3D-Kamera-, Detektions- und Konfigurations-*Softwaremodule* (Abschnitt 6) der Kamerapipeline des *rc_reason_stack* mit der angegebenen Nummer `number`, mit ihren jeweiligen Laufzeitzuständen, Parametern und verfügbaren Services.
- **`/pipelines`** Zugriff auf den Status und die Konfiguration der Kamerapipelines.
- **`/templates`** Zugriff auf die im *rc_reason_stack* hinterlegten Objekttemplates.
- **`/cad`** Zugriff auf die CAD-Elemente, z.B. für Greifer, im *rc_reason_stack*.
- **`/presets`** Zugriff auf die benutzerdefinierten 2D und 3D Voreinstellungen für *zivid* Kameras.
- **`/system`** Zugriff auf Systemzustand, Netzwerkkonfiguration, Konfiguration der Kamerapipelines, und Verwaltung der Lizenzen sowie der Firmware-Updates.
- **`/logs`** Zugriff auf die im *rc_reason_stack* hinterlegten Logdateien.
- **`/generic_robot_interface`** Zugriff auf die Jobs und Hand-Auge-Kalibrierkonfigurationen für das Generic Robot Interface auf dem *rc_reason_stack*.

7.2.2.1 Module, Parameter und Services

Die *Softwaremodule* (Abschnitt 6) des *rc_reason_stack* heißen in der REST-API *Nodes* und vereinen jeweils bestimmte algorithmische Funktionen. Über folgenden Befehl lassen sich alle globalen Datenbankmodule der REST-API mit ihren jeweiligen Services und Parametern auflisten:

```
curl -X GET http://<host>/api/v2/nodes
```

Informationen zu einem bestimmten Modul (z.B. `rc_load_carrier_db`) lassen sich mit folgendem Befehl abrufen:

```
curl -X GET http://<host>/api/v2/nodes/rc_load_carrier_db
```

Alle verfügbaren 3D-Kamera-, Detektions- und Konfigurationsmodule der REST-API lassen sich mit ihren Services und Parametern wie folgt auflisten:

```
curl -X GET http://<host>/api/v2/pipelines/<pipeline number>/nodes
```

Informationen zu einem bestimmten Modul (z.B. `rc_camera` in Kamerapipeline 1) lassen sich mit folgendem Befehl abrufen:

```
curl -X GET http://<host>/api/v2/pipelines/1/nodes/rc_camera
```

Status: Während der Laufzeit stellt jedes Modul Informationen zu seinem aktuellen Status bereit. Dies umfasst nicht nur den aktuellen **Verarbeitungsstatus** des Moduls (z.B. `running` oder `stale`), sondern die meisten Module melden auch Laufzeitstatistiken oder schreibgeschützte Parameter, sogenannte **Statuswerte**. Die Statuswerte des `rc_camera`-Moduls lassen sich beispielsweise wie folgt abrufen:

```
curl -X GET http://<host>/api/v2/pipelines/<pipeline number>/nodes/rc_camera/status
```

Bemerkung: Die zurückgegebenen **Statuswerte** sind modulspezifisch und werden im jeweiligen *Softwaremodul* (Abschnitt 6) dokumentiert.

Bemerkung: Statuswerte werden nur gemeldet, wenn sich das jeweilige Modul im Zustand `running` befindet.

Parameter: Die meisten Module stellen Parameter über die REST-API des *rc_reason_stack* zur Verfügung, damit ihr Laufzeitverhalten an den Anwendungskontext oder die Anforderungen angepasst werden kann. Die REST-API ermöglicht es, den Wert eines Parameters zu setzen und abzufragen.

Darüber hinaus stellt sie weitere Angaben, wie z.B. den jeweiligen Standardwert und zulässige Minimal- bzw. Maximalwerte von Parametern, zur Verfügung.

Die `rc_stereomatching`-Parameter lassen sich beispielsweise wie folgt abrufen:

```
curl -X GET http://<host>/api/v2/pipelines/<pipeline number>/nodes/rc_stereomatching/
↪parameters
```

Der `quality`-Parameter dieses Moduls könnte wie folgt auf den Wert `Full` gesetzt werden:

```
curl -X PUT http://<host>/api/v2/pipelines/<pipeline number>/nodes/rc_stereomatching/
↪parameters?quality=Full
```

oder äquivalent

```
curl -X PUT --header 'Content-Type: application/json' -d '{ "value": "Full" }' http://<host>
↪/api/v2/pipelines/<pipeline number>/nodes/rc_stereomatching/parameters/quality
```

Bemerkung: Laufzeitparameter sind modulspezifisch und werden in dem jeweiligen *Softwaremodul* (Abschnitt 6) dokumentiert.

Bemerkung: Die meisten Parameter, die die Module über die REST-API anbieten, lassen sich auch über die benutzerfreundliche *Web GUI* (Abschnitt 7.1) des *rc_reason_stack* erkunden und austesten.

Zudem bietet jedes Modul, das Laufzeitparameter bereitstellt, auch einen Service, um die Werkseinstellungen aller Parameter wiederherzustellen.

Services: Die meisten Module bieten auch Services, die sich über die REST-API aufrufen lassen. Hierzu gehört beispielsweise das oben bereits genannte Wiederherstellen von Parametern oder auch das Starten und Stoppen von Modulen. Die *Services des Moduls zur Hand-Auge-Kalibrierung* (Abschnitt 6.4.1.5) lassen sich beispielsweise wie folgt aufrufen:

```
curl -X GET http://<host>/api/v2/pipelines/<pipeline number>/nodes/rc_hand_eye_calibration/
↪services
```

Um einen Service eines Moduls aufzurufen, wird eine `PUT`-Anfrage mit servicespezifischen Argumenten für die jeweilige Ressource gestellt (siehe das `"args"`-Feld des *Service-Datenmodells*, Abschnitt 7.2.3). Beispielsweise lässt sich folgendermaßen eine Bildaufnahme mit dem Stereo-Matching-Modul auslösen:

```
curl -X PUT --header 'Content-Type: application/json' -d '{ "args": {} }' http://<host>/api/
↪v2/pipelines/<pipeline number>/nodes/rc_stereomatching/services/acquisition_trigger
```

Bemerkung: Die Services und zugehörigen Argumente sind modulspezifisch und werden im jeweiligen *Softwaremodul* (Abschnitt 6) dokumentiert.

Die folgende Liste enthält alle REST-API-Anfragen zum Status der globalen Datenbankmodule und ihrer Parameter und Services:

`GET /nodes`

Abruf einer Liste aller verfügbaren globalen Nodes.

Musteranfrage

```
GET /api/v2/nodes HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "name": "rc_roi_db",
    "parameters": [],
    "services": [
      "set_region_of_interest",
      "get_regions_of_interest",
      "delete_regions_of_interest",
      "set_region_of_interest_2d",
      "get_regions_of_interest_2d",
      "delete_regions_of_interest_2d"
    ],
    "status": "running"
  },
  {
    "name": "rc_load_carrier_db",
    "parameters": [],
    "services": [
      "set_load_carrier",
      "get_load_carriers",
      "delete_load_carriers"
    ],
    "status": "running"
  },
  {
    "name": "rc_gripper_db",
    "parameters": [],
    "services": [
      "set_gripper",
      "get_grippers",
      "delete_grippers"
    ],
    "status": "running"
  }
]
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo-Array)*

Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 7.2.3)

`GET /nodes/{node}`

Abruf von Informationen zu einem einzelnen globalen Modul.

Musteranfrage

```
GET /api/v2/nodes/<node> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
{
  "name": "rc_roi_db",
  "parameters": [],
  "services": [
    "set_region_of_interest",
    "get_regions_of_interest",
    "delete_regions_of_interest",
    "set_region_of_interest_2d",
    "get_regions_of_interest_2d",
    "delete_regions_of_interest_2d"
  ],
  "status": "running"
}
```


Parameter

- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 7.2.3)

GET /nodes/{node}/services

Abruf von Beschreibungen aller von einem globalen Modul angebotenen Services.

Musteranfrage

```
GET /api/v2/nodes/<node>/services HTTP/1.1
```

Musteranfrage


```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "args": {},
    "description": "string",
    "name": "string",
    "response": {}
  }
]
```


Parameter

- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Service-Array)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 7.2.3)

GET /nodes/{node}/services/{service}

Abruf der Beschreibung eines Services eines globalen Moduls.

Musteranfrage

```
GET /api/v2/nodes/<node>/services/<service> HTTP/1.1
```

Musteranfrage

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "args": {},
  "description": "string",
  "name": "string",
  "response": {}
}
```

Parameter

- **node** (*string*) – Modulname *(obligatorisch)*
- **service** (*string*) – Name des Service *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Service)*
- 404 Not Found – Modul oder Service nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 7.2.3)

PUT /nodes/{node}/services/{service}

Aufruf des Services eines Moduls: Die benötigten Argumente und die zugehörige Antwort hängt vom Modul und vom Service ab.

Musteranfrage

```
PUT /api/v2/nodes/<node>/services/<service> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Musteranfrage

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "args": {},
  "description": "string",
  "name": "string",
  "response": {}
}
```

Parameter

- **node** (*string*) – Modulname *(obligatorisch)*
- **service** (*string*) – Name des Service *(obligatorisch)*

JSON-Objekt zur Anfrage

- **service args** (*object*) – Beispielargumente *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Serviceaufruf erledigt *(Rückgabe: Service)*
- 403 Forbidden – Service-Aufruf verboten, z.B. weil keine valide Lizenz für dieses Modul vorliegt.
- 404 Not Found – Modul oder Service nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 7.2.3)

GET /nodes/{node}/status

Abruf des Status eines globalen Datenbankmoduls.

Musteranfrage

```
GET /api/v2/nodes/<node>/status HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "running",
  "timestamp": 1503075030.2335997,
  "values": []
}
```

Parameter

- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeStatus)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *NodeStatus* (Abschnitt 7.2.3)

Die folgende Liste enthält alle REST-API-Anfragen zum Status der pipelinespezifischen 3D-Kamera-, Detektions- und KonfigurationsModule und ihrer Parameter und Services:

GET /pipelines/{pipeline}/nodes

Abruf einer Liste aller verfügbaren Module.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "name": "rc_camera",
    "parameters": [
      "fps",
      "exp_auto",
      "exp_value",
      "exp_max"
    ],
    "services": [
      "reset_defaults"
    ],
    "status": "running"
  },
  {
    "name": "rc_hand_eye_calibration",
    "parameters": [
      "grid_width",
      "grid_height",
      "robot_mounted"
    ],
    "services": [
      "reset_defaults",
      "set_pose",
      "reset",
      "save",
      "calibrate",
      "get_calibration"
    ],
    "status": "idle"
  },
  {
    "name": "rc_stereomatching",
    "parameters": [
      "quality",
      "seg",
      "fill",
      "minconf",
      "mindepth",
      "maxdepth",
      "maxdeptherr"
    ],
    "services": [
      "reset_defaults"
    ],
    "status": "running"
  }
]
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`, `1`, `2` oder `3`) *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo-Array)*

Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 7.2.3)

`GET /pipelines/{pipeline}/nodes/{node}`

Abruf von Informationen zu einem einzelnen Modul.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "name": "rc_camera",
  "parameters": [
    "fps",
    "exp_auto",
    "exp_value",
    "exp_max"
  ],
  "services": [
    "reset_defaults"
  ],
  "status": "running"
}
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`, `1`, `2` oder `3`) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *NodeInfo* (Abschnitt 7.2.3)

`GET /pipelines/{pipeline}/nodes/{node}/parameters`

Abruf von Parametern eines Moduls.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/parameters?name=<name> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "default": 25,
    "description": "Frames per second in Hz",
    "max": 25,
    "min": 1,
    "name": "fps",
    "type": "float64",
    "value": 25
  },
  {
    "default": true,
    "description": "Switching between auto and manual exposure",
    "max": true,
    "min": false,
    "name": "exp_auto",
    "type": "bool",
    "value": true
  },
  {
    "default": 0.007,
    "description": "Maximum exposure time in s if exp_auto is true",
    "max": 0.018,
    "min": 6.6e-05,
    "name": "exp_max",
    "type": "float64",
    "value": 0.007
  }
]
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`, `1`, `2` oder `3`) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*

Anfrageparameter

- **name** (*string*) – Schränkt Ergebnisse auf Parameter mit diesem Namen ein *(optional)*.

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo-Array)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Parameter* (Abschnitt 7.2.3)

PUT /pipelines/{pipeline}/nodes/{node}/parameters

Aktualisierung mehrerer Parameter.

Musteranfrage

```
PUT /api/v2/pipelines/<pipeline>/nodes/<node>/parameters HTTP/1.1
Accept: application/json application/ubjson
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
[
  {
    "name": "string",
    "value": {}
  }
]
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "default": 25,
    "description": "Frames per second in Hz",
    "max": 25,
    "min": 1,
    "name": "fps",
    "type": "float64",
    "value": 10
  },
  {
    "default": true,
    "description": "Switching between auto and manual exposure",
    "max": true,
    "min": false,
    "name": "exp_auto",
    "type": "bool",
    "value": false
  },
  {
    "default": 0.005,
    "description": "Manual exposure time in s if exp_auto is false",
    "max": 0.018,
    "min": 6.6e-05,
    "name": "exp_value",
    "type": "float64",
    "value": 0.005
  }
]
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (0, 1, 2 oder 3) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*

JSON-Objekt-Array zur Anfrage

- **parameters** (*ParameterNameValue*) – Liste von Parametern *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeInfo-Array)*

- 400 Bad Request – Ungültiger Parameterwert
- 403 Forbidden – Aktualisierung des Parameters verboten, z.B. weil er aufgrund einer laufenden GigE Vision-Anwendung gesperrt ist oder keine valide Lizenz für dieses Modul vorliegt.
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *ParameterNameValue* (Abschnitt 7.2.3)
- *Parameter* (Abschnitt 7.2.3)

`GET /pipelines/{pipeline}/nodes/{node}/parameters/{param}`
Abruf eines bestimmten Parameters eines Moduls.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/parameters/<param> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "default": 25,
  "description": "Frames per second in Hertz",
  "max": 25,
  "min": 1,
  "name": "fps",
  "type": "float64",
  "value": 10
}
```

Parameter

- **`pipeline`** (*`string`*) – Name der Pipeline (`0`, `1`, `2` oder `3`) *(obligatorisch)*
- **`node`** (*`string`*) – Modulname *(obligatorisch)*
- **`param`** (*`string`*) – Name des Parameters *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Parameter)*
- 404 Not Found – Modul oder Parameter nicht gefunden

Referenzierte Datenmodelle

- *Parameter* (Abschnitt 7.2.3)

`PUT /pipelines/{pipeline}/nodes/{node}/parameters/{param}`
Aktualisierung eines bestimmten Parameters eines Moduls.

Musteranfrage

```
PUT /api/v2/pipelines/<pipeline>/nodes/<node>/parameters/<param> HTTP/1.1
Accept: application/json application/ubjson

{
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  "value": {}
}
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "default": 25,
  "description": "Frames per second in Hertz",
  "max": 25,
  "min": 1,
  "name": "fps",
  "type": "float64",
  "value": 10
}
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`, `1`, `2` oder `3`) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*
- **param** (*string*) – Name des Parameters *(obligatorisch)*

JSON-Objekt zur Anfrage

- **parameter** (*ParameterValue*) – zu aktualisierender Parameter als JSON-Objekt *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Parameter)*
- 400 Bad Request – Ungültiger Parameterwert
- 403 Forbidden – Aktualisierung des Parameters verboten, z.B. weil er aufgrund einer laufenden GigE Vision-Anwendung gesperrt ist oder keine valide Lizenz für dieses Modul vorliegt.
- 404 Not Found – Modul oder Parameter nicht gefunden

Referenzierte Datenmodelle

- *ParameterValue* (Abschnitt 7.2.3)
- *Parameter* (Abschnitt 7.2.3)

GET /pipelines/{pipeline}/nodes/{node}/services

Abruf von Beschreibungen aller von einem Modul angebotenen Services.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/services HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "args": {},
    "description": "Restarts the module.",
    "name": "restart",
    "response": {
      "accepted": "bool",
      "current_state": "string"
    }
  },
  {
    "args": {},
    "description": "Starts the module.",
    "name": "start",
    "response": {
      "accepted": "bool",
      "current_state": "string"
    }
  },
  {
    "args": {},
    "description": "Stops the module.",
    "name": "stop",
    "response": {
      "accepted": "bool",
      "current_state": "string"
    }
  }
]
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`, `1`, `2` oder `3`) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Service-Array)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 7.2.3)

GET /pipelines/{pipeline}/nodes/{node}/services/{service}

Abruf der Beschreibung eines modulspezifischen Services.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/services/<service> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
{
  "args": {
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "slot": "int32"
  },
  "description": "Save a pose (grid or gripper) for later calibration.",
  "name": "set_pose",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```


Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`, `1`, `2` oder `3`) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*
- **service** (*string*) – Name des Service *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Service)*
- 404 Not Found – Modul oder Service nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 7.2.3)

PUT /pipelines/{pipeline}/nodes/{node}/services/{service}

Aufruf des Services eines Moduls: Die benötigten Argumente und die zugehörige Antwort hängt vom Modul und vom Service ab.

Musteranfrage

```
PUT /api/v2/pipelines/<pipeline>/nodes/<node>/services/<service> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
{
  "name": "set_pose",
  "response": {
    "message": "Grid detected, pose stored.",
    "status": 1,
    "success": true
  }
}
```


Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`, `1`, `2` oder `3`) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*
- **service** (*string*) – Name des Service *(obligatorisch)*

JSON-Objekt zur Anfrage

- **service args** (*object*) – Beispielargumente *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Serviceaufruf erledigt *(Rückgabe: Service)*
- 403 Forbidden – Service-Aufruf verboten, z.B. weil keine valide Lizenz für dieses Modul vorliegt.
- 404 Not Found – Modul oder Service nicht gefunden

Referenzierte Datenmodelle

- *Service* (Abschnitt 7.2.3)

GET /pipelines/{pipeline}/nodes/{node}/status

Abruf des Status eines Moduls.

Musteranfrage

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/status HTTP/1.1
```

Beispielantwort


```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "running",
  "timestamp": 1503075030.2335997,
  "values": {
    "baseline": "0.0650542",
    "color": "0",
    "exp": "0.00426667",
    "focal": "0.844893",
    "fps": "25.1352",
    "gain": "12.0412",
    "height": "960",
    "temp_left": "39.6",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "temp_right": "38.2",
    "time": "0.00406513",
    "width": "1280"
  }
}
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (0, 1, 2 oder 3) *(obligatorisch)*
- **node** (*string*) – Modulname *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: NodeStatus)*
- 404 Not Found – Modul nicht gefunden

Referenzierte Datenmodelle

- *NodeStatus* (Abschnitt 7.2.3)

7.2.2.2 Pipelines

Pipelines repräsentieren die Kamerapipelines des *rc_reason_stack*.

Die folgende Liste enthält alle REST-API Anfragen bezüglich der Konfiguration der Kamerapipelines:

GET /pipelines
Abruf der aktiven Pipelines

Musteranfrage

```
GET /api/v2/pipelines HTTP/1.1
```

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

GET /pipelines/{pipeline}
Abruf des Pipeline-Typs und -Status

Musteranfrage

```
GET /api/v2/pipelines/<pipeline> HTTP/1.1
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (0, 1, 2 oder 3) *(obligatorisch)*

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

GET /system/pipelines
Abruf der Pipelinekonfiguration

Musteranfrage

```
GET /api/v2/system/pipelines HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "config": {
    "0": {
      "type": "rc_visard"
    }
  },
  "max_pipelines": 4,
  "pending_changes": false
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

`GET /system/pipelines/config/{pipeline}`
Abruf der Konfiguration einer spezifischen Pipeline.

Musteranfrage

```
GET /api/v2/system/pipelines/config/<pipeline> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "0": {
    "type": "rc_visard"
  }
}
```

Parameter

- **`pipeline`** (*`string`*) – Name der Pipeline (`0`, `1`, `2` oder `3`) *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

`PUT /system/pipelines/config/{pipeline}`
Updaten der Konfiguration einer spezifischen Pipeline.

Musteranfrage

```
PUT /api/v2/system/pipelines/config/<pipeline>?type=<type> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "type": "rc_visard"
}
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`, `1`, `2` oder `3`) *(obligatorisch)*

Anfrageparameter

- **type** (*string*) – Pipelinetyp (`rc_visard`, `rc_viscore`,“blaze“, `zivid`, `stereo_ace`) *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 400 Bad Request – ungültiger Pipelinename oder -typ

`DELETE /system/pipelines/config/{pipeline}`

Lösche spezifische Pipelines

Musteranfrage

```
DELETE /api/v2/system/pipelines/config/<pipeline> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "message": "Pipeline 1 deleted"
}
```

Parameter

- **pipeline** (*string*) – Name der Pipeline (`0`, `1`, `2` oder `3`) *(obligatorisch)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 400 Bad Request – ungültiger Pipelinename, z.B. kann Pipeline 0 nicht gelöscht werden

7.2.2.3 UserSpace

UserSpace Informationen einschließlich laufender Apps und ihre veröffentlichten Ports können über den userspace Endpunkt abgefragt werden. Eine App kann vom Typ (`type`) `container` oder `compose` (Compose-Stack mit potenziell mehreren Containern) sein.

7.2.2.4 System und Logs

Die folgenden Ressourcen und Anfragen sind für die System-Level-API des *rc_reason_stack* verfügbar. Sie ermöglichen Folgendes:

- Zugriff auf Logdateien (systemweit oder modulspezifisch),
- Abruf von Informationen zum Gerät und zur Laufzeitstatistik, wie Datum, MAC-Adresse, Uhrzeitsynchronisierungsstatus und verfügbare Ressourcen,
- Verwaltung installierter Softwarelizenzen, und
- Aktualisierung des Firmware-Images des *rc_reason_stack*.

`GET /logs`
Abruf einer Liste aller verfügbaren Logdateien.

Musteranfrage

```
GET /api/v2/logs HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "date": 1503060035.0625782,
    "name": "rcsense-api.log",
    "size": 730
  },
  {
    "date": 1503060035.741574,
    "name": "stereo.log",
    "size": 39024
  },
  {
    "date": 1503060044.0475223,
    "name": "camera.log",
    "size": 1091
  }
]
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: LogInfo-Array)*

Referenzierte Datenmodelle

- *LogInfo* (Abschnitt 7.2.3)

`GET /logs/{log}`
Abruf einer Logdatei: Die Art des Inhalts der Antwort richtet sich nach dem *format*-Parameter.

Musteranfrage

```
GET /api/v2/logs/<log>?format=<format>&limit=<limit> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "date": 1581609251.8168414,
  "log": [
    {
      "component": "rc_gev_server",
      "level": "INFO",
      "message": "Application from IP 10.0.1.7 registered with control access.",
      "timestamp": 1581609249.61
    },
    {
      "component": "rc_gev_server",
      "level": "INFO",
      "message": "Application from IP 10.0.1.7 deregistered.",
      "timestamp": 1581609249.739
    },
    {
      "component": "rc_gev_server",
      "level": "INFO",
      "message": "Application from IP 10.0.1.7 registered with control access.",
      "timestamp": 1581609250.94
    },
    {
      "component": "rc_gev_server",
      "level": "INFO",
      "message": "Application from IP 10.0.1.7 deregistered.",
      "timestamp": 1581609251.819
    }
  ],
  "name": "gev.log",
  "size": 42112
}
```

Parameter

- **log** (*string*) – Name der Logdatei *(obligatorisch)*

Anfrageparameter

- **format** (*string*) – Rückgabe des Logs im JSON- oder Rohdatenformat (mögliche Werte: `json` oder `raw`; Voreinstellung: `json`) *(optional)*
- **limit** (*integer*) – Beschränkung auf die letzten x Zeilen im JSON-Format (Voreinstellung: `100`) *(optional)*

Antwort-Headers

- Content-Type – text/plain application/json

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: Log)*
- 404 Not Found – Log nicht gefunden

Referenzierte Datenmodelle

- *Log* (Abschnitt 7.2.3)

GET /system

Abruf von Systeminformationen zum Gerät.

Musteranfrage

```
GET /api/v2/system HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "dongle_id": "wibu:1234",
  "firmware": {
    "active_image": {
      "image_version": "26.01.0"
    }
  },
  "model_name": "rc_reason_stack",
  "pipelines": {
    "config": {
      "0": {
        "type": "rc_visard"
      },
      "1": {
        "type": "rc_visard"
      }
    },
    "max_pipelines": 4,
    "pending_changes": false
  },
  "ready": true,
  "reboot_required": false,
  "time": 1649678734.0306993,
  "uptime": 336455.25
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: SysInfo)*

Referenzierte Datenmodelle

- *SysInfo* (Abschnitt 7.2.3)

GET /system/backup

Abruf eines Backups der Einstellungen.

Musteranfrage

```
GET /api/v2/system/backup?pipelines=<pipelines>&load_carriers=<load_carriers>&regions_of_
↪interest=<regions_of_interest>&grippers=<grippers> HTTP/1.1
```

Anfrageparameter

- **pipelines** (*boolean*) – Backup der Pipelines mit Moduleinstellungen, d.h. Parameter und bevorzugte TCP-Orientierung (Standardwert: `True`) *(optional)*
- **load_carriers** (*boolean*) – Backup der Load Carrier (Standardwert: `True`) *(optional)*
- **regions_of_interest** (*boolean*) – Backup der Regions of Interest (Standardwert: `True`) *(optional)*
- **grippers** (*boolean*) – Backup der Greifer (Standardwert: `True`) *(optional)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

`POST /system/backup`
Backup einspielen.

Musteranfrage

```
POST /api/v2/system/backup HTTP/1.1
Accept: application/json application/ubjson

{}
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "return_code": {
    "message": "backup restored",
    "value": 0
  },
  "warnings": []
}
```

Request JSON Object

- **`backup`** (*`object`*) – Backup-Daten als json-Objekt *(erforderlich)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

`GET /system/disk_info`
Abruf der Speicherinformation

Musteranfrage

```
GET /api/v2/system/disk_info HTTP/1.1
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

`GET /system/license`
Abruf von Informationen zu den auf dem Gerät installierten Lizenzen.

Musteranfrage

```
GET /api/v2/system/license HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "components": {
    "hand_eye_calibration": true,
    "rectification": true,
    "stereo": true
  },
  "valid": true
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: LicenseInfo)*

Referenzierte Datenmodelle

- *LicenseInfo* (Abschnitt 7.2.3)

`POST /system/license`
Aktualisierung der auf dem Gerät installierten Lizenz mithilfe einer Lizenzdatei.

Musteranfrage

```
POST /api/v2/system/license HTTP/1.1
Accept: multipart/form-data
```

Formularparameter

- **`file`** – Lizenzdatei *(obligatorisch)*

Anfrage-Header

- Accept – Multipart/Formulardaten

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 400 Bad Request – Keine gültige Lizenz

`GET /system/ui_lock`
Abruf des UI Lock Status

Musteranfrage

```
GET /api/v2/system/ui_lock HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": false
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung *(Rückgabe: UILock)*

Referenzierte Datenmodelle

- *UILock* (Abschnitt 7.2.3)

`DELETE /system/ui_lock`

UI Lock entfernen.

Musteranfrage

```
DELETE /api/v2/system/ui_lock HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": false,
  "valid": false
}
```

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

`POST /system/ui_lock`

Verifizieren oder Setzen des UI Locks.

Musteranfrage

```
POST /api/v2/system/ui_lock?hash=<hash>&set=<set> HTTP/1.1
```

Beispielantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": true,
  "valid": true
}
```

Anfrageparameter

- **`hash`** (*`string`*) – Hash des UI Lock Passworts *(obligatorisch)*
- **`set`** (*`boolean`*) – neuen Hash setzen anstatt zu verifizieren *(optional)*

Antwort-Headers

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

7.2.3 Datentyp-Definitionen

Die REST-API definiert folgende Datenmodelle, die verwendet werden, um auf die *verfügbaren Ressourcen* (Abschnitt 7.2.2) zuzugreifen oder diese zu ändern, entweder als benötigte Attribute/Parameter oder als Rückgabewerte.

FirmwareInfo: Informationen zu aktuell aktiven und inaktiven Firmware-Images und dazu, welches Image für den Boot-Vorgang verwendet wird.

Ein Objekt des Typs FirmwareInfo besitzt folgende Eigenschaften:

- **active_image** (*ImageInfo*): siehe Beschreibung von *ImageInfo*.

Musterobjekt

```
{
  "active_image": {
    "image_version": "string"
  }
}
```

FirmwareInfo-Objekte sind in *SysInfo* enthalten.

GripperElement: CAD-Greiferelement

Ein Objekt des Typs GripperElement besitzt folgende Eigenschaften:

- **id** (string): Eindeutiger Name des Elements

Musterobjekt

```
{
  "id": "string"
}
```

GripperElement-Objekte werden in folgenden Anfragen verwendet:

- *GET /cad/gripper_elements*
- *GET /cad/gripper_elements/{id}*
- *PUT /cad/gripper_elements/{id}*

ImageInfo: Informationen zu einem bestimmten Firmware-Image.

Ein Objekt des Typs ImageInfo besitzt folgende Eigenschaften:

- **image_version** (string): Image-Version.

Musterobjekt

```
{
  "image_version": "string"
}
```

ImageInfo-Objekte sind in *FirmwareInfo* enthalten.

LicenseComponentConstraint: Einschränkungen für die Modul-Version.

Ein Objekt des Typs LicenseComponentConstraint besitzt folgende Eigenschaften:

- **max_version** (string) - optionale höchste unterstützte Version (exclusive)
- **min_version** (string) - optionale minimale unterstützte Version (inclusive)

Musterobjekt

```
{
  "max_version": "string",
  "min_version": "string"
}
```


LicenseComponentConstraint-Objekte sind in *LicenseConstraints* enthalten.

LicenseComponents: Liste der Lizenzstatus-Angaben der einzelnen Softwaremodule: Der zugehörige Statusindikator ist auf TRUE gesetzt, wenn das entsprechende Modul mit einer installierten Softwarelizenz entsperrt ist.

Ein Objekt des Typs LicenseComponents besitzt folgende Eigenschaften:

- **hand_eye_calibration** (boolean): Modul zur Hand-Auge-Kalibrierung.
- **rectification** (boolean): Modul zur Bildrektifizierung.
- **stereo** (boolean): Stereo-Matching-Modul.

Musterobjekt


```
{
  "hand_eye_calibration": false,
  "rectification": false,
  "stereo": false
}
```


LicenseComponents-Objekte sind in *LicenseInfo* enthalten.

LicenseConstraints: Versionseinschränkungen für Module.

Ein Objekt des Typs LicenseConstraints besitzt folgende Eigenschaften:

- **image_version** (*LicenseComponentConstraint*) - siehe Beschreibung von *LicenseComponentConstraint*

Musterobjekt


```
{
  "image_version": {
    "max_version": "string",
    "min_version": "string"
  }
}
```


LicenseConstraints-Objekte sind in *LicenseInfo* enthalten.

LicenseInfo: Informationen zur aktuell auf dem Gerät angewandten Softwarelizenz.

Ein Objekt des Typs LicenseInfo besitzt folgende Eigenschaften:

- **components** (*LicenseComponents*): siehe Beschreibung von *LicenseComponents*.
- **components_constraints** (*LicenseConstraints*) - siehe Beschreibung von *LicenseConstraints*
- **valid** (boolean): Angabe, ob eine Lizenz gültig ist oder nicht.

Musterobjekt


```
{
  "components": {
    "hand_eye_calibration": false,
    "rectification": false,
    "stereo": false
  },
  "components_constraints": {
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)


```
    "image_version": {
      "max_version": "string",
      "min_version": "string"
    }
  },
  "valid": false
}
```


LicenseInfo-Objekte werden in folgenden Anfragen verwendet:

- *GET /system/license*

Log: Inhalt einer bestimmten Logdatei im JSON-Format.

Ein Objekt des Typs Log besitzt folgende Eigenschaften:

- **date** (float): UNIX-Uhrzeit, zu der das Log zuletzt geändert wurde.
- **log** (*LogEntry*-Array): die eigentlichen Logeinträge.
- **name** (string): Name der Logdatei.
- **size** (Integer): Größe der Logdatei in Bytes.

Musterobjekt


```
{
  "date": 0,
  "log": [
    {
      "component": "string",
      "level": "string",
      "message": "string",
      "timestamp": 0
    },
    {
      "component": "string",
      "level": "string",
      "message": "string",
      "timestamp": 0
    }
  ],
  "name": "string",
  "size": 0
}
```


Log-Objekte werden in folgenden Anfragen verwendet:

- *GET /logs/{log}*

LogEntry: Darstellung eines einzelnen Logeintrags in einer Logdatei.

Ein Objekt des Typs LogEntry besitzt folgende Eigenschaften:

- **component** (string): Name des Moduls, das diesen Eintrag angelegt hat.
- **level** (string): Logstufe (mögliche Werte: `DEBUG`, `INFO`, `WARN`, `ERROR` oder `FATAL`)
- **message** (string): eigentliche Lognachricht.
- **timestamp** (float): UNIX-Uhrzeit des Logeintrags.

Musterobjekt


```
{
  "component": "string",
  "level": "string",
```


(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  "message": "string",
  "timestamp": 0
}
```

LogEntry-Objekte sind in *Log* enthalten.

LogInfo: Informationen zu einer bestimmten Logdatei.

Ein Objekt des Typs LogInfo besitzt folgende Eigenschaften:

- **date** (float): UNIX-Uhrzeit, zu der das Log zuletzt geändert wurde.
- **name** (string): Name der Logdatei.
- **size** (Integer): Größe der Logdatei in Bytes.

Musterobjekt

```
{
  "date": 0,
  "name": "string",
  "size": 0
}
```

LogInfo-Objekte werden in folgenden Anfragen verwendet:

- *GET /logs*

NodeInfo: Beschreibung eines auf dem Gerät laufenden Softwaremoduls.

Ein Objekt des Typs NodeInfo besitzt folgende Eigenschaften:

- **name** (string): Name des Moduls.
- **parameters** (string-Array): Liste der Laufzeitparameter des Moduls.
- **services** (string-Array): Liste der von diesem Modul angebotenen Services.
- **status** (string): Status des Moduls (mögliche Werte: `unknown`, `down`, `idle` oder `running`).

Musterobjekt

```
{
  "name": "string",
  "parameters": [
    "string",
    "string"
  ],
  "services": [
    "string",
    "string"
  ],
  "status": "string"
}
```

NodeInfo-Objekte werden in folgenden Anfragen verwendet:

- *GET /nodes*
- *GET /nodes/{node}*
- *GET /pipelines/{pipeline}/nodes*
- *GET /pipelines/{pipeline}/nodes/{node}*

NodeStatus: Detaillierter aktueller Status des Moduls, einschließlich Laufzeitstatistik.

Ein Objekt des Typs NodeStatus besitzt folgende Eigenschaften:

- **status** (string): Status des Moduls (mögliche Werte: `unknown`, `down`, `idle` oder `running`).
- **timestamp** (float): UNIX-Uhrzeit, zu der die Werte zuletzt aktualisiert wurden.
- **values** (object): Dictionary (Schlüssel-Werte-Auflistung) mit den aktuellen Statuswerten/Statistiken des Moduls.

Musterobjekt


```
{
  "status": "string",
  "timestamp": 0,
  "values": {}
}
```


NodeStatus-Objekte werden in folgenden Anfragen verwendet:

- *GET /nodes/{node}/status*
- *GET /pipelines/{pipeline}/nodes/{node}/status*

Parameter: Darstellung der Laufzeitparameter eines Moduls: Der Datentyp des Werts („value“) eines Parameters (und damit der Datentyp der Felder „min“, „max“ und „default“) lässt sich vom Feld „type“ ableiten und kann ein primitiver Datentyp sein.

Ein Objekt des Typs Parameter besitzt folgende Eigenschaften:

- **default** (Typ nicht definiert): ab Werk voreingestellter Wert des Parameters.
- **description** (string): Beschreibung des Parameters.
- **max** (Typ nicht definiert): Höchstwert, der diesem Parameter zugewiesen werden kann.
- **min** (Typ nicht definiert): Mindestwert, der diesem Parameter zugewiesen werden kann.
- **name** (string): Name des Parameters.
- **type** (string): als Zeichenfolge dargestellter primitiver Datentyp des Parameters (mögliche Werte: `bool`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `int64`, `uint64`, `float32`, `float64` oder `string`).
- **value** (Typ nicht definiert): aktueller Wert des Parameters.

Musterobjekt


```
{
  "default": {},
  "description": "string",
  "max": {},
  "min": {},
  "name": "string",
  "type": "string",
  "value": {}
}
```


Parameter-Objekte werden in folgenden Anfragen verwendet:

- *GET /pipelines/{pipeline}/nodes/{node}/parameters*
- *PUT /pipelines/{pipeline}/nodes/{node}/parameters*
- *GET /pipelines/{pipeline}/nodes/{node}/parameters/{param}*
- *PUT /pipelines/{pipeline}/nodes/{node}/parameters/{param}*

ParameterNameValue: Parametername und -wert. Der Typ des Parameterwerts (Felder ‚value‘ und ‚min‘, ‚max‘, ‚default‘) ist durch das Feld ‚type‘ angegeben und kann einer der eingebauten primitiven Datentypen sein.

Ein Objekt des Typs ParameterNameValue besitzt folgende Eigenschaften:

- **name** (string): Name des Parameters.
- **value** (Typ nicht definiert): aktueller Wert des Parameters.

Musterobjekt


```
{
  "name": "string",
  "value": {}
}
```


ParameterNameValue-Objekte werden in folgenden Anfragen verwendet:

- *PUT /pipelines/{pipeline}/nodes/{node}/parameters*

ParameterValue: Parameterwert. Der Typ des Parameterwerts (Felder ‚value' und ‚min', ‚max', ‚default') ist durch das Feld ‚type' angegeben und kann einer der eingebauten primitiven Datentypen sein.

Ein Objekt des Typs ParameterValue besitzt folgende Eigenschaften:

- **value** (Typ nicht definiert): aktueller Wert des Parameters.

Musterobjekt


```
{
  "value": {}
}
```


ParameterValue-Objekte werden in folgenden Anfragen verwendet:

- *PUT /pipelines/{pipeline}/nodes/{node}/parameters/{param}*

Service: Darstellung eines von einem Modul angebotenen Services.

Ein Objekt des Typs Service besitzt folgende Eigenschaften:

- **args** (*ServiceArgs*): siehe Beschreibung von *ServiceArgs*.
- **description** (string): Kurzbeschreibung des Services.
- **name** (string): Name des Services.
- **response** (*ServiceResponse*): siehe Beschreibung von *ServiceResponse*.

Musterobjekt


```
{
  "args": {},
  "description": "string",
  "name": "string",
  "response": {}
}
```


Service-Objekte werden in folgenden Anfragen verwendet:

- *GET /nodes/{node}/services*
- *GET /nodes/{node}/services/{service}*
- *PUT /nodes/{node}/services/{service}*
- *GET /pipelines/{pipeline}/nodes/{node}/services*
- *GET /pipelines/{pipeline}/nodes/{node}/services/{service}*
- *PUT /pipelines/{pipeline}/nodes/{node}/services/{service}*

ServiceArgs: Argumente, die für den Aufruf eines Services benötigt werden: Diese Argumente werden in der Regel in einem (verschachtelten) Dictionary (Schlüssel-Werte-Auflistung) dargestellt. Der genaue Inhalt dieses Dictionarys hängt vom jeweiligen Modul und vom Serviceaufruf ab.

ServiceArg-Objekte sind in *Service* enthalten.

ServiceResponse: Die von dem Serviceaufruf zurückgegebene Antwort: Die Antwort wird in der Regel in einem (verschachtelten) Dictionary (Schlüssel-Werte-Auflistung) dargestellt. Der genaue Inhalt dieses Dictionarys hängt vom jeweiligen Modul und von dem Serviceaufruf ab.

ServiceResponse-Objekte sind in *Service* enthalten.

SysInfo: Systeminformationen über das Gerät.

Ein Objekt des Typs SysInfo besitzt folgende Eigenschaften:

- **firmware** (*FirmwareInfo*): siehe Beschreibung von *FirmwareInfo*.
- **ready** (boolean): Das System ist vollständig hochgefahren und betriebsbereit.
- **time** (float): Systemzeit als UNIX-Zeitstempel.
- **ui_lock** (*UILock*): siehe Beschreibung von *UILock*
- **uptime** (float): Betriebszeit in Sekunden.

Musterobjekt


```
{
  "firmware": {
    "active_image": {
      "image_version": "string"
    }
  },
  "ready": false,
  "time": 0,
  "ui_lock": {
    "enabled": false
  },
  "uptime": 0
}
```


SysInfo-Objekte werden in folgenden Anfragen verwendet:

- *GET /system*

Template: Template für die Erkennung

Ein Objekt des Typs Template besitzt folgende Eigenschaften:

- **id** (string): Eindeutiger Name des Templates

Musterobjekt


```
{
  "id": "string"
}
```


Template-Objekte werden in folgenden Anfragen verwendet:

- *GET /templates/rc_boxpick*
- *GET /templates/rc_boxpick/{id}*
- *PUT /templates/rc_boxpick/{id}*
- *GET /templates/rc_cadmatch*
- *GET /templates/rc_cadmatch/{id}*
- *PUT /templates/rc_cadmatch/{id}*
- *GET /templates/rc_silhouettematch*
- *GET /templates/rc_silhouettematch/{id}*

- *PUT /templates/rc_silhouettematch/{id}*

UILock: UI Lock Status.

Ein Objekt des Typs UILock besitzt folgende Eigenschaften:

- **enabled** (boolean)

Musterobjekt

```
{
  "enabled": false
}
```

UILock-Objekte sind in *SysInfo* enthalten und werden für folgende Anfragen verwendet:

- *GET /system/ui_lock*

7.2.4 Swagger UI

Die Swagger UI des *rc_reason_stack* ermöglicht es Entwicklern, die REST-API – beispielsweise zu Entwicklungs- und Testzwecken – leicht darzustellen und zu verwenden. Der Zugriff auf `http://<host>/api/` oder auf `http://<host>/api/swagger` (der erste Link leitet automatisch auf den zweiten Link weiter) öffnet eine Vorschau der allgemeinen API-Struktur des *rc_reason_stack*, einschließlich aller *verfügbaren Ressourcen und Anfragen* (Abschnitt 7.2.2). Auf dieser vereinfachten Benutzeroberfläche lassen sich alle Funktionen erkunden und austesten.

Bemerkung: Der Benutzer muss bedenken, dass die Swagger UI des *rc_reason_stack*, auch wenn sie zur Erprobung der REST-API bestimmt ist, eine voll funktionstüchtige Schnittstelle ist. Das bedeutet, dass alle ausgelösten Anfragen tatsächlich bearbeitet werden und den Zustand und/oder das Verhalten des Geräts beeinflussen. Dies gilt insbesondere für Anfragen des Typs `PUT`, `POST` und `DELETE`.

Mithilfe dieser Schnittstelle können alle verfügbaren Ressourcen und Anfragen erprobt werden, indem diese durch Klick auf- und zugeklappt werden. Die folgende Abbildung zeigt ein Beispiel dafür, wie sich der aktuelle Zustand eines Moduls abrufen lässt, indem die erforderlichen Parameter (`pipeline`-Nummer und `node`-Name) ausgefüllt werden und anschließend *Execute* geklickt wird. Daraufhin zeigt die Swagger UI unter anderem den `curl`-Befehl an, der bei Auslösung der Anfrage ausgeführt wurde, sowie den Antworttext, in dem der aktuelle Status des angefragten Moduls in einer Zeichenfolge im JSON-Format enthalten ist.

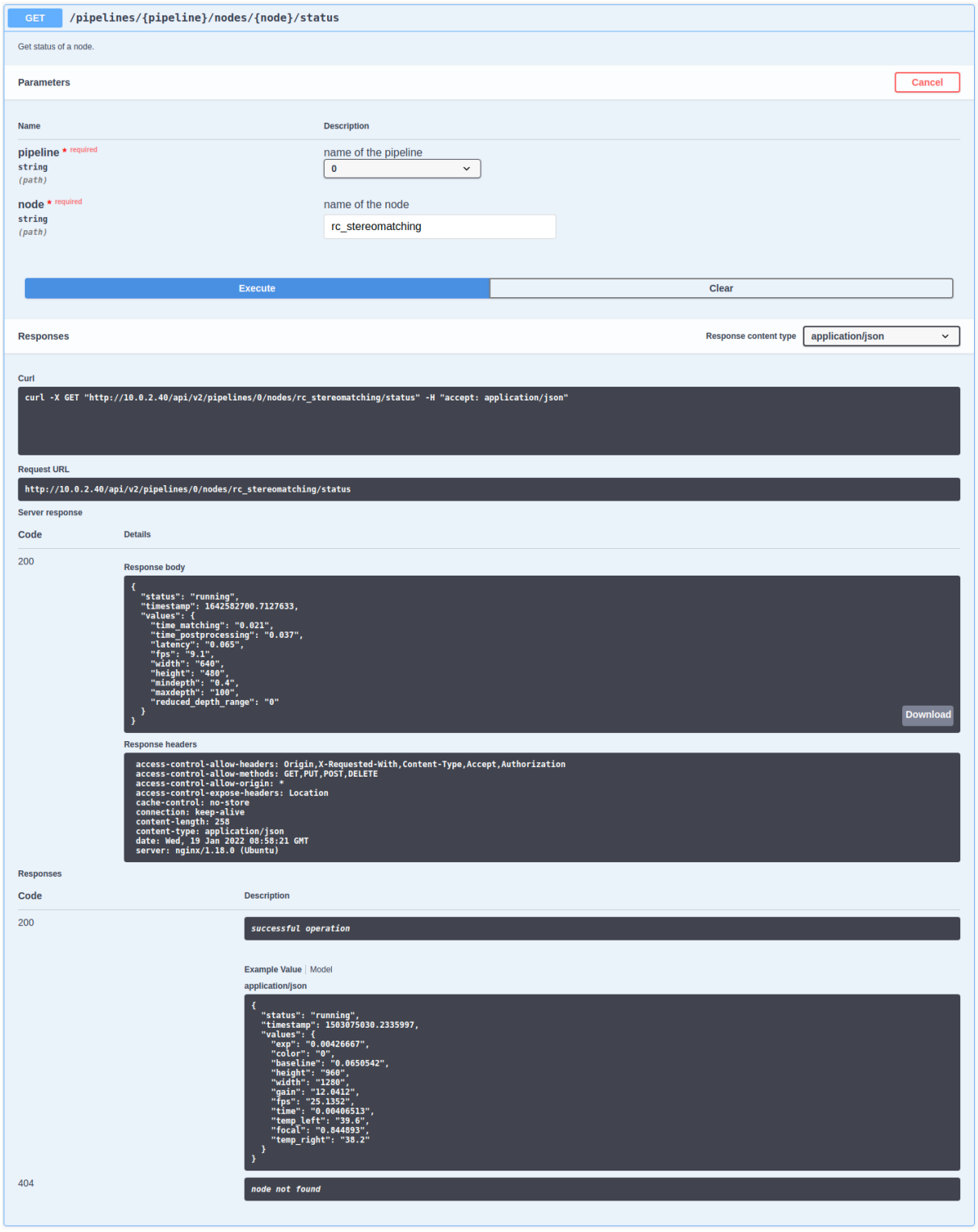


Abb. 7.3: Ergebnis nach Abfrage des Status des `rc_stereomatching`-Moduls

Einige Aktionen, wie das Setzen von Parametern oder der Aufruf von Services, bedürfen komplexerer Parameter als eine HTTP-Anfrage. Die Swagger UI erlaubt es Entwicklern, die für diese Aktionen benötigten Attribute, wie im nächsten Beispiel gezeigt, während der Laufzeit zu erkunden. In der folgenden Abbildung werden die Attribute, die für den `set_pose`-Service des `rc_hand_eye_calibration`-Moduls benötigt werden, erkundet, indem eine `GET`-Anfrage zu dieser Ressource durchgeführt wird. Die Antwort enthält eine vollständige Beschreibung des angebotenen Services, einschließlich aller erforderlichen Argumente mit ihren Namen und Typen in einer Zeichenfolge im JSON-Format.

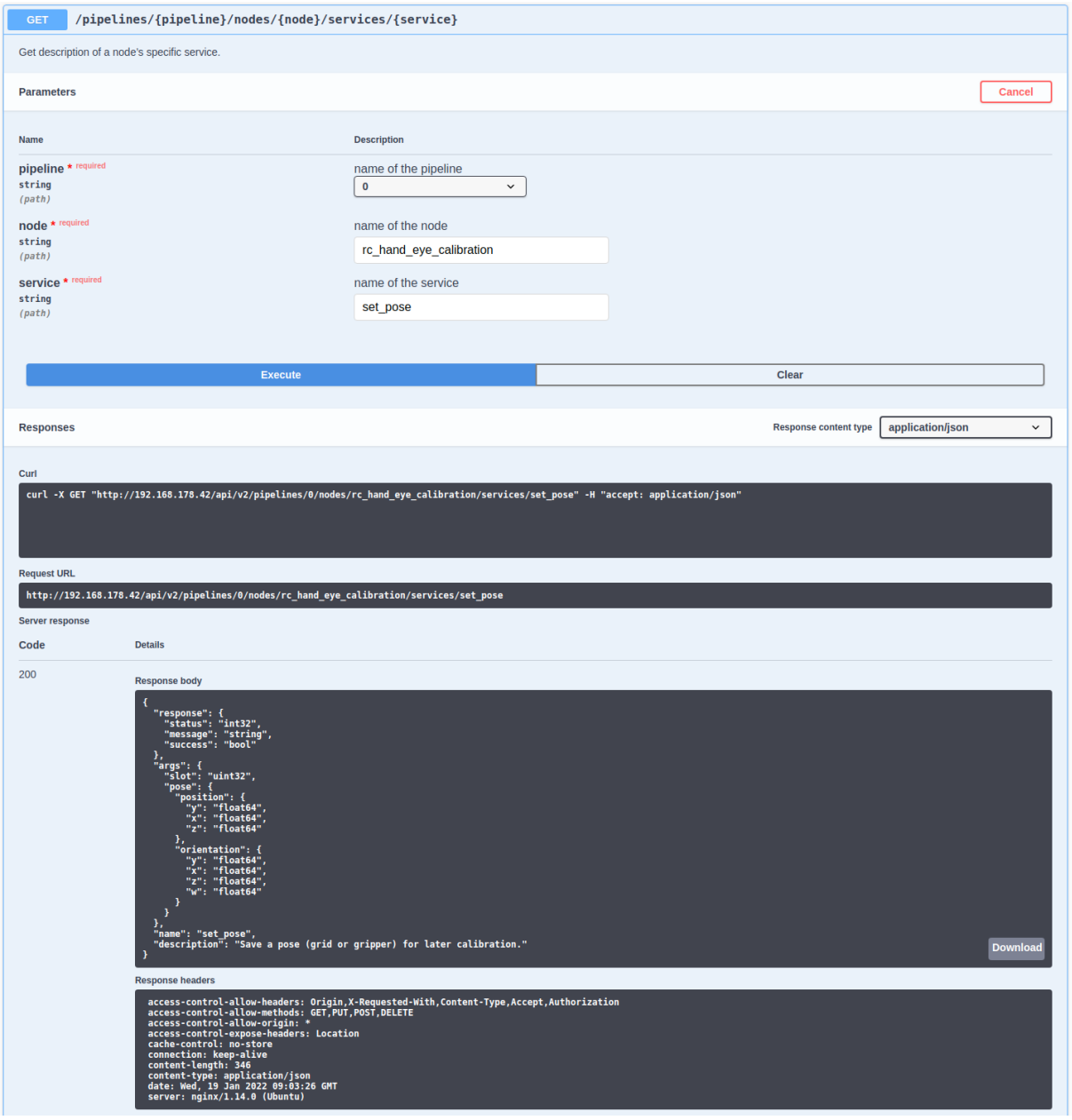


Abb. 7.4: Ergebnis der `GET`-Anfrage zum `set_pose`-Service zeigt die für diesen Service benötigten Argumente

Der Benutzer kann diesen vorformatierten JSON-Text als Muster für die Argumente nutzen, um damit den Service tatsächlich aufzurufen:

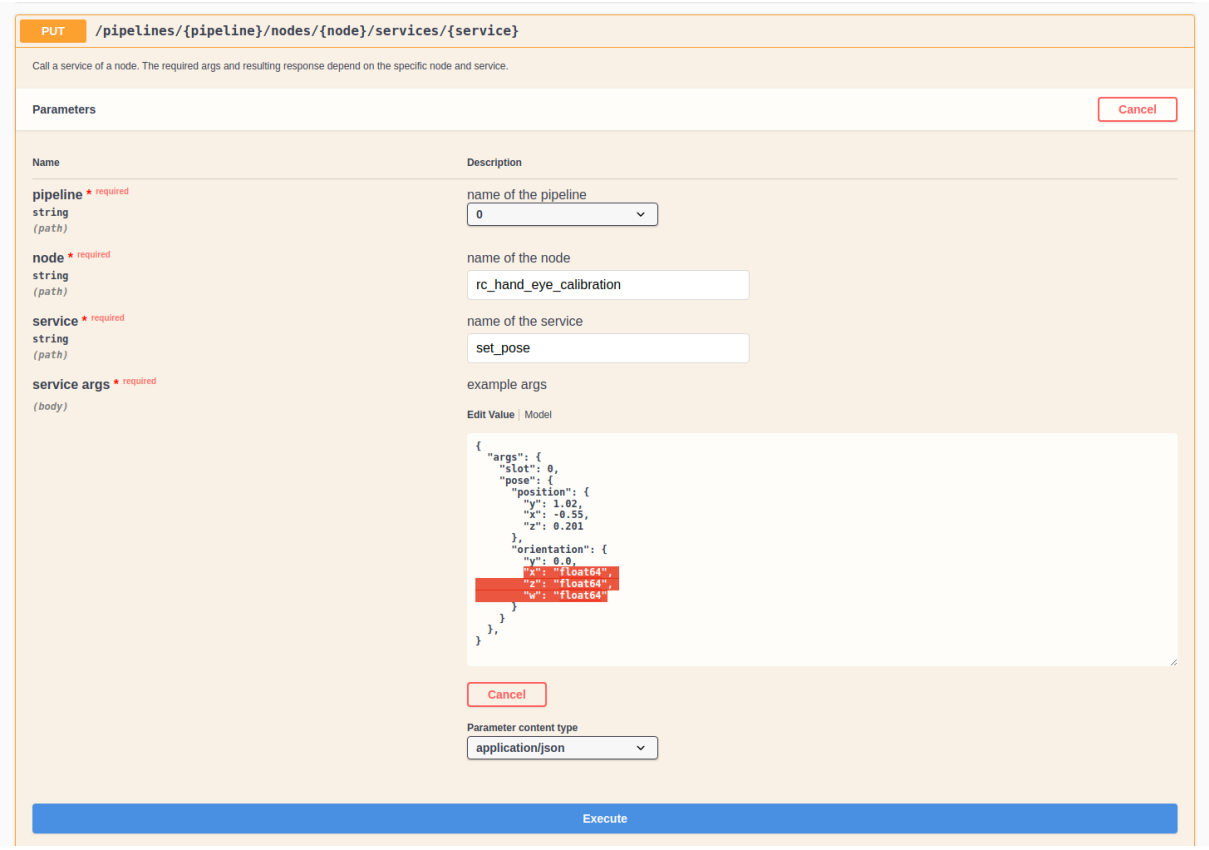


Abb. 7.5: Ausfüllen der Argumente des `set_pose`-Services

7.3 Generic Robot Interface

Das Generic Robot Interface (GRI) ist ein Integrationslayer, der die *REST-API v2* (Abschnitt 7.2) überbrückt und eine standardisierte Kommunikationsschnittstelle zu den Softwaremodulen über eine einfache TCP-Socket-Kommunikation auf Port 7100 bietet.

Das GRI ermöglicht es, Konfigurationen zu erstellen und als nummerierte Jobs zu speichern. Diese Jobs können durch einfache Befehle des Roboters über TCP-Socket-Kommunikation getriggert werden. Das GRI übernimmt intern die REST-API-Kommunikation und liefert die ausgewählten Posenergebnisse in einem roboterspezifisch wählbaren Format.

7.3.1 Job Definition

Jobs sind vorkonfigurierte Aufgaben, die von der Roboteranwendung getriggert werden können. Jeder Job hat eine eindeutige ID und enthält alle notwendigen Informationen für eine bestimmte Operation, z.B. das Berechnen von Greifpunkten für das Bin Picking oder das Ändern von Laufzeitparametern eines Moduls. Einmal konfiguriert, kann der Roboter diese Jobs mit einfachen Socket-Befehlen ausführen und gegebenenfalls die zurückgegebenen Posen empfangen.

7.3.1.1 Job Arten

Das Generic Robot Interface unterstützt drei Arten von Jobs:

Pipeline Service Job (CALL_PIPELINE_SERVICE)

Dieser Job ruft einen Service auf einer bestimmten Kamera-Pipeline auf, um beispielsweise Objekte zu erkennen oder Greifpunkte zu berechnen, und gibt Posendaten an den Roboter zurück (z.B. Greifposen).

Ein Pipeline-Service-Job besteht aus:

- `job_type`: die Art des Jobs `CALL_PIPELINE_SERVICE`
- `name`: Name des Jobs (beschreibender Name zur Unterscheidung von Jobs)
- `pipeline`: die Kamerapipeline, die für den Job verwendet werden soll (z.B. „0“)
- `node`: der REST-API Name der Pipeline-Node die genutzt werden soll (z.B. `rc_load_carrier`)
- `service`: der REST-API Name des Services, der aufgerufen werden soll
- `args`: die REST-API Json Argumente, die dem Service übergeben werden sollen
- `selected_return`: der REST-API Name des Felds, das zurückgeliefert werden soll

Eine Beispieldefinition für einen Pipeline-Service-Job ist:

```
{
   "args": {
      "pose_frame": "external",
      "suction_surface_length": 0.02,
      "suction_surface_width": 0.02
   },
   "job_type": "CALL_PIPELINE_SERVICE",
   "name": "Compute Grasps",
   "node": "rc_itempick",
   "pipeline": "0",
   "selected_return": "grasps",
   "service": "compute_grasps"
}
```

Die verfügbaren Werte für `selected_return` hängen von der gewählten Node (Modul) ab und können z.B. `grasps` oder `matches` sein. Details zu `node`, `service`, `args` und `selected_return` sind in den Servicedefinitionen des entsprechenden Moduls beschrieben.

Globaler Service-Job (CALL_GLOBAL_SERVICE)

Dieser Job ruft einen Service auf, der nicht an eine bestimmte Pipeline gebunden ist, z.B. Datenbank Services zum Festlegen von Regions of Interest oder Load Carriern. Globale Service Jobs geben keine Posen zurück.

Ein globaler Service-Job besteht aus:

- `job_type`: die Art des Jobs `CALL_GLOBAL_SERVICE`
- `name`: Name des Jobs (beschreibender Name zur Unterscheidung von Jobs)
- `node`: der REST-API Name der globalen Node die genutzt werden soll (z.B. `rc_load_carrier_db`)
- `service`: der REST-API Name des Services, der aufgerufen werden soll
- `args`: die REST-API Json Argumente, die dem Service übergeben werden sollen

Eine Beispieldefinition für einen globalen Service-Job ist:

```
{
   "args": {
      "region_of_interest_2d": {
         "id": "2d_roi",
         "width": 526,
         "height": 501,
         "offset_x": 558,
         "offset_y": 307
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      }
   },
   "job_type": "CALL_GLOBAL_SERVICE",
   "name": "Set 2D ROI",
   "node": "rc_roi_db",
   "service": "set_region_of_interest_2d"
}
```

Details zu `node`, `service` und `args` sind in den Servicedefinitionen des entsprechenden Moduls beschrieben.

Parameter-Job (SET_PIPELINE_PARAMETER)

Dieser Job setzt Laufzeitparameter von Pipeline-Nodes, z.B. zum Anpassen von Kamera- oder Detektionsmoduleinstellungen. Parameter-Jobs geben keine Posen zurück.

Ein Parameter-Job besteht aus:

- `job_type`: die Art des Jobs `SET_PIPELINE_PARAMETER`
- `name`: Name des Jobs (beschreibender Name zur Unterscheidung von Jobs)
- `pipeline`: die Kamerapipeline, die für den Job verwendet werden soll (z.B. „0“)
- `node`: der REST-API Name der Pipeline-Node die genutzt werden soll (z.B. `rc_stereomatching`)
- `parameters`: die zu setzenden Parameter als Key-Value Paare

Eine Beispieldefinition für einen Parameter-Job ist:

```
{
   "job_type": "SET_PIPELINE_PARAMETERS",
   "name": "Set Stereo Parameters",
   "node": "rc_stereomatching",
   "parameters": {
      "maxdepth": 2,
      "quality": "High"
   },
   "pipeline": "0"
}
```

Details zu `node` und `parameters` sind in den Laufzeitparameterdefinitionen des entsprechenden Moduls beschrieben.

Die Jobs können über die Web GUI oder über die REST-API definiert werden (siehe *Job und HEC_config API*).

7.3.1.2 Primäre und zugehörige Objekte

Die *primären Objekte* sind die Objekte, die als `selected_return` festgelegt wurden, z.B. die Greifpunkte `grasps`. Die *zugehörigen Objekte* sind dann die gefundenen Objekte `items` oder `matches`, auf denen der zurückgegebene Greifpunkt liegt. Während ein primäres Objekt `grasp` genau ein zugehöriges Objekt `item` oder `match` hat, kann ein primäres Objekt `match` mehrere zugehörige Objekte `grasp` haben.

7.3.1.3 Ausführungsmodi

Das Generic Robot Interface unterstützt zwei Ausführungsmodi zur Optimierung der Roboterzykluszeit:

- Synchrone Ausführung: Der Roboter startet einen Job und wartet auf das erste Ergebnis. Dieser Modus empfiehlt sich, wenn die Ergebnisse sofort benötigt werden.

- Asynchrone Ausführung: Der Roboter startet einen Job und kann mit anderen Vorgängen fortfahren, während der Job im Hintergrund läuft. Der Jobstatus kann abgefragt und Ergebnisse abgerufen werden, sobald diese vorliegen. Dieser Modus maximiert die Effizienz bei langen Erkennungszeiten.

7.3.2 Hand-Auge-Kalibrierung

Für jede Kamera-Pipeline kann eine Hand-Auge-Kalibrierkonfiguration definiert werden, um eine programmgesteuerte Hand-Auge-Kalibrierung mithilfe des GRI zu ermöglichen. Jede Hand-Auge-Kalibrierkonfiguration besteht aus den folgenden Informationen:

- `grid_height`: Höhe des Kalibriermusters in Metern
- `grid_width`: Breite des Kalibriermusters in Metern
- `robot_mounted`: Boolean, das festlegt, ob die Kamera am Roboter montiert ist
- `tcp_offset`: 0 für 6DOF-Roboter. Für 4DOF-Roboter: der vorzeichenbehaftete Offset vom TCP zum Kamerakoordinatensystem (am Roboter montierter Sensor) oder zur sichtbaren Oberfläche des Kalibriermusters (statisch montierter Sensor) entlang der TCP-Rotationsachse in Metern.
- `tcp_rotation_axis`: -1 für 6DOF-Roboter. Für 4DOF-Roboter: Bestimmt die Achse des Roboterkoordinatensystems, um die der Roboter seinen TCP drehen kann (0 wird für X, 1 für Y und 2 für die Z-Achse verwendet).

Nähere Informationen zu diesen Einstellungen und zur Hand-Auge-Kalibrierung im Allgemeinen sind in *Hand-Auge-Kalibrierung* beschrieben.

Die Hand-Auge-Kalibrierkonfigurationen können über die Web GUI oder über die REST-API (siehe *Job und HEC_config API*) gesetzt werden.

7.3.3 Spezifikation des Binären GRI Protokolls

Diese Spezifikation definiert das genaue On-Wire-Format für Client-Server-Nachrichten. Eine Nachricht besteht aus einem festen 8-Byte-Header und einem Body, dessen Layout von der Protokollversion abhängt. Derzeit gibt es nur die Protokollversion 1.

Bemerkung: Alle Mehrbyte-Ganzzahlen sind **Little-Endian**. Die Typen sind `uint8` (8 Bit ohne Vorzeichen), `int16` (16 Bit mit Vorzeichen) und `int32` (32 Bit mit Vorzeichen).

7.3.3.1 Header (8 Bytes)

Tab. 7.1: Header Definition

Feld	Typ	Größe	Beschreibung
Magic Number	uint32	4	ASCII tag *„GRI0“*, Bytes *47 52 49 00* (Little-Endian)
protocol_version	uint8	1	Protokollversion: derzeit *1*
message_length	uint8	1	Gesamte Nachrichtengröße (Bytes), inkl. Header + Body
pose_format	uint8	1	Datenformat für Posen (siehe *Posenformate*)
Aktion	uint8	1	Kommando/Aktion (siehe *Aktionen*)

7.3.3.2 Posenformate

Das GRI verwendet zur Positionsdarstellung immer **Millimeter**. Die folgenden Tabellen zeigen verschiedene Rotationsformate, die passend zur Rotationsdarstellung des verwendeten Roboters ausgewählt werden können. Die Formate sind in Nicht-Euler-Rotationsformate, Tait-Bryan-Euler-Rotationsformate

(alle drei Achsen werden verwendet) und reine Euler-Rotationsformate (erste und letzte Rotationsachse sind identisch) unterteilt.

Tab. 7.2: Nicht-Euler Rotationsformate

Name	Wert	rot_1	rot_2	rot_3	rot_4	Einheit	Beispielroboter
QUAT_WXYZ	1	w	x	y	z	–	ABB
QUAT_XYZW	2	x	y	z	w	–	Fruitcore HORST
AXIS_ANGLE_RAD	3	rx	ry	rz	–	rad	Universal Robots

In der folgenden Notation kennzeichnen Hochstriche aufeinanderfolgende Rotationen im intrinsischen Bezugssystem (z. B. Y‘ = Rotation um die neue Y-Achse nach der ersten Rotation). `_B` und `_F` bestimmen die Reihenfolge der Rotationskomponenten. `F` steht für vorwärts (forward), d.h. die Rotationskomponenten werden in derselben Reihenfolge angegeben, in der die Rotation angewendet wird, und `B` steht für rückwärts (backward), d.h. die Rotationskomponenten werden in umgekehrter Reihenfolge angegeben. `_RAD` und `_DEG` bestimmen, ob die Rotationskomponenten gegebenenfalls in Radian oder Grad angegeben werden. Das Format `EULER_ZYX_B_DEG` bedeutet also, dass die intrinsische Rotationsreihenfolge z-y‘-x‘ ist (zuerst Rotation um die z-Achse, dann Rotation um die neue y-Achse, dann Rotation um die neue x-Achse), die Reihenfolge der Rotationskomponenten rückwärts ist (das erste Rotationselement ist also der Winkel um die x-Achse), und die Winkel in Grad angegeben werden.

Tab. 7.3: Tait-Bryan-Euler-Rotationsformate. Hochstriche zeigen aufeinanderfolgende Rotationen im intrinsischen Koordinatensystem an (z.B. ist Y‘ eine Rotation um die neue Y-Achse nach der ersten Rotation). **_F (Forward)**: *[1., 2., 3.]* | **_B (Backward)**: *[3., 2., 1.]*, **_DEG (degrees): Grad** | **_RAD (radian): Radian**.

Name	Wert	rot_1	rot_2	rot_3	rot_4	Einheit	Beispielroboter
EULER_XYZ_F_DEG	4	X	Y‘	Z“	–	deg	
EULER_XYZ_F_RAD	5	X	Y‘	Z“	–	rad	
EULER_XYZ_B_DEG	6	Z“	Y‘	X	–	deg	
EULER_XYZ_B_RAD	7	Z“	Y‘	X	–	rad	
EULER_XZY_F_DEG	8	X	Z‘	Y“	–	deg	
EULER_XZY_F_RAD	9	X	Z‘	Y“	–	rad	
EULER_XZY_B_DEG	10	Y“	Z‘	X	–	deg	
EULER_XZY_B_RAD	11	Y“	Z‘	X	–	rad	
EULER_YXZ_F_DEG	12	Y	X‘	Z“	–	deg	
EULER_YXZ_F_RAD	13	Y	X‘	Z“	–	rad	
EULER_YXZ_B_DEG	14	Z“	X‘	Y	–	deg	
EULER_YXZ_B_RAD	15	Z“	X‘	Y	–	rad	
EULER_YZX_F_DEG	16	Y	Z‘	X“	–	deg	
EULER_YZX_F_RAD	17	Y	Z‘	X“	–	rad	
EULER_YZX_B_DEG	18	X“	Z‘	Y	–	deg	
EULER_YZX_B_RAD	19	X“	Z‘	Y	–	rad	
EULER_ZXY_F_DEG	20	Z	X‘	Y“	–	deg	
EULER_ZXY_F_RAD	21	Z	X‘	Y“	–	rad	
EULER_ZXY_B_DEG	22	Y“	X‘	Z	–	deg	
EULER_ZXY_B_RAD	23	Y“	X‘	Z	–	rad	
EULER_ZYX_F_DEG	24	Z	Y‘	X“	–	deg	KUKA
EULER_ZYX_F_RAD	25	Z	Y‘	X“	–	rad	
EULER_ZYX_B_DEG	26	X“	Y‘	Z	–	deg	FANUC, Mitsubishi, Yaskawa
EULER_ZYX_B_RAD	27	X“	Y‘	Z	–	rad	

Tab. 7.4: Euler-Rotationsformate. Hochstriche zeigen aufeinanderfolgende Rotationen im intrinsischen Koordinatensystem an (z.B. ist Y‘ eine Rotation um die neue Y-Achse nach der ersten Rotation). **_F (Forward)**: *[1., 2., 3.]* | **_B (Backward)**: *[3., 2., 1.]*, **_DEG (degrees): Grad** | **_RAD (radian): Radian**.

Name	Wert	rot_1	rot_2	rot_3	rot_4	Einheit	Beispielroboter
EULER_XYX_F_DEG	28	X	Y‘	X“	–	deg	
EULER_XYX_F_RAD	29	X	Y‘	X“	–	rad	
EULER_XYX_B_DEG	30	X“	Y‘	X	–	deg	
EULER_XYX_B_RAD	31	X“	Y‘	X	–	rad	
EULER_XZX_F_DEG	32	X	Z‘	X“	–	deg	
EULER_XZX_F_RAD	33	X	Z‘	X“	–	rad	
EULER_XZX_B_DEG	34	X“	Z‘	X	–	deg	
EULER_XZX_B_RAD	35	X“	Z‘	X	–	rad	
EULER_YXY_F_DEG	36	Y	X‘	Y“	–	deg	
EULER_YXY_F_RAD	37	Y	X‘	Y“	–	rad	
EULER_YXY_B_DEG	38	Y“	X‘	Y	–	deg	
EULER_YXY_B_RAD	39	Y“	X‘	Y	–	rad	
EULER_YZY_F_DEG	40	Y	Z‘	Y“	–	deg	
EULER_YZY_F_RAD	41	Y	Z‘	Y“	–	rad	
EULER_YZY_B_DEG	42	Y“	Z‘	Y	–	deg	
EULER_YZY_B_RAD	43	Y“	Z‘	Y	–	rad	
EULER_ZXZ_F_DEG	44	Z	X‘	Z“	–	deg	
EULER_ZXZ_F_RAD	45	Z	X‘	Z“	–	rad	
EULER_ZXZ_B_DEG	46	Z“	X‘	Z	–	deg	
EULER_ZXZ_B_RAD	47	Z“	X‘	Z	–	rad	
EULER_ZYZ_F_DEG	88	Z	Y‘	Z“	–	deg	Kawasaki
EULER_ZYZ_F_RAD	49	Z	Y‘	Z“	–	rad	
EULER_ZYZ_B_DEG	50	Z“	Y‘	Z	–	deg	
EULER_ZYZ_B_RAD	51	Z“	Y‘	Z	–	rad	

Alle Posenkomponenten (Position **und** Rotation) sind `int32` mit 1.000.000 skaliert.

- Float zu Int: `int = round(float * 1000000)`
- Int zu Float: `float = int / 1000000.0`
- Positionen werden **vor der Skalierung** in Millimetern erwartet.
- Winkel werden vor der Skalierung in Grad/Radian (je nach Format) erwartet.
- Quaternion-Komponenten haben keine Einheit, verwenden aber dieselbe Skalierung.
- `rot_4` ist bei Euler oder Axis-Angle format ungenutzt (auf 0 gesetzt)

7.3.3.3 Aktionen

Die folgenden Aktionen können gesendet werden.

Tab. 7.5: GRI Aktionen

Name	Wert	Beschreibung
STATUS	1	Ready-Zustand des Systems abfragen; schreibt den Ready-Zustand auf `data_2` (1 oder 0)
TRIG-GER_JOB_SYNC	2	Führt einen Job synchron aus
TRIG-GER_JOB_ASYNC	3	Startet einen Job asynchron
GET_JOB_STATUS	4	Abfrage des Jobstatus (siehe *Jobstatus*)
GET_NEXT_POSE	5	Abfrage des nächsten verfügbaren Ergebnisses
GET_RELATED_POSE	6	Abtrage der nächsten zugehörigen Pose
HEC_INIT	7	Hand-Auge-Kalibrierung initialisieren
HEC_SET_POSE	8	Kalibrierpose abspeichern
HEC_CALIBRATE	9	Kalibrierung durchführen und Ergebnis speichern

STATUS (1)

Liefert den Ready-Zustand des *rc_reason_stack* in `data_2` (1 wenn ready, 0 wenn nicht).

TRIGGER_JOB_SYNC (2)

Führt den Job aus und gibt sofort das **erste** Ergebnis zurück. Weitere Ergebnisse werden für einen späteren Abruf gespeichert. Wenn der Job erfolgreich ist und Ergebnisse zurückliefert, ist der `error_code` Null und die Pose befüllt. Wenn keine Ergebnisse zurückgeliefert werden, liefert der `error_code` den Wert `NO_POSES_FOUND` (positiver Wert als Warnung). Außerdem wird Folgendes gemeldet:

- `data_1` = `return_code` Wert der Node
- `data_2` = Anzahl der verbleibenden *primären Objekte* (siehe *Primäre und zugehörige Objekte*)
- `data_3` = Anzahl der verbleibenden *zugehörigen Objekte* (ref. *Primäre und zugehörige Objekte*)

TRIGGER_JOB_ASYNC (3)

Startet den Job und endet sofort. Der Status des Jobs kann mit `GET_JOB_STATUS` (4) (siehe *Jobstatus*) abgefragt und die Ergebnisse mit `GET_NEXT_POSE` (5) abgerufen werden, sobald der Job abgeschlossen (`DONE`) ist

GET_JOB_STATUS (4)

Liefert den Jobstatus. Es wird gemeldet:

- `data_1` = `return_code` Wert der Node
- `data_2` = Jobstatus (siehe Tabelle *Job Statuswerte*)

Fehlerdetails sind in `error_code` enthalten.

GET_NEXT_POSE (5)

Gibt das nächste Ergebnis des *primären Objekts* zurück. Außerdem wird Folgendes gemeldet:

- `data_1` = `return_code` Wert der Node
- `data_2` = Anzahl der verbleibenden *primären Objekte* (siehe *Primäre und zugehörige Objekte*)
- `data_3` = Anzahl der verbleibenden *zugehörigen Objekte* (ref. *Primäre und zugehörige Objekte*)

Wenn keine primären Objekte mehr verfügbar sind, wird `NO_POSES_FOUND` zurückgegeben und der Job zurückgesetzt.

GET_RELATED_POSE (6)

Gibt die nächste Pose des *zugehörigen Objekts* zum aktuellen *primären Objekt* zurück. Außerdem wird Folgendes gemeldet:

- `data_1` = `return_code` Wert der Node

- `data_2` = Anzahl der verbleibenden *primären Objekte* (siehe *Primäre und zugehörige Objekte*)
- `data_3` = Anzahl der verbleibenden *zugehörigen Objekte* (ref. *Primäre und zugehörige Objekte*)

Wenn keine zugehörigen Posen gefunden wurden, wird `NO_RELATED_POSES` zurückgeliefert.

HEC_INIT (7)

Diese Aktion initialisiert die Hand-Auge-Kalibrierung. Sie löscht existierende Kalibrierdaten, wendet die Hand-Auge-Kalibrierkonfiguration der Pipeline an und bereitet das System zum Aufnehmen von Kalibrierposen vor. Der Wert in `data_1` gibt die Zielpipeline für die Kalibrierung an.

HEC_SET_POSE (8)

Diese Aktion wird achtmal verwendet, um unterschiedliche Roboterposen mit sichtbarem Kalibriermuster aufzuzeichnen. Das Feld `data_2` dient zur Angabe des Bildspeicherplatzes (Slot) (0-7). Eine vorherige Pose in einem Slot wird überschrieben, wenn dieser wiederverwendet wird. Jede Pose muss eine andere Ansicht des Kalibriermusters liefern, wie in *Hand-Auge-Kalibrierung* beschrieben. Der Inhalt von `data_1` gibt die Zielpipeline an.

HEC_CALIBRATE (9)

Diese Aktion verarbeitet alle aufgezeichneten Posen und berechnet die Transformation zwischen Kamera und Roboter. Erfolgreiche Kalibrierergebnisse werden automatisch gespeichert. Der Inhalt von `data_1` gibt die Zielpipeline an.

7.3.3.4 Jobstatus

Die folgenden Statuswerte für Jobs können zurückgeliefert werden.

Tab. 7.6: Job Statuswerte

Name	Wert
INACTIVE	1
RUNNING	2
DONE	3
FAILED	4

7.3.3.5 Body Definitionen

Es gibt unterschiedliche Body-Definitionen, je nachdem, ob eine Anfrage gesendet oder eine Antwort empfangen wird. Der Anfrage-Body besteht aus insgesamt 54 Bytes und seine Definition ist in der Tabelle *Anfrage-Body Definition* angegeben.

Tab. 7.7: Anfrage-Body Definition

Feld	Typ	Größe	Beschreibung
Header	struct	8	Nachrichtenheader (siehe *Header (8 Bytes)*)
job_id	uint16	2	Eindeutige Job ID aus der Job Konfiguration
pos_x	int32	4	Position X (skaliert imt 10^6)
pos_y	int32	4	Position Y (skaliert imt 10^6)
pos_z	int32	4	Position Z (skaliert imt 10^6)
rot_1	int32	4	Rotationskomponente 1 (skaliert mit 10^6)
rot_2	int32	4	Rotationskomponente 2 (skaliert mit 10^6)
rot_3	int32	4	Rotationskomponente 3 (skaliert mit 10^6)
rot_4	int32	4	Rotationskomponente 4 (skaliert mit 10^6)
data_1	int32	4	Zusätzlicher Parameter 1
data_2	int32	4	Zusätzlicher Parameter 2
data_3	int32	4	Zusätzlicher Parameter 3
data_4	int32	4	Zusätzlicher Parameter 4

Die Job-ID ist die eindeutige Kennung aus der Job-Konfiguration. Die Verwendung der Felder `data_1. ..data_4` ist abhängig von der Aktion und vom Job. Bei Nichtverwendung werden sie auf *0* gesetzt.

Der Antwort-Body besteht aus insgesamt 80 Bytes, Seine Definition ist in Tabelle *Antwort-Body Definition* angegeben.

Tab. 7.8: Antwort-Body Definition

Feld	Typ	Größe	Beschreibung
Header	struct	8	Protokollheader
job_id	uint16	2	Verarbeitete Job Nummer
error_code	int16	2	GRI Ergebnisstatus (Schweregrad nach Vorzeichen)
pos_x	int32	4	Position X (skaliert imt 10^6)
pos_y	int32	4	Position Y (skaliert imt 10^6)
pos_z	int32	4	Position Z (skaliert imt 10^6)
rot_1	int32	4	Rotationskomponente 1 (skaliert mit 10^6)
rot_2	int32	4	Rotationskomponente 2 (skaliert mit 10^6)
rot_3	int32	4	Rotationskomponente 3 (skaliert mit 10^6)
rot_4	int32	4	Rotationskomponente 4 (skaliert mit 10^6)
data_1	int32	4	Rückgabecode der Node (0 wenn keiner)
data_2	int32	4	Zusätzliches Ergebnis 2
data_3	int32	4	Zusätzliches Ergebnis 3
data_4	int32	4	Zusätzliches Ergebnis 4
data_5	int32	4	Zusätzliches Ergebnis 5
data_6	int32	4	Zusätzliches Ergebnis 6
data_7	int32	4	Zusätzliches Ergebnis 7
data_8	int32	4	Zusätzliches Ergebnis 8
data_9	int32	4	Zusätzliches Ergebnis 9
data_10	int32	4	Zusätzliches Ergebnis 10

Bemerkung: Für `rc_measure` wird `mean_z` auf pos_x/pos_y/pos_z ausgegeben.

7.3.3.6 Fehlercodes und Bedeutung

Der Fehlercode `error_code` ist ein `int16` und kodiert Fehler/Warnungen durch Vorzeichen:

- Negativ *< 0* = **error** (Fehler)
- Null = *0* = **success** (Erfolg)
- Positiv *> 0* = **warning** (Erfolg mit Warnung)

Die folgenden Tabellen geben die verschiedenen Fehlercodes an und sind nach Vorzeichen aufgeteilt und sortiert.

Erfolg

Name	Wert	Beschreibung
NO_ERROR	0	Verarbeitung erfolgreich

Negative Fehlercodes

Name	Wert	Beschreibung
UNKNOWN_ERROR	-1	GRI intern, nicht spezifiziert
INTERNAL_ERROR	-2	GRI interner Systemfehler
API_NOT_REACHABLE	-3	API nicht erreichbar
API_RESPONSE_ERROR	-4	API hat negativen Code zurückgeliefert
PIPELINE_NOT_AVAILABLE	-5	Pipeline nicht verfügbar
INVALID_REQUEST_ERROR	-6	Fehlerhafte Anfrage
INVALID_REQUEST_LENGTH	-7	Falsche Nachrichtenlänge
INVALID_ACTION	-8	Nicht unterstützte Aktion
PROCESSING_TIMEOUT	-9	Timeout während der Verarbeitung
UNKNOWN_PROTOCOL_VERSION	-10	Protokollversion nicht unterstützt
WRONG_PROTOCOL_FOR_JOB	-11	Job passt nicht zur Protokollversion
JOB_DOES_NOT_EXIST	-12	Invalid job ID
MISCONFIGURED_JOB	-13	Ungültige Job Konfiguration
HEC_CONFIG_ERROR	-14	Ungültige Konfigurationsparameter
HEC_INIT_ERROR	-15	Initialisierung der Kalibrierung fehlgeschlagen
HEC_SET_POSE_ERROR	-16	Pose konnte nicht in angegebenem Slot aufgenommen werden
HEC_CALIBRATE_ERROR	-17	Kalibrierung konnte aus aufgenommenen Posen nicht berechnet werden
HEC_INSUFFICIENT_DETECTION	-18	Kalibriermuster nicht sichtbar oder nicht erkannt

Positive Codes

Name	Wert	Beschreibung
NO_POSES_FOUND	1	Keine Ergebnisse verfügbar
NO_RELATED_POSES	2	Keine zugehörigen Objekte gefunden
NO_RETURN_SPECIFIED	3	Job ohne Rückgabewerte konfiguriert
JOB_STILL_RUNNING	4	Asynchroner Job nicht beendet

Node Rückgabecode Bedeutung

Die Module/Nodes können einen `return_code` zurückgeben. Dieser Node-Rückgabecode wird im Antwortfeld `data_1` platziert (standardmäßig 0, wenn kein Code vorhanden ist). Der primäre Status des GRI wird in `error_code` zurückgegeben (vorzeichenbasierte Bedeutung).

7.3.4 Integration mit einem Roboter

Die Generic Robot Interface bietet die Kommunikation auf Port 7100 an.

Für die Integration der GRI-Kommunikation mit einem Roboter werden Beispiele für verschiedene Robotersprachen unter https://github.com/roboception/rc_generic_robot_interface_robot angeboten.

Unterschiedliche Roboterplattformen können durch die Implementierung eines TCP-Socket-Clients unterstützt werden, der dem GRI-Binärprotokoll folgt (siehe *Spezifikation des Binären GRI Protokolls*). Dies erfordert einen Robotercontroller mit TCP/IP-Unterstützung und der Fähigkeit, Roboterposen in Binärnachrichten zu packen und Binärnachrichten in Roboterposen zu parsen.

Die Implementierungsschritte sind wie folgt:

1. TCP Socketverbindung aufbauen
2. Anfragenachricht zusammenstellen:
   - Nachrichtenheader setzen (8 Bytes)
   - Job ID setzen (2 Bytes)

- Position verpacken (12 Bytes, 3x int32)
- Rotation verpacken (16 Bytes, 4x int32)
- Zusätzliche Daten verpacken (16 Bytes, 4x int32)

3. Anfrage senden (54 Bytes insgesamt)
4. Antwort empfangen (80 Bytes insgesamt)
5. Antwort parsen:
   - Header (8 Bytes)
   - Job ID (2 Bytes)
   - Fehlercode (2 Bytes)
   - Position (12 Bytes, 3x int32)
   - Rotation (16 Bytes, 4x int32)
   - Zusätzliche Daten (40 bytes, 10x int32)

7.3.4.1 Byte-Interpretation in der Socket-Kommunikation

Einige Skriptsprachen für Roboter interpretieren einzelne Socket-Bytes als vorzeichenbehaftete Werte im Bereich `[-128, 127]` anstatt als vorzeichenlose Werte im Bereich `[0, 255]`. Falls dies der Fall ist, muss jedes Byte **vor** der Rekonstruktion von `int32`-Werten in einen vorzeichenlosen Wert konvertiert werden.

```
# Convert signed byte to unsigned
if byte_value < 0:
    byte_value = byte_value + 256
```

Nach der Konvertierung muss der `int32` Wert in Little-Endian-Byte-Reihenfolge rekonstruiert werden. Anschließend wird die Vorzeicheninterpretation auf das höchstwertige Byte (most significant byte, MSB) angewendet, um festzustellen, ob der Gesamtwert des `int32` negativ ist.

Bemerkung: Alle Posenkomponenten verwenden die in *Posenformate* beschriebene Skalierung.

7.3.5 Job und HEC_config API

Die Jobdefinitionen und die Definitionen von HEC_configs für die Hand-Auge-Kalibrierung können über die folgenden REST-API-Endpunkte gesetzt, abgerufen und gelöscht werden.

`GET /generic_robot_interface/hec_configs`
Liefert die definierten Hand-Auge-Kalibrierkonfigurationen zurück

Musteranfrage

```
GET /api/v2/generic_robot_interface/hec_configs HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "0": {
    "grid_height": 0.18,
    "grid_width": 0.26,
    "robot_mounted": true,
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "tcp_offset": 0,
    "tcp_rotation_axis": -1
  }
}
```

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

GET /generic_robot_interface/hec_configs/{pipeline}
Liefert die Hand-Auge-Kalibrierkonfiguration für die ausgewählte Pipeline

Musteranfrage

```
GET /api/v2/generic_robot_interface/hec_configs/<pipeline> HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "grid_height": 0.18,
  "grid_width": 0.26,
  "robot_mounted": true,
  "tcp_offset": 0,
  "tcp_rotation_axis": -1
}
```

Parameter

- **pipeline** (*string*) – Pipeline der Hand-Auge-Kalibrierkonfiguration *(obligatorisch)*

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

PUT /generic_robot_interface/hec_configs/{pipeline}
Setzt eine Hand-Auge-Kalibrierkonfiguration für die ausgewählte Pipeline.

Musteranfrage

```
PUT /api/v2/generic_robot_interface/hec_configs/<pipeline> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "return_code": {
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    "message": "HEC configuration saved successfully",
    "value": 0
  }
}
```

Parameter

- **pipeline** (*string*) – Pipeline der Hand-Auge-Kalibrierkonfiguration *(obligatorisch)*

JSON-Objekt zur Anfrage

- **hand-eye calibration configuration** (*object*) – Beispielargumente *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

DELETE /generic_robot_interface/hec_configs/{pipeline}

Entfernt eine Hand-Auge-Kalibrierkonfiguration.

Musteranfrage

```
DELETE /api/v2/generic_robot_interface/hec_configs/<pipeline> HTTP/1.1
Accept: application/json application/ubjson
```

Parameter

- **pipeline** (*string*) – Pipeline der Hand-Auge-Kalibrierkonfiguration *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Konfiguration für die angegebene Pipeline nicht gefunden

GET /generic_robot_interface/jobs

Liefert die definierten Jobs zurück

Musteranfrage

```
GET /api/v2/generic_robot_interface/jobs HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "0": {
    "args": {
      "pose_frame": "external",
      "tags": []
    },
    "job_type": "CALL_PIPELINE_SERVICE",
    "name": "detect_qr_code",
    "node": "rc_qr_code_detect",
    "pipeline": "0",
    "selected_return": "tags",
    "service": "detect"
  },
  "1": {
    "job_type": "SET_PARAMETERS_SERVICE",
    "name": "set_depth_full_quality",
    "node": "rc_stereomatching",
    "parameters": {
      "double_shot": true,
      "quality": "Full"
    },
    "pipeline": "0"
  }
}
```

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

GET /generic_robot_interface/jobs/{job_id}

Liefert die Definition des ausgewählten Jobs zurück

Musteranfrage

```
GET /api/v2/generic_robot_interface/jobs/<job_id> HTTP/1.1
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "args": {
    "pose_frame": "camera",
    "tags": []
  },
  "job_type": "CALL_PIPELINE_SERVICE",
  "name": "detect_qr_code",
  "node": "rc_qr_code_detect",
  "pipeline": "0",
  "selected_return": "tags",
  "service": "detect"
}
```

Parameter

- **job_id** (*string*) – ID des Jobs *(obligatorisch)*

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

PUT /generic_robot_interface/jobs/{job_id}

Legt eine Jobdefinition für die ausgewählte Job-Art fest. Die erforderlichen Felder hängen vom gewählten Job-Art ab.

Musteranfrage

```
PUT /api/v2/generic_robot_interface/jobs/<job_id> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Musterantwort

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "job_id": "1",
  "return_code": {
    "message": "Job configuration updated successfully",
    "value": 0
  }
}
```

Parameter

- **job_id** (*string*) – ID des Jobs *(obligatorisch)*

JSON-Objekt zur Anfrage

- **job definition** (*object*) – Beispielargumente *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung

DELETE /generic_robot_interface/jobs/{job_id}

Entfernt eine Jobdefinition

Musteranfrage

```
DELETE /api/v2/generic_robot_interface/jobs/<job_id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameter

- **job_id** (*string*) – ID des Jobs *(obligatorisch)*

Anfrage-Header

- Accept – application/json application/ubjson

Antwort-Header

- Content-Type – application/json application/ubjson

Statuscodes

- 200 OK – Erfolgreiche Verarbeitung
- 403 Forbidden – Verboten, z.B. weil keine gültige Lizenz für das CADMatch-Modul vorliegt.
- 404 Not Found – Job mit angegebener ID nicht gefunden

7.4 OPC UA Interface

Der *rc_reason_stack* bietet auch ein optionales OPC UA Interface auf dem TCP Port 4840 an. Der OPC UA Server kann über eine separate *Lizenz* (Abschnitt 8.2) aktiviert werden.

Der OPC UA Server ermöglicht Zugriff auf die Parameter und Services aller verfügbaren Softwaremodule analog zur REST-API. Um den OPC UA Adressraum zu durchsuchen kann z.B. der frei erhältliche UAExpert GUI Client verwendet werden.

Der OPC UA Server nutzt das `DataTypeDefinition` Attribut (verfügbar in OPC UA Version 1.04) für benutzerdefinierte Datentypen und verwendet auch Methoden und Arrays variabler Länge. Bitte überprüfen Sie, ob Ihr OPC UA Client dies unterstützt.

Bemerkung: Der OPC UA Server unterstützt aktuell nur das Äquivalent zu API Version 1 (d.h. nur Kamerapipeline 0).

Bitte kontaktieren Sie support@roboception.de wenn Sie Interesse haben den OPC UA Server zu nutzen.

7.5 KUKA Ethernet KRL Schnittstelle

Der *rc_reason_stack* stellt ein Ethernet KRL Interface (EKI-Bridge) zur Verfügung, welches eine Kommunikation von KUKA KRL via KUKA.EthernetKRL XML mit dem *rc_reason_stack* erlaubt.

Bemerkung: Dieses Modul ist optional und benötigt eine gesonderte EKIBridge-*Lizenz* (Abschnitt 8.2).

Bemerkung: Das KUKA.EthernetKRL add-on Software-Paket Version 2.2 bis maximal 5.x muss auf der Robotersteuerung aktiviert sein, um dieses Modul zu benutzen.

Die EKI-Bridge kann benutzt werden, um programmatisch

- Serviceanfragen auszuführen, z.B. um individuelle Module zu starten und stoppen, oder um angebotene Services wie z.B. die Hand-Auge-Kalibrierung oder Berechnung von Greifposen zu nutzen,
- Laufzeitparameter abzufragen und zu ändern, z.B. der Kamera oder Disparitätsberechnung.

Bemerkung: Eine bekannte Einschränkung der EKI Bridge ist, dass Strings, die valide Zahlen darstellen, nach int/float konvertiert werden. Daher sollten benutzerdefinierte Namen (wie ROI IDs, etc.) immer mindestens einen Buchstaben enthalten, sodass diese als Serviceargumente benutzt werden können.

7.5.1 Konfiguration der Ethernet-Verbindung

Die EKI-Bridge hört auf Port 7000 auf EKI-XML-Nachrichten und übersetzt diese transparent zur *rc_reason_stack REST-API v2* (Abschnitt 7.2). Die empfangenen EKI-Nachrichten werden in JSON umgewandelt und an die *rc_reason_stack* REST-API weitergeleitet. Die Antwort der REST-API wird anschließend zurück in EKI-XML gewandelt.

Die EKI-Bridge erlaubt den Zugriff auf Laufzeitparameter und Services aller Module, die in *Softwaremodule* (Abschnitt 6) beschrieben sind.

Die Ethernet-Verbindung zum *rc_reason_stack* wird auf der Robotersteuerung mit XML-Dateien konfiguriert.

Die Ethernet-Verbindung zum *rc_reason_stack* wird auf der Robotersteuerung mit XML-Dateien konfiguriert. Die EKI-XML-Konfigurationsdateien aller Module auf dem *rc_reason_stack* können hier heruntergeladen werden:

https://doc.rc-visard.com/latest/de/eki.html#eki-xml-configuration-files

Für jedes Softwaremodul, das Laufzeitparameter anbietet, gibt es eine XML-Konfigurationsdatei, um die Parameter abzufragen und zu setzen. Diese sind nach dem Schema `<node_name>-parameters.xml` benannt. Für jeden Service eines Softwaremoduls gibt eine eigene XML-Konfigurationsdatei. Diese ist nach dem Schema `<node_name>-<service_name>.xml` benannt.

Die IP des Host-PC, auf dem der *rc_reason_stack* läuft, muss in der XML Datei eingetragen werden.

Der Port ist bereits auf 7000 gesetzt, was der Pipeline 0 entspricht. Dieser muss angepasst werden falls eine andere Pipeline benutzt werden soll. Die Port Nummer ist 7000 + Pipeline Nummer, also 7001 für Pipeline 1, etc.

Diese Konfigurationsdateien müssen im Verzeichnis `C:\KRC\ROBOTER\Config\User\Common\EthernetKRL` auf der Robotersteuerung abgelegt werden. Sie werden gelesen, sobald eine Verbindung initialisiert wird.

Um z.B. eine Ethernet-Verbindung mit dem Ziel aufzubauen, um die `rc_stereomatching`-Parameter zu konfigurieren, ist der folgende KRL-Code notwendig.

```
DECL EKI_Status RET
RET = EKI_INIT("rc_stereomatching-parameters")
RET = EKI_Open("rc_stereomatching-parameters")

; ----------- Desired operation -----------

RET = EKI_Close("rc_stereomatching-parameters")
```

Bemerkung: Die EKI-Bridge terminiert automatisch die Verbindung zum Client, wenn eine empfangene XML-Nachricht ungültig ist.

7.5.2 Allgemeine XML-Struktur

Für die Datenanfrage nutzt die EKI-Bridge `<req>` als Wurzelelement (kurz für „Request“).

Das Wurzelelement enthält immer die folgenden Elemente.

- `<node>`: Dieses enthält ein Unterelement, über das die EKI-Bridge das Ziel-Softwaremodul identifiziert. Der Modulname ist bereits in der XML-Konfigurationsdatei vorausgefüllt.
- `<end_of_request>`: „End-of-Request“ Flag, das das Ende der Anfrage markiert und diese auslöst.

Die generische XML-Struktur sieht wie folgt aus.

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

Für den Datenempfang nutzt die EKI-Bridge `<res>` als Wurzelelement (kurz für „Response“). Das Wurzelelement enthält immer ein `<return_code>` Unterelement.

```
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

Bemerkung: Standardmäßig ist in den Konfigurationsdateien 998 als Flag angegeben, über welches KRL benachrichtigt wird, sobald eine Antwortnachricht empfangen wurde. Falls dieser Wert bereits in Benutzung ist, sollte dieser in der entsprechenden Konfigurationsdatei geändert werden.

7.5.2.1 Rückgabecode

Das `<return_code>`-Element enthält die Attribute `value` und `message`.

Wie für alle anderen Softwaremodule gibt eine erfolgreiche Anfrage ein `res/return_code/@value` mit dem Wert 0 zurück. Negative Werte geben an, dass die Anfrage fehlgeschlagen ist. Die Fehlermeldung ist in `res/return_code/@message` enthalten. Positive Werte geben an, dass die Anfrage erfolgreich war, aber weitere Informationen in `res/return_code/@message` enthalten sind.

Die folgenden Rückgabecodes können von der EKI-Bridge zurückgegeben werden:

Tab. 7.9: Rückgabecodes der EKI-Bridge

Code	Beschreibung
0	Erfolgreich
-1	Parsing-Fehler in der Konvertierung von XML zu JSON
-2	Interner Fehler
-5	Verbindungsfehler von der REST-API
-9	Fehlende oder ungültige Lizenz für das EKI-Bridge-Modul

Bemerkung: Die EKI-Bridge liefert auch Rückgabecodes spezifisch zu den individuellen Softwaremodulen zurück. Diese sind im jeweiligen *Softwaremodul* (Abschnitt 6) dokumentiert.

Bemerkung: Aufgrund von Limitierungen in KRL ist die maximale Länge eines Strings, der von der EKI-Bridge zurückgegeben wird, auf 512 Zeichen begrenzt. Alle längeren Strings werden gekürzt.

7.5.3 Services

Das XML-Schema für die Services der Softwaremodule wird aus den Argumenten und der Antwort in *JavaScript Object Notation (JSON)* generiert, wie in *Softwaremodule* (Abschnitt 6) beschrieben. Diese Umwandlung ist bis auf die unten beschriebenen Regeln transparent.

Konvertierung von Posen:

Eine Pose ist ein JSON-Objekt, das die Schlüssel `position` und `orientation` enthält.

```
{
  "pose": {
    "position": {
      "x": "float64",
      "y": "float64",
      "z": "float64",
    },
    "orientation": {
      "x": "float64",
      "y": "float64",
      "z": "float64",
      "w": "float64",
    }
  }
}
```

Dieses JSON-Objekt wird zu einem KRL `FRAME` in der XML-Nachricht konvertiert.

```
<pose X="..." Y="..." Z="..." A="..." B="..." C="..."></pose>
```

Positionen werden von Metern in Millimetern umgerechnet und Orientierungen von Quaternionen in das KUKA-ABC-Format (in Grad).

Bemerkung: Es werden in der EKI-Bridge keine anderen Größenumrechnungen vorgenommen. Alle Abmessungen und 3D-Koordinaten, die nicht zu einer Pose gehören, werden in Metern erwartet und zurückgegeben.

Arrays:

Arrays enthalten die Unterelemente `<le>` (kurz für „List Element“). Als Beispiel wird das JSON-Objekt

```
{
  "rectangles": [
    {
      "x": "float64",
      "y": "float64"
    }
  ]
}
```

in das folgende XML-Fragment konvertiert

```
<rectangles>
  <le>
    <x>...</x>
    <y>...</y>
  </le>
</rectangles>
```

XML-Attribute:

Alle JSON-Schlüssel, deren Wert ein primitiver Datentyp ist und die nicht zu einem Array gehören, werden in XML-Attributen gespeichert. Als Beispiel wird das JSON-Objekt

```
{
  "item": {
    "uuid": "string",
    "confidence": "float64",
    "rectangle": {
      "x": "float64",
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
      "y": "float64"
    }
  }
}
```

in das folgende XML-Fragment konvertiert

```
<item uuid="..." confidence="...">
  <rectangle x="..." y="...">
  </rectangle>
</item>
```

7.5.3.1 Anfrage-XML-Struktur

Das <SEND>-Element in der XML-Konfigurationsdatei für einen generischen Service folgt der folgenden Spezifikation:

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/service/<service_name>" Type="STRING"/>
    <ELEMENT Tag="req/args/<argX>" Type="<argX_type>"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

Das <service>-Element hat ein XML-Unterelement, über das die EKI-Bridge den angefragten Service identifiziert. Es ist bereits vorausgefüllt in der Konfigurationsdatei enthalten.

Das <args> Element beinhaltet die Service-Argumente. Diese können jeweils mit der KRL-Instruktion EKI_Set<Type> gesetzt werden.

Beispielsweise sieht das <SEND>-Element des rc_load_carrier_db get_load_carriers Services (siehe *LoadCarrierDB*, Abschnitt 6.5.1) wie folgt aus.

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/rc_load_carrier_db" Type="STRING"/>
    <ELEMENT Tag="req/service/get_load_carriers" Type="STRING"/>
    <ELEMENT Tag="req/args/load_carrier_ids/le" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

Das <end_of_request>-Element erlaubt es, Anfragen mit Arrays zu übermitteln. Um ein Array zu senden, wird die Anfrage in so viele Nachrichten wie Array-Elemente aufgeteilt. Die letzte Nachricht beinhaltet alle XML-Tags inklusive dem <end_of_request>-Flag, während alle anderen Nachrichten jeweils nur ein Array-Element enthalten.

Um z.B. zwei Load-Carrier-Modelle mit dem get_load_carriers Service vom rc_load_carrier_db abzufragen, muss der Nutzer zwei XML-Nachrichten senden. Die erste XML-Nachricht lautet:

```
<req>
  <args>
    <load_carrier_ids>
      <le>load_carrier1</le>
    </load_carrier_ids>
  </args>
</req>
```

Diese Nachricht kann über KRL mit dem `EKI_Send` Kommando gesendet werden, indem das Listenelement als Pfad angegeben wird.

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_load_carrier_db-get_load_carriers", "req/args/load_carrier_ids/
↪le", "load_carrier1")
RET = EKI_Send("rc_load_carrier_db-get_load_carriers", "req/args/load_carrier_ids/le")
```

Die zweite Nachricht beinhaltet alle XML-Tags und löst die Anfrage beim `rc_load_carrier_db` Softwaremodul aus.

```
<req>
  <node>
    <rc_load_carrier_db></rc_load_carrier_db>
  </node>
  <service>
    <get_load_carriers></get_load_carriers>
  </service>
  <args>
    <load_carrier_ids>
      <le>load_carrier2</le>
    </load_carrier_ids>
  </args>
  <end_of_request></end_of_request>
</req>
```

Diese Nachricht kann über KRL gesendet werden, indem `req` als Pfad für `EKI_Send` angegeben wird:

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_load_carrier_db-get_load_carriers", "req/args/load_carrier_ids/
↪le", "load_carrier2")
RET = EKI_Send("rc_load_carrier_db-get_load_carriers", "req")
```

7.5.3.2 Antwort-XML-Struktur

Das `<SEND>`-Element in der XML-Konfigurationsdatei für einen generischen Service folgt der folgenden Spezifikation:

```
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/<resX>" Type="<resX_type>"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

Beispielsweise sieht das `<RECEIVE>`-Element des `rc_april_tag_detect detect` Services (siehe *TagDetect*, Abschnitt 6.3.3) wie folgt aus.

```
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/timestamp/@sec" Type="INT"/>
    <ELEMENT Tag="res/timestamp/@nsec" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/tags/le/pose_frame" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/timestamp/@sec" Type="INT"/>
    <ELEMENT Tag="res/tags/le/timestamp/@nsec" Type="INT"/>
    <ELEMENT Tag="res/tags/le/pose/@X" Type="REAL"/>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    <ELEMENT Tag="res/tags/le/pose/@Y" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@Z" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@A" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@B" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/pose/@C" Type="REAL"/>
    <ELEMENT Tag="res/tags/le/instance_id" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/id" Type="STRING"/>
    <ELEMENT Tag="res/tags/le/size" Type="REAL"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

Bei Arrays beinhaltet die Antwort mehrere Instanzen des gleichen XML-Elements. Jedes Element wird in einen separaten Puffer in EKI geschrieben und kann daraus mit KRL-Instruktionen ausgelesen werden. Die Anzahl an Instanzen (Array-Elementen) kann über `EKI_CheckBuffer` abgefragt werden und jede Instanz mit `EKI_Get<Type>` ausgelesen werden.

Beispielsweise können die Ergebnisposen aus einer Antwort des `rc_april_tag_detect detect` Services in KRL wie folgt ausgelesen werden:

```
DECL EKI_STATUS RET
DECL INT i
DECL INT num_instances
DECL FRAME poses[32]

DECL FRAME pose = {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}

RET = EKI_CheckBuffer("rc_april_tag_detect-detect", "res/tags/le/pose")
num_instances = RET.Buff
for i=1 to num_instances
  RET = EKI_GetFrame("rc_april_tag_detect-detect", "res/tags/le/pose", pose)
  poses[i] = pose
endfor
RET = EKI_ClearBuffer("rc_april_tag_detect-detect", "res")
```

Bemerkung: Vor jeder Anfrage über EKI zum *rc_reason_stack* sollten alle Puffer geleert werden, um sicherzustellen, dass nur die aktuelle Antwort in den EKI-Puffern enthalten ist.

7.5.4 Parameter

Die Parameter aller Softwaremodule können über die EKI-Bridge ausgelesen und gesetzt werden. Die XML-Konfigurationsdatei für ein generisches Softwaremodul folgt dieser Spezifikation:

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/parameters/<parameter_x>/@value" Type="INT"/>
    <ELEMENT Tag="req/parameters/<parameter_y>/@value" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
<RECEIVE>
  <XML>
    <ELEMENT Tag="res/parameters/<parameter_x>/@value" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@default" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@min" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_x>/@max" Type="INT"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@value" Type="REAL"/>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
    <ELEMENT Tag="res/parameters/<parameter_y>/@default" Type="REAL"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@min" Type="REAL"/>
    <ELEMENT Tag="res/parameters/<parameter_y>/@max" Type="REAL"/>
    <ELEMENT Tag="res/return_code/@value" Type="INT"/>
    <ELEMENT Tag="res/return_code/@message" Type="STRING"/>
    <ELEMENT Tag="res" Set_Flag="998"/>
  </XML>
</RECEIVE>
```

Die Anfrage wird als Anfrage zum *Lesen* von Parametern interpretiert, wenn die `value`-Attribute aller Parameter leer sind. Falls mindestens ein `value`-Attribut befüllt ist, wird die Anfrage als Anfrage zum *Setzen* von Parametern interpretiert und die befüllten Parameter gesetzt.

Beispielsweise können die aktuellen Werte aller Parameter von `rc_stereomatching` mit der folgenden XML-Nachricht abgefragt werden:

```
<req>
  <node>
    <rc_stereomatching></rc_stereomatching>
  </node>
  <parameters></parameters>
  <end_of_request></end_of_request>
</req>
```

Diese XML-Nachricht kann folgendermaßen über KRL gesendet werden:

```
DECL EKI_STATUS RET
RET = EKI_Send("rc_stereomatching-parameters", "req")
```

Die Antwort der EKI-Bridge enthält alle Parameter:

```
<res>
  <parameters>
    <acquisition_mode default="Continuous" max="" min="" value="Continuous"/>
    <quality default="High" max="" min="" value="High"/>
    <static_scene default="0" max="1" min="0" value="0"/>
    <seg default="200" max="4000" min="0" value="200"/>
    <smooth default="1" max="1" min="0" value="1"/>
    <fill default="3" max="4" min="0" value="3"/>
    <minconf default="0.5" max="1.0" min="0.5" value="0.5"/>
    <mindepth default="0.1" max="100.0" min="0.1" value="0.1"/>
    <maxdepth default="100.0" max="100.0" min="0.1" value="100.0"/>
    <maxdeptherr default="100.0" max="100.0" min="0.01" value="100.0"/>
  </parameters>
  <return_code message="" value="0"/>
</res>
```

Der `quality`-Parameter von `rc_stereomatching` kann mit folgender XML-Nachricht auf `Low` gesetzt werden:

```
<req>
    <node>
      <rc_stereomatching></rc_stereomatching>
    </node>
    <parameters>
      <quality value="Low"></quality>
    </parameters>
    <end_of_request></end_of_request>
</req>
```

Diese XML-Nachricht kann folgendermaßen über KRL gesendet werden:

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_stereomatching-parameters", "req/parameters/quality/@value",
↪"Low")
RET = EKI_Send("rc_stereomatching-parameters", "req")
```

In diesem Fall wird nur der gesetzte Wert von `quality` zurückgegeben:

```
<res>
  <parameters>
    <quality default="High" max="" min="" value="Low"/>
  </parameters>
  <return_code message="" value="0"/>
</res>
```

7.5.5 Beispielanwendungen

Ausführlichere Beispielanwendungen können unter https://github.com/roboception/eki_examples abgerufen werden.

7.5.6 Fehlerbehebung

SmartPad Fehlermeldung: Limit of element memory reached

Dieser Fehler kann auftreten, wenn die Anzahl der Matches das Speicherlimit überschreitet.

- Erhöhen Sie den Wert BUFFERING und setzen Sie BUFFSIZE in den EKI Konfigurationsdateien. Passen Sie diese Einstellungen an Ihre spezielle KRC an.
- Verringern Sie den Parameter ‚Maximale Matches' im Detektionsmodul.
- Selbst wenn das Gesamtspeicherlimit (BUFFSIZE) einer Nachricht nicht erreicht wird, kann die KRC die Anzahl der Elemente im XML-Baum möglicherweise nicht analysieren, wenn das BUFFERING-Limit zu klein ist. Wenn Ihre Anwendung beispielsweise 50 verschiedene Greifpunkte vorschlägt, muss das BUFFERING-Limit ebenfalls 50 betragen.

7.6 gRPC Bilddatenschnittstelle

Die gRPC Bilddatenschnittstelle kann zum Streamen von Kamerabildern und synchronisierten Bilddaten (z.B. linkes Kamerabild und das dazugehörige Disparitätsbild) genutzt werden.

gRPC ist ein System zur Interprozesskommunikation über Rechnergrenzen hinweg, welches auch das Streamen von Daten unterstützt. Es benutzt Protocol Buffers (siehe https://developers.google.com/protocol-buffers/) als Beschreibungssprache und zur Datenserialisierung. Eine Einführung und mehr Details zu gRPC sind auf der offiziellen Webseite verfügbar (https://grpc.io/).

Die Vorteile der gRPC Schnittstelle gegenüber GigE Vision sind:

- Es ist in eigenen Programmen einfacher zu benutzen als GigE Vision.
- Es gibt gRPC Unterstützung für sehr viele Programmiersprachen (siehe https://grpc.io/).
- Die Kommunikation basiert auf TCP statt auf UDP und funktioniert deshalb besser über weniger stabile Netzwerke wie z.B. WLAN.

Die Nachteile der gRPC Schnittstelle im Vergleich zu GigE Vision sind:

- Es unterstützt nicht das Ändern von Parametern. Allerdings können alle Parameter über die *REST-API-Schnittstelle* (Abschnitt 7.2) geändert werden.
- Es ist keine Standard-Bildverarbeitungsschnittstelle wie z.B. GigE Vision.

Der *rc_reason_stack* bietet synchronisierte Bilddaten über gRPC Serverstreams auf einem separaten Port pro Pipeline an. Der Port ist 50051 + Pipeline Nummer, also 50051 für Pipeline 0, 50052 für Pipeline 1, etc.

Die Kommunikation wird gestartet indem eine `ImageSetRequest` Nachricht an den Server geschickt wird. Die Nachricht enthält die Information über angeforderte Bilder, d.h. linkes, rechtes, Disparitäts-, Konfidenz- oder Fehlerbild, die separat an- und abgeschaltet werden können.

Nach dem Empfangen der Anfrage sendet der Server kontinuierlich `ImageSet` Nachrichten, welche alle angeforderten Bilder mit allen Parametern enthalten, die notwendig sind, um die Bilder zu interpretieren. Die Bilder in einer `ImageSet` Nachricht sind synchronisiert, d.h. sie sind alle zum selben Zeitpunkt aufgenommen. Die einzige Ausnahme von dieser Regel besteht, wenn der *out1_mode* (Abschnitt 6.4.4.1) auf `AlternateExposureActive` gesetzt ist. In diesem Fall werden die Kamera- und Disparitätsbilder um 40 ms versetzt aufgenommen, sodass GPIO Out1 auf *aus* (LOW) steht, wenn das linke und rechte Bild aufgenommen werden, und auf *an* (HIGH) für das Disparitäts-, Konfidenz- und Fehlerbild. Dies ist sinnvoll, wenn ein Musterprojektor genutzt wird, da der Projektor dann bei der Aufnahme des linken und rechten Bildes aus ist und für das Disparitätsbild an, wodurch die Kamerabilder ungestört sind, aber das Disparitätsbild deutlich dichter und genauer wird.

Das Streamen von Bildern wird beendet, sobald der Client die Verbindung schließt.

Eine `ImageEventsRequest` Nachricht kann gesendet werden, um das Streaming von `ImageEvents` zu starten. Diese Nachricht enthält das Ereignis `depth_acquisition_done`, welches signalisiert, dass die Bildaufnahme zur Tiefenberechnung abgeschlossen ist. Sie enthält außerdem den `imageset_timestamp` des zugehörigen `ImageSet`. Dieses Ereignis kann verwendet werden, um die Zykluszeit in einer Roboteranwendung zu optimieren, da es signalisiert, ab wann die Kamera oder die Szene nach dem Auslösen einer Erkennung sicher bewegt werden können.

7.6.1 gRPC Servicedefinition

```
syntax = "proto3";

message Time
{
  int32 sec = 1; ///< Seconds
  int32 nsec = 2; ///< Nanoseconds
}

message Gpios
{
  uint32 inputs  = 1; ///< bitmask of available inputs
  uint32 outputs = 2; ///< bitmask of available outputs
  uint32 values  = 3; ///< bitmask of GPIO values
}

message Image
{
  Time timestamp           = 1; ///< Acquisition timestamp of the image
  uint32 height            = 2; ///< image height (number of rows)
  uint32 width             = 3; ///< image width (number of columns)
  float focal_length       = 4; ///< focal length in pixels
  float principal_point_u  = 5; ///< horizontal position of the principal point
  float principal_point_v  = 6; ///< vertical position of the principal point
  string encoding          = 7; ///< Encoding of pixels ["mono8", "mono16", "rgb8"]
  bool is_bigendian        = 8; ///< is data bigendian, (in our case false)
  uint32 step              = 9; ///< full row length in bytes
  bytes data               = 10; ///< actual matrix data, size is (step * height)
  Gpios gpios              = 11; ///< GPIOs as of acquisition timestamp
  float exposure_time      = 12; ///< exposure time in seconds
  float gain               = 13; ///< gain factor in decibel
  float noise              = 14; ///< noise
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  float out1_reduction     = 16; ///< Fraction of reduction (0.0 - 1.0) of exposure time for␣
↪images with GPIO Out1=Low in exp_auto_mode=AdaptiveOut1
  float brightness         = 17; ///< Current brightness of the image as value between 0 and 1
}

message DisparityImage
{
  Time timestamp           = 1; ///< Acquisition timestamp of the image
  float scale              = 2; ///< scale factor
  float offset             = 3; ///< offset in pixels (in our case 0)
  float invalid_data_value = 4; ///< value used to mark pixels as invalid (in our case 0)
  float baseline           = 5; ///< baseline in meters
  float delta_d            = 6; ///< Smallest allowed disparity increment. The smallest␣
↪achievable depth range resolution is delta_Z = (Z^2/image.focal_length*baseline)*delta_d.
  Image image              = 7; ///< disparity image
}

message Mesh
{
  Time timestamp           = 1; ///< Acquisition timestamp of disparity image from which the mesh␣
↪is computed
  string format            = 2; ///< currently only "ply" is supported
  bytes data               = 3; ///< actual mesh data
}

message ImageSet
{
  Time timestamp             = 1;
  Image left                 = 2;
  Image right                = 3;
  DisparityImage disparity   = 4;
  Image disparity_error      = 5;
  Image confidence           = 6;
  Mesh mesh                  = 7;
}

message MeshOptions
{
  uint32 max_points            = 1; ///< limit maximum number of points, zero means default (up␣
↪to 3.1MP), minimum is 1000
  enum BinningMethod {
    AVERAGE = 0;                   ///< average over all points in bin
    MIN_DEPTH = 1;                 ///< use point with minimum depth (i.e. closest to camera) in␣
↪bin
  }
  BinningMethod binning_method = 2; ///< method used for binning if limited by max_points
  bool watertight              = 3; ///< connect all edges and fill all holes, e.g. for collision␣
↪checking
  bool textured                = 4; ///< add texture information to mesh
}

message ImageSetRequest
{
  bool left_enabled            = 1;
  bool right_enabled           = 2;
  bool disparity_enabled       = 3;
  bool disparity_error_enabled = 4;
  bool confidence_enabled      = 5;
  bool mesh_enabled            = 6;
  MeshOptions mesh_options     = 7;
  bool color                   = 8; ///< send left/right image as color (rgb8) images
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
}

service ImageInterface
{
  // A server-to-client streaming RPC.
  rpc StreamImageSets(ImageSetRequest) returns (stream ImageSet) {}
}

message Event
{
  Time timestamp = 1; ///< timestamp of the event
  string message = 2; ///< optional message of the event
}

message ImageEvents
{
  Time imageset_timestamp      = 1; ///< timestamp of the ImageSet that the event belongs to
  Event depth_acquisition_done = 2; ///< Depth image acquisition is done (e.g. stereo images
↪captured)
}

message ImageEventsRequest
{
  bool depth_acquisition_done_enabled = 1; ///< send event when depth acquisition is done
}

service ImageEventsInterface
{
  // A server-to-client streaming RPC.
  rpc StreamImageEvents(ImageEventsRequest) returns (stream ImageEvents) {}
}
```

7.6.1.1 Umwandlung von Bild-Streams

Das Disparitätsbild enthält vorzeichenlose 16-Bit-Ganzzahlwerte. Diese Werte müssen mit dem `scale` Wert der `DisparityImage` Nachricht multipliziert werden, um die Disparitätswerte d in Pixeln zu ermitteln. Um die 3D-Objektkoordinaten aus den Disparitätswerten berechnen zu können, werden der Basisabstand `baseline` = t aus der `DisparityImage` Nachricht, die Brennweite `focal_length` = f, und der Bildhauptpunkt `principal_point_u` = c_x und `principal_point_v` = c_y aus der `ImageData` Nachricht benötigt. Die Brennweite und der Bildhauptpunkt hängen von der Bildauflösung der Kamera ab und müssen auf die Auflösung des Disparitätsbilds skaliert werden. Sind diese Werte bekannt, können die Pixel-Koordinaten und die Disparitätswerte mithilfe der im Abschnitt *Berechnung von Tiefenbildern und Punktwolken* (Abschnitt 4.2.2) angegebenen Gleichungen in 3D-Objektkoordination im Kamera-Koordinatensystem umgerechnet werden.

Unter der Annahme, dass es sich bei d_{ik} um den 16-Bit-Disparitätswert in der Spalte i und Zeile k eines Disparitätsbildes handelt, ist der Fließkomma-Disparitätswert in Pixeln d_{ik} gegeben durch

$$d_{ik} = d16_{ik} \cdot \text{scale}$$

Die 3D-Rekonstruktion (in Metern) kann wie folgt durchgeführt werden:

$$P_x = (i + 0.5 - c_x)\,\frac{t}{d_{ik}},$$
$$P_y = (k + 0.5 - c_y)\,\frac{t}{d_{ik}},$$
$$P_z = f\,\frac{t}{d_{ik}}.$$

Das Konfidenzbild umfasst vorzeichenlose 8-Bit-Ganzzahlwerte. Diese Werte müssen durch 255 geteilt werden, um die zwischen 0 und 1 liegenden Konfidenzwerte zu berechnen.

Das Fehlerbild umfasst vorzeichenlose 8-Bit-Ganzzahlwerte. Der Fehler e_{ik} muss mit dem im `scale` Wert der `DisparityImage` Nachricht angegebenen Skalierungsfaktor multipliziert werden, um die Disparitätsfehlerwerte d_{eps} in Pixeln zu ermitteln. Der Beschreibung in *Konfidenz- und Fehlerbilder* (Abschnitt 4.2.3) zufolge lässt sich der Tiefenfehler z_{eps} (in Metern) wie folgt berechnen:

$$d_{ik} = d16_{ik} \cdot \text{scale},$$
$$z_{eps} = \frac{e_{ik} \cdot \text{scale} \cdot f \cdot t}{(d_{ik})^2}.$$

Für nähere Informationen zu Disparitäts-, Fehler- und Konfidenzbildern siehe *Stereo-Matching Modul* (Abschnitt 6.2.2).

7.6.2 Beispielclient

Ein einfacher C++ Client kann von https://github.com/roboception/grpc_image_client_example heruntergeladen werden.

8 Wartung

8.1 Backup der Einstellungen

Der *rc_reason_stack* bietet die Möglichkeit, die aktuellen Einstellungen als Backup oder zum Übertragen auf einen anderen *rc_visard* oder *rc_cube* herunterzuladen.

Die aktuellen Einstellungen des *rc_reason_stack* können über die *Web GUI* (Abschnitt 7.1) auf der Seite *System* im Abschnitt *rc_reason_stack Einstellungen* heruntergeladen werden, oder über die *REST-API-Schnittstelle* (Abschnitt 7.2) des *rc_reason_stack* mit Hilfe des Aufrufs `GET /system/backup`.

Beim Herunterladen des Backups kann der Nutzer entscheiden, welche Einstellungen das Backup enthalten soll:

- `nodes`: die Einstellungen aller Module (Parameter, bevorzugte TCP-Orientierungen und Sortierstrategien)
- `load_carriers`: die erstellten Load Carrier
- `regions_of_interest`: die erstellten 2D und 3D Regions of Interest
- `grippers`: die erstellten Greifer (ohne CAD Elemente)

Das zurückgelieferte Backup sollte als .json-Datei gespeichert werden.

Die Templates der SilhouetteMatch und CADMatch Module sind nicht im Backup enthalten, aber können manuell über die REST-API oder die Web GUI heruntergeladen werden (siehe *Template API*, Abschnitt 6.3.6.14 und *Template API*, Abschnitt 6.3.7.13).

Ein Backup kann auf dem *rc_reason_stack* über die *Web GUI* (Abschnitt 7.1) auf der Seite *System* im Abschnitt *rc_reason_stack Einstellungen* eingespielt werden, indem die Backup .json-Datei hochgeladen wird. In der *Web GUI* werden die im Backup enthaltenen Einstellungen angezeigt und können für das Einspielen ausgewählt werden. Der zugehörige Aufruf der *REST-API-Schnittstelle* (Abschnitt 7.2) ist `POST /system/backup`.

Warnung: Wenn ein Backup von Load Carriern eingespielt wird, gehen alle bestehenden Load Carrier auf dem *rc_reason_stack* verloren und werden durch die Load Carrier im Backup ersetzt. Das gleiche trifft auf das Einspielen von Greifern und Regions of Interest zu.

Wenn ein Backup eingespielt wird, werden nur die Einstellungen gesetzt, die für den jeweiligen *rc_reason_stack* zutreffend sind. Parameter für Module, die nicht existieren oder keine gültige Lizenz haben, werden ignoriert. Wenn ein Backup nur teilweise eingespielt werden konnte, wird der Benutzer über Warnungen darüber informiert.

8.2 Aktualisierung der Softwarelizenz

Lizenzen zur Aktivierung zusätzlicher Funktionen können von Roboception erworben werden.

8.3 Download der Logdateien

Während des Betriebs dokumentiert der *rc_reason_stack* wichtige Informationen, Hinweise und Fehler in sogenannten Logdateien. Zeigt der *rc_reason_stack* ein unerwartetes oder fehlerhaftes Verhalten, kann mithilfe der Logdateien nach der Fehlerursache geforscht werden. Logeinträge lassen sich über die Seite *System* → *Logs* auf der *Web GUI* (Abschnitt 7.1) ansehen und filtern. Wird der Support kontaktiert (*Kontakt*, Abschnitt 10), sind die Logdateien sehr hilfreich, um Probleme aufzuspüren. Um diese als tar.gz-Datei herunterzuladen, ist der Button *Alle Logs herunterladen* auf der Seite *System* → *Logs* der Web GUI unter *System* zu klicken.

Die Logs sind nicht nur über die Web GUI, sondern auch über die *REST-API-Schnittstelle* (Abschnitt 7.2) des *rc_reason_stack* zugänglich. Hierfür können die Anfragen des Typs `GET /logs` und `GET /logs/{log}` verwendet werden.

9 Fehlerbehebung

9.1 Probleme mit den Kamerabildern

Kamerabild ist zu hell

- Wenn die Kamera im manuellen Belichtungsmodus arbeitet, versuchen Sie, die Belichtungszeit zu verkürzen oder
- schalten Sie auf automatische Belichtung um.

Kamerabild ist zu dunkel

- Wenn die Kamera im manuellen Belichtungsmodus arbeitet, versuchen Sie, die Belichtungszeit zu verlängern oder
- schalten Sie auf automatische Belichtung um.

Kamerabild rauscht zu stark

Große Gain-Faktoren verursachen ein Bildrauschen mit hoher Amplitude. Wollen Sie das Bildrauschen verringern,

- verwenden Sie eine zusätzliche Lichtquelle, um die Lichtintensität der Aufnahme zu erhöhen, oder
- stellen Sie eine größere maximale Autobelichtungszeit ein.

Kamerabild ist unscharf

- Überprüfen Sie, ob das Objekt zu nahe an der Linse liegt, und erhöhen Sie bei Bedarf den Abstand zwischen dem Objekt und der Linse.
- Überprüfen Sie, ob die Kameralinsen verschmutzt sind, und reinigen Sie diese bei Bedarf.
- Trifft keiner der vorstehenden Punkte zu, kann es sein, dass ein schweres Hardware-Problem vorliegt. Bitte wenden Sie sich an den *Support* (Abschnitt 10).

Kamerabild ist verschwommen

Schnelle Bewegungen können in Kombination mit langen Belichtungszeiten zu Unschärfe führen. Um Bewegungsunschärfe zu verringern,

- verringern Sie die Bewegungsgeschwindigkeit der Kamera,
- verringern Sie die Bewegungsgeschwindigkeit von Objekten im Sichtfeld der Kamera oder
- verkürzen Sie die Belichtungszeit der Kameras.

Bildwiederholrate ist zu niedrig

- Erhöhen Sie die Bildwiederholrate.
- Die maximale Bildwiederholrate der Kameras beträgt 25 Hz.

9.2 Probleme mit Tiefen-/Disparitäts-, Fehler- oder Konfidenzbildern

Die folgenden Hinweise gelten auch für Fehler- und Konfidenzbilder, da sie direkt mit den Disparitätsbildern zusammenhängen.

Disparitätsbild spärlich befüllt oder leer

- Überprüfen Sie, ob die Kamerabilder gut belichtet und scharf sind. Befolgen Sie bei Bedarf die Anweisungen in *Probleme mit den Kamerabildern* (Abschnitt 9.1).
- Überprüfen Sie, ob die Szene genügend Textur hat und installieren Sie bei Bedarf einen Musterprojektor.
- Senken Sie den *Minimalen Abstand* (Abschnitt 6.2.2.1).
- Erhöhen Sie den *Maximalen Abstand* (Abschnitt 6.2.2.1).
- Überprüfen Sie, ob das Objekt zu nahe an der Kamera liegt. Beachten Sie die unterschiedlichen Tiefenmessbereiche der Kameravarianten.
- Senken Sie die *Minimale Konfidenz* (Abschnitt 6.2.2.1).
- Erhöhen Sie den *Maximalen Fehler* (Abschnitt 6.2.2.1).
- Wählen Sie eine geringere *Qualität des Disparitätsbilds* (Abschnitt 6.2.2.1). Disparitätsbilder mit einer geringeren Auflösung sind in der Regel nicht so spärlich befüllt.
- Überprüfen Sie die Kalibrierung der Kameras und führen Sie bei Bedarf eine Neukalibrierung durch (siehe *Kamerakalibrierung*, Abschnitt 6.4.3).

Bildwiederholrate der Disparitätsbilder ist zu niedrig

- Überprüfen und erhöhen Sie die Bildwiederholrate der Kamerabilder. Die Bildwiederholrate der Disparitätsbilder kann nicht größer sein als die Bildwiederholrate der Kamerabilder.
- Wählen Sie eine geringere *Qualität des Disparitätsbilds* (Abschnitt 6.2.2.1).
- Erhöhen Sie den *Minimalen Abstand* (Abschnitt 6.2.2.1) so viel wie für die Applikation möglich.

Disparitätsbild zeigt keine nahe liegenden Objekte

- Überprüfen Sie, ob das Objekt zu nahe an der Linse liegt. Beachten Sie die unterschiedlichen Tiefenmessbereiche der Kameravarianten.
- Senken Sie den *Minimalen Abstand* (Abschnitt 6.2.2.1).

Disparitätsbild zeigt keine weit entfernten Objekte

- Erhöhen Sie den *Maximalen Abstand* (Abschnitt 6.2.2.1).
- Erhöhen Sie den *Maximalen Fehler* (Abschnitt 6.2.2.1).
- Senken Sie die *Minimale Konfidenz* (Abschnitt 6.2.2.1).

Disparitätsbild rauscht zu stark

- Erhöhen Sie den *Segmentierungswert* (Abschnitt 6.2.2.1).
- Erhöhen Sie den *Füllen-Wert* (Abschnitt 6.2.2.1).

Disparitätswerte oder resultierende Tiefenwerte sind zu ungenau

- Verringern Sie den Abstand zwischen der Kamera und der Szene. Der Tiefenmessfehler nimmt quadratisch mit dem Abstand zu den Kameras zu.
- Überprüfen Sie, ob die Szene wiederkehrende Muster enthält und entfernen Sie diese bei Bedarf. Diese könnten falsche Disparitätsmessungen verursachen.

Disparitätsbild ist zu glatt

- Senken Sie den *Füllen-Wert* (Abschnitt 6.2.2.1).

Disparitätsbild zeigt keine feinen Strukturen

- Senken Sie den *Segmentierungswert* (Abschnitt 6.2.2.1).
- Senken Sie den *Füllen-Wert* (Abschnitt 6.2.2.1).

10 Kontakt

10.1 Support

Support-Anfragen können Sie uns entweder über die Seite http://www.roboception.com/support oder per E-Mail an support@roboception.de zukommen lassen.

10.2 Downloads

Software-SDKs usw. können von der Roboception-Homepage heruntergeladen werden: http://www.roboception.com/download.

10.3 Adresse

Roboception GmbH
Kaflerstraße 2
81241 München
Deutschland

Web: http://www.roboception.com
E-Mail: info@roboception.de
Telefon: +49 89 889 50 79-0

11 Anhang

11.1 Formate für Posendaten

Eine Pose besteht aus einer Translation und einer Rotation. Die Translation definiert die Verschiebung entlang der x, y und z-Achsen. Die Rotation kann auf viele verschiedene Arten definiert werden. Der *rc_reason_stack* benutzt Quaternionen, um Rotationen zu definieren, und Translationen werden in Metern angegeben. Dies wird als XYZ+Quaternion Format bezeichnet. Dieses Kapitel erklärt Umrechnungen zwischen verschiedenen üblichen Posenformaten und dem XYZ+Quaternion Format.

Es ist weit verbreitet, Rotationen in 3D durch drei Winkel als Drehungen um die Koordinatenachsen zu definieren. Leider existieren hierfür viele verschiedene Möglichkeiten. Übliche Konventionen sind Euler- oder Kardanwinkel (letztere werden auch als Tait-Bryan Winkel bezeichnet). In beiden Konventionen können die drei Rotationen auf die bereits gedrehten Achsen (intrinsische Rotation) oder auf die Achsen des festen Koordinatensystems (extrinsische Rotation) angewendet werden.

Wir benutzen x, y und z zur Bezeichnung der drei Koordinatenachsen. x', y' und z' bezeichnen die Achsen, die einmal rotiert wurden. x'', y'' und z'' bezeichnen die Achsen nach zwei Rotationen.

In der ursprünglichen Eulerwinkelkonvention ist die erste und dritte Drehachse immer identisch. Die Rotationsreihenfolge z-x'-z'' bedeutet z.B. eine Drehung um die z-Achse, dann eine Drehung um die gedrehte x-Achse und schließlich eine Drehung um die (zweimal) gedrehte z-Achse. In der Kardanwinkelkonvention sind alle drei Drehachsen unterschiedlich, z.B. z-y'-x''. Kardanwinkel werden häufig ebenfalls als Eulerwinkel bezeichnet.

Für jede intrinsische Rotationsreihenfolge gibt es eine äquivalente extrinsische Rotationsreihenfolge, die genau umgekehrt ist. Die intrinsische Rotationsreihenfolge z-y'-x'' ist zum Beispiel äquivalent zu der extrinsischen Rotationsreihenfolge x-y-z.

Rotationen um die x, y und z-Achse können mit Quaternionen definiert werden als

$$r_x(\alpha) = \begin{pmatrix} \sin\frac{\alpha}{2} \\ 0 \\ 0 \\ \cos\frac{\alpha}{2} \end{pmatrix}, \qquad r_y(\beta) = \begin{pmatrix} 0 \\ \sin\frac{\beta}{2} \\ 0 \\ \cos\frac{\beta}{2} \end{pmatrix}, \qquad r_z(\gamma) = \begin{pmatrix} 0 \\ 0 \\ \sin\frac{\gamma}{2} \\ \cos\frac{\gamma}{2} \end{pmatrix},$$

oder durch Rotationsmatrizen als

$$r_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix},$$
$$r_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix},$$
$$r_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Die extrinsische Rotationsreihenfolge x-y-z kann durch die Multiplikation einzelner Rotationen in umge-

kehrter Reihenfolge berechnet werden, d.h. $r_z(\gamma)r_y(\beta)r_x(\alpha)$.

Basierend auf diesen Definitionen beschreiben die folgenden Abschnitte die Umrechnung zwischen üblichen Konventionen und dem XYZ+Quaternion Format.

Bemerkung: Zu beachten sind stets die Einheiten für Positionen und Orientierungen. *rc_reason_stack* Geräte benutzen stets Meter als Längeneinheit, während die meisten Roboterhersteller Längen in Millimeter oder Inch angeben. Winkel werden üblicherweise in Grad angegeben, können aber auch im Bogenmaß angegeben sein.

11.1.1 Rotationsmatrix und Translationsvektor

Eine Pose kann mit einer Rotationsmatrix R und einem Translationsvektor T definiert werden.

$$R = \begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix}, \qquad T = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

Die Posentransformation für einen Punkt P ist

$$P' = RP + T.$$

11.1.1.1 Umrechnung von Rotationsmatrizen in Quaternionen

Die Umrechnung von einer Rotationsmatrix (mit $det(R) = 1$) in eine Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ kann wie folgt durchgeführt werden.

$$\begin{aligned} x &= \text{sign}(r_{21} - r_{12})\frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} - r_{11} - r_{22})} \\ y &= \text{sign}(r_{02} - r_{20})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} + r_{11} - r_{22})} \\ z &= \text{sign}(r_{10} - r_{01})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} - r_{11} + r_{22})} \\ w &= \frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} + r_{11} + r_{22})} \end{aligned}$$

Der sign Operator gibt -1 zurück, falls sein Argument negativ ist. Sonst wird 1 zurück gegeben. Er wird zur Wiederherstellung das Vorzeichens der Wurzel benutzt. Die max Funktion stellt sicher, dass das Argument der Wurzel nicht negativ ist, was in der Praxis durch Rundungsfehler passieren kann.

11.1.1.2 Umrechnung von Quaternionen in Rotationsmatrizen

Die Umrechnung von einer Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ mit $||q|| = 1$ in eine Rotationsmatrix kann wie folgt durchgeführt werden.

$$R = 2\begin{pmatrix} \frac{1}{2} - y^2 - z^2 & xy - zw & xz + yw \\ xy + zw & \frac{1}{2} - x^2 - z^2 & yz - xw \\ xz - yw & yz + xw & \frac{1}{2} - x^2 - y^2 \end{pmatrix}$$

11.1.2 ABB Posenformat

ABB Roboter beschreiben eine Pose durch Position X, Y, Z und Quaternion $Q1$, $Q2$, $Q3$, $Q4$, ähnlich wie *rc_reason_stack* Geräte. Jedoch muss die Position in Millimetern angegeben werden und die Quaternion Reihenfolge ist wie folgt definiert:

$q = \begin{pmatrix} x & y & z & w \end{pmatrix} = \begin{pmatrix} Q2 & Q3 & Q4 & Q1 \end{pmatrix}$.

11.1.3 FANUC XYZ-WPR Format

Das Posenformat, welches von FANUC Robotern benutzt wird, besteht aus einer Position XYZ in Millimetern und einer Orientierung WPR, welche durch drei Winkel in Grad gegeben ist. W rotiert um die x-Achse, P rotiert um die y-Achse und R rotiert um die z-Achse. Die Rotationsreihenfolge ist x-y-z und wird berechnet durch $r_z(R)r_y(P)r_x(W)$.

11.1.3.1 Umrechnung von FANUC-WPR in Quaternionen

Zur Umrechnung von WPR Winkeln in Grad in eine Quaternion $q = (\ x \quad y \quad z \quad w\ )$ werden zunächst die Winkel ins Bogenmaß umgerechnet

$$\begin{aligned} W_r &= W\frac{\pi}{180}, \\ P_r &= P\frac{\pi}{180}, \\ R_r &= R\frac{\pi}{180}, \end{aligned}$$

und damit wird die Quaternion berechnet als

$$\begin{aligned} x &= \cos(R_r/2)\cos(P_r/2)\sin(W_r/2) - \sin(R_r/2)\sin(P_r/2)\cos(W_r/2), \\ y &= \cos(R_r/2)\sin(P_r/2)\cos(W_r/2) + \sin(R_r/2)\cos(P_r/2)\sin(W_r/2), \\ z &= \sin(R_r/2)\cos(P_r/2)\cos(W_r/2) - \cos(R_r/2)\sin(P_r/2)\sin(W_r/2), \\ w &= \cos(R_r/2)\cos(P_r/2)\cos(W_r/2) + \sin(R_r/2)\sin(P_r/2)\sin(W_r/2). \end{aligned}$$

11.1.3.2 Umrechnung von Quaternionen in FANUC-WPR

Die Umrechnung von einer Quaternion $q = (\ x \quad y \quad z \quad w\ )$ mit $||q|| = 1$ in WPR Winkel in Grad kann wie folgt durchgeführt werden.

$$\begin{aligned} R &= \text{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi} \\ P &= \text{asin}(2(wy - zx))\frac{180}{\pi} \\ W &= \text{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi} \end{aligned}$$

11.1.4 Franka Emika Posenformat

Franka Emika Roboter nutzen eine Transformationsmatrix T um eine Pose zu definieren. Eine Transformationsmatrix kombiniert eine Rotationsmatrix R und einen Translationsvektor $t = (\ x \quad y \quad z\ )^T$.

$$T = \begin{pmatrix} r_{00} & r_{01} & r_{02} & x \\ r_{10} & r_{11} & r_{12} & y \\ r_{20} & r_{21} & r_{22} & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Die Posen, die Franka Emika's „Measure Pose“ App ausgibt, bestehen aus einer Translation x, y, z in Millimetern und einer Rotation x, y, z in Grad. Die Rotationsreihenfolge ist z-y'-x'' (d.h. x-y-z) und die Rotation wird berechnet durch $r_z(z)r_y(y)r_x(x)$.

11.1.4.1 Umrechnung von Transformation in Quaternion

Die Umrechnung von einer Rotationsmatrix (mit $det(R) = 1$) in eine Quaternion $q = \begin{pmatrix} q_x & q_y & q_z & q_w \end{pmatrix}$ kann wie folgt durchgeführt werden.

$$
\begin{aligned}
q_x &= \text{sign}(r_{21} - r_{12})\frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} - r_{11} - r_{22})} \\
q_y &= \text{sign}(r_{02} - r_{20})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} + r_{11} - r_{22})} \\
q_z &= \text{sign}(r_{10} - r_{01})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} - r_{11} + r_{22})} \\
q_w &= \frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} + r_{11} + r_{22})}
\end{aligned}
$$

Der sign Operator gibt -1 zurück, falls sein Argument negativ ist. Sonst wird 1 zurück gegeben. Er wird zur Wiederherstellung das Vorzeichens der Wurzel benutzt. Die max Funktion stellt sicher, dass das Argument der Wurzel nicht negativ ist, was in der Praxis durch Rundungsfehler passieren kann.

11.1.4.2 Umrechnung von Rotation-XYZ in Quaternion

Zur Umrechnung von der Rotationswinkel x, y, z in Grad in eine Quaternion $q = \begin{pmatrix} q_x & q_y & q_z & q_w \end{pmatrix}$ werden zuerst alle Winkel in das Bogenmaß umgerechnet mit

$$
\begin{aligned}
X_r &= x\frac{\pi}{180}, \\
Y_r &= y\frac{\pi}{180}, \\
Z_r &= z\frac{\pi}{180},
\end{aligned}
$$

und damit die Quaternion berechnet durch

$$
\begin{aligned}
q_x &= \cos(Z_r/2)\cos(Y_r/2)\sin(X_r/2) - \sin(Z_r/2)\sin(Y_r/2)\cos(X_r/2), \\
q_y &= \cos(Z_r/2)\sin(Y_r/2)\cos(X_r/2) + \sin(Z_r/2)\cos(Y_r/2)\sin(X_r/2), \\
q_z &= \sin(Z_r/2)\cos(Y_r/2)\cos(X_r/2) - \cos(Z_r/2)\sin(Y_r/2)\sin(X_r/2), \\
q_w &= \cos(Z_r/2)\cos(Y_r/2)\cos(X_r/2) + \sin(Z_r/2)\sin(Y_r/2)\sin(X_r/2).
\end{aligned}
$$

11.1.4.3 Umrechnung von Quaternion und Translation in Transformation

Die Umrechnung von einer Quaternion $q = \begin{pmatrix} q_x & q_y & q_z & q_w \end{pmatrix}$ und einem Translationsvektor $t = \begin{pmatrix} x & y & z \end{pmatrix}^T$ in eine Transformationsmatrix T kann wie folgt durchgeführt werden.

$$
T = \begin{pmatrix}
1 - 2s(q_y^2 + q_z^2) & 2s(q_x q_y - q_z q_w) & 2s(q_x q_z + q_y q_w) & x \\
2s(q_x q_y + q_z q_w) & 1 - 2s(q_x^2 + q_z^2) & 2s(q_y q_z - q_x q_w) & y \\
2s(q_x q_z - q_y q_w) & 2s(q_y q_z + q_x q_w) & 1 - 2s(q_x^2 + q_y^2) & z \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

wobei $s = ||q||^{-2} = \frac{1}{q_x^2 + q_y^2 + q_z^2 + q_w^2}$ und $s = 1$ wenn q eine Einheitsquaternion ist.

11.1.4.4 Umrechnung von Quaternion in Rotation-XYZ

Die Umrechnung von einer Quaternion $q = (\ q_x \quad q_y \quad q_z \quad q_w\ )$ mit $||q|| = 1$ in x, y, z Winkel in Grad kann wie folgt durchgeführt werden.

$$x = \text{atan}_2(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2))\frac{180}{\pi}$$
$$y = \text{asin}(2(q_w q_y - q_z q_x))\frac{180}{\pi}$$
$$z = \text{atan}_2(2(q_w q_x + q_y q_z), 1 - 2(q_x^2 + q_y^2))\frac{180}{\pi}$$

11.1.4.5 Posenrepräsentation in RaceCom Messages und Statemachines

In RaceCom Messages und in Statemachines wird eine Pose normalerweise als eindimensionales Array aus 16 Floatwerten definiert, die in spaltenweiser Anordnung eine Transformationsmatrix repräsentieren. Die Indizes der Einträge der folgenden Matrix entsprechen den Array-Indizes.

$$T = \begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix}$$

11.1.5 Fruitcore HORST Posenformat

Fruitcore HORST Roboter beschreiben eine Pose durch eine Position in Metern und ein Quaternion mit $q_0 = w$, $q_1 = x$, $q_2 = y$ und $q_3 = z$ wie auch *rc_reason_stack* Geräte. Es ist keine Konvertierung notwendig.

11.1.6 Kawasaki XYZ-OAT Format

Das Posenformat, welches von Kawasaki Robotern benutzt wird, besteht aus einer Position XYZ in Millimetern und einer Orientierung OAT, welche durch drei Winkel in Grad angegeben wird. O rotiert um die z-Achse, A rotiert um die gedrehte y-Achse und T rotiert um die gedrehte z-Achse. Die Rotationsreihenfolge ist z-y'-z'' (d.h. z-y-z) und wird berechnet durch $r_z(O)r_y(A)r_z(T)$.

11.1.6.1 Umrechnung von Kawasaki-OAT in Quaternionen

Zur Umrechnung von OAT Winkeln in Grad in eine Quaternion $q = (\ x \quad y \quad z \quad w\ )$ werden zunächst alle Winkel in das Bogenmaß umgerechnet durch

$$O_r = O\frac{\pi}{180},$$
$$A_r = A\frac{\pi}{180},$$
$$T_r = T\frac{\pi}{180},$$

und damit wird die Quaternion berechnet durch

$$x = \cos(O_r/2)\sin(A_r/2)\sin(T_r/2) - \sin(O_r/2)\sin(A_r/2)\cos(T_r/2),$$
$$y = \cos(O_r/2)\sin(A_r/2)\cos(T_r/2) + \sin(O_r/2)\sin(A_r/2)\sin(T_r/2),$$
$$z = \sin(O_r/2)\cos(A_r/2)\cos(T_r/2) + \cos(O_r/2)\cos(A_r/2)\sin(T_r/2),$$
$$w = \cos(O_r/2)\cos(A_r/2)\cos(T_r/2) - \sin(O_r/2)\cos(A_r/2)\sin(T_r/2).$$

11.1.6.2 Umrechnung von Quaternionen in Kawasaki-OAT

Die Umrechnung von einer Quaternion $q = (\ x \quad y \quad z \quad w\ )$ mit $||q|| = 1$ in OAT Winkel in Grad kann wie folgt durchgeführt werden.

Wenn $x = 0$ **und** $y = 0$ ist die Umrechnung

$$O = \text{atan}_2(2(z - w), 2(z + w))\frac{180}{\pi}$$
$$A = \text{acos}(w^2 + z^2)\frac{180}{\pi}$$
$$T = \text{atan}_2(2(z + w), 2(w - z))\frac{180}{\pi}$$

Wenn $z = 0$ **und** $w = 0$ ist die Umrechnung

$$O = \text{atan}_2(2(y - x), 2(x + y))\frac{180}{\pi}$$
$$A = \text{acos}(-1.0)\frac{180}{\pi}$$
$$T = \text{atan}_2(2(y + x), 2(y - x))\frac{180}{\pi}$$

In allen anderen Fällen ist die Umrechnung

$$O = \text{atan}_2(2(yz - wx), 2(xz + wy))\frac{180}{\pi}$$
$$A = \text{acos}(w^2 - x^2 - y^2 + z^2)\frac{180}{\pi}$$
$$T = \text{atan}_2(2(yz + wx), 2(wy - xz))\frac{180}{\pi}$$

11.1.7 KUKA XYZ-ABC Format

KUKA Roboter nutzen das sogenannte XYZ-ABC Format. XYZ ist die Position in Millimetern. ABC sind Winkel in Grad, wobei A um die z-Achse rotiert, B rotiert um die y-Achse und C rotiert um die x-Achse. Die Rotationsreihenfolge ist z-y'-x'' (i.e. x-y-z) und wird berechnet durch $r_z(A)r_y(B)r_x(C)$.

11.1.7.1 Umrechnung von KUKA-ABC in Quaternionen

Zur Umrechnung von ABC Winkeln in Grad in eine Quaternion $q = (\ x \quad y \quad z \quad w\ )$ werden zuerst alle Winkel in das Bogenmaß umgerechnet mit

$$A_r = A\frac{\pi}{180},$$
$$B_r = B\frac{\pi}{180},$$
$$C_r = C\frac{\pi}{180},$$

und damit die Quaternion berechnet durch

$$x = \cos(A_r/2)\cos(B_r/2)\sin(C_r/2) - \sin(A_r/2)\sin(B_r/2)\cos(C_r/2),$$
$$y = \cos(A_r/2)\sin(B_r/2)\cos(C_r/2) + \sin(A_r/2)\cos(B_r/2)\sin(C_r/2),$$
$$z = \sin(A_r/2)\cos(B_r/2)\cos(C_r/2) - \cos(A_r/2)\sin(B_r/2)\sin(C_r/2),$$
$$w = \cos(A_r/2)\cos(B_r/2)\cos(C_r/2) + \sin(A_r/2)\sin(B_r/2)\sin(C_r/2).$$

11.1.7.2 Umrechnung von Quaternionen in KUKA-ABC

Die Umrechnung von einer Quaternion $q = (\ x \quad y \quad z \quad w\ )$ mit $||q|| = 1$ in ABC Winkel in Grad kann wie folgt durchgeführt werden.

$$A = \text{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi}$$
$$B = \text{asin}(2(wy - zx))\frac{180}{\pi}$$
$$C = \text{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi}$$

11.1.8 Mitsubishi XYZ-ABC Format

Das Posenformat, welches von Mitsubishi Robotern benutzt wird, ist das gleiche wie für KUKA Roboter (siehe *KUKA XYZ-ABC Format*, Abschnitt 11.1.7), außer, dass der Winkel A um die x-Achse rotiert und C eine Rotation um die z-Achse ist. Damit wird die Rotation berechnet durch $r_z(C)r_y(B)r_x(A)$.

11.1.8.1 Umrechnung von Mitsubishi-ABC in Quaternionen

Zur Umrechnung von ABC Winkeln in Grad in eine Quaternion $q = (\ x \quad y \quad z \quad w\ )$ werden die Winkel zunächst ins Bogenmaß umgerechnet mit

$$A_r = A\frac{\pi}{180},$$
$$B_r = B\frac{\pi}{180},$$
$$C_r = C\frac{\pi}{180},$$

und damit die Quaternion berechnet durch

$$x = \cos(C_r/2)\cos(B_r/2)\sin(A_r/2) - \sin(C_r/2)\sin(B_r/2)\cos(A_r/2),$$
$$y = \cos(C_r/2)\sin(B_r/2)\cos(A_r/2) + \sin(C_r/2)\cos(B_r/2)\sin(A_r/2),$$
$$z = \sin(C_r/2)\cos(B_r/2)\cos(A_r/2) - \cos(C_r/2)\sin(B_r/2)\sin(A_r/2),$$
$$w = \cos(C_r/2)\cos(B_r/2)\cos(A_r/2) + \sin(C_r/2)\sin(B_r/2)\sin(A_r/2).$$

11.1.8.2 Umrechnung von Quaternionen in Mitsubishi-ABC

Die Umrechnung von einer Quaternion $q = (\ x \quad y \quad z \quad w\ )$ mit $||q|| = 1$ in ABC Winkel in Grad kann wie folgt durchgeführt werden.

$$A = \text{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi}$$
$$B = \text{asin}(2(wy - zx))\frac{180}{\pi}$$
$$C = \text{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi}$$

11.1.9 Universal Robots Posenformat

Das Posenformat, welches von Universal Robots verwendet wird, besteht aus einer Position XYZ in Millimetern und einer Orientierung im Angle-Axis Format $V = (\ RX \quad RY \quad RZ\ )^T$. Der Rotationswin-

kel θ im Bogenmaß ist die Länge der Rotationsachse U.

$$V = \begin{pmatrix} RX \\ RY \\ RZ \end{pmatrix} = \begin{pmatrix} \theta u_x \\ \theta u_y \\ \theta u_z \end{pmatrix}$$

V wird als Rotationsvektor bezeichnet.

11.1.9.1 Umrechnung vom Angle-Axis Format in Quaternionen

Die Umrechnung von einem Rotationsvektor V in eine Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ kann wie folgt durchgeführt werden.

Zunächst wird der Winkel θ im Bogenmaß aus dem Rotationsvektor V gewonnen durch

$$\theta = \sqrt{RX^2 + RY^2 + RZ^2}.$$

Wenn $\theta = 0$, dann ist die Quaternion gleich $q = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$, sonst wird sie berechnet durch

$$\begin{aligned} x &= RX \frac{\sin(\theta/2)}{\theta}, \\ y &= RY \frac{\sin(\theta/2)}{\theta}, \\ z &= RZ \frac{\sin(\theta/2)}{\theta}, \\ w &= \cos(\theta/2). \end{aligned}$$

11.1.9.2 Umrechnung von Quaternionen ins Angle-Axis Format

Die Umrechnung von einer Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ mit $||q|| = 1$ in einen Rotationsvektor im Angle-Axis Format kann wie folgt durchgeführt werden.

Zunächst wird der Winkel θ im Bogenmaß aus dem Quaternion gewonnen durch

$$\theta = 2 \cdot \mathrm{acos}(w).$$

Wenn $\theta = 0$, dann ist der Rotationsvektor $V = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$, sonst wird er berechnet durch

$$\begin{aligned} RX &= \theta \frac{x}{\sqrt{1 - w^2}}, \\ RY &= \theta \frac{y}{\sqrt{1 - w^2}}, \\ RZ &= \theta \frac{z}{\sqrt{1 - w^2}}. \end{aligned}$$

11.1.10 Yaskawa Posenformat

Das Posenformat, welches von Yaskawa Robotern benutzt wird, besteht aus einer Position XYZ in Millimetern und einer Orientierung, welche durch drei Winkel in Grad gegeben ist. Rx rotiert um die x-Achse, Ry rotiert um die y-Achse und Rz rotiert um die z-Achse. Die Rotationsreihenfolge ist x-y-z und wird berechnet durch $r_z(Rz)r_y(Ry)r_x(Rx)$.

11.1.10.1 Umrechnung von Yaskawa Rx, Ry, Rz in Quaternionen

Zur Umrechnung von Rx, Ry, Rz Winkeln in Grad in eine Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ werden zunächst die Winkel ins Bogenmaß umgerechnet

$$\begin{aligned} X_r &= Rx\frac{\pi}{180}, \\ Y_r &= Ry\frac{\pi}{180}, \\ Z_r &= Rz\frac{\pi}{180}, \end{aligned}$$

und damit wird die Quaternion berechnet als

$$\begin{aligned} x &= \cos(Z_r/2)\cos(Y_r/2)\sin(X_r/2) - \sin(Z_r/2)\sin(Y_r/2)\cos(X_r/2), \\ y &= \cos(Z_r/2)\sin(Y_r/2)\cos(X_r/2) + \sin(Z_r/2)\cos(Y_r/2)\sin(X_r/2), \\ z &= \sin(Z_r/2)\cos(Y_r/2)\cos(X_r/2) - \cos(Z_r/2)\sin(Y_r/2)\sin(X_r/2), \\ w &= \cos(Z_r/2)\cos(Y_r/2)\cos(X_r/2) + \sin(Z_r/2)\sin(Y_r/2)\sin(X_r/2). \end{aligned}$$

11.1.10.2 Umrechnung von Quaternionen in Yaskawa Rx, Ry, Rz

Die Umrechnung von einer Quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ mit $||q|| = 1$ in Rx, Ry, Rz Winkel in Grad kann wie folgt durchgeführt werden.

$$\begin{aligned} Rx &= \text{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi} \\ Ry &= \text{asin}(2(wy - zx))\frac{180}{\pi} \\ Rz &= \text{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi} \end{aligned}$$

HTTP Routing Table

/cad

/generic_robot_interface

/logs

/nodes

/pipelines

/presets

/system

/templates

```
DELETE /templates/rc_silhouettematch/{id},
        206
```

Stichwortverzeichnis

Sonderzeichen

A

B

C

D

E

F

G

H

I

Q

R

S

T

U

V

W

Z

roboception

rc_reason_stack 3D Vision Software Platform

INSTALLATIONS- UND BEDIENUNGSANLEITUNG

Roboception GmbH

Kaflerstraße 2
81241 München
Deutschland

info@roboception.de
www.roboception.com

Tutorials: https://tutorials.roboception.com
GitHub: https://github.com/roboception
Dokumentation: https://doc.rc-visard.com
https://doc.rc-viscore.com
https://doc.rc-cube.com
https://doc.rc-randomdot.com
Shop: https://roboception.com/shop

Für Kundensupport kontaktieren Sie

+49 89 889 50 790
(09:00-17:00 CET)

support@roboception.de