roboception

Roboception GmbH | November 2025

rc_reason_stack

3D Vision Software Platform

INSTALLATION AND OPERATING MANUAL

Revisions

This product may be modified without notice, when necessary, due to product improvements, modifications, or changes in specifications. If such modification is made, the manual will also be revised; see revision information.

DOCUMENTATION REVISION 25.10.3 Nov 05, 2025

Applicable to *rc_reason_stack* firmware 25.10.x

MANUFACTURER

Roboception GmbH

Kaflerstrasse 2

81241 Munich

Germany

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

Please read the operating manual in full and keep it with the product.

Contents

1 Introduction **5**
1.1 Overview . . . 5

2 Safety **6**
2.1 General warnings . . . 6
2.2 Intended use . . . 6

3 Installation **8**
3.1 Offline installation guide . . . 8
3.1.1 Prerequisites . . . 8
3.1.2 Install Ubuntu 24.04 . . . 8
3.1.3 Install NVIDIA driver . . . 9
3.1.4 Install Docker . . . 9
3.1.5 Install NVIDIA Container Toolkit . . . 10
3.1.6 Install WIBU CodeMeter runtime . . . 11
3.1.7 Create a `sensor0` network . . . 11
3.1.8 Ensure network settings for GigE Vision . . . 12
3.1.9 Load container images . . . 12
3.1.10 Start the Docker stack . . . 12
3.1.11 Access the Web GUI . . . 13
3.1.12 Troubleshooting . . . 13
3.2 Software license . . . 13
3.3 Connection of cameras . . . 13

4 Measurement principles **14**
4.1 Stereo vision . . . 14
4.2 General information on 3D data . . . 15
4.2.1 Computing disparity images . . . 15
4.2.2 Computing depth images and point clouds . . . 16
4.2.3 Confidence and error images . . . 16

5 Camera pipelines **18**
5.1 Configuration of camera pipelines . . . 18
5.2 Configuration of connected cameras . . . 19

6 Software modules **22**
6.1 Camera module . . . 23
6.1.1 Rectification . . . 23
6.1.2 Viewing and downloading images . . . 23
6.1.3 Pipeline types *rc_visard* and *rc_viscore* . . . 23
6.1.4 Pipeline type *stereo_ace* . . . 31
6.1.5 Pipeline type *orbbec* . . . 41
6.1.6 Pipeline type *zivid* . . . 46
6.2 3D modules . . . 51
6.2.1 Viewing and downloading images and point clouds . . . 51
6.2.2 Stereo matching module . . . 51

6.2.3 Zivid module . . . 60
6.2.4 Orbbec module . . . 65
6.3 Detection & Measure modules . . . 68
6.3.1 Measure . . . 68
6.3.2 LoadCarrier . . . 73
6.3.3 TagDetect . . . 87
6.3.4 ItemPick . . . 99
6.3.5 BoxPick . . . 119
6.3.6 SilhouetteMatch . . . 147
6.3.7 CADMatch . . . 184
6.4 Configuration modules . . . 218
6.4.1 Hand-eye calibration . . . 219
6.4.2 CollisionCheck . . . 241
6.4.3 Camera calibration . . . 250
6.4.4 IO and Projector Control . . . 258
6.5 Database modules . . . 262
6.5.1 LoadCarrierDB . . . 262
6.5.2 RoiDB . . . 270
6.5.3 GripperDB . . . 277

7 Interfaces 288
7.1 Web GUI . . . 288
7.1.1 Accessing the Web GUI . . . 288
7.1.2 Exploring the Web GUI . . . 288
7.1.3 Web GUI access control . . . 290
7.1.4 Downloading camera images . . . 290
7.1.5 Downloading depth images and point clouds . . . 291
7.2 REST-API interface . . . 291
7.2.1 General API structure . . . 292
7.2.2 Available resources and requests . . . 293
7.2.3 Data type definitions . . . 319
7.2.4 Swagger UI . . . 326
7.3 Generic Robot Interface . . . 329
7.3.1 Job definition . . . 329
7.3.2 Hand-Eye Calibration . . . 331
7.3.3 GRI binary protocol specification . . . 332
7.3.4 Integration with a robot . . . 339
7.3.5 Job and HEC_config API . . . 340
7.4 OPC UA interface . . . 344
7.5 KUKA Ethernet KRL Interface . . . 345
7.5.1 Ethernet connection configuration . . . 345
7.5.2 Generic XML structure . . . 346
7.5.3 Services . . . 347
7.5.4 Parameters . . . 351
7.5.5 Example applications . . . 352
7.5.6 Troubleshooting . . . 352
7.6 gRPC image stream interface . . . 352
7.6.1 gRPC service definition . . . 353
7.6.2 Example client . . . 356

8 Maintenance 357
8.1 Creating and restoring backups of settings . . . 357
8.2 Updating the software license . . . 357
8.3 Downloading log files . . . 358

9 Troubleshooting 359
9.1 Camera-image issues . . . 359
9.2 Depth/Disparity, error, and confidence image issues . . . 359

10 Contact 361
10.1 Support . . . 361
10.2 Downloads . . . 361
10.3 Address . . . 361

11 Appendix 362
11.1 Pose formats . . . 362
11.1.1 Rotation matrix and translation vector . . . 363
11.1.2 ABB pose format . . . 363
11.1.3 FANUC XYZ-WPR format . . . 363
11.1.4 Franka Emika Pose Format . . . 364
11.1.5 Fruitcore HORST pose format . . . 366
11.1.6 Kawasaki XYZ-OAT format . . . 366
11.1.7 KUKA XYZ-ABC format . . . 367
11.1.8 Mitsubishi XYZ-ABC format . . . 368
11.1.9 Universal Robots pose format . . . 368
11.1.10 Yaskawa Pose Format . . . 369

HTTP Routing Table 371

Index 373

1 Introduction

Indications in the manual

To prevent damage to the equipment and ensure the user's safety, this manual indicates each precaution related to safety with *Warning*. Supplementary information is provided as a *Note*.

Warning: Warnings in this manual indicate procedures and actions that must be observed to avoid danger of injury to the operator/user, or damage to the equipment. Software-related warnings indicate procedures that must be observed to avoid malfunctions or unexpected behavior of the software.

Note: Notes are used in this manual to indicate supplementary relevant information.

1.1 Overview

The *rc_reason_stack* is a high-performance 3D-image-processing Docker software stack for deployment on Ubuntu machines. It enhances the computing capabilities of the Roboception stereo camera *rc_visard* and supports the *rc_viscore*, the Basler *Stereo ace* camera, the *Orbbec* camera and the *zivid* camera.

The *rc_reason_stack* provides real-time camera images and depth images, which can be used to compute 3D point clouds. Additionally, it provides confidence and error images as quality measures for each image acquisition. It offers an intuitive web UI (user interface) and standardized interfaces, making it compatible with all major image processing libraries.

With optionally available software modules the *rc_reason_stack* provides out-of-the-box solutions for object detection and robotic pick-and-place applications.

Note: This manual uses the metric system and mostly uses the units meter and millimeter. Unless otherwise specified, all dimensions in technical drawings are in millimeters.

2 Safety

Warning: The operator must have read and understood all of the instructions in this manual before handling the *rc_reason_stack* product.

Warning: If operating the *rc_reason_stack* with *rc_visard* product(s), the operator must have read and understood all of the safety, installation, and maintenance instructions given in the *rc_visard* manual.

Note: The term "operator" refers to anyone responsible for any of the following tasks performed in conjunction with *rc_reason_stack*:

- Installation
- Maintenance
- Inspection
- Calibration
- Programming
- Decommissioning

This manual explains the *rc_reason_stack*'s various components and general operations regarding the product's whole life-cycle, from installation through operation to decommissioning.

The drawings and photos in this documentation are representative examples; differences may exist between them and the delivered product.

2.1 General warnings

Note: Any use of the *rc_reason_stack* in noncompliance with these warnings is inappropriate and may cause injury or damage as well as void the warranty.

2.2 Intended use

Warning: The *rc_reason_stack* is **NOT** intended for safety critical applications.

The GigE Vision® industry standard used by the *rc_reason_stack* does not support authentication and encryption. All data from and to the device is transmitted without authentication and encryption and could be monitored or manipulated by a third party. It is the operator's responsibility to connect the *rc_reason_stack* only to a secured internal network.

Warning: The *rc_reason_stack* must be connected to secured internal networks.

The *rc_reason_stack* may be used only within the scope of its technical specification. Any other use of the product is deemed unintended use. Roboception will not be liable for any damages resulting from any improper or unintended use.

Warning: Always comply with local and/or national laws, regulations and directives on automation safety and general machine safety.

3 Installation

Warning: The instructions on *Safety* (Section 2) related to the *rc_reason_stack* must be read and understood prior to installation.

The *rc_reason_stack* is a Docker-based software stacks that can be installed on machines meeting the prerequisites given in *Prerequisites*. This chapter provides detailed information about installing the *rc_reason_stack* software.

3.1 Offline installation guide

This section explains the manual installation of *rc_reason_stack* on a host system. Unlike the automated Docker-compose workflow, the Docker images are first copied to the host machine and then loaded into Docker manually. Follow the steps below to get the stack up and running, ready for your development or production environment.

All commands must be executed on the host machine (not inside a container).

3.1.1 Prerequisites

Component	Minimum Version
Ubuntu	24.04 LTS
NVIDIA GPU	Any RTX with minimum 8GB VRAM, or better Nvidia RTX A4000, RTX 4000 Ada, RTX 3080, RTX 4070, RTX 4080
Docker	20.10+
NVIDIA Driver	535+ (the guide uses *nvidia-driver-575-server*)

The following files are provided by Roboception and needed for installation.

File	Description
rc_container-latest.tar	rc_container docker image
tritonserver-23.10.tar	triton server docker image
docker-compose.yml	The docker compose file
docker-compose.json	The docker compose file in JSON format

3.1.2 Install Ubuntu 24.04

This section can be skipped if a working Ubuntu 24.04 installation is present.

For installing Ubuntu, follow the official Ubuntu installation guide under https://ubuntu.com/download/desktop or https://ubuntu.com/download/server.

3.1.3 Install NVIDIA driver

The NVIDIA driver is required for the host to expose the GPU to Docker containers. After installing the driver, the GPU and its capabilities should be visible with `nvidia-smi`. If the driver is not installed or not loaded correctly, `nvidia-smi` will either not be found or will report "No devices were found".

```
# Update package lists
sudo apt update

# Install the latest NVIDIA driver (replace 525 with the version that matches your GPU)
sudo apt install -y nvidia-driver-525

# Reboot to load the driver
sudo reboot
```

After the reboot, verify that the driver is active:

```
$ nvidia-smi
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.125.02   Driver Version: 525.125.02   CUDA Version: 12.5     |
|-------------------------------+----------------------+----------------------+
| 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  GeForce RTX 3080    Off  | 00000000:01:00.0 Off |                  N/A |
| 30%   38C    P8    12W / 350W |    12MiB / 11264MiB  |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

The table shows:

- **GPU**: the device ID (0, 1, . . .)
- **Name**: the GPU model (e.g., GeForce RTX 3080)
- **Driver Version**: the installed NVIDIA driver
- **CUDA Version**: the CUDA toolkit that ships with the driver
- **Memory-Usage**: total RAM allocated to the GPU
- **GPU-Util**: current GPU utilization percentage

If this output is displayed, the driver is correctly installed and the GPU is ready to be used by the NVIDIA Container Toolkit and the containers.

3.1.4 Install Docker

```
# 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 \
```

(continues on next page)

(continued from previous page)

```
    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 Install NVIDIA Container Toolkit

The NVIDIA Container Toolkit gives Docker the ability to see, expose, and sandbox NVIDIA GPUs inside containers. Without it CUDA workloads cannot run in the container. The toolkit is the bridge between Docker's container runtime and the NVIDIA driver stack on the host

```
# 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
```

For fixing an issue that lets the GPU fail after some time in the container (can be seen when `nvidia-smi` in the container fails), open the file `/etc/nvidia-container-runtime/config.toml` and set `no-cgroups = false`. After changing the configuration, start docker with:

```
sudo systemctl restart docker
```

Test that Docker can access the GPU:

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

3.1.6 Install WIBU CodeMeter runtime

Install the CodeMeter runtime (https://www.wibu.com/de/support/anwendersoftware/anwendersoftware.html) on the host system.

After installing, stop the runtime:

```
sudo service codemeter stop
```

Switch network licensing on by setting `IsNetworkServer` to 1 in the file `/etc/wibu/CodeMeter/Server.ini`.

Start the runtime:

```
sudo service codemeter start
```

A firewall may be used to not expose the WIBU network license server in the external network (WIBU uses the ports 22350-22352), since the license server must only be visible to the docker container.

3.1.7 Create a `sensor0` network

Example network setup using separate ethernet port (`enp9s0`) for `sensor0` interface via macvlan. Adjust the name of the ethernet port accordingly, in this example port `enp9s0` will be used.

Create `/etc/netplan/40-sensor0.yaml` with content

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

Alternative network configuration for multiple cameras via bridged network interfaces can be created as follows. In this example the interfaces enp7s0, ens4f0 and ens4f1 are bridged to br0.

```
network:
version: 2
renderer: networkd
ethernets:
   enp7s0:
      dhcp4: false
      dhcp6: false
   ens4f0:
      dhcp4: false
      dhcp6: false
   ens4f1:
      dhcp4: false
      dhcp6: false
bridges:
   br0:
      interfaces: [enp7s0, ens4f0, ens4f1]
      addresses: [172.23.42.1/28]
      parameters:
```

(continues on next page)

(continued from previous page)

```
    stp: true
    forward-delay: 4
    dhcp4: false
    dhcp6: false
```

Change permissions and apply with:

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

Create docker network with macvlan driver:

```
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 bridged network
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=br0 sensor0
```

3.1.8 Ensure network settings for GigE Vision

GigE Vision cameras stream images with high bandwidth via UDP packages. Lost packages lead to image loss that degrades the performance of the application. To avoid that, the Ethernet read buffers should be increased on the host. For Ubuntu, create the file `/etc/sysctl.d/10-gev-perf.conf` with the following content:

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

Apply settings with:

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

3.1.9 Load container images

```
gunzip -c ./rc_container-latest.tar.gz | docker load
gunzip -c ./tritonserver-23.10.tar.gz | docker load
```

3.1.10 Start the Docker stack

The preferred method to start the docker-compose stack is

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

If the host system requires a docker-compose file in JSON format, use the following command.

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

Wait a few minutes for all containers to start. The status can be monitored with:

```
docker compose ps
```

3.1.11 Access the Web GUI

Once the stack is running, the Web GUI can be accessed via:

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

3.1.12 Troubleshooting

Symptom	Likely Cause	Fix
docker: error: driver nvidia does not support the requested device	NVIDIA driver / Docker integration mismatch	Re-run the NVIDIA Container Toolkit installation and reboot
Containers fail to start	Wrong network name	Ensure a Docker network named `sensor0` exists
Web GUI not reachable	Containers not up	`docker compose logs` to inspect errors
Very low depth image frame rate	GPU does not work in container	Verify by running `nvidia-smi` on the host and inside the container and fix problems **[1]**

[1] If `nvidia-smi` on the host fails, ensure that packages are consistent, because an unattended upgrade under Ubuntu may upgrade the Nvidia driver, but not the Nvidia toolkit. This can be fixed by running `sudo apt update && sudo apt upgrade` manually. Unattended upgrades may be disabled. If `nvidia-smi` fails inside the container, ensure that `no-cgroups = false` in `/etc/nvidia-container-runtime/config.toml` and restart docker if the configuration had to be changed. This configuration file may have been overwritten by an update of the Nvidia container toolkit.

3.2 Software license

The *rc_reason_stack* ships with a USB dongle for licensing and protection of the installed software packages. The purchased software licenses are installed on and are bound to this dongle and its ID.

The functionality of the *rc_reason_stack* can be enhanced anytime by *upgrading the license* (Section 8.2), e.g., for optionally available software modules.

Note: The *rc_reason_stack* requires to be restarted whenever the installed licenses have changed.

Note: The dongle ID and the license status can be retrieved via the *rc_reason_stack*'s various interfaces such as the *System* → *Firmware & License* page of the *Web GUI* (Section 7.1).

Note: For the software components to be properly licensed, the USB dongle must be plugged to the *rc_reason_stack* **before power up**.

Note: The *rc_reason_stack* requires to be restarted, whenever the license dongle is plugged to or unplugged from the device.

3.3 Connection of cameras

The *rc_reason_stack* offers up to four software *camera pipelines* for processing data from the connected sensors. The configuration of the camera pipelines is explained in *Camera pipelines* (see Section 4.2.3).

4 Measurement principles

The *rc_reason_stack* is a high-performance 3D-image-processing software stack that is used in combination with one or more 3D cameras such as Roboception's 3D camera *rc_visard* or *rc_viscore*. Together, they provide rectified camera, disparity, confidence, and error images, which allow the viewed scene's depth values along with their uncertainties to be computed.

In the following, the underlying measurement principles are explained in more detail.

4.1 Stereo vision

In *stereo vision*, 3D information about a scene can be extracted by comparing two images taken from different viewpoints. The main idea behind using a camera pair for measuring depth is the fact that object points appear at different positions in the two camera images depending on their distance from the camera pair. Very distant object points appear at approximately the same position in both images, whereas very close object points occupy different positions in the left and right camera image. The object points' displacement in the two images is called *disparity*. The larger the disparity, the closer the object is to the camera. The principle is illustrated in Fig. 4.1.

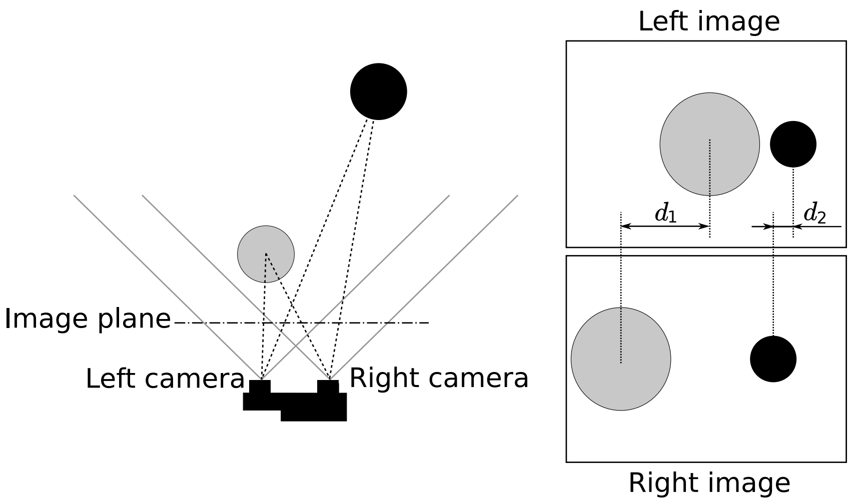

Fig. 4.1: Sketch of the stereo-vision principle: The more distant object (black) exhibits a smaller disparity d_2 than that of the close object (gray), d_1.

Stereo vision is a form of passive sensing, meaning that it emits neither light nor other signals to measure distances, but uses only light that the environment emits or reflects. Thus, the Roboception products utilizing this sensing principle can work indoors and outdoors and multiple devices can work together without interferences.

To compute the 3D information, the stereo matching algorithm must be able to find corresponding object points in the left and right camera images. For this, the algorithm requires texture, meaning changes in image intensity values due to patterns or the objects' surface structure, in the images. Stereo matching is not possible for completely untextured regions, such as a flat white wall without any visible surface structure. The stereo matching method used by the *rc_reason_stack* is *SGM* (*Semi-Global Matching*), which provides the best trade-off between runtime and accuracy, even for fine structures.

The following software modules are required to compute 3D information:

- *Camera module*: This module is responsible for capturing synchronized image pairs and transforming them into images approaching those taken by an ideal camera (rectification).
- *Stereo matching module*: This module computes disparities for the rectified stereo image pair using *SGM* (Section 6.2.2).

4.2 General information on 3D data

While on stereo pipelines, such as `rc_visard`, `rc_viscore` and `stereo_ace`, disparity images are computed by matching the left and right camera images, on `zivid` or `orbbec` pipelines the 3D data is internally converted into a disparity image that can be used to compute depth information using a provided virtual baseline.

The following sections describe how disparity images are computed from stereo image pairs and how disparity, error and confidence images can be used to compute depth data and depth errors.

4.2.1 Computing disparity images

After rectification, an object point is guaranteed to be projected onto the same pixel row in both left and right image. That point's pixel column in the right image is always lower than or equal to the same point's pixel column in the left image. The term disparity signifies the difference between the pixel columns in the right and left images and expresses the depth or distance of the object point from the camera. The disparity image stores the disparity values of all pixels in the left camera image.

The larger the disparity, the closer the object point. A disparity of 0 means that the projections of the object point are in the same image column and the object point is at infinite distance. Often, there are pixels for which disparity cannot be determined. This is the case for occlusions that appear on the left sides of objects, because these areas are not seen from the right camera. Furthermore, disparity cannot be determined for textureless areas. Pixels for which the disparity cannot be determined are marked as invalid with the special disparity value of 0. To distinguish between invalid disparity measurements and disparity measurements of 0 for objects that are infinitely far away, the disparity value for the latter is set to the smallest possible disparity value above 0.

To compute disparity values, the stereo matching algorithm has to find corresponding object points in the left and right camera images. These are points that represent the same object point in the scene. For stereo matching, the *rc_reason_stack* uses *SGM* (*Semi-Global Matching*), which offers quick run times and great accuracy, especially at object borders, fine structures, and in weakly textured areas.

A key requirement for any stereo matching method is the presence of texture in the image, i.e., image-intensity changes due to patterns or surface structure within the scene. In completely untextured regions such as a flat white wall without any structure, disparity values can either not be computed or the results are erroneous or have low confidence (see *Confidence and error images*, Section 4.2.3). The texture in the scene should not be an artificial, repetitive pattern, since those structures may lead to ambiguities and hence to wrong disparity measurements.

When working with poorly textured objects or in untextured environments, a static artificial texture can be projected onto the scene using an external pattern projector. This pattern should be random-like and not contain repetitive structures. The *rc_reason_stack* provides the IOControl module (see *IO and Projector Control*, Section 6.4.4) as optional software module which can control a pattern projector connected to the sensor.

4.2.2 Computing depth images and point clouds

The following equations show how to compute an object point's actual 3D coordinates P_x, P_y, P_z in the camera coordinate frame from the disparity image's pixel coordinates p_x, p_y and the disparity value d in pixels:

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

where f is the focal length after rectification in pixels and t is the stereo baseline in meters, which was determined during calibration.

Note: The *rc_reason_stack* reports a focal length factor via its various interfaces. It relates to the image width for supporting different image resolutions. The focal length f in pixels can be easily obtained by multiplying the focal length factor by the image width in pixels.

Please note that equations (4.1) assume that the coordinate frame is centered in the principal point that is typically in the center of the image, and p_x, p_y refer to the middle of the pixel, i.e. by adding 0.5 to the integer pixel coordinates. The following figure shows the definition of the image coordinate frame.

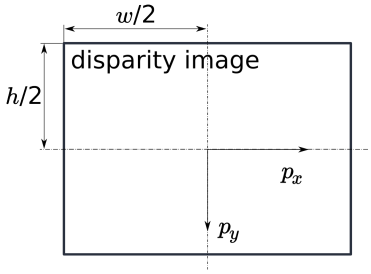

Fig. 4.2: The image coordinate frame's origin is defined to be at the image center – w is the image width and h is the image height.

The set of all object points computed from the disparity image gives the point cloud, which can be used for 3D modeling applications. The disparity image is converted into a depth image by replacing the disparity value in each pixel with the value of P_z.

Note: Roboception provides software and examples for receiving disparity images from the *rc_reason_stack* via GigE Vision and computing depth images and point clouds. See http://www.roboception.com/download.

4.2.3 Confidence and error images

For each disparity image, additionally an error image and a confidence image are provided, which give uncertainty measures for each disparity value. These images have the same resolution and the same frame rate as the disparity image. The error image contains the disparity error d_{eps} in pixels corresponding to the disparity value at the same image coordinates in the disparity image. The confidence image contains the corresponding confidence value c between 0 and 1. The confidence is defined as the probability of the true disparity value being within the interval of three times the error around the measured disparity d, i.e., $[d - 3d_{eps}, d + 3d_{eps}]$. Thus, the disparity image with error and confidence values can be used in applications requiring probabilistic inference. The confidence and error values corresponding to an invalid disparity measurement will be 0.

The disparity error d_{eps} (in pixels) can be converted to a depth error z_{eps} (in meters) using the focal length f (in pixels), the baseline t (in meters), and the disparity value d (in pixels) of the same pixel in the disparity image:

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

Combining equations (4.1) and (4.2) allows the depth error to be related to the depth:

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

5 Camera pipelines

The *rc_reason_stack* supports multiple cameras at the same time. For this, it offers up to four *camera pipelines* that can be configured by the user.

A camera pipeline contains several software modules which are responsible for acquiring data of the camera connected to that pipeline, performing detections or configuring modules used in this pipeline, e.g. by hand-eye calibration.

The *rc_reason_stack* supports cameras of type *rc_visard*, *rc_viscore*, *zivid*, *Orbbec* and *Stereo ace*. The type of the corresponding camera pipeline has to be configured to match the connected device.

5.1 Configuration of camera pipelines

The camera pipelines can be configured via the *Web GUI* (Section 7.1) under *System* → *Camera Pipelines*. This page shows the running pipelines with their types and the connected devices.

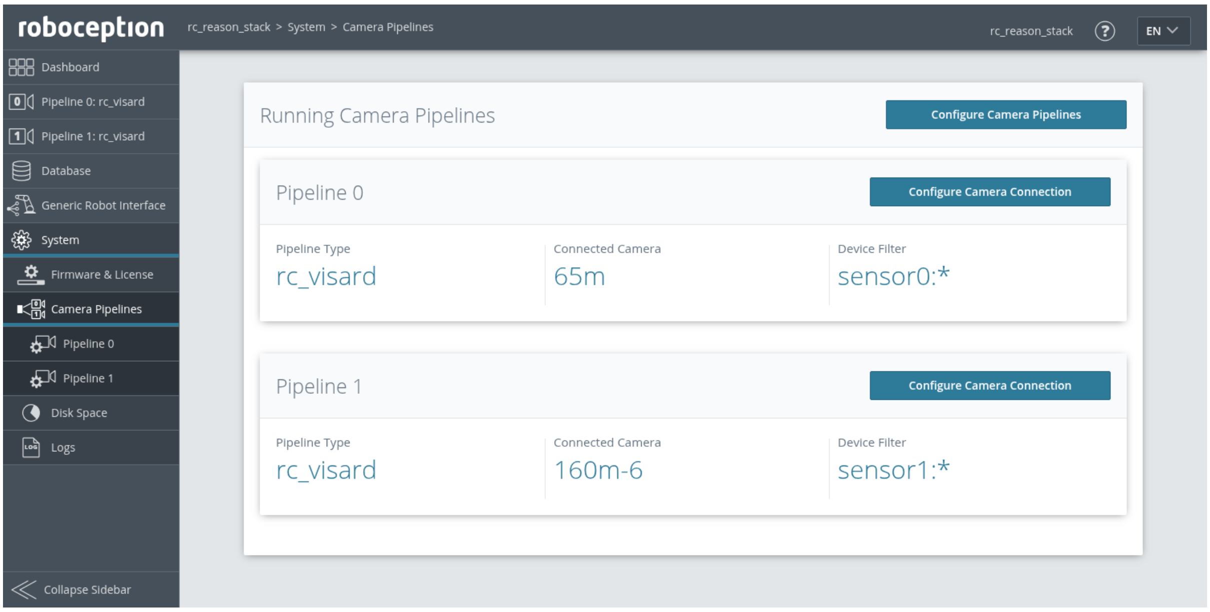

Fig. 5.1: Example of the *Camera Pipelines* page on an *rc_reason_stack* with two running pipelines of type `rc_visard`

Clicking on *Configure Camera Pipelines* allows to configure the number and type of running pipelines as shown in the next figure.

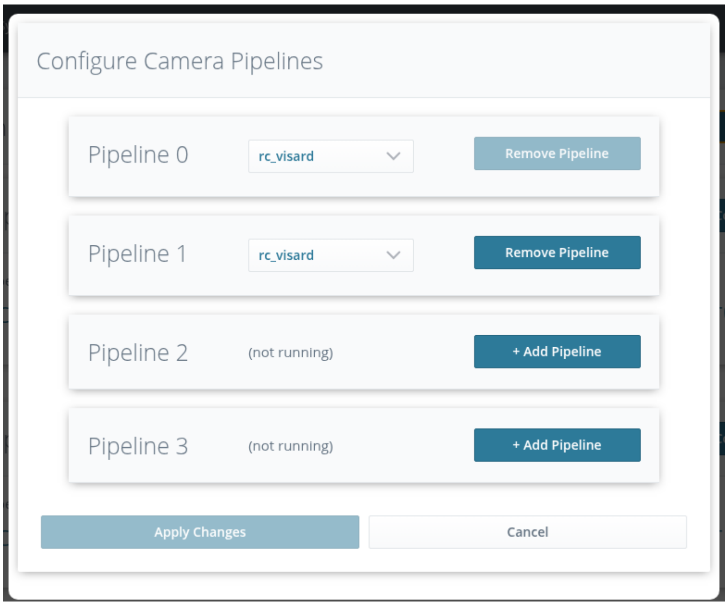

Fig. 5.2: Configuring the camera pipelines

The type of a running pipeline can be changed by selecting a different type in the drop down field. A running pipeline can be removed by clicking *Remove Pipeline*. Only pipeline 0 can never be removed, because this is the primary pipeline. Clicking on *+ Add Pipeline* allows to choose the type for the new pipeline and creates a new pipeline of the chosen type.

Once the pipelines are configured as desired, clicking *Apply Changes* will apply the new configuration. Then, the *rc_reason_stack* has to be restarted for the changes to take effect.

5.2 Configuration of connected cameras

A pipeline of a certain type can only discover devices of the same type. That means, a pipeline of type `rc_visard` can only connect to an *rc_visard*. In case multiple cameras of the same type are connected to the *rc_reason_stack*, the user can set a *device filter* to choose a specific camera for each pipeline. The current device filter value is displayed for each running pipeline as shown in Fig. 5.1. By default, the device filter is set to $*$, which means that any device matching the pipeline type will automatically be connected, but only if there is a unique match. Otherwise, no camera will be connected to that pipeline and an error will be shown.

To adjust the device filter and select the camera to be connected to a pipeline, click on *Configure Camera Connection* on the *Camera Pipelines* page, or select the corresponding pipeline in the menu, e.g. under *System* → *Camera Pipelines* → *Pipeline 1*. This will show the current device filter value and more information about the connected camera.

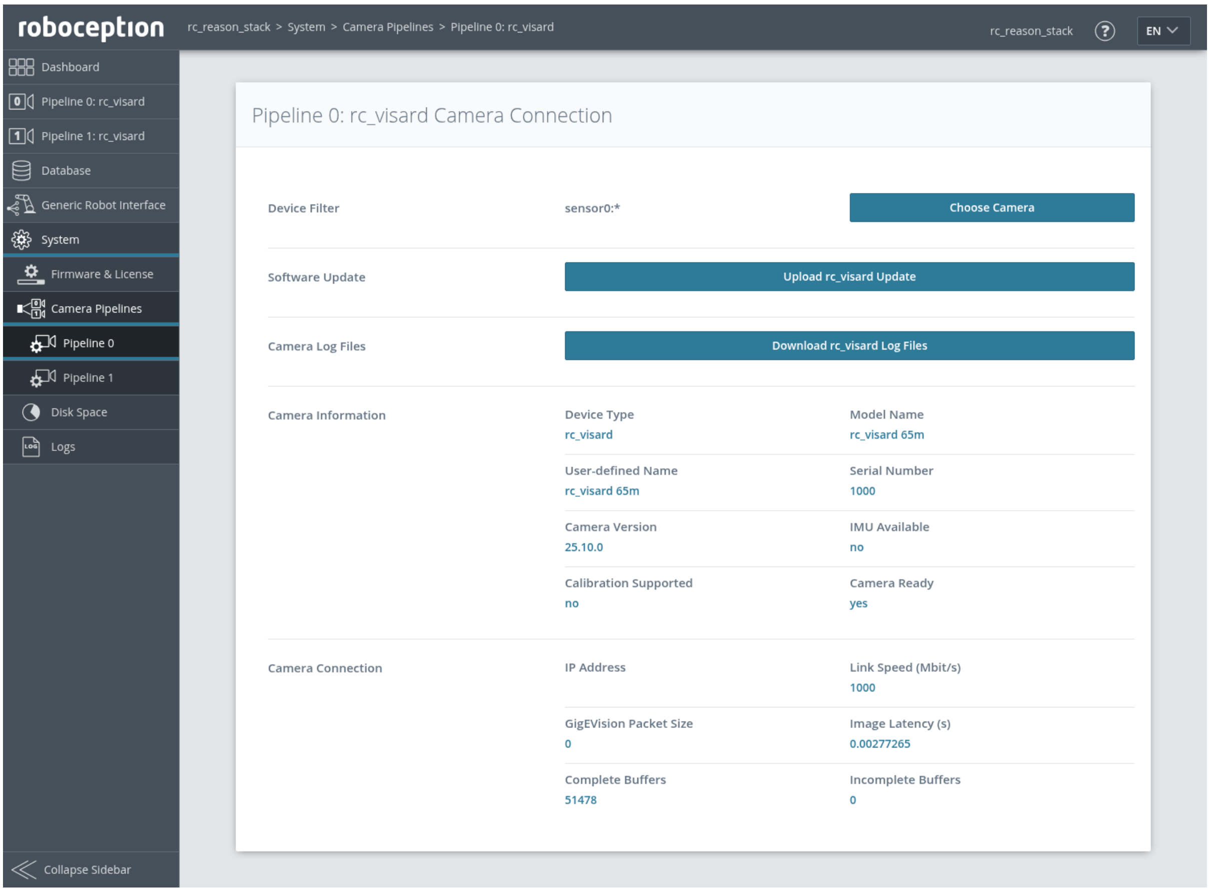

Fig. 5.3: Configuring the camera connection of pipeline 0

Clicking *Choose Camera* opens a dialog to edit the device filter.

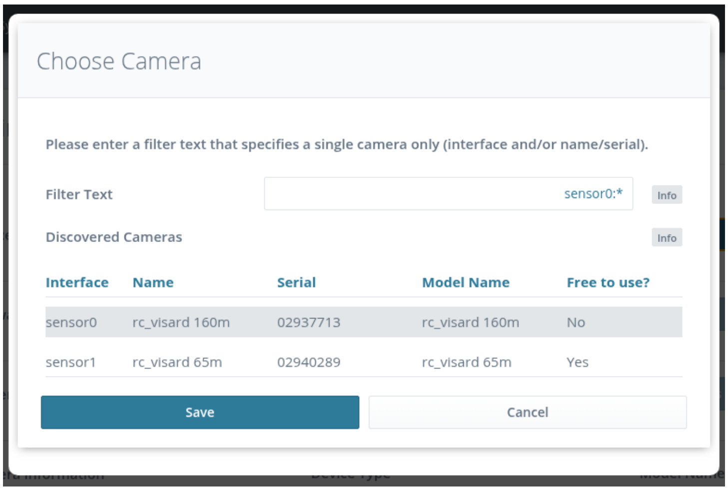

Fig. 5.4: Choosing the camera by setting a device filter

This dialog also shows a list of all discovered devices matching the pipeline type and highlights the ones that match the current value entered for the device filter. It also indicates if the devices are already in use in a different pipeline. Device filters can be selected by clicking on an *Interface*, *Name* or *Serial* of the desired device in the list. The following table shows possible device filter values.

Table 5.1: Possible device filter values

Device filter	Description
*	selects any device matching the pipeline type
sensor<n>:*	selects any device connected via the sensor<n> interface that matches the pipeline type
<name>	selects the device by the user-defined name
<serial>	selects the device by the full serial number
sensor<n>:<serial>	selects the device connected via the sensor<n> interface with the given serial
sensor<n>:<name>	selects the device connected via the sensor<n> interface with the given user-defined name
	if empty, no camera will be connected

By pressing *Save*, the entered device filter is applied and a camera matching the device filter is connected to this pipeline, if possible. Changing the device filter does not require a restart of the *rc_reason_stack*.

6 Software modules

The *rc_reason_stack* comes with several software modules, each of which corresponds to a certain functionality and can be interfaced via its respective *node* in the *REST-API interface* (Section 7.2) or in the *Generic Robot Interface* (Section 7.3).

The *rc_reason_stack* offers the possibility to connect multiple 3D cameras such as the *rc_visard*. The image data from each device is processed in a separate *camera pipeline*, which consists of several different software modules. The modules inside each pipeline are pipeline specific, which means that they can have different parameters for each pipeline. The modules running outside the pipelines are global and provide data for all modules in all pipelines. An overview is given in Fig. 6.1. The port numbers are given for the default installation as described in *Installation* (Section **??**) and may be changed in the docker compose file.

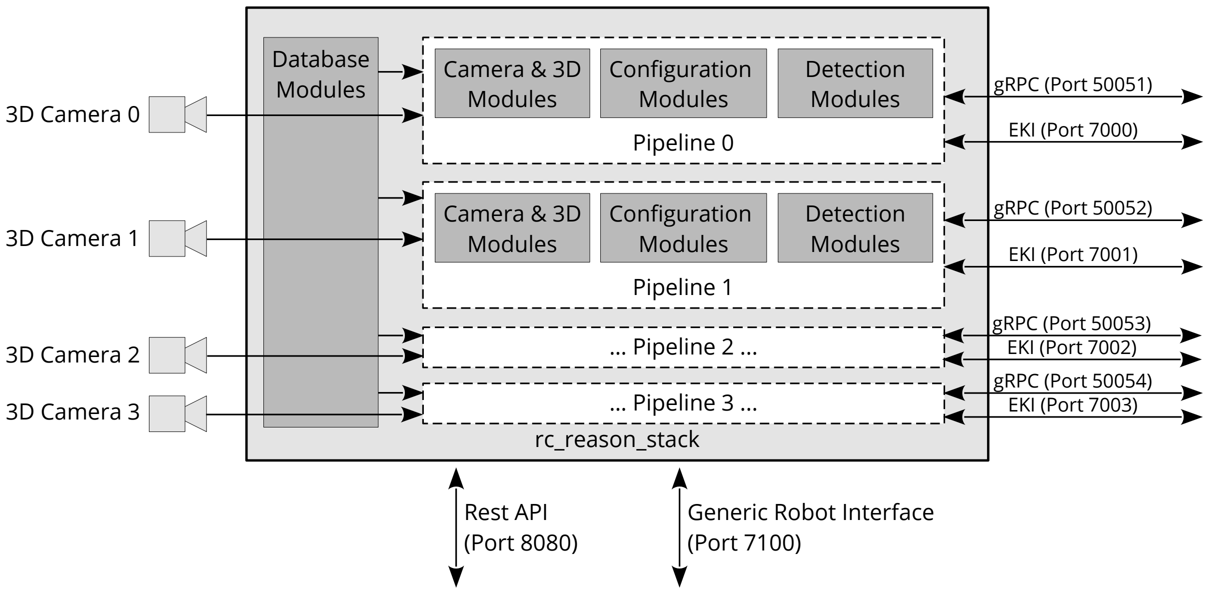

Fig. 6.1: Overview of the pipeline-specific and global software modules on the *rc_reason_stack*

The *rc_reason_stack*'s pipeline-specific software modules can be divided into

- ***Camera module*** **(Section 6.1)** acquires images and performs planar rectification for using the camera as a measurement device. Depending on the chosen camera pipeline type, this module offers different run-time parameters.
- ***3D modules*** **(Section 6.2)** which provide 3D depth information such as disparity, error, and confidence images,
- ***Detection & Measure modules*** **(Section 6.3)** which provide a variety of detection functionalities, such as grasp point computation and object detection,
- ***Configuration modules*** **(Section 6.4)** which enable the user to perform calibrations and configure the *rc_reason_stack* for specific applications.

The modules that are global for all camera pipelines running on the *rc_reason_stack* are the

- ***Database modules* (Section 6.5)** which enable the user to configure global data available to all other modules, such as load carriers, regions of interest and grippers.

6.1 Camera module

The camera module is a base module which is available on every *rc_reason_stack* and is responsible for image acquisition and rectification. It provides various parameters, e.g. to control exposure and frame rate.

Note: Depending on the chosen camera pipeline type, this module offers different run-time parameters.

6.1.1 Rectification

To simplify image processing, the camera module rectifies all camera images based on the camera calibration. This means that lens distortion is removed and the principal point is located exactly in the middle of the image.

The model of a rectified camera is described with just one value, which is the focal length. The *rc_reason_stack* reports a focal length factor via its various interfaces. It relates to the image width for supporting different image resolutions. The focal length f in pixels can be easily obtained by multiplying the focal length factor by the image width in pixels.

In case of a stereo camera, rectification also aligns images such that an object point is always projected onto the same image row in both images. The cameras' optical axes become exactly parallel.

Note: If a `zivid` or `orbbec` camera is used instead of a stereo camera, only one camera image is provided. However, the image is rectified, i.e. lens distortion is removed and the principal point is in the image center.

6.1.2 Viewing and downloading images

The *rc_reason_stack* provides time-stamped rectified images via the *gRPC image stream interface* (see Section 7.6).

Live streams of the images are provided with reduced quality in the *Web GUI* (Section 7.1).

The Web GUI also provides the possibility to download a snapshot of the current scene as a .tar.gz file as described in *Downloading camera images* (Section 7.1.4).

6.1.3 Pipeline types *rc_visard* and *rc_viscore*

6.1.3.1 Parameters

The camera module is called `rc_camera` and is represented by the *Camera* page in the desired pipeline in the *Web GUI* (Section 7.1). The user can change the camera parameters there, or directly via the REST-API (*REST-API interface*, Section 7.2).

Parameter overview

Note: The minimum, maximum and default values in the parameter table below show the values of the *rc_visard*. The values will be different for other camera models and for the `rc_viscore` pipeline.

This module offers the following run-time parameters:

Table 6.1: The `rc_camera` module's run-time parameters on a pipeline of type `rc_visard`

Name	Type	Min	Max	Default	Description
`acquisition_mode`	string	-	-	Continuous	Acquisition mode: [Continuous, Trigger]
`exp_auto`	bool	false	true	true	Switching between auto and manual exposure (deprecated, please use exp_control instead)
`exp_auto_average_max`	float64	0.0	1.0	0.75	Maximum average intensity in Auto exposure mode
`exp_auto_average_min`	float64	0.0	1.0	0.25	Minimum average intensity in Auto exposure mode
`exp_auto_mode`	string	-	-	Normal	Auto-exposure mode: [Normal, Out1High, AdaptiveOut1]
`exp_control`	string	-	-	Auto	Exposure control mode: [Manual, Auto, HDR]
`exp_height`	int32	0	959	0	Height of auto exposure region. 0 for whole image.
`exp_max`	float64	6.6e-05	0.018	0.018	Maximum exposure time in seconds in Auto exposure mode
`exp_offset_x`	int32	0	1279	0	First column of auto exposure region
`exp_offset_y`	int32	0	959	0	First row of auto exposure region
`exp_value`	float64	6.6e-05	0.018	0.005	Exposure time in seconds in Manual exposure mode
`exp_width`	int32	0	1279	0	Width of auto exposure region. 0 for whole image.
`fps`	float64	1.0	25.0	25.0	Frames per second in Hertz
`gain_value`	float64	0.0	18.0	0.0	Gain value in decibel if not in Auto exposure mode
`gamma`	float64	0.1	10.0	1.0	Gamma factor
`trigger_activation`	string	-	-	RisingEdge	Trigger activation: [RisingEdge, FallingEdge, AnyEdge]
`trigger_source`	string	-	-	Software	Trigger source: [Software, In1, In2, In3, In4]
`wb_auto`	bool	false	true	true	Switching white balance on and off (only for color camera)
`wb_ratio_blue`	float64	0.125	8.0	2.4	Blue balance ratio if wb_auto is false (only for color camera)
`wb_ratio_red`	float64	0.125	8.0	1.2	Red balance ratio if wb_auto is false (only for color camera)

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's *Camera* page. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI.

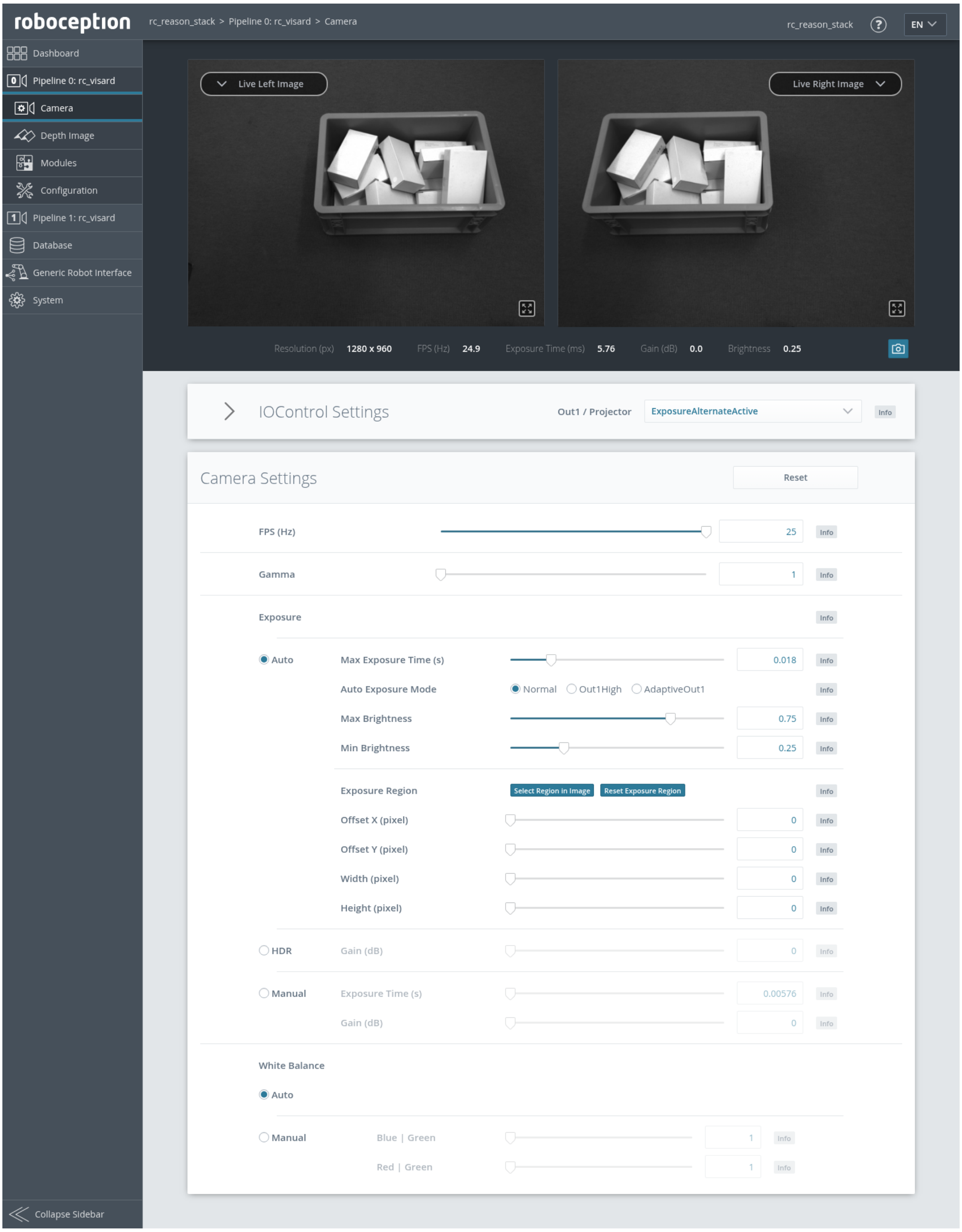

Fig. 6.2: The Web GUI's *Camera* page

fps (*FPS (Hz)*)

This value is the cameras' frame rate (fps, frames per second), which determines the upper frequency at which depth images can be computed. This is also the frequency at which the *rc_reason_stack* delivers images via GigE Vision. Reducing this frequency also reduces the network bandwidth required to transmit the images.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`gamma` (*Gamma*)

The gamma value determines the mapping of perceived light to the brightness of a pixel. A gamma value of 1 corresponds to a linear relationship. Lower gamma values let dark image parts appear brighter. A value around 0.5 corresponds to human vision.

> **Note:** For a pipeline of type `rc_visard` this value can only be changed when the connected *rc_visard* has at least firmware version 22.07. Otherwise the gamma value will always be 1.0.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`exp_control` (Exposure *Auto*, *HDR* or *Manual*)

The exposure control mode can be set to *Auto*, *HDR* or *Manual*. This replaces the deprecated *exp_auto* parameter.

Auto: This is the default mode in which the exposure time and gain factor is chosen automatically to correctly expose the image. The last automatically determined exposure and gain values are set into `exp_value` and `gain_value` when switching auto-exposure off.

HDR: The HDR mode computes high-dynamic-range images by combining images with different exposure times to avoid under-exposed and over-exposed areas. This decreases the frame rate and is only suitable for static scenes.

Manual: In the manual exposure mode the exposure time and gain are kept fixed independent of the resulting image brightness.

> **Note:** For a pipeline of type `rc_visard` the *HDR* mode is only available when the connected *rc_visard* has at least firmware version 23.01.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

exp_auto_mode (*Auto Exposure Mode*)

The auto exposure mode can be set to *Normal*, *Out1High* or *AdaptiveOut1*. These modes are relevant when the *rc_reason_stack* is used with an external light source or projector connected to the camera's GPIO Out1, which can be controlled by the IOControl module (*IO and Projector Control*, Section 6.4.4).

Normal: All images are considered for exposure control, except if the IOControl mode for GPIO Out1 is *ExposureAlternateActive*: then only images where GPIO Out1 is HIGH will be considered, since these images may be brighter in case GPIO Out1 is used to trigger an external light source.

Out1High: This exposure mode adapts the exposure time using only images with GPIO Out1 HIGH. Images where GPIO Out1 is LOW are not considered at all, which means, that the exposure time does not change when only images with Out1 LOW are acquired. This mode is recommended for using the `acquisition_mode` `SingleFrameOut1` in the stereo matching module as described in *Stereo Matching Parameters* (Section 6.2.2.1) and having an external projector connected to GPIO Out1, when changes in the brightness of the scene should only be considered when Out1 is HIGH. This is the case, for example, when a bright part of the robot moves through the field of view of the camera just before a detection is triggered, which should not affect the exposure time.

AdaptiveOut1: This exposure mode uses all camera images and tracks the exposure difference between images with GPIO Out1 LOW and HIGH. While the IOControl mode for GPIO Out1 is LOW, the images are under-exposed by this exposure difference to avoid over-exposure for when GPIO Out1 triggers an external projector. The resulting exposure difference is given as *Out1 Reduction* below the live images. This mode is recommended for using the `acquisition_mode` `SingleFrameOut1` in the stereo matching module as described in *Stereo Matching Parameters* (Section 6.2.2.1) and having an external projector connected to GPIO Out1, when changes in the brightness of the scene should be considered at all times. This is the case, for example, in applications where the external lighting changes.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

exp_max (*Max Exposure*)

This value is the maximal exposure time in auto-exposure mode in seconds. The actual exposure time is adjusted automatically so that the images are exposed correctly. If the maximum exposure time is reached, but the images are still underexposed, the *rc_reason_stack* stepwise increases the gain to increase the images' brightness. Limiting the exposure time is useful for avoiding or reducing motion blur during fast movements. However, higher gain introduces noise into the image. The best trade-off depends on the application.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

exp_auto_average_max (*Max Brightness*) and exp_auto_average_min (*Min Brightness*)

The auto-exposure tries to set the exposure time and gain factor such that the average intensity (i.e. brightness) in the image or exposure region is between a maximum and a minimum. The maximum brightness will be used if there is no saturation, e.g. no over-exposure due to bright surfaces or reflections. In case of saturation, the exposure time and gain factor are reduced, but only down to the minimum brightness.

The maximum brightness has precedence over the minimum brightness parameter. If the minimum brightness is larger than the maximum brightness, the auto-exposure always tries to make the average intensity equal to the maximum brightness.

The current brightness is always shown in the status bar below the images.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
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 (*Exposure Region*)

These values define a rectangular region in the left rectified image for limiting the area used for computing the auto exposure. The exposure time and gain factor of both images are chosen to optimally expose the defined region. This can lead to over- or underexposure of image parts outside the defined region. If either the width or height is 0, then the whole left and right images are considered by the auto exposure function. This is the default.

The region is visualized in the Web GUI by a rectangle in the left rectified image. It can be defined using the sliders or by selecting it in the image after pressing the button `Select Region in Image`.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

exp_value (*Exposure*)

This value is the exposure time in manual exposure mode in seconds. This exposure time is kept constant even if the images are underexposed.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

gain_value (*Gain (dB)*)

This value is the gain factor in decibel that can be set in manual exposure mode. Higher gain factors reduce the required exposure time but introduce noise.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

wb_auto (*White Balance Auto* or *Manual*)

This value can be set to *true* for automatic white balancing or *false* for manually setting the ratio between the colors using `wb_ratio_red` and `wb_ratio_blue`. The last automatically determined ratios are set into `wb_ratio_red` and `wb_ratio_blue` when switching automatic white balancing off. White balancing is without function for monochrome cameras and will not be displayed in the Web GUI in this case.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

wb_ratio_blue and wb_ratio_red (*Blue* / *Green* and *Red* / *Green*)

These values are used to set blue to green and red to green ratios for manual white balance. White balancing is without function for monochrome cameras and will not be displayed in the Web GUI in this case.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

6.1.3.2 Status values

This module reports the following status values:

Table 6.2: The `rc_camera` module's status values

Name	Description
`baseline`	Stereo baseline t in meters
`brightness`	Current brightness of the image as value between 0 and 1
`color`	0 for monochrome cameras, 1 for color cameras
`exp`	Current exposure time in seconds. This value is shown below the image preview in the Web GUI as *Exposure (ms)*.
`device_trigger_sources`	Gives the available trigger sources, in case the device can be triggered
`focal`	Focal length factor normalized to an image width of 1
`fps`	Current frame rate of the camera images in Hertz. This value is shown in the Web GUI below the image preview as *FPS (Hz)*.
`gain`	Current gain factor in decibel. This value is shown in the Web GUI below the image preview as *Gain (dB)*.
`gamma`	Current gamma value.
`height`	Height of the camera image in pixels. This value is shown in the Web GUI below the image preview as the second part of *Resolution (px)*.
`last_timestamp_grabbed`	Timestamp of the last image acquired in case the camera is in trigger mode
`out1_reduction`	Fraction of reduction (0.0 - 1.0) of brightness for images with GPIO Out1=LOW in exp_auto_mode=AdaptiveOut1 or exp_auto_mode=Out1High. This value is shown in the Web GUI below the image preview as *Out1 Reduction (%)*.
`params_override_active`	1 if parameters are temporarily overwritten by a running calibration process
`selfcalib_counter`	How often a correction has been performed by the self-calibration
`selfcalib_offset`	Current offset determined by the self-calibration
`test`	0 for live images and 1 for test images
`width`	Width of the camera image in pixels. This value is shown in the Web GUI below the image preview as the first part of *Resolution (px)*.

6.1.3.3 Services

The camera module offers the following services.

`reset_defaults`

Restores and applies the default values for this module's parameters ("factory reset").

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.1.4 Pipeline type *stereo_ace*

6.1.4.1 Parameters

The camera module on a pipeline of type `stereo_ace` is called `rc_camera` and is represented by the *Camera* page in the desired pipeline in the *Web GUI* (Section 7.1). The user can change the camera parameters there, or directly via the REST-API (*REST-API interface*, Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.3: The rc_camera module's run-time parameters on a pipeline of type stereo_ace

Name	Type	Min	Max	Default	Description
acquisition_mode	string	-	-	Continuous	Acquisition mode: [Continuous, Trigger]
brightness	float64	-1.0	1.0	0.0	Brightness
contrast	float64	-1.0	1.0	0.0	Contrast
contrast_mode	string	-	-	Linear	Contrast mode [Linear, SCurve]
exp_auto	bool	false	true	false	Switching between auto and manual exposure (deprecated, please use exp_control instead)
exp_auto_average_max	float64	0.0	1.0	0.75	Maximum average intensity in Auto exposure mode
exp_auto_average_min	float64	0.0	1.0	0.25	Minimum average intensity in Auto exposure mode
exp_auto_mode	string	-	-	Normal	Auto-exposure mode: [Normal, Out1High, AdaptiveOut1]
exp_control	string	-	-	Manual	Exposure control mode: [Manual, Auto, HDR]
exp_height	int32	0	2047	0	Height of auto exposure region. 0 for whole image.
exp_max	float64	6.6e-05	0.1	0.018	Maximum exposure time in seconds in Auto exposure mode
exp_offset_x	int32	0	2447	0	First column of auto exposure region
exp_offset_y	int32	0	2047	0	First row of auto exposure region
exp_value	float64	6.6e-05	0.1	0.005	Exposure time in seconds in Manual exposure mode
exp_width	int32	0	2447	0	Width of auto exposure region. 0 for whole image.
fps	float64	1.0	50.0	25.0	Frames per second in Hertz
gain_value	float64	0.0	48.0	0.0	Gain value in decibel if not in Auto exposure mode
gamma	float64	0.1	3.99998	1.0	Gamma factor
light_source_preset	string	-	-	Daylight6500K	Light source preset [Off, Tungsten, Daylight5000K, Daylight6500K, FactoryLED6000K]
saturation	float64	0.0	2.0	1.0	Saturation
trigger_activation	string	-	-	RisingEdge	Trigger activation: [RisingEdge, FallingEdge, AnyEdge]
trigger_source	string	-	-	Software	Trigger source: [Software, In1, In2, In3, In4]
wb_auto	bool	false	true	true	Switching white balance on and off (only for color camera)
wb_ratio_blue	float64	0.125	16.0	2.4	Blue balance ratio if wb_auto is false (only for color camera)
wb_ratio_green	float64	0.125	16.0	1.0	Green balance ratio if wb_auto is false (only for some color cameras)
wb_ratio_red	float64	0.125	16.0	1.2	Red balance ratio if wb_auto is false (only for color camera)

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's *Camera* page. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI.

acquisition_mode (*Acquisition Mode*)

This values determines the camera acquisition mode. In Continuous mode, the camera will acquire images at the specified frame rate fps. In Trigger mode, images are only acquired when the camera receives a trigger signal.

Note: This parameter only has an effect when used in a pipeline with an *rc_viscore* or *rc_visard NG*.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

trigger_source (*Trigger Source*)

This value is only used when acquisition_mode is set to Trigger and determines the source for the trigger. In Software mode a trigger can be sent via the rc_camera/acquisition_trigger service. When the acquisition_mode for the depth images is set to SingleFrame or SingleFrameOut1 (see *Parameters*, Section 6.2.2.1), the camera software trigger is sent automatically whenever a depth image acquisition is triggered. The modes In1 and In2 are hardware trigger modes. An image is acquired whenever a signal on the chosen input is received.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

trigger_activation (*Trigger Activation*)

This value is only used when acquisition_mode is set to Trigger and trigger_source is set to In1 or In2. It determines the signal edge that should be used to trigger an acquisition. Possible values are RisingEdge, FallingEdge or AnyEdge.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

fps (*FPS (Hz)*)

This value is the cameras' frame rate (fps, frames per second), which determines the upper frequency at which depth images can be computed. This is also the frequency at which the *rc_reason_stack* delivers images via GigE Vision. Reducing this frequency also reduces the network bandwidth required to transmit the images.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

gamma (*Gamma*)

The gamma value determines the mapping of perceived light to the brightness of a pixel. A gamma value of 1 corresponds to a linear relationship. Lower gamma values let dark image parts appear brighter. A value around 0.5 corresponds to human vision.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

contrast_mode (*Contrast Mode*)

The contrast mode can be set to `Linear` (*Linear*) or `SCurve` (*S-Curve*) and determines how the image intensity values are scaled when the contrast is adjusted. In the `Linear` mode, the camera uses a linear function to adjust the contrast. Increasing or decreasing the contrast, increases or decreases the gradient of the linear function. When increasing the contrast, the darkest and lightest regions of the image will appear completely black or completely white, but the other areas will appear more defined. Decreasing the contrast has the opposite effect. In the `SCurve` mode, the camera uses an S-curve function to adjust the contrast. Increasing the contrast darkens dark pixels and lightens light pixels, but the dynamic range of the image is preserved.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`contrast` (*Contrast*)

Adjusting the contrast increases or decreases the difference between light and dark areas in the image. The way the light and dark regions change when adjusting the contrast depends on the specified `contrast_mode`.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`exp_control` (Exposure *Auto*, *HDR* or *Manual*)

The exposure control mode can be set to *Auto*, *HDR* or *Manual*.

Auto: This is the default mode in which the exposure time and gain factor is chosen automatically to correctly expose the image. The last automatically determined exposure and gain values are set into `exp_value` and `gain_value` when switching auto-exposure off.

HDR: The HDR mode computes high-dynamic-range images by combining images with different exposure times to avoid under-exposed and over-exposed areas. This decreases the frame rate and is only suitable for static scenes.

Manual: In the manual exposure mode the exposure time and gain are kept fixed independent of the resulting image brightness.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`exp_auto_mode` (*Auto Exposure Mode*)

The auto exposure mode can be set to *Normal*, *Out1High* or *AdaptiveOut1*. These modes are relevant when the *rc_reason_stack* is used with an external light source or projector connected to the camera's GPIO Out1, which can be controlled by the IOControl module (*IO and Projector Control*, Section 6.4.4).

Normal: All images are considered for exposure control, except if the IOControl mode for GPIO Out1 is *ExposureAlternateActive*: then only images where GPIO Out1 is HIGH will be considered, since these images may be brighter in case GPIO Out1 is used to trigger an external light source.

Out1High: This exposure mode adapts the exposure time using only images with GPIO Out1 HIGH. Images where GPIO Out1 is LOW are not considered at all, which means, that the exposure time does not change when only images with Out1 LOW are acquired. This mode is recommended for using the `acquisition_mode` `SingleFrameOut1` in the stereo matching module as described in *Stereo Matching Parameters* (Section 6.2.2.1) and having an external projector connected to GPIO Out1, when changes in the brightness of the scene should only be considered when Out1 is HIGH. This is the case, for example, when a bright part of the robot moves through the field of view of the camera just before a detection is triggered, which should not affect the exposure time.

AdaptiveOut1: This exposure mode uses all camera images and tracks the exposure difference between images with GPIO Out1 LOW and HIGH. While the IOControl mode for GPIO Out1 is LOW, the images are under-exposed by this exposure difference to avoid over-exposure for when GPIO Out1 triggers an external projector. The resulting exposure difference is given as *Out1 Reduction* below the live images. This mode is recommended for using the `acquisition_mode` `SingleFrameOut1` in the stereo matching module as described in *Stereo Matching Parameters* (Section 6.2.2.1) and having an external projector connected to GPIO Out1, when changes in the brightness of the scene should be considered at all times. This is the case, for example, in applications where the external lighting changes.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`exp_max` (*Max Exposure*)

This value is the maximal exposure time in auto-exposure mode in seconds. The actual exposure time is adjusted automatically so that the images are exposed correctly. If the maximum exposure time is reached, but the images are still underexposed, the *rc_reason_stack* stepwise increases the gain to increase the images' brightness. Limiting the exposure time is useful for avoiding or reducing motion blur during fast movements. However, higher gain introduces noise into the image. The best trade-off depends on the application.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`exp_auto_average_max` (*Max Brightness*) and `exp_auto_average_min` (*Min Brightness*)

The auto-exposure tries to set the exposure time and gain factor such that the average intensity (i.e. brightness) in the image or exposure region is between a maximum and a minimum. The maximum brightness will be used if there is no saturation, e.g. no over-exposure due to bright surfaces or reflections. In case of saturation, the exposure time and gain factor are reduced, but only down to the minimum brightness.

The maximum brightness has precedence over the minimum brightness parameter. If the minimum brightness is larger than the maximum brightness, the auto-exposure always tries to make the average intensity equal to the maximum brightness.

The current brightness is always shown in the status bar below the images.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
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` (*Exposure Region*)

These values define a rectangular region in the left rectified image for limiting the area used for computing the auto exposure. The exposure time and gain factor of both images are chosen to optimally expose the defined region. This can lead to over- or underexposure of image parts outside the defined region. If either the width or height is 0, then the whole left and right images are considered by the auto exposure function. This is the default.

The region is visualized in the Web GUI by a rectangle in the left rectified image. It can be defined using the sliders or by selecting it in the image after pressing the button `Select Region in Image`.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

`exp_value` (*Exposure*)

This value is the exposure time in manual exposure mode in seconds. This exposure time is kept constant even if the images are underexposed.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`gain_value` (*Gain (dB)*)

This value is the gain factor in decibel that can be set in manual exposure mode. Higher gain factors reduce the required exposure time but introduce noise.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

brightness (*Brightness*)

Adjusting the brightness lightens or darkens the entire image.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

wb_auto (*White Balance Auto* or *Manual*, only available for color cameras)

This value can be set to *true* for automatic white balancing or *false* for manually setting the ratio between the colors using `wb_ratio_red` and `wb_ratio_blue`. The last automatically determined ratios are set into `wb_ratio_red` and `wb_ratio_blue` when switching automatic white balancing off. White balancing is without function for monochrome cameras and will not be displayed in the Web GUI in this case.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

wb_ratio_blue, wb_ratio_red and wb_ratio_green (*Blue Ratio*, *Red Ratio* and *Green Ratio*, only available for color cameras)

These values are used to set blue, red and green ratios for manual white balance. White balancing is without function for monochrome cameras and will not be displayed in the Web GUI in this case.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

light_source_preset (*Light Source Preset*, only available for color cameras)

The light source preset parameter allows to correct color shifts caused by certain light sources. Depending on its specific color temperature, the light used for image acquisition can cause color shifts in the image. These color shifts can be corrected by selecting the related light source preset. Possible values are *Off*, *Tungsten*, *Daylight5000K*, *Daylight6500K* and *FactoryLED6000K*.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

saturation (*Saturation*, only available for color cameras)

Adjusting the saturation changes the colorfulness (intensity) of the colors. A higher saturation, e.g., makes colors easier to distinguish.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

6.1.4.2 Status values

The rc_camera module reports the following status values on a pipeline of type stereo_ace.

Table 6.4: The `rc_camera` module's status values

Name	Description
`baseline`	Stereo baseline t in meters
`brightness`	Current brightness of the image as value between 0 and 1
`color`	0 for monochrome cameras, 1 for color cameras
`exp`	Current exposure time in seconds. This value is shown below the image preview in the Web GUI as *Exposure (ms)*.
`device_trigger_sources`	Gives the available trigger sources, in case the device can be triggered
`focal`	Focal length factor normalized to an image width of 1
`fps`	Current frame rate of the camera images in Hertz. This value is shown in the Web GUI below the image preview as *FPS (Hz)*.
`gain`	Current gain factor in decibel. This value is shown in the Web GUI below the image preview as *Gain (dB)*.
`gamma`	Current gamma value.
`height`	Height of the camera image in pixels. This value is shown in the Web GUI below the image preview as the second part of *Resolution (px)*.
`last_timestamp_grabbed`	Timestamp of the last image acquired in case the camera is in trigger mode
`out1_reduction`	Fraction of reduction (0.0 - 1.0) of brightness for images with GPIO Out1=LOW in exp_auto_mode=AdaptiveOut1 or exp_auto_mode=Out1High. This value is shown in the Web GUI below the image preview as *Out1 Reduction (%)*.
`params_override_active`	1 if parameters are temporarily overwritten by a running calibration process
`selfcalib_counter`	How often a correction has been performed by the self-calibration
`selfcalib_offset`	Current offset determined by the self-calibration
`test`	0 for live images and 1 for test images
`width`	Width of the camera image in pixels. This value is shown in the Web GUI below the image preview as the first part of *Resolution (px)*.

6.1.4.3 Services

The `rc_camera` module offers the following services on a pipeline of type `stereo_ace`.

`acquisition_trigger`

Triggers an image acquisition when `acquisition_mode` is set to `Trigger` and `trigger_source` is set to `Software`.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

reset_defaults

Restores and applies the default values for this module's parameters ("factory reset").

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.1.5 Pipeline type *orbbec*

Note: The firmware version of the connected *Orbbec* camera must be at least 1.6.00, otherwise the *Orbbec* cannot be used.

6.1.5.1 Parameters

The camera module on a pipeline of type `orbbec` is called `rc_camera` and is represented by the *Camera* page in the desired pipeline in the *Web GUI* (Section 7.1). The user can change the camera parameters there, or directly via the REST-API (*REST-API interface*, Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.5: The rc_camera module's run-time parameters on a pipeline of type orbbec

Name	Type	Min	Max	Default	Description
exp_control	string	-	-	Auto	Exposure control mode: [Manual, Auto]
exp_height	int32	0	799	0	Height of auto exposure region. 0 for whole image.
exp_max	float64	1.0	1999.0	665.0	Maximum exposure time in seconds in Auto exposure mode
exp_offset_x	int32	0	1279	0	First column of auto exposure region
exp_offset_y	int32	0	799	0	First row of auto exposure region
exp_value	float64	1.0	1999.0	156.0	Exposure time in seconds in Manual exposure mode
exp_width	int32	0	1279	0	Width of auto exposure region. 0 for whole image.
gain_value	float64	0.0	128.0	16.0	Gain value in decibel if not in Auto exposure mode
gamma	float64	100.0	500.0	300.0	Gamma factor
wb_auto	bool	false	true	true	Switching white balance on and off
wb_value	float64	2800.0	6500.0	4600.0	White balance value

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's *Camera* page. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI.

gamma (*Gamma*)

The gamma value determines the mapping of perceived light to the brightness of a pixel. Lower gamma values let dark image parts appear brighter.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

exp_control (Exposure *Auto* or *Manual*)

The exposure control mode can be set to *Auto* or *Manual*.

Auto: This is the default mode in which the exposure time and gain factor is chosen automatically to correctly expose the image. The last automatically determined exposure and gain values are set into exp_value and gain_value when switching auto-exposure off.

Manual: In the manual exposure mode the exposure time and gain are kept fixed independent of the resulting image brightness.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

exp_max (*Max Exposure*)

This value is the maximal exposure in auto-exposure mode in seconds. The actual exposure time is adjusted automatically so that the images are exposed correctly. If the maximum exposure is reached, but the images are still underexposed, the gain is stepwise increased to increase the image brightness. Limiting the exposure is useful for avoiding or reducing motion blur during fast movements. However, higher gain introduces noise into the image. The best trade-off depends on the application.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

exp_offset_x, exp_offset_y, exp_width, exp_height (*Exposure Region*)

These values define a rectangular region in the left rectified image for limiting the area used for computing the auto exposure. The exposure time and gain factor of both images are chosen to optimally expose the defined region. This can lead to over- or underexposure of image parts outside the defined region. If either the width or height is 0, then the whole left and right images are considered by the auto exposure function. This is the default.

The region is visualized in the Web GUI by a rectangle in the left rectified image. It can be defined using the sliders or by selecting it in the image after pressing the button `Select Region in Image`.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

exp_value (*Exposure*)

This value is the exposure in manual exposure mode. This exposure is kept constant even if the images are underexposed.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

gain_value (*Gain*)

This value is the gain factor that can be set in manual exposure mode. Higher gain factors reduce the required exposure time but introduce noise.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

wb_auto (*White Balance Auto* or *Manual*)

This value can be set to *true* for automatic white balancing or *false* for manually setting the white balance using `wb_value`.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

wb_value (*White Balance Manual Value*)

This value determines the white balance when `wb_auto` is *false*.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

6.1.5.2 Status values

The rc_camera module of an orbbec pipeline reports the following status values:

Table 6.6: The rc_camera module's status values

Name	Description
baseline	Internally assumed stereo baseline t in meters for disparity image computation
brightness	Current brightness of the image as value between 0 and 1
color	0 for monochrome cameras, 1 for color cameras
exp	Current exposure. This value is shown below the image preview in the Web GUI as *Exposure*.
focal	Focal length factor normalized to an image width of 1
fps	Current frame rate of the camera images in Hertz. This value is shown in the Web GUI below the image preview as *FPS (Hz)*.
gain	Current gain factor. This value is shown in the Web GUI below the image preview as *Gain*.
height	Height of the camera image in pixels. This value is shown in the Web GUI below the image preview as the second part of *Resolution (px)*.
last_capture_ok	1 if the last image capture was successful
last_timestamp_grabbed	Timestamp of the last image acquired
test	0 for live images and 1 for test images
width	Width of the camera image in pixels. This value is shown in the Web GUI below the image preview as the first part of *Resolution (px)*.

6.1.5.3 Services

In a pipeline of type orbbec the rc_camera module offers the following services.

reset_defaults

Restores and applies the default values for this module's parameters ("factory reset").

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.1.6 Pipeline type *zivid*

Note: The firmware version of the connected *zivid* camera has to match the version required by the *rc_reason_stack*, otherwise the *zivid* cannot be used. To update the *zivid* to the required firmware version, open the *Web GUI* (Section 7.1), navigate to *System* → *Camera Pipelines* and select the *zivid* pipeline. Then click on *Update zivid firmware* and wait for the update process to finish.

6.1.6.1 User-defined presets

The *zivid* camera comes with multiple pre-configured settings for capturing images, so-called presets. The 2D presets are tailored specifically for 2D image captures, focusing on settings like resolution, exposure time, brightness, and gain. They are optimized for applications requiring detailed color or monochrome images.

Users can also create own 2D presets using the *Zivid Studio* software (https://www.zivid.com/zivid-studio-software) and save them as .yml files. These preset files can be uploaded to the *rc_reason_stack* on the *Camera* page of the Web GUI or using the REST-API as described in *Presets API*. User-defined presets can then be selected for image acquisition in the same way as the pre-defined presets via the `preset_name` run-time parameter. Also 3D presets including 2D settings can be uploaded and used as 2D preset. In this case, only the 2D settings will be applied.

6.1.6.2 Parameters

The camera module on a pipeline of type `zivid` is called `rc_camera` and is represented by the *Camera* page in the desired pipeline in the *Web GUI* (Section 7.1). The user can change the camera parameters there, or directly via the REST-API (*REST-API interface*, Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.7: The `rc_camera` module's run-time parameters on a pipeline of type `zivid`

Name	Type	Min	Max	Default	Description
`acquisition_mode`	string	-	-	Trigger	Acquisition mode: [Continuous, Trigger]
`fps`	float64	1.0	25.0	25.0	Frames per second in Hertz
`preset_name`	string	-	-	-	Name of preset configuration

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's *Camera* page. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI.

acquisition_mode (*Acquisition Mode*)

This parameter determines the acquisition mode of the 2D camera images. In `Continuous` mode, the camera will acquire images at the specified frame rate `fps`. In `Trigger` mode, images are only acquired when the camera receives a software trigger signal, either by clicking on the *Acquire* button in the Web GUI, or by calling the `rc_camera/acquisition_trigger` service (see *Services* (Section 6.1.6.4)).

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

fps (*FPS (Hz)*)

This value is the camera's frame rate (fps, frames per second), which determines the upper frequency at which camera images can be acquired.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

preset_name (*Preset Name*)

This parameter allows to select a preset for 2D image acquisition. The preset can be any of the *zivid*'s pre-configured presets which depend on the *zivid* model and are read from the connected device, or a user-defined preset that has been uploaded to the *rc_reason_stack* (*User-defined presets*, Section 6.1.6.1).

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

6.1.6.3 Status values

The `rc_camera` module of a `zivid` pipeline reports the following status values:

Table 6.8: The `rc_camera` module's status values

Name	Description
`baseline`	Internally assumed stereo baseline t in meters for disparity image computation
`brightness`	Current brightness of the image as value between 0 and 1
`color`	0 for monochrome cameras, 1 for color cameras
`exp`	Current exposure time in seconds. This value is shown below the image preview in the Web GUI as *Exposure (ms)*.
`focal`	Focal length factor normalized to an image width of 1
`fps`	Current frame rate of the camera images in Hertz. This value is shown in the Web GUI below the image preview as *FPS (Hz)*.
`height`	Height of the camera image in pixels. This value is shown in the Web GUI below the image preview as the second part of *Resolution (px)*.
`last_capture_ok`	1 if the last image capture was successful
`last_timestamp_grabbed`	Timestamp of the last image acquired
`test`	0 for live images and 1 for test images
`width`	Width of the camera image in pixels. This value is shown in the Web GUI below the image preview as the first part of *Resolution (px)*.

6.1.6.4 Services

In a pipeline of type `zivid` the `rc_camera` module offers the following services.

`acquisition_trigger`

Triggers an image acquisition when `acquisition_mode` is set to `Trigger`.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

(continues on next page)

(continued from previous page)

```
  }
}
```

reset_defaults

Restores and applies the default values for this module's parameters ("factory reset").

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.1.6.5 Presets API

The 2D presets can be set, retrieved and deleted via the following REST-API endpoints.

GET /presets/rc_zivid/2d_presets

Get zivid 2D presets.

Template request

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

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – successful operation

GET /presets/rc_zivid/2d_presets/{id}

Get zivid 2D preset yml file.

Template request

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

Parameters

- **id** (*string*) – ID/filename without extension *(required)*

Response Headers

- Content-Type – application/octet-stream

Status Codes

- 200 OK – successful operation
- 404 Not Found – yml file not found

PUT /presets/rc_zivid/2d_presets/{id}

Create or update a zivid 2D preset yml file.

Template request

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

Parameters

- **id** (*string*) – ID/filename without extension *(required)*

Form Parameters

- **file** – preset yml file *(required)*

Request Headers

- Accept – multipart/form-data application/json

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – successful operation
- 400 Bad Request – yml is not valid or max number of elements reached
- 413 Request Entity Too Large – File too large

DELETE /presets/rc_zivid/2d_presets/{id}

Remove a zivid 2D preset yml file.

Template request

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

Parameters

- **id** (*string*) – ID/filename without extension *(required)*

Request Headers

- Accept – application/json

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – successful operation
- 404 Not Found – element not found

6.2 3D modules

The *rc_reason_stack*'s 3D camera software consists of the following modules:

- ***Stereo matching module*** **(`rc_stereomatching`, Section 6.2.2)** uses the rectified stereo image pairs of the connected stereo camera, e.g. the *rc_visard*, to compute 3D depth information such as disparity, error, and confidence images. This module only runs on pipeline types for stereo cameras, i.e. `rc_visard`, `rc_viscore` and `stereo_ace`.
- ***Zivid module*** **(`rc_zivid`, Section 6.2.3)** provides 3D depth information such as disparity, error, and confidence images of the connected *zivid* structured light camera. This module only runs on pipelines of type `zivid`.
- ***Orbbec module*** **(`rc_orbbec`, Section 6.2.4)** provides 3D depth information such as disparity, error, and confidence images of the connected *Orbbec* Gemini 335Le stereo camera. This module only runs on pipelines of type `orbbec`.

These modules are pipeline specific, which means that they run inside each camera pipeline. Changes to their settings or parameters only affect the corresponding pipeline and have no influence on the other camera pipelines running on the *rc_reason_stack*.

6.2.1 Viewing and downloading images and point clouds

The *rc_reason_stack* provides time-stamped disparity, error, and confidence images via the *gRPC image stream interface* (see Section 7.6).

Live streams of the images are provided with reduced quality on the *Depth Image* page in the desired pipeline of the *Web GUI* (Section 7.1).

The Web GUI also provides the possibility to download a snapshot of the current scene containing the depth, error and confidence images, as well as a point cloud in ply format as described in *Downloading depth images and point clouds* (Section 7.1.5).

6.2.2 Stereo matching module

The stereo matching module is a base module which is available on every *rc_reason_stack* and uses the rectified stereo-image pair to compute disparity, error, and confidence images.

Note: This module is not available in camera pipelines of type `zivid` or `orbbec`.

To compute full resolution disparity, error and confidence images, an additional StereoPlus *license* (Section 8.2) is required. This license is included in every *rc_reason_stack* purchased after 31.01.2019.

6.2.2.1 Parameters

The stereo matching module is called `rc_stereomatching` in the REST-API and it is represented by the *Depth Image* page in the desired pipeline in the *Web GUI* (Section 7.1). The user can change the stereo matching parameters there, or use the REST-API (*REST-API interface*, Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.9: The rc_stereomatching module's run-time parameters

Name	Type	Min	Max	Default	Description
acquisition_mode	string	-	-	Continuous	Acquisition mode: [Continuous, SingleFrame, SingleFrameOut1]
double_shot	bool	false	true	false	Combination of disparity images from two subsequent stereo image pairs
exposure_adapt_timeout	float64	0.0	2.0	0.0	Maximum time in seconds to wait after triggering in SingleFrame modes until auto exposure has finished adjustments
fill	int32	0	4	3	Disparity tolerance for hole filling in pixels
maxdepth	float64	0.1	100.0	100.0	Maximum depth in meters
maxdeptherr	float64	0.01	100.0	100.0	Maximum depth error in meters
minconf	float64	0.5	1.0	0.5	Minimum confidence
mindepth	float64	0.1	100.0	0.1	Minimum depth in meters
quality	string	-	-	High	Quality: [Low, Medium, High, Full]. Full requires 'stereo_plus' license.
seg	int32	0	4000	200	Minimum size of valid disparity segments in pixels
smooth	bool	false	true	true	Smoothing of disparity image (requires 'stereo_plus' license)
static_scene	bool	false	true	false	Accumulation of images in static scenes to reduce noise

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's *Depth Image* page. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI:

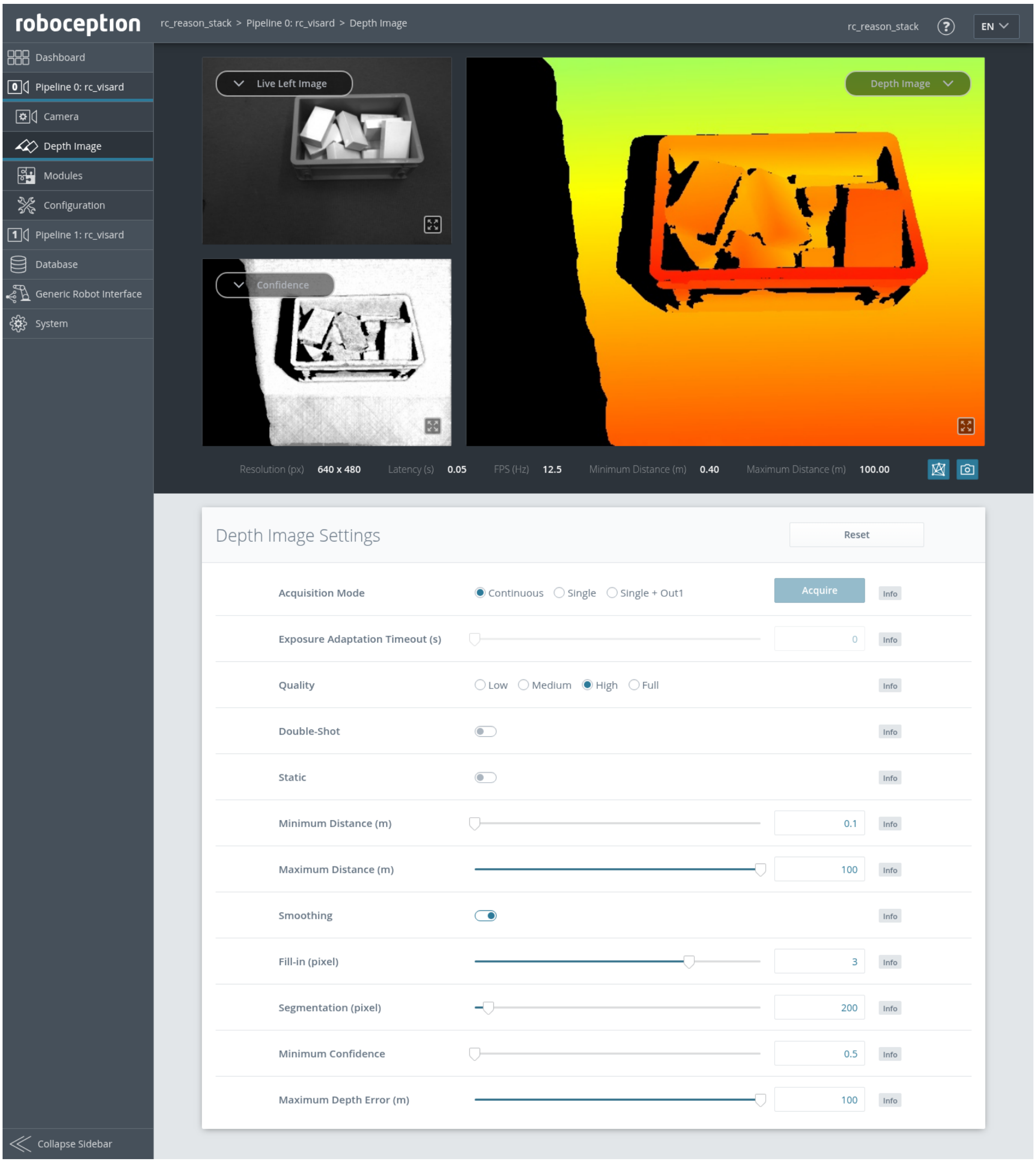

Fig. 6.3: The Web GUI's *Depth Image* page

`acquisition_mode` (*Acquisition Mode*)

The acquisition mode can be set to `Continuous`, `SingleFrame` (*Single*) or `SingleFrameOut1` (*Single + Out1*). The first one is the default, which performs stereo matching continuously according to the user defined frame rate and the available computation resources. The two other modes perform stereo matching upon each click of the *Acquire* button. The *Single + Out1* mode additionally controls an external projector that is connected to GPIO Out1 (*IO and Projector Control*, Section 6.4.4). In this mode, `out1_mode` of the IOControl module is automatically set to `ExposureAlternateActive` upon each trigger call and reset to `Low` after receiving images for stereo matching.

Note: The `Single + Out1` mode can only change the `out1_mode` if the IOControl license is available on the *rc_reason_stack*.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`exposure_adapt_timeout` (*Exposure Adaptation Timeout*)

The exposure adaptation timeout gives the maximum time in seconds that the system will wait after triggering an image acquisition until auto exposure has found the optimal exposure time. This timeout is only used in `SingleFrame` (*Single*) or `SingleFrameOut1` (*Single + Out1*) acquisition mode with auto exposure active. This value should be increased in applications with changing lighting conditions, when images are under- oder overexposed and the resulting disparity images are too sparse. In these cases multiple images are acquired until the auto-exposure mode has adjusted or the timeout is reached, and only then the actual image acquisition is triggered.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`quality` (*Quality*)

Disparity images can be computed in different resolutions: `Full` (full image resolution), `High` (half of the full image resolution), `Medium` (quarter of the full image resolution) and `Low` (sixth of the full image resolution). Full resolution matching (`Full`) is only possible with a valid StereoPlus license. The lower the resolution, the higher the frame rate of the disparity image. Please note that the frame rate of the disparity, confidence, and error images will always be less than or equal to the camera frame rate. In case the projector is in `ExposureAlternateActive` mode, the frame rate of the images can be at most half of the camera frame rate.

If full resolution is selected, the depth range is internally limited due to limited onboard memory resources. It is recommended to adjust `mindepth` and `maxdepth` to the depth range that is required by the application.

Table 6.10: Depth image resolutions (pixel) depending on the chosen quality

Connected Camera	Full Quality	High Quality	Medium Quality	Low Quality
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

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

double_shot (*Double-Shot*)

Enabling this option will lead to denser disparity images, but will increase processing time.

For scenes recorded with a projector in `Single + Out1` acquisition mode, or in continuous acquisition mode with the projector in `ExposureAlternateActive` mode, holes caused by reflections of the projector are filled with depth information computed from the images without projector pattern. In this case, the `double_shot` parameter must only be enabled if the scene does not change during the acquisition of the images.

For all other scenes, holes are filled with depth information computed from a downscaled version of the same image.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

static_scene (*Static*)

This option averages 8 consecutive camera images before matching. This reduces noise, which improves the stereo matching result. However, the latency increases significantly. The timestamp of the first image is taken as timestamp of the disparity image. This option only affects matching in full or high quality. It must only be enabled if the scene does not change during the acquisition of the 8 images.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

mindepth (*Minimum Distance*)

The minimum distance is the smallest distance from the camera at which measurements should be possible. Larger values implicitly reduce the disparity range, which also reduces the computation time. The minimum distance is given in meters.

Depending on the capabilities of the sensor, the actual minimum distance can be higher than the user setting. The actual minimum distance will be reported in the status values.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

maxdepth (*Maximum Distance*)

The maximum distance is the largest distance from the camera at which measurements should be possible. Pixels with larger distance values are set to invalid in the disparity image. Setting this value to its maximum permits values up to infinity. The maximum distance is given in meters.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

smooth (*Smoothing*)

This option activates advanced smoothing of disparity values. It is only available with a valid StereoPlus license.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`fill` (*Fill-in*)

This option is used to fill holes in the disparity image by interpolation. The fill-in value is the maximum allowed disparity step on the border of the hole. Larger fill-in values can decrease the number of holes, but the interpolated values can have larger errors. At most 5% of pixels are interpolated. Interpolation of small holes is preferred over interpolation of larger holes. The confidence for the interpolated pixels is set to a low value of 0.5. A fill-in value of 0 switches hole filling off.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`seg` (*Segmentation*)

The segmentation parameter is used to set the minimum number of pixels that a connected disparity region in the disparity image must fill. Isolated regions that are smaller are set to invalid in the disparity image. The value is related to the high quality disparity image with half resolution and does not have to be scaled when a different quality is chosen. Segmentation is useful for removing erroneous disparities. However, larger values may also remove real objects.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`minconf` (*Minimum Confidence*)

The minimum confidence can be set to filter potentially false disparity measurements. All pixels with less confidence than the chosen value are set to invalid in the disparity image.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

maxdeptherr (*Maximum Depth Error*)

The maximum depth error is used to filter measurements that are too inaccurate. All pixels with a larger depth error than the chosen value are set to invalid in the disparity image. The maximum depth error is given in meters. The depth error generally grows quadratically with an object's distance from the camera (see *Confidence and error images*, Section 4.2.3).

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

6.2.2.2 Status values

This module reports the following status values:

Table 6.11: The `rc_stereomatching` module's status values

Name	Description
`fps`	Actual frame rate of the disparity, error, and confidence images. This value is shown in the Web GUI below the image preview as *FPS (Hz)*.
`latency`	Time in seconds between image acquisition and publishing of disparity image. This value is shown in the Web GUI below the image preview as *Latency (s)*.
`width`	Current width of the disparity, error, and confidence images in pixels. This value is shown in the Web GUI below the image preview as the first number of *Resolution (px)*.
`height`	Current height of the disparity, error, and confidence images in pixels. This value is shown in the Web GUI below the image preview as the second number of *Resolution (px)*.
`mindepth`	Actual minimum working distance in meters. This value is shown in the Web GUI below the image preview as *Minimum Distance (m)*.
`maxdepth`	Actual maximum working distance in meters. This value is shown in the Web GUI below the image preview as *Maximum Distance (m)*.
`time_matching`	Time in seconds for performing stereo matching using *SGM* on the GPU
`time_postprocessing`	Time in seconds for postprocessing the matching result on the CPU
`reduced_depth_range`	Indicates whether the depth range is reduced due to computation resources

6.2.2.3 Services

The stereo matching module offers the following services.

acquisition_trigger

Signals the module to perform stereo matching of the next available images, if the parameter `acquisition_mode` is set to `SingleFrame` or `SingleFrameOut1`.

Details

An error is returned if the `acquisition_mode` is set to `Continuous`.

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

Possible return codes are shown below.

Table 6.12: Possible return codes of the acquisition_trigger service call.

Code	Description
0	Success
-8	Triggering is only possible in SingleFrame acquisition mode
101	Trigger is ignored, because there is a trigger call pending
102	Trigger is ignored, because there are no subscribers

The definition for the response with corresponding datatypes is:

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

reset_defaults

Restores and applies the default values for this module's parameters ("factory reset").

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.2.3 Zivid module

Note: The firmware version of the connected *zivid* camera has to match the version required by the *rc_reason_stack*, otherwise the *zivid* cannot be used. To update the *zivid* to the required firmware version, open the *Web GUI* (Section 7.1), navigate to *System* → *Camera Pipelines* and select the *zivid* pipeline. Then click on *Update zivid firmware* and wait for the update process to finish.

The zivid module is a base module which is available on every *rc_reason_stack* and provides disparity, confidence and error images of a connected *zivid* structured light camera. It only runs in camera pipelines of type `zivid`.

6.2.3.1 User-defined presets

The *zivid* camera comes with multiple pre-configured settings for capturing images, so-called presets.

The 3D presets that come with the *zivid* camera include both 2D and 3D settings, enabling simultaneous capture of color images and depth data. However, the 2D image settings are ignored and instead the 2D image is captured with the preset chosen in `rc_camera` (see *User-defined presets*, Section 6.1.6.1). The 3D presets are categorized based on application needs, such as *Consumer Goods*, *Manufacturing* etc.

Users can also create own 3D presets using the *Zivid Studio* software (https://www.zivid.com/zivid-studio-software) and save them as .yml files. These preset files can be uploaded to the *rc_reason_stack* on the *Depth Image* page of the Web GUI or using the REST-API as described in *Presets API*. User-defined presets can then be selected for depth image acquisition in the same way as the pre-defined presets via the `preset_name` run-time parameter. If the 2D preset included in the user-defined 3D preset should be used, then this preset also has to be uploaded as 2D preset and selected as the camera preset name.

6.2.3.2 Parameters

The zivid module is called `rc_zivid` in the REST-API and it is represented by the *Depth Image* page in the desired pipeline in the *Web GUI* (Section 7.1), when a *zivid* camera is connected and running in the corresponding pipeline. The user can change the zivid parameters there, or use the REST-API (*REST-API interface*, Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.13: The rc_zivid module's run-time parameters

Name	Type	Min	Max	Default	Description
acquisition_mode	string	-	-	SingleFrame	Acquisition mode: [Continuous, SingleFrame]
maxdepth	float64	0.3	100.0	100.0	Maximum depth in meters
mindepth	float64	0.3	100.0	0.3	Minimum depth in meters
preset_name	string	-	-	-	Name of preset configuration

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's *Depth Image* page. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI.

acquisition_mode (*Acquisition Mode*)

This parameter determines the acquisition mode for 3D data. The acquisition mode can be set to `Continuous` or `SingleFrame`. The latter is the default, which acquires a depth image upon each click of the *Acquire* button or when calling the `rc_zivid/acquisition_trigger` service (see *Services of the rc_zivid module*, Section 6.2.3.4). In `Continuous` mode, depth images are acquired continuously, when the 2D image acquisition mode is also set to `Continuous`. Otherwise, depth images are only acquired when a 2D images acquisition is triggered.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

preset_name (*Preset Name*)

This parameter allows to select a preset for 3D image acquisition. The preset can be any of the *zivid*'s pre-configured presets which depend on the *zivid* model and are read from the connected device, or a user-defined preset that has been uploaded to the *rc_reason_stack* (see *User-defined presets*, Section 6.2.3.1).

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

6.2.3.3 Status values

The rc_zivid module reports the following status values:

Table 6.14: The rc_zivid module's status values

Name	Description
fps	Actual frame rate of the disparity, error, and confidence images. This value is shown in the Web GUI below the image preview as *FPS (Hz)*.
height	Current height of the disparity, error, and confidence images in pixels. This value is shown in the Web GUI below the image preview as the second number of *Resolution (px)*.
last_capture_ok	1 if the last image capture was successful
last_timestamp_grabbed	Timestamp of the last depth data acquired
latency	Time in seconds between image acquisition and publishing of disparity image. This value is shown in the Web GUI below the image preview as *Latency (s)*.
width	Current width of the disparity, error, and confidence images in pixels. This value is shown in the Web GUI below the image preview as the first number of *Resolution (px)*.

6.2.3.4 Services of the rc_zivid module

The rc_zivid module offers the following services.

acquisition_trigger

Signals the module to acquire a depth image, if the parameter acquisition_mode is set to SingleFrame.

Details

An error is returned if the acquisition_mode is set to Continuous.

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

Possible return codes are shown below.

Table 6.15: Possible return codes of the acquisition_trigger service call.

Code	Description
0	Success
-8	Triggering is only possible in SingleFrame acquisition mode
101	Trigger is ignored, because there is a trigger call pending
102	Trigger is ignored, because there are no subscribers

The definition for the response with corresponding datatypes is:

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

reset_defaults

Restores and applies the default values for this module's parameters ("factory reset").

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.2.3.5 Presets API

The 3D presets can be set, retrieved and deleted via the following REST-API endpoints.

GET /presets/rc_zivid/3d_presets

Get zivid 3D presets.

Template request

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

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – successful operation

GET /presets/rc_zivid/3d_presets/{id}

Get zivid 3D preset yml file.

Template request

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

Parameters

- **id** (*string*) – ID/filename without extension *(required)*

Response Headers

- Content-Type – application/octet-stream

Status Codes

- 200 OK – successful operation
- 404 Not Found – yml file not found

PUT /presets/rc_zivid/3d_presets/{id}

Create or update a zivid 3D preset yml file.

Template request

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

Parameters

- **id** (*string*) – ID/filename without extension *(required)*

Form Parameters

- **file** – preset yml file *(required)*

Request Headers

- Accept – multipart/form-data application/json

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – successful operation
- 400 Bad Request – yml is not valid or max number of elements reached
- 413 Request Entity Too Large – File too large

DELETE /presets/rc_zivid/3d_presets/{id}

Remove a zivid 3D preset yml file.

Template request

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

Parameters

- **id** (*string*) – ID/filename without extension *(required)*

Request Headers

- Accept – application/json

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – successful operation
- 404 Not Found – element not found

6.2.4 Orbbec module

Note: The firmware version of the connected *Orbbec* camera has to be at least 1.6.00, otherwise the *Orbbec* cannot be used.

The orbbec module is a base module which is available on every *rc_reason_stack* and provides disparity, confidence and error images of a connected *Orbbec* stereo camera. It only runs in camera pipelines of type `orbbec`.

6.2.4.1 Parameters

The orbbec module is called `rc_orbbec` in the REST-API and it is represented by the *Depth Image* page in the desired pipeline in the *Web GUI* (Section 7.1), when an *Orbbec* camera is connected and running in the corresponding pipeline. The user can change the orbbec parameters there, or use the REST-API (*REST-API interface*, Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.16: The `rc_orbbec` module's run-time parameters

Name	Type	Min	Max	Default	Description
`fill`	int32	0	4	3	Disparity tolerance for hole filling in pixels
`maxdepth`	float64	0.1	100.0	100.0	Maximum depth in meters
`mindepth`	float64	0.1	100.0	0.1	Minimum depth in meters
`seg`	int32	0	4000	200	Minimum size of valid disparity segments in pixels
`smooth`	bool	false	true	true	Smoothing of disparity image (requires 'stereo_plus' license)

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's *Depth Image* page. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI.

`mindepth` (*Minimum Distance*)

The minimum distance is the smallest distance from the camera at which measurements should be possible. The minimum distance is given in meters.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

maxdepth (*Maximum Distance*)

The maximum distance is the largest distance from the camera at which measurements should be possible. Pixels with larger distance values are set to invalid in the disparity image. Setting this value to its maximum permits values up to infinity. The maximum distance is given in meters.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

smooth (*Smoothing*)

This option activates advanced smoothing of disparity values.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

fill (*Fill-in*)

This option is used to fill holes in the disparity image by interpolation. The fill-in value is the maximum allowed disparity step on the border of the hole. Larger fill-in values can decrease the number of holes, but the interpolated values can have larger errors. At most 5% of pixels are interpolated. Interpolation of small holes is preferred over interpolation of larger holes. The confidence for the interpolated pixels is set to a low value of 0.5. A fill-in value of 0 switches hole filling off.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

seg (*Segmentation*)

The segmentation parameter is used to set the minimum number of pixels that a connected disparity region in the disparity image must fill. Isolated regions that are smaller are set to invalid in the disparity image. The value is related to the high quality disparity image with half resolution and does not have to be scaled when a different quality is chosen. Segmentation is useful for removing erroneous disparities. However, larger values may also remove real objects.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

6.2.4.2 Status values

The `rc_orbbec` module reports the following status values:

Table 6.17: The `rc_orbbec` module's status values

Name	Description
`fps`	Actual frame rate of the disparity, error, and confidence images. This value is shown in the Web GUI below the image preview as *FPS (Hz)*.
`height`	Current height of the disparity, error, and confidence images in pixels. This value is shown in the Web GUI below the image preview as the second number of *Resolution (px)*.
`last_capture_ok`	1 if the last image capture was successful
`last_timestamp_grabbed`	Timestamp of the last depth data acquired
`latency`	Time in seconds between image acquisition and publishing of disparity image. This value is shown in the Web GUI below the image preview as *Latency (s)*.
`mindepth`	Actual minimum working distance in meters. This value is shown in the Web GUI below the image preview as *Minimum Distance (m)*.
`maxdepth`	Actual maximum working distance in meters. This value is shown in the Web GUI below the image preview as *Maximum Distance (m)*.
`width`	Current width of the disparity, error, and confidence images in pixels. This value is shown in the Web GUI below the image preview as the first number of *Resolution (px)*.

6.2.4.3 Services of the rc_orbbec module

The `rc_orbbec` module offers the following services.

reset_defaults

Restores and applies the default values for this module's parameters ("factory reset").

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.3 Detection & Measure modules

The *rc_reason_stack* offers software modules for different detection and measuring applications:

- ***Measure*** **(`rc_measure`, Section 6.3.1)** provides measure functionalities, such as depth measurements.
- ***LoadCarrier*** **(`rc_load_carrier`, Section 6.3.2)** allows detecting load carriers and their filling levels.
- ***TagDetect*** **(`rc_april_tag_detect` and `rc_qr_code_detect`, Section 6.3.3)** allows the detection of AprilTags and QR codes, as well as the estimation of their poses.
- ***ItemPick*** **(`rc_itempick`, Section 6.3.4)** provides an out-of-the-box perception solution for robotic pick-and-place applications of objects of an object category or unknown objects.
- ***BoxPick*** **(`rc_boxpick`, Section 6.3.5)** provides an out-of-the-box perception solution for robotic pick-and-place applications of boxes or textured boxes.
- ***SilhouetteMatch*** **(`rc_silhouettematch`, Section 6.3.6)** provides an object detection solution for objects placed on a plane or stacked planar objects.
- ***CADMatch*** **(`rc_cadmatch`, Section 6.3.7)** provides an object detection solution for 3D objects.

These modules are pipeline specific, which means that they run inside each camera pipeline. Changes to their settings or parameters only affect the corresponding pipeline and have no influence on the other camera pipelines running on the *rc_reason_stack*.

These modules are optional and can be activated by purchasing a separate *license* (Section 8.2).

6.3.1 Measure

6.3.1.1 Introduction

The Measure module allows measuring of depth values in a specific region of interest.

The Measure module is a base module which is available on every *rc_reason_stack*.

6.3.1.2 Measuring Depth

The Measure module provides functionality to measure depth values in the current scene in a 2D region of interest. Optionally, the region of interest can be subdivided into up to 100 cells, for which separate depth measurements are returned in addition to the overall depth measurements of the whole region.

A depth measurement consist of the average depth `mean_z`, the minimum depth `min_z` and the maximum depth `max_z`, each containing 3D coordinates. The coordinates of the `min_z` and `max_z` measurements correspond to the point in the cell or overall region with the minimum and maximum distance from the camera, respectively. The `x` and `y` coordinates of the `mean_z` measurements define a point in the center of the cell or the overall region and the `z` coordinate is determined by the average of all depth value measurements (distances from the camera) in this region. Additionally, a `coverage` value is returned for each cell and the overall region, which is a number between 0 and 1 that reflects the fraction of valid depth values inside the respective region. A coverage value of 0 means that the cell is invalid and no depth value could be computed.

When the external `pose_frame` is used for the depth measurements, all 3D coordinates are computed as described above, but then transformed to the external frame. That means, the depth is always measured along the line of sight of the camera, independently of the chosen pose frame.

6.3.1.3 Interaction with other modules

Internally, the Measure module depends on, and interacts with other on-board modules as listed below.

Note: All changes and configuration updates to these modules will affect the performance of the Measure module.

Depth Images

The Measure module internally makes use of the following data:

- Disparity images from the *Stereo matching module* (`rc_stereomatching`, Section 6.2.2), in case a stereo camera is used
- Disparity, error, and confidence images from the *Orbbec module* (`rc_orbbec`, Section 6.2.4), in case an *Orbbec* camera is used
- Disparity images from the *Zivid module* (`rc_zivid`, Section 6.2.3), in case a *zivid* camera is used

Hand-eye calibration

In case the camera has been calibrated to a robot, the Measure module can automatically provide points in the robot coordinate frame. For the Measure node's *Services* (Section 6.3.1.6), the frame of the output points can be controlled with the `pose_frame` argument.

Two different `pose_frame` values can be chosen:

1. **Camera frame** (`camera`). All points provided by the module are in the camera frame, and no prior knowledge about the pose of the camera in the environment is required. It is the user's responsibility to update the configured points if the camera frame moves (e.g. with a robot-mounted camera).
2. **External frame** (`external`). All points provided by the module are in the external frame, configured by the user during the hand-eye calibration process. The module relies on the on-board *Hand-eye calibration module* (Section 6.4.1) to retrieve the sensor mounting (static or robot mounted) and the hand-eye transformation. If the mounting is static, no further information is needed. If the

sensor is robot-mounted, the `robot_pose` is required to transform poses to and from the `external` frame.

Note: If no hand-eye calibration is available, all `pose_frame` values should be set to `camera`.

All `pose_frame` values that are not `camera` or `external` are rejected.

6.3.1.4 Parameters

The Measure module is called `rc_measure` in the REST-API and is represented in the *Web GUI* (Section 7.1) in the desired pipeline under *Modules* → *Measure*.

Parameter overview

This module has no run-time parameters.

6.3.1.5 Status values

The Measure module reports the following status values:

Table 6.18: The `rc_measure` module's status values

Name	Description
`data_acquisition_time`	Time in seconds required to acquire depth image
`last_timestamp_processed`	The timestamp of the last processed depth image
`processing_time`	Processing time of the last measurement in seconds

6.3.1.6 Services

The user can explore and call the Measure module's services, e.g. for development and testing, using the *REST-API interface* (Section 7.2) or the *rc_reason_stack* *Web GUI* (Section 7.1) on the *Measure* page under *Modules*.

The Measure module offers the following services.

`measure_depth`

Computes the mean, minimum and maximum depth in a given region of interest, which can optionally be subdivided into cells.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

Required arguments:

`pose_frame`: see *Hand-eye calibration* (Section 6.3.1.3).

Optional arguments:

region_of_interest_2d_id is the ID of the 2D region of interest (see *RoiDB*, Section 6.5.2) that will be used for the depth measurements.

region_of_interest_2d is an alternative on-the-fly definition of the region of interest for the depth measurements. This region of interest will be ignored if a region_of_interest_2d_id is given. The region of interest is always defined on the camera image with full resolution, where offset_x and offset_y are the pixel coordinates of the upper left corner of the rectangular region of interest, and width and height are the width and height of it in pixels. Default is a region of interest covering the whole image.

cell_count is the number of cells in x and y direction into which the region of interest is divided. If not given, a cell count of 0, 0 is assumed and only the overall values will be computed. The total cell count is computed as product of the x and y values must not exceed 100.

data_acquisition_mode: if set to CAPTURE_NEW (default), a new image dataset will be used for the measurement. If set to USE_LAST, the previous dataset will be used for the measurement.

Potentially required arguments:

robot_pose: see *Hand-eye calibration* (Section 6.3.1.3).

The definition for the request arguments with corresponding datatypes is:

```
{
  "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 contains the depth measurements of all requested cells. The cells are always ordered from left to right and top to bottom in image coordinates.

overall contains the depth measurements of the full region of interest.

coverage contains the valid pixel ratio as described in *Measuring Depth* (Section 6.3.1.2).

mean_z, min_z and max_z contains the measurement coordinates as described in *Measuring Depth* (Section 6.3.1.2).

region_of_interest_2d returns the definition of the requested region of interest for the depth measurement.

pose_frame contains the pose frame of the depth measurement coordinates.

The definition for the response with corresponding datatypes is:

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

(continues on next page)

(continued from previous page)

```
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

6.3.1.7 Return codes

Each service response contains a `return_code`, which consists of a `value` plus an optional `message`. A successful service returns with a `return_code` value of `0`. Negative `return_code` values indicate that the service failed. Positive `return_code` values indicate that the service succeeded with additional information. The smaller value is selected in case a service has multiple `return_code` values, but all messages are appended in the `return_code` message.

The following table contains a list of common codes:

Table 6.19: Return codes of the Measure module's services

Code	Description
0	Success
-1	An invalid argument was provided

6.3.2 LoadCarrier

6.3.2.1 Introduction

The LoadCarrier module allows the detection of load carriers, which is usually the first step when objects or grasp points inside a bin should be found. The models of the load carriers to be detected have to be defined in the *LoadCarrierDB* (Section 6.5.1) module.

The LoadCarrier module is an optional on-board module of the *rc_reason_stack* and is licensed with any of the modules *ItemPick* (Section 6.3.4) and *BoxPick* (Section 6.3.5) or *SilhouetteMatch* (Section 6.3.6). Otherwise it requires a separate LoadCarrier *license* (Section 8.2) to be purchased.

Note: This module is pipeline specific. Changes to its settings or parameters only affect the respective camera pipeline and have no influence on other pipelines running on the *rc_reason_stack*.

6.3.2.2 Detection of load carriers

The load carrier detection algorithm detects load carriers that match a specific load carrier model, which must be defined in the *LoadCarrierDB* (Section 6.5.1) module. The load carrier model is referenced by its ID, which is passed to the load carrier detection. The detection of a load carrier is based on the detection of its rectangular rim. For this, it uses lines detected in the left camera image and the depth values of the load carrier rim. Thus, the rim should form a contrast to the background and the disparity image must be dense on the rim.

If multiple load carriers of the specified load carrier ID are visible in the scene, all of them will be detected and returned by the load carrier detection.

By default, when `assume_gravity_aligned` is true and gravity measurements are available, the algorithm searches for load carriers whose rim planes are perpendicular to the measured gravity vector. To detect tilted load carriers, `assume_gravity_aligned` must be set to false or the load carrier's approximate orientation must be specified as `pose` and the `pose_type` should be set to `ORIENTATION_PRIOR`.

Load carriers can be detected at a distance of up to 3 meters from the camera.

When a 3D region of interest (see *RoiDB*, Section 6.5.2) is used to limit the volume in which load carriers should be detected, only the load carriers' rims must be fully included in the region of interest.

The detection algorithm returns the poses of the load carriers' origins (see *Load carrier definition*, Section 6.5.1.2) in the desired pose frame.

The detection functionality also determines if the detected load carriers are `overfilled`, which means, that objects protrude from the plane defined by the load carrier's outer part of the rim.

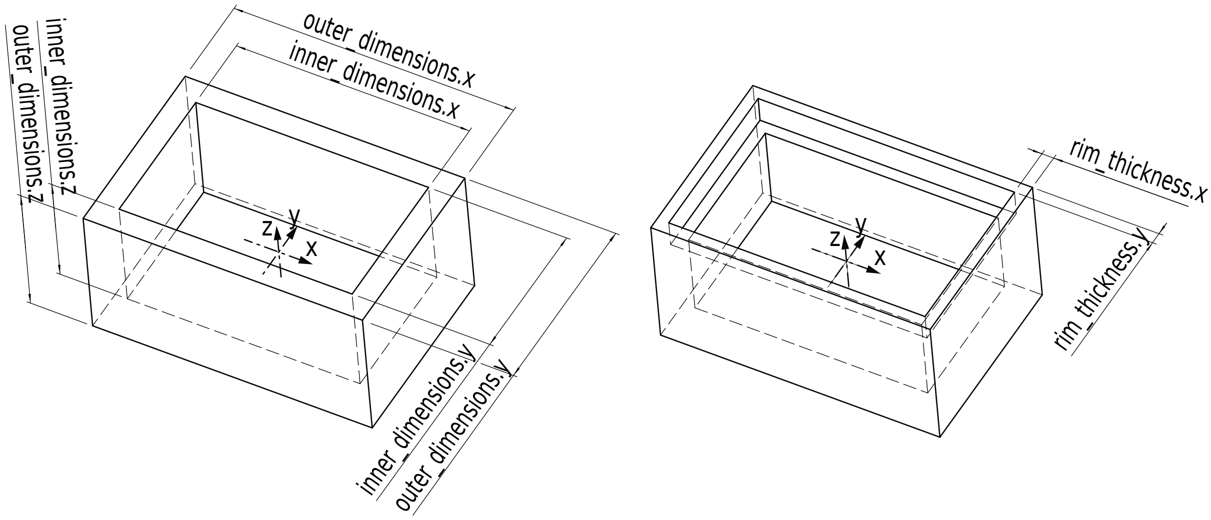

Fig. 6.4: Load carrier models and reference frames

6.3.2.3 Detection of filling level

The LoadCarrier module offers the `detect_filling_level` service to compute the filling level of all detected load carriers.

The load carriers are subdivided into a configurable number of cells in a 2D grid. The maximum number of cells is 10x10. For each cell, the following values are reported:

- `level_in_percent`: minimum, maximum and mean cell filling level in percent from the load carrier floor. These values can be larger than 100% if the cell is overfilled.
- `level_free_in_meters`: minimum, maximum and mean cell free level in meters from the load carrier rim. These values can be negative if the cell is overfilled.
- `cell_size`: dimensions of the 2D cell in meters.
- `cell_position`: position of the cell center in meters (either in `camera` or `external` frame, see *Hand-eye calibration*, Section 6.3.2.4). The z-coordinate is on the level of the load carrier rim.
- `coverage`: represents the proportion of valid pixels in this cell. It varies between 0 and 1 with steps of 0.1. A low coverage indicates that the cell contains several missing data (i.e. only a few points were actually measured in this cell).

These values are also calculated for the whole load carrier itself. If no cell subdivision is specified, only the overall filling level is computed.

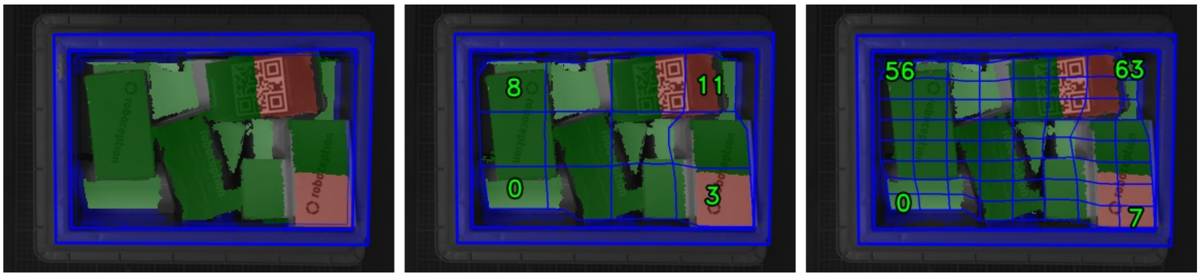

Fig. 6.5: Visualizations of the load carrier filling level: overall filling level without grid (left), 4x3 grid (center), 8x8 grid (right). The load carrier content is shown in a green gradient from white (on the load carrier floor) to dark green. The overfilled regions are visualized in red. Numbers indicate cell IDs.

6.3.2.4 Interaction with other modules

Internally, the LoadCarrier module depends on, and interacts with other on-board modules as listed below.

Note: All changes and configuration updates to these modules will affect the performance of the LoadCarrier module.

Camera and depth data

The LoadCarrier module makes internally use of the following data:

- Rectified images from the *Camera module* (`rc_camera`, Section 6.1);
- Disparity, error, and confidence images from the *Stereo matching module* (`rc_stereomatching`, Section 6.2.2), in case a stereo camera is used
- Disparity, error, and confidence images from the *Orbbec module* (`rc_orbbec`, Section 6.2.4), in case an *Orbbec* camera is used
- Disparity, error, and confidence images from the *Zivid module* (`rc_zivid`, Section 6.2.3), in case a *zivid* camera is used

All processed images are guaranteed to be captured after the module trigger time.

IO and Projector Control

In case the *rc_reason_stack* is used in conjunction with an external random dot projector and the *IO and Projector Control* module (`rc_iocontrol`, Section 6.4.4), it is recommended to connect the projector to GPIO Out 1 and set the stereo-camera module's acquisition mode to `SingleFrameOut1` (see *Stereo matching parameters*, Section 6.2.2.1), so that on each image acquisition trigger an image with and without projector pattern is acquired.

Alternatively, the output mode for the GPIO output in use should be set to `ExposureAlternateActive` (see *Description of run-time parameters*, Section 6.4.4.1).

In either case, the *Auto Exposure Mode* `exp_auto_mode` should be set to `AdaptiveOut1` to optimize the exposure of both images.

No additional changes are required to use the LoadCarrier module in combination with a random dot projector.

Hand-eye calibration

In case the camera has been calibrated to a robot, the LoadCarrier module can automatically provide poses in the robot coordinate frame. For the LoadCarrier node's *Services* (Section 6.3.2.7), the frame of the output poses can be controlled with the `pose_frame` argument.

Two different `pose_frame` values can be chosen:

1. **Camera frame** (`camera`). All poses provided by the module are in the camera frame, and no prior knowledge about the pose of the camera in the environment is required. This means that the configured load carriers move with the camera. It is the user's responsibility to update the configured poses if the camera frame moves (e.g. with a robot-mounted camera).
2. **External frame** (`external`). All poses provided by the module are in the external frame, configured by the user during the hand-eye calibration process. The module relies on the on-board *Hand-eye calibration module* (Section 6.4.1) to retrieve the sensor mounting (static or robot mounted) and the hand-eye transformation. If the mounting is static, no further information is needed. If the sensor is robot-mounted, the `robot_pose` is required to transform poses to and from the `external` frame.

Note: If no hand-eye calibration is available, all `pose_frame` values should be set to `camera`.

All `pose_frame` values that are not `camera` or `external` are rejected.

6.3.2.5 Parameters

The LoadCarrier module is called `rc_load_carrier` in the REST-API and is represented in the *Web GUI* (Section 7.1) in the desired pipeline under *Modules* → *LoadCarrier*. The user can explore and configure the LoadCarrier module's run-time parameters, e.g. for development and testing, using the Web GUI or the *REST-API interface* (Section 7.2).

Parameter overview

Note: The default values in the parameter table below show the values of the *rc_visard*. The values can be different for other sensors.

This module offers the following run-time parameters:

Table 6.20: The `rc_load_carrier` module's run-time parameters

Name	Type	Min	Max	Default	Description
`assume_gravity_aligned`	bool	false	true	true	When true, only gravity-aligned load carriers are detected, if gravity measurement is available.
`crop_distance`	float64	0.0	0.05	0.005	Safety margin in meters by which the load carrier inner dimensions are reduced to define the region of interest for detection
`min_plausibility`	float64	0.0	0.99	0.8	Indicates how much of the plane surrounding the load carrier rim must be free to count as valid detection
`model_tolerance`	float64	0.003	0.025	0.008	Indicates how much the estimated load carrier dimensions are allowed to differ from the load carrier model dimensions in meters

Description of run-time parameters

Each run-time parameter is represented by a row on the *LoadCarrier Settings* section of the Web GUI's *LoadCarrier* page under *Modules*. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI. The parameters are prefixed with `load_carrier_` when they are used outside the `rc_load_carrier` module from another detection module using the *REST-API interface* (Section 7.2).

`assume_gravity_aligned` (*Assume Gravity Aligned*)

If this parameter is set to true, then only load carriers without tilt will be detected. This can speed up the detection. If this parameter is set to false, tilted load carriers will also be detected.

This parameter is ignored for load carriers with an orientation prior.

Note: Gravity alignment is only available for pipelines of type `rc_visard`. The gravity vector is estimated from linear acceleration readings from the on-board IMU.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

`model_tolerance` (*Model Tolerance*)

indicates how much the estimated load carrier dimensions are allowed to differ from the load carrier model dimensions in meters.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`crop_distance` (*Crop Distance*)

sets the safety margin in meters by which the load carrier's inner dimensions are reduced to define the region of interest for detection (ref. Fig. 6.37).

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

min_plausibility (***Minimum Plausibility***):

The minimum plausibility defines how much of the plane around the load carrier rim must at least be free to count as valid detection. Increase the minimal plausibility to reject false positive detections and decrease the value in case a clearly visible load carrier cannot be detected.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

6.3.2.6 Status values

The LoadCarrier module reports the following status values:

Table 6.21: The rc_load_carrier module's status values

Name	Description
data_acquisition_time	Time in seconds required to acquire image pair
last_timestamp_processed	The timestamp of the last processed image pair
load_carrier_detection_time	Processing time of the last detection in seconds

6.3.2.7 Services

The user can explore and call the LoadCarrier module's services, e.g. for development and testing, using the *REST-API interface* (Section 7.2) or the *rc_reason_stack* *Web GUI* (Section 7.1) on the *Load-Carrier* page under *Modules*.

The LoadCarrier module offers the following services.

detect_load_carriers

Triggers a load carrier detection as described in *Detection of load carriers* (Section 6.3.2.2).

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

Required arguments:

pose_frame: see *Hand-eye calibration* (Section 6.3.2.4).

load_carrier_ids: IDs of the load carriers which should be detected. Currently only one ID can be specified.

Potentially required arguments:

robot_pose: see *Hand-eye calibration* (Section 6.3.2.4).

Optional arguments:

region_of_interest_id: ID of the 3D region of interest where to search for the load carriers.

region_of_interest_2d_id: ID of the 2D region of interest where to search for the load carriers.

Note: Only one type of region of interest can be set.

The definition for the request arguments with corresponding datatypes is:

```
{
  "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: list of detected load carriers.

timestamp: timestamp of the image set the detection ran on.

return_code: holds possible warnings or error codes and messages.

The definition for the response with corresponding datatypes is:

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

(continues on next page)

(continued from previous page)

```
        "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"
    }
  }
}
```

detect_filling_level

Triggers a load carrier filling level detection as described in *Detection of filling level* (Section 6.3.2.3).

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

Required arguments:

> `pose_frame`: see *Hand-eye calibration* (Section 6.3.2.4).
>
> `load_carrier_ids`: IDs of the load carriers which should be detected. Currently only one ID can be specified.

Potentially required arguments:

> `robot_pose`: see *Hand-eye calibration* (Section 6.3.2.4).

Optional arguments:

> `filling_level_cell_count`: Number of cells in the filling level grid.
>
> `region_of_interest_id`: ID of the 3D region of interest where to search for the load carriers.
>
> `region_of_interest_2d_id`: ID of the 2D region of interest where to search for the load carriers.
>
> **Note:** Only one type of region of interest can be set.

The definition for the request arguments with corresponding datatypes is:

```
{
  "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`: list of detected load carriers and their filling levels.

`timestamp`: timestamp of the image set the detection ran on.

`return_code`: holds possible warnings or error codes and messages.

The definition for the response with corresponding datatypes is:

```
{
  "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"
          },
```

(continues on next page)

(continued from previous page)

```
          "level_in_percent": {
            "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

Restores and applies the default values for this module's parameters ("factory reset").

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

`set_load_carrier` (deprecated)

Persistently stores a load carrier on the *rc_reason_stack*.

API version 2

This service is not available in API version 2. Use *set_load_carrier* (Section 6.5.1.5) in `rc_load_carrier_db` instead.

API version 1 (deprecated)

This service can be called as follows.

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

The definitions of the request and response are the same as described in *set_load_carrier* (Section 6.5.1.5) in `rc_load_carrier_db`.

`get_load_carriers` (deprecated)

Returns the configured load carriers with the requested `load_carrier_ids`.

API version 2

This service is not available in API version 2. Use *get_load_carriers* (Section 6.5.1.5) in `rc_load_carrier_db` instead.

API version 1 (deprecated)

This service can be called as follows.

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

The definitions of the request and response are the same as described in *get_load_carriers* (Section 6.5.1.5) in `rc_load_carrier_db`.

`delete_load_carriers` (deprecated)

Deletes the configured load carriers with the requested `load_carrier_ids`.

API version 2

This service is not available in API version 2. Use *delete_load_carriers* (Section 6.5.1.5) in `rc_load_carrier_db` instead.

API version 1 (deprecated)

This service can be called as follows.

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

The definitions of the request and response are the same as described in *delete_load_carriers* (Section 6.5.1.5) in `rc_load_carrier_db`.

`set_region_of_interest` (deprecated)

Persistently stores a 3D region of interest on the *rc_reason_stack*.

API version 2

This service is not available in API version 2. Use *set_region_of_interest* (Section 6.5.2.4) in `rc_roi_db` instead.

API version 1 (deprecated)

This service can be called as follows.

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

The definitions of the request and response are the same as described in *set_region_of_interest* (Section 6.5.2.4) in `rc_roi_db`.

`get_regions_of_interest` (deprecated)

Returns the configured 3D regions of interest with the requested `region_of_interest_ids`.

API version 2

This service is not available in API version 2. Use *get_regions_of_interest* (Section 6.5.2.4) in `rc_roi_db` instead.

API version 1 (deprecated)

This service can be called as follows.

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

The definitions of the request and response are the same as described in *get_regions_of_interest* (Section 6.5.2.4) in `rc_roi_db`.

`delete_regions_of_interest` (deprecated)

Deletes the configured 3D regions of interest with the requested `region_of_interest_ids`.

API version 2

This service is not available in API version 2. Use *delete_regions_of_interest* (Section 6.5.2.4) in `rc_roi_db` instead.

API version 1 (deprecated)

This service can be called as follows.

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

The definitions of the request and response are the same as described in *delete_regions_of_interest* (Section 6.5.2.4) in `rc_roi_db`.

`set_region_of_interest_2d` (deprecated)

Persistently stores a 2D region of interest on the *rc_reason_stack*.

API version 2

This service is not available in API version 2. Use *set_region_of_interest_2d* (Section 6.5.2.4) in `rc_roi_db` instead.

API version 1 (deprecated)

This service can be called as follows.

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

The definitions of the request and response are the same as described in *set_region_of_interest_2d* (Section 6.5.2.4) in `rc_roi_db`.

`get_regions_of_interest_2d` (deprecated)

Returns the configured 2D regions of interest with the requested `region_of_interest_2d_ids`.

API version 2

This service is not available in API version 2. Use *get_regions_of_interest_2d* (Section 6.5.2.4) in `rc_roi_db` instead.

API version 1 (deprecated)

This service can be called as follows.

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

The definitions of the request and response are the same as described in *get_regions_of_interest_2d* (Section 6.5.2.4) in `rc_roi_db`.

`delete_regions_of_interest_2d` (deprecated)

Deletes the configured 2D regions of interest with the requested `region_of_interest_2d_ids`.

API version 2

This service is not available in API version 2. Use *delete_regions_of_interest_2d* (Section 6.5.2.4) in `rc_roi_db` instead.

API version 1 (deprecated)

This service can be called as follows.

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

The definitions of the request and response are the same as described in *delete_regions_of_interest_2d* (Section 6.5.2.4) in `rc_roi_db`.

6.3.2.8 Return codes

Each service response contains a `return_code`, which consists of a `value` plus an optional `message`. A successful service returns with a `return_code` value of `0`. Negative `return_code` values indicate that the service failed. Positive `return_code` values indicate that the service succeeded with additional information. The smaller value is selected in case a service has multiple `return_code` values, but all messages are appended in the `return_code` message.

The following table contains a list of common codes:

Table 6.22: Return codes of the LoadCarrier module's services

Code	Description
0	Success
-1	An invalid argument was provided
-4	Data acquisition took longer than allowed
-10	New element could not be added as the maximum storage capacity of load carriers has been exceeded
-11	Sensor not connected, not supported or not ready
-302	More than one load carrier provided to the `detect_load_carriers` or `detect_filling_level` services, but only one is supported
3	The detection timeout during load carrier detection has been reached. Consider reducing the model tolerance.
10	The maximum storage capacity of load carriers has been reached
11	An existent persistent model was overwritten by the call to `set_load_carrier`
100	The requested load carriers were not detected in the scene
102	The detected load carrier has no points inside
300	A valid `robot_pose` was provided as argument but it is not required

6.3.3 TagDetect

6.3.3.1 Introduction

The TagDetect modules are optional on-board modules of the *rc_reason_stack* and require separate *licenses* (Section 8.2) to be purchased. The licenses are included in every *rc_reason_stack* purchased after 01.07.2020.

The TagDetect modules run on board the *rc_reason_stack* and allow the detection of 2D matrix codes and tags. Currently, there are TagDetect modules for *QR codes* and *AprilTags*. The modules, furthermore, compute the position and orientation of each tag in the 3D camera coordinate system, making it simple to manipulate a tag with a robot or to localize the camera with respect to a tag.

Note: These modules are not available in camera pipelines of type `zivid` or `orbbec`.

Note: These modules are pipeline specific. Changes to their settings or parameters only affect the respective camera pipeline and have no influence on other pipelines running on the *rc_reason_stack*.

Tag detection is made up of three steps:

1. Tag reading on the 2D image pair (see *Tag reading*, Section 6.3.3.2).
2. Estimation of the pose of each tag (see *Pose estimation*, Section 6.3.3.3).
3. Re-identification of previously seen tags (see *Tag re-identification*, Section 6.3.3.4).

In the following, the two supported tag types are described, followed by a comparison.

QR code

Fig. 6.6: Sample QR code

QR codes are two-dimensional matrix codes that contain arbitrary user-defined data. There is wide support for decoding of QR codes on commodity hardware such as smartphones. Also, many online and offline tools are available for the generation of such codes.

The "pixels" of a QR code are called *modules*. Appearance and resolution of QR codes change with the amount of data they contain. While the special patterns in the three corners are always 7 modules wide, the number of modules between them increases the more data is stored. The lowest-resolution QR code is of size 21x21 modules and can contain up to 152 bits.

Even though many QR code generation tools support generation of specially designed QR codes (e.g., containing a logo, having round corners, or having dots as modules), a reliable detection of these tags by the *rc_reason_stack*'s TagDetect module is not guaranteed. The same holds for QR codes which contain characters that are not part of regular ASCII.

AprilTag

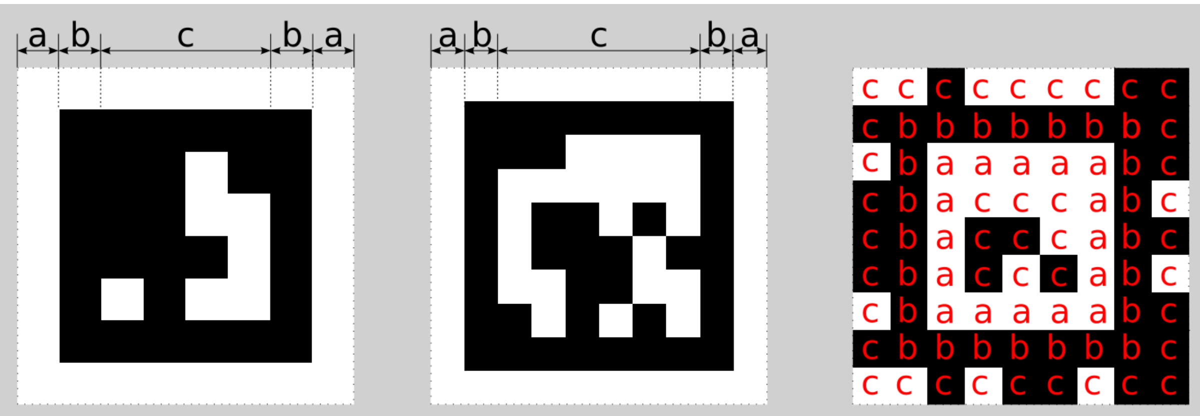

Fig. 6.7: A 16h5 tag (left), a 36h11 tag (center) and a 41h12 tag (right). AprilTags consist of a mandatory white (a) and black (b) border and a variable amount of data bits (c).

AprilTags are similar to QR codes. However, they are specifically designed for robust identification at large distances. As for QR codes, we will call the tag pixels *modules*. Fig. 6.7 shows how AprilTags are structured. They have a mandatory white and black border, each one module wide. The tag families 16h5, 25h9, 36h10 and 36h11 are surrounded by this border and carry a variable amount of data modules in the center. For tag family 41h12, the black and white border is shifted towards the inside and the data modules are in the center and also at the border of the tags. Other than QR codes, AprilTags do not contain any user-defined information but are identified by a predefined *family* and *ID*. The tags in Fig. 6.7 for example are of family 16h5, 36h11 and 41h12 have id 0, 11 and 0, respectively. All supported families are shown in Table 6.23.

Table 6.23: AprilTag families

Family	Number of tag IDs	Recommended
16h5	30	-
25h9	35	o
36h10	2320	o
36h11	587	+
41h12	2115	+

For each family, the number before the "h" states the number of data modules contained in the tag: While a 16h5 tag contains 16 (4x4) data modules ((c) in Fig. 6.7), a 36h11 tag contains 36 (6x6) modules and a 41h12 tag contains 41 (3x3 inner + 4x8 outer) modules. The number behind the "h" refers to the Hamming distance between two tags of the same family. The higher, the more robust is the detection, but the fewer individual tag IDs are available for the same number of data modules (see Table 6.23).

The advantage of fewer modules (as for 16h5 compared to 36h11) is the lower resolution of the tag. Hence, each tag module is larger and the tag therefore can be detected from a larger distance. This, however, comes at a price: Firstly, fewer data modules lead to fewer individual tag IDs. Secondly, and more importantly, detection robustness is significantly reduced due to a higher false positive rate; i.e, tags are mixed up or nonexistent tags are detected in random image texture or noise. The 41h12 family has its border shifted towards the inside, which gives it more data modules at a lower number of total modules compared to the 36h11 family.

For these reasons we recommend using the 41h12 and 36h11 families and highly discourage the use of the 16h5 family. The latter family should only be used if a large detection distance really is necessary for an application. However, the maximum detection distance increases only by approximately 25% when using a 16h5 tag instead of a 36h11 tag.

Pre-generated AprilTags can be downloaded from the website https://github.com/AprilRobotics/apriltag-imgs. There, each family consists of multiple PNGs containing single tags. Each pixel in the PNGs corresponds to one AprilTag module. When printing the tags of the families 36h11, 36h10, 25h9 and 16h5 special care must be taken to also include the white border around the tag that is contained in the PNG (see (a) in Fig. 6.7). Moreover, all tags should be scaled to the desired printing size without any interpolation, so that the sharp edges are preserved.

Comparison

Both QR codes and AprilTags have their up and down sides. While QR codes allow arbitrary user-defined data to be stored, AprilTags have a pre-defined and limited set of tags. On the other hand, AprilTags have a lower resolution and can therefore be detected at larger distances. Moreover, the continuous white to black border in AprilTags allow for more precise pose estimation.

Note: If user-defined data is not required, AprilTags should be preferred over QR codes.

6.3.3.2 Tag reading

The first step in the tag detection pipeline is reading the tags on the 2D image pair. This step takes most of the processing time and its precision is crucial for the precision of the resulting tag pose. To control the speed of this step, the `quality` parameter can be set by the user. It results in a downscaling of the image pair before reading the tags. `High` yields the largest maximum detection distance and highest precision, but also the highest processing time. `Low` results in the smallest maximum detection distance and lowest precision, but processing requires less than half of the time. `Medium` lies in between.

Please note that this quality parameter has no relation to the quality parameter of *Stereo matching module* (Section 6.2.2).

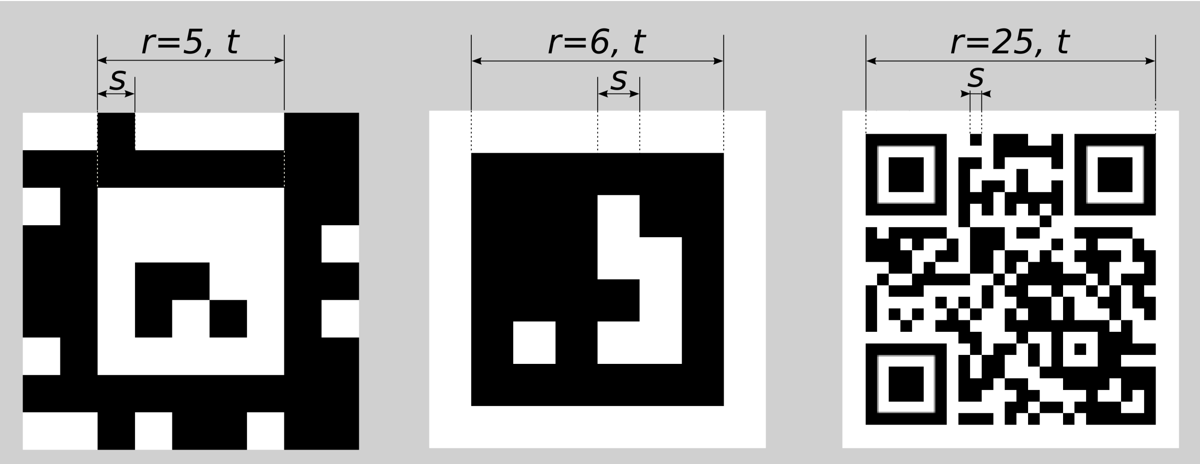

Fig. 6.8: Visualization of module size s, size of a tag in modules r, and size of a tag in meters t for AprilTags (left and center) and QR codes (right)

The maximum detection distance z at quality `High` can be approximated by using the following formulae,

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

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

where f is the *focal length* (Section 6.1.1) in pixels and s is the size of a module in meters. s can easily be calculated by the latter formula, where t is the size of the tag in meters and r is the width of the code in modules (for AprilTags without the white border). Fig. 6.8 visualizes these variables. p denotes the number of image pixels per module required for detection. It is different for QR codes and AprilTags. Moreover, it varies with the tag's angle to the camera and illumination. Approximate values for robust detection are:

- AprilTag: $p = 5$ pixels/module
- QR code: $p = 6$ pixels/module

The following tables give sample maximum distances for different situations, assuming a focal length of 1075 pixels and the parameter `quality` to be set to `High`.

Table 6.24: Maximum detection distance examples for AprilTags with a width of $t = 4$ cm

AprilTag family	Tag width	Maximum distance
36h11 (recommended)	8 modules	1.1 m
16h5	6 modules	1.4 m
41h12 (recommended)	5 modules	1.7 m

Table 6.25: Maximum detection distance examples for QR codes with a width of $t = 8$ cm

Tag width	Maximum distance
29 modules	0.49 m
21 modules	0.70 m

6.3.3.3 Pose estimation

For each detected tag, the pose of this tag in the camera coordinate frame is estimated. A requirement for pose estimation is that a tag is fully visible in the left and right camera image. The coordinate frame of the tag is aligned as shown below.

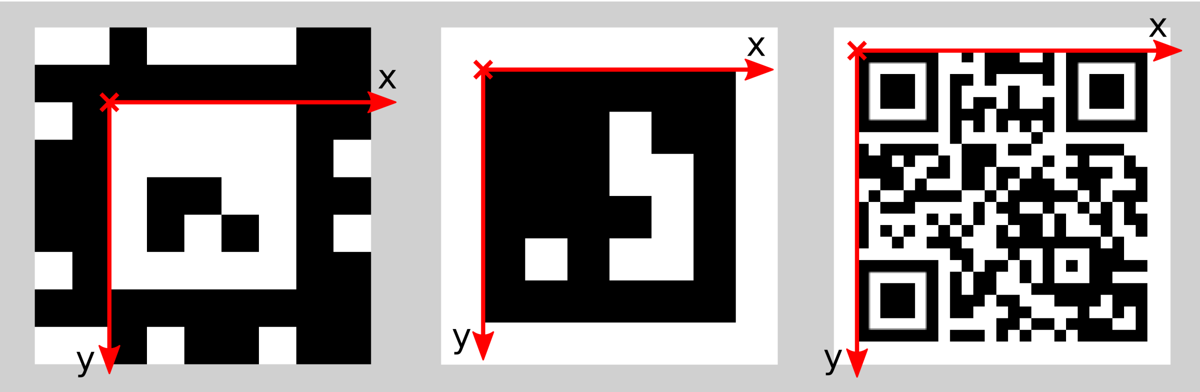

Fig. 6.9: Coordinate frames of AprilTags (left and center) and QR codes (right)

The z-axis is pointing "into" the tag. Please note that for AprilTags, although having the white border included in their definition, the coordinate system's origin is placed exactly at the transition from the white to the black border. Since AprilTags do not have an obvious orientation, the origin is defined as the upper left corner in the orientation they are pre-generated in.

During pose estimation, the tag's size is also estimated, while assuming the tag to be square. For QR codes, the size covers the full tag. For AprilTags, the size covers only the part inside the border defined by the transition from the black to the white border modules, hence ignoring the outermost white border for the tag families 16h5, 25h9, 36h10 and 36h11.

The user can also specify the approximate size $(\pm 10\%)$ of tags. All tags not matching this size constraint are automatically filtered out. This information is further used to resolve ambiguities in pose estimation that may arise if multiple tags with the same ID are visible in the left and right image and these tags are aligned in parallel to the image rows.

Note: For best pose estimation results one should make sure to accurately print the tag and to attach it to a rigid and as planar as possible surface. Any distortion of the tag or bump in the surface will degrade the estimated pose.

Note: It is highly recommended to set the approximate size of a tag. Otherwise, if multiple tags with the same ID are visible in the left or right image, pose estimation may compute a wrong pose if these tags have the same orientation and are approximately aligned in parallel to the image rows. However, even if the approximate size is not given, the TagDetect modules try to detect such situations and filter out affected tags.

Below tables give approximate precisions of the estimated poses of AprilTags. We distinguish between lateral precision (i.e., in x and y direction) and precision in z direction. It is assumed that `quality` is set to `High`, that the camera's viewing direction is parallel to the tag's normal and that the images are well exposed and do not suffer from motion blur. The size of a tag does not have a significant effect on the lateral or z precision; however, in general, larger tags improve precision. With respect to precision of the orientation especially around the x and y axes, larger tags clearly outperform smaller ones.

Table 6.26: Approximate orientation precision for AprilTag detections with `High` quality in an ideal scenario for different tag sizes

Distance	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 re-identification

Each tag has an ID; for AprilTags it is the *family* plus *tag ID*, for QR codes it is the contained data. However, these IDs are not unique, since the same tag may appear multiple times in a scene.

For distinction of these tags, the TagDetect modules also assign each detected tag a unique identifier. To help the user identifying an identical tag over multiple detections, tag detection tries to re-identify tags; if successful, a tag is assigned the same unique identifier again.

Tag re-identification compares the positions of the corners of the tags in the camera coordinate frame to find identical tags. Tags are assumed identical if they did not or only slightly move in that frame.

By setting the `max_corner_distance` threshold, the user can specify how much a tag is allowed move in the static coordinate frame between two detections to be considered identical. This parameter defines the maximum distance between the corners of two tags, which is shown in Fig. 6.10. The Euclidean distances of all four corresponding tag corners are computed in 3D. If none of these distances exceeds the threshold, the tags are considered identical.

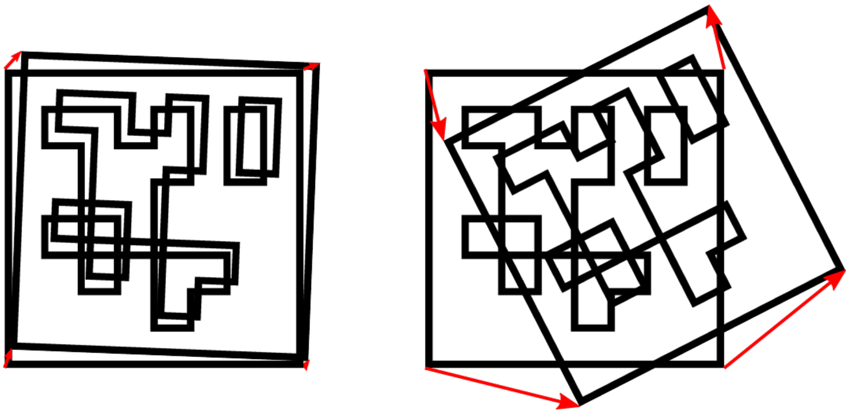

Fig. 6.10: Simplified visualization of tag re-identification. Euclidean distances between associated tag corners in 3D are compared (red arrows).

After a number of tag detection runs, previously detected tag instances will be discarded if they are not detected in the meantime. This can be configured by the parameter `forget_after_n_detections`.

6.3.3.5 Hand-eye calibration

In case the camera has been calibrated to a robot, the TagDetect module can automatically provide poses in the robot coordinate frame. For the TagDetect node's *Services* (Section 6.3.3.8), the frame of the output poses can be controlled with the `pose_frame` argument.

Two different `pose_frame` values can be chosen:

1. **Camera frame** (`camera`). All poses provided by the module are in the camera frame.
2. **External frame** (`external`). All poses provided by the module are in the external frame, configured by the user during the hand-eye calibration process. The module relies on the on-board *Hand-eye*

calibration module (Section 6.4.1) to retrieve the sensor mounting (static or robot mounted) and the hand-eye transformation. If the sensor mounting is static, no further information is needed. If the sensor is robot-mounted, the `robot_pose` is required to transform poses to and from the `external` frame.

All `pose_frame` values that are not `camera` or `external` are rejected.

6.3.3.6 Parameters

There are two separate modules available for tag detection, one for detecting AprilTags and one for QR codes, named `rc_april_tag_detect` and `rc_qr_code_detect`, respectively. Apart from the module names they share the same interface definition.

In addition to the *REST-API interface* (Section 7.2), the TagDetect modules provide pages on the Web GUI in the desired pipeline under *Modules* → *AprilTag* and *Modules* → *QR Code*, on which they can be tried out and configured manually.

In the following, the parameters are listed based on the example of `rc_qr_code_detect`. They are the same for `rc_april_tag_detect`.

This module offers the following run-time parameters:

Table 6.27: The `rc_qr_code_detect` module's run-time parameters

Name	Type	Min	Max	Default	Description
`detect_inverted_tags`	bool	false	true	false	Detect tags with black and white exchanged
`forget_after_n_detections`	int32	1	1000	30	Number of detection runs after which to forget about a previous tag during tag re-identification
`max_corner_distance`	float64	0.001	0.01	0.005	Maximum distance of corresponding tag corners in meters during tag re-identification
`quality`	string	-	-	High	Quality of tag detection: [Low, Medium, High]
`use_cached_images`	bool	false	true	false	Use most recently received image pair instead of waiting for a new pair

Via the REST-API, these parameters can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/parameters?
↪<parameter-name>=<value>
```

6.3.3.7 Status values

The TagDetect modules reports the following status values:

Table 6.28: The `rc_qr_code_detect` and `rc_april_tag_detect` module's status values

Name	Description
`data_acquisition_time`	Time in seconds required to acquire image pair
`last_timestamp_processed`	The timestamp of the last processed image pair
`processing_time`	Processing time of the last detection in seconds
`state`	The current state of the node

The reported `state` can take one of the following values.

Table 6.29: Possible states of the TagDetect modules

State name	Description
IDLE	The module is idle.
RUNNING	The module is running and ready for tag detection.
FATAL	A fatal error has occurred.

6.3.3.8 Services

The TagDetect modules implement a state machine for starting and stopping. The actual tag detection can be triggered via `detect`.

The user can explore and call the `rc_qr_code_detect` and `rc_april_tag_detect` modules' services, e.g. for development and testing, using the *REST-API interface* (Section 7.2) or the *rc_reason_stack Web GUI* (Section 7.1).

`detect`

Triggers a tag detection.

Details

Depending on the `use_cached_images` parameter, the module will use the latest received image pair (if set to true) or wait for a new pair that is captured after the service call was triggered (if set to false, this is the default). Even if set to true, tag detection will never use one image pair twice.

It is recommended to call `detect` in state `RUNNING` only. It is also possible to be called in state `IDLE`, resulting in an auto-start and stop of the module. This, however, has some drawbacks: First, the call will take considerably longer; second, tag re-identification will not work. It is therefore highly recommended to manually start the module before calling `detect`.

Tags might be omitted from the `detect` response due to several reasons, e.g., if a tag is visible in only one of the cameras or if pose estimation did not succeed. These filtered-out tags are noted in the log, which can be accessed as described in *Downloading log files* (Section 8.3).

A visualization of the latest detection is shown on the Web GUI tabs of the TagDetect modules. Please note that this visualization will only be shown after calling the detection service at least once. On the Web GUI, one can also manually try the detection by clicking the *Detect* button.

Due to changes in system time on the *rc_reason_stack* there might occur jumps of timestamps, forward as well as backward. Forward jumps do not have an effect on the TagDetect module. Backward jumps, however, invalidate already received images. Therefore, in case a backwards time jump is detected, an error of value -102 will be issued on the next `detect` call, also to inform the user that the timestamps included in the response will jump back. This service can be called as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪services/detect
```

Request

Optional arguments:

`tags` is the list of tag IDs that the TagDetect module should detect. For QR codes, the ID is the contained data. For AprilTags, it is *"<family>_<id>"*, so, e.g., for a tag of family 36h11 and ID 5, it is *"36h11_5"*. Naturally, the AprilTag module can only be triggered for AprilTags, and the QR code module only for QR codes.

The `tags` list can also be left empty. In that case, all detected tags will be returned. This feature should be used only during development and debugging of an application. Whenever possible, the concrete tag IDs should be listed, on the one hand avoiding some false positives, on the other hand speeding up tag detection by filtering tags not of interest.

For AprilTags, the user can not only specify concrete tags but also a complete family by setting the ID to "<family>", so, e.g., "36h11". All tags of this family will then be detected. It is further possible to specify multiple complete tag families or a combination of concrete tags and complete tag families; for instance, triggering for "36h11", "25h9_3", and "36h10" at the same time.

In addition to the ID, the approximate size ($\pm 10\%$) of a tag can be set with the `size` parameter. As described in *Pose estimation* (Section 6.3.3.3), this information helps to resolve ambiguities in pose estimation that may arise in certain situations and can be used to filter out tags not fulfilling the given size constraint.

The `tags` list is OR-connected. All tags will be returned that match any of `id-size` pair elements in the `tags` list.

`pose_frame` controls whether the poses of the detected tags are returned in the camera or external frame, as detailed in *Hand-eye calibration* (Section 6.3.3.5). The default is `camera`.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "pose_frame": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
```

(continues on next page)

(continued from previous page)

```
    },
    "tags": [
      {
        "id": "string",
        "size": "float64"
      }
    ]
  }
}
```

Response

`timestamp` is set to the timestamp of the image pair the tag detection ran on.

`tags` contains all detected tags.

`id` is the ID of the tag, similar to `id` in the request.

`instance_id` is the random unique identifier of the tag assigned by tag re-identification.

`pose` contains `position` and `orientation`. The orientation is in quaternion format.

`pose_frame` is set to the coordinate frame above pose refers to. It will either be "camera" or "external".

`size` will be set to the estimated tag size in meters.

`return_code` holds possible warnings or error codes.

The definition for the response with corresponding datatypes is:

```
{
  "name": "detect",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "tags": [
      {
        "id": "string",
        "instance_id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "size": "float64",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        }
      }
    ],
```

(continues on next page)

(continued from previous page)

```
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

start

Starts the module by transitioning from `IDLE` to `RUNNING`.

Details

When running, the module receives images from the stereo camera and is ready to perform tag detections. To save computing resources, the module should only be running when necessary.

This service can be called as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/services/start
```

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

```
{
  "name": "start",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

Stops the module by transitioning to `IDLE`.

Details

This transition can be performed from state `RUNNING` and `FATAL`. All tag re-identification information is cleared during stopping.

This service can be called as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪services/stop
```

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

restart

Restarts the module.

Details

If in RUNNING or FATAL, the module will be stopped and then started. If in IDLE, the module will be started.

This service can be called as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/
↪services/restart
```

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

```
{
  "name": "restart",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

reset_defaults

Resets all parameters of the module to its default values, as listed in above table.

Details

This service can be called as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/<rc_qr_code_detect|rc_april_tag_detect>/services/reset_
↪defaults
```

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.3.3.9 Return codes

Each service response contains a `return_code`, which consists of a `value` plus an optional `message`. A successful service returns with a `return_code` value of `0`. Negative `return_code` values indicate that the service failed. Positive `return_code` values indicate that the service succeeded with additional information. The smaller value is selected in case a service has multiple `return_code` values, but all messages are appended in the `return_code` message.

The following table contains a list of common return codes:

Code	Description
0	Success
-1	An invalid argument was provided
-4	A timeout occurred while waiting for the image pair
-9	The license is not valid
-11	Sensor not connected, not supported or not ready
-101	Internal error during tag detection
-102	There was a backwards jump of system time
-103	Internal error during tag pose estimation
-200	A fatal internal error occurred
200	Multiple warnings occurred; see list in `message`
201	The module was not in state `RUNNING`

6.3.4 ItemPick

6.3.4.1 Introduction

The ItemPick module provides an out-of-the-box perception solution for robotic pick-and-place applications. ItemPick targets the detection of flat surfaces on unknown objects for picking with a suction gripper. ItemPickAI uses neural networks to segment objects of a given object category and computes oriented and object-centered grasp points for suction grippers.

In addition, the module offers:

- A dedicated page on the *rc_reason_stack* *Web GUI* (Section 7.1) for easy setup, configuration, testing, and application tuning.
- The definition of regions of interest to select relevant volumes in the scene (see *RoiDB*, Section 6.5.2).
- A load carrier detection functionality for bin-picking applications (see *LoadCarrier*, Section 6.3.2), to provide grasps for items inside a bin only.
- The definition of compartments inside a load carrier to provide grasps for specific volumes of the bin only.
- Collision checking between the gripper and the load carrier and/or the point cloud.
- Support for static and robot-mounted cameras and optional integration with the *Hand-eye calibration* (Section 6.4.1) module, to provide grasps in the user-configured external reference frame.
- A quality value associated to each suggested grasp and related to the flatness of the grasping surface.
- Selection of a sorting strategy to sort the returned grasps.
- 3D visualization of the detection results with grasp points and gripper animations in the Web GUI.

Note: This module is pipeline specific. Changes to its settings or parameters only affect the respective camera pipeline and have no influence on other pipelines running on the *rc_reason_stack*.

Note: In this chapter, cluster and surface are used as synonyms and identify a set of points (or pixels) with defined geometrical properties.

The module is an optional on-board module of the *rc_reason_stack* and requires a separate ItemPick or ItemPickAI *license* (Section 8.2) to be purchased.

6.3.4.2 Computation of grasps

The ItemPick module offers a service for computing grasps for suction grippers. The gripper is defined by its suction surface length and width.

The ItemPick module identifies flat surfaces in the scene and supports flexible and/or deformable items. The `type` of these `item_models` is called `UNKNOWN` since they don't need to have a standard geometrical shape. Optionally, the user can also specify the minimum and maximum size of the item.

For ItemPickAI, the grasps are computed in the center of the top surface of the segmented objects (`items`) of the given object category. The object category is chosen by setting the `type` of `item_models`. Currently the types `BAG` and `CONSUMER_GOODS` are supported. `BAG` refers to deformable and flexible bag-like objects with various filling levels, such as pouch packs, packets, bulk bags, shipping bags, paper bags and sacks. `CONSUMER_GOODS` includes general packaged consumer products, such as packaged food, beverages, toiletries, cleaning supplies, and other affordable household goods.

Note: The first detection call with the `BAG` or `CONSUMER_GOODS` item model takes longer than the following detection calls, because the model has to be loaded into the ItemPickAI module first.

Optionally, further information can be given to the modules in a grasp computation request:

- The ID of the load carrier which contains the items to be grasped.
- A compartment inside the load carrier where to compute grasps (see *Load carrier compartments*, Section 6.5.1.3).
- The ID of the 3D region of interest where to search for the load carriers if a load carrier is set. Otherwise, the ID of the 3D region of interest where to compute grasps.

- Collision detection information: The ID of the gripper to enable collision checking and optionally a pre-grasp offset to define a pre-grasp position. Details on collision checking are given below in *CollisionCheck* (Section 6.3.4.4).

A grasp provided by the ItemPick module represents the recommended pose of the TCP (Tool Center Point) of the suction gripper. The grasp `type` is always set to `SUCTION`.

For ItemPick with an `UNKNOWN` item model, the computed grasp pose is the center of the biggest ellipse that can be inscribed in each surface.

For ItemPickAI with the `BAG` or `CONSUMER_GOODS` item model, the grasp position corresponds to the center of the top surface of the segmented objects.

The grasp orientation is a right-handed coordinate system and is defined such that its z axis is normal to the surface pointing inside the object at the grasp position and its x axis is directed along the maximum elongation of the ellipse. Since the x axis can have two possible directions, the one that better fits to the preferred TCP orientation (see *Setting the preferred orientation of the TCP*, Section 6.3.4.3) is selected. If the run-time parameter `allow_any_grasp_z_rotation` is set to true, the x axis will not be forced to be aligned with the maximum elongation of the graspable ellipse, but can have any rotation around the z axis. In this case, the returned grasp will have the orientation that best fits to the preferred TCP orientation and is collision free, if collision checking.

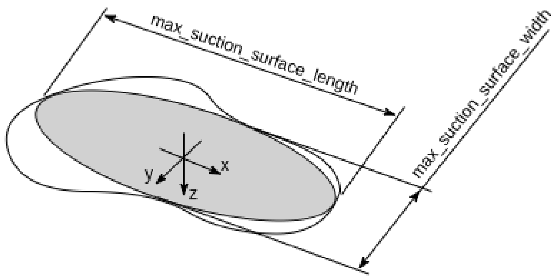

Fig. 6.11: Illustration of a suction grasp with coordinate system and ellipse representing the maximum suction surface

Each grasp includes the dimensions of the maximum suction surface available, modelled as an ellipse of axes `max_suction_surface_length` and `max_suction_surface_width`. The user is enabled to filter grasps by specifying the minimum suction surface required by the suction device in use. If the run-time parameter `allow_any_grasp_z_rotation` is set to true, `max_suction_surface_length` and `max_suction_surface_width` will be equal and correspond to the shortest axis of the largest graspable ellipse.

Each grasp also includes a `quality` value, which gives an indication of the flatness of the grasping surface. The `quality` value varies between 0 and 1, where higher numbers correspond to a flatter reconstructed surface.

The grasp definition is complemented by a `uuid` (Universally Unique Identifier) and the `timestamp` of the oldest image that was used to compute the grasp.

Grasp sorting is performed based on the selected sorting strategy. The following sorting strategies are available and can be set in the *Web GUI* (Section 7.1) or using the `set_sorting_strategies` service call:

- `gravity`: highest grasp points along the gravity direction are returned first,
- `surface_area`: grasp points with the largest surface area are returned first,
- `direction`: grasp points with the shortest distance along a defined direction `vector` in a given `pose_frame` are returned first.
- `distance_to_point`: grasp points with the shortest or farthest (if `farthest_first` is true) distance from a `point` in a given `pose_frame` are returned first.

If no sorting strategy is set or default sorting is chosen in the Web GUI, sorting is done based on a combination of `gravity` and `surface_area`.

6.3.4.3 Setting the preferred orientation of the TCP

The ItemPick module determines the reachability of grasp points based on the *preferred orientation* of the TCP. The preferred orientation can be set via the `set_preferred_orientation` service or on the ItemPick or ItemPickAI page in the Web GUI. The resulting direction of the TCP's z axis is used to reject grasps which cannot be reached by the gripper. Furthermore, the preferred orientation is used to select one grasp of several possible symmetries that is best reachable for the robot.

The preferred orientation can be set in the camera coordinate frame or in the external coordinate frame, in case a hand-eye calibration is available. If the preferred orientation is specified in the external coordinate frame and the sensor is robot mounted, the current robot pose has to be given to each object detection call. If no preferred orientation is set, the orientation of the left camera (see Coordinate frames in the rc_visard manual) will be used as the preferred orientation of the TCP.

6.3.4.4 Interaction with other modules

Internally, the ItemPick module depends on, and interacts with other on-board modules as listed below.

Note: All changes and configuration updates to these modules will affect the performance of the ItemPick module.

Camera and depth data

The ItemPick module makes internally use of the following data:

- Rectified images from the *Camera module* (`rc_camera`, Section 6.1);
- Disparity, error, and confidence images from the *Stereo matching module* (`rc_stereomatching`, Section 6.2.2), in case a stereo camera is used
- Disparity, error, and confidence images from the *Orbbec module* (`rc_orbbec`, Section 6.2.4), in case an *Orbbec* camera is used
- Disparity, error, and confidence images from the *Zivid module* (`rc_zivid`, Section 6.2.3), in case a *zivid* camera is used

All processed images are guaranteed to be captured after the module trigger time.

IO and Projector Control

In case the *rc_reason_stack* is used in conjunction with an external random dot projector and the *IO and Projector Control* module (`rc_iocontrol`, Section 6.4.4), it is recommended to connect the projector to GPIO Out 1 and set the stereo-camera module's acquisition mode to `SingleFrameOut1` (see *Stereo matching parameters*, Section 6.2.2.1), so that on each image acquisition trigger an image with and without projector pattern is acquired.

Alternatively, the output mode for the GPIO output in use should be set to `ExposureAlternateActive` (see *Description of run-time parameters*, Section 6.4.4.1).

In either case, the *Auto Exposure Mode* `exp_auto_mode` should be set to `AdaptiveOut1` to optimize the exposure of both images.

Hand-eye calibration

In case the camera has been calibrated to a robot, the ItemPick module can automatically provide poses in the robot coordinate frame. For the ItemPick node's *Services* (Section 6.3.4.7), the frame of the output poses can be controlled with the `pose_frame` argument.

Two different `pose_frame` values can be chosen:

1. **Camera frame** (`camera`). All poses provided by the modules are in the camera frame, and no prior knowledge about the pose of the camera in the environment is required. This means that the configured regions of interest and load carriers move with the camera. It is the user's responsibility to update the configured poses if the camera frame moves (e.g. with a robot-mounted camera).
2. **External frame** (`external`). All poses provided by the modules are in the external frame, configured by the user during the hand-eye calibration process. The module relies on the on-board *Hand-eye calibration module* (Section 6.4.1) to retrieve the sensor mounting (static or robot mounted) and the hand-eye transformation. If the mounting is static, no further information is needed. If the sensor is robot-mounted, the `robot_pose` is required to transform poses to and from the `external` frame.

Note: If no hand-eye calibration is available, all `pose_frame` values should be set to `camera`.

All `pose_frame` values that are not `camera` or `external` are rejected.

If the sensor is robot-mounted, the current `robot_pose` has to be provided depending on the value of `pose_frame` and the definition of the sorting direction or sorting point:

- If `pose_frame` is set to `external`, providing the robot pose is obligatory.
- If the sorting direction is defined in `external`, providing the robot pose is obligatory.
- If the distance-to-point sorting strategy is defined in `external`, providing the robot pose is obligatory.
- In all other cases, providing the robot pose is optional.

LoadCarrier

The ItemPick module uses the load carrier detection functionality provided by the *LoadCarrier* module (`rc_load_carrier`, Section 6.3.2), with the run-time parameters specified for this module. However, only one load carrier will be returned and used in case multiple matching load carriers could be found in the scene. In case multiple load carriers of the same type are visible, a 3D region of interest should be set to ensure that always the same load carrier is used for the ItemPick module.

CollisionCheck

Collision checking can be easily enabled for grasp computation of the ItemPick module by passing the ID of the used gripper and optionally a pre-grasp offset to the `compute_grasps` service call. The gripper has to be defined in the GripperDB module (see *Setting a gripper*, Section 6.5.3.2) and details about collision checking are given in *Collision checking within other modules* (Section 6.4.2.2).

If collision checking is enabled, only grasps which are collision free will be returned. However, the visualization images on the ItemPick or ItemPickAI page of the Web GUI also show colliding grasp points as black ellipses.

The CollisionCheck module's run-time parameters affect the collision detection as described in *CollisionCheck Parameters* (Section 6.4.2.3).

6.3.4.5 Parameters

ItemPick is represented by the `rc_itempick` node in the REST-API and are reached in the *Web GUI* (Section 7.1) in the desired pipeline under *Modules* → *ItemPick* and *Modules* → *ItemPickAI*. If both licenses, ItemPick, are present on a device, the ItemPick functionality will be integrated into the *ItemPickAI* page of the Web GUI. The user can explore and configure the `rc_itempick` module's run-time parameters, e.g. for development and testing, using the Web GUI or the *REST-API interface* :(Section 7.2).

The user can explore and configure the `rc_itempick` module's run-time parameters, e.g. for development and testing, using the Web GUI or the *REST-API interface* (Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.30: The `rc_itempick` module's run-time parameters

Name	Type	Min	Max	Default	Description
`allow_any_grasp_pose`	bool	false	true	false	Whether the grasps are allowed to be placed anywhere on the objects where planar surfaces are detected
`allow_any_grasp_z_rotation`	bool	false	true	false	Whether the grasps are allowed to have arbitrary rotation instead being aligned with the major axis of the graspable ellipse
`check_collisions_with_point_cloud`	bool	false	true	false	Whether to check for collisions between gripper and the point cloud
`cluster_max_curvature`	float64	0.005	0.5	0.11	Maximum curvature allowed within one cluster. The smaller this value, the more clusters will be split apart.
`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	Factor used to discriminate depth discontinuities within a patch. The smaller this value, the more clusters will be split apart.
`clustering_max_surface_rmse`	float64	0.0005	0.01	0.004	Maximum root-mean-square error (RMSE) in meters of points belonging to a surface
`clustering_patch_size`	int32	3	10	4	Size in pixels of the square patches the depth map is subdivided into during the first clustering step
`grasp_filter_orientation_threshold`	float64	0.0	180.0	45.0	Maximum allowed orientation change between grasp and preferred orientation in degrees
`max_grasps`	int32	1	100	5	Maximum number of provided grasps

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's ItemPick or ItemPickAI page. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI:

max_grasps (*Maximum Grasps*)

sets the maximum number of provided grasps.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?max_grasps=<value>
```

cluster_max_dimension (*Cluster Maximum Dimension*, Only for 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.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?cluster_max_dimension=<value>
```

cluster_max_curvature (*Cluster Maximum Curvature*, Only for ItemPick)

is the maximum curvature allowed within one cluster. The smaller this value, the more clusters will be split apart.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?cluster_max_curvature=<value>
```

clustering_patch_size (*Patch Size*, Only for ItemPick)

is the size of the square patches the depth map is subdivided into during the first clustering step in pixels.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?clustering_patch_size=<value>
```

`clustering_discontinuity_factor` (*Discontinuity Factor*, Only for ItemPick)

is the factor used to discriminate depth discontinuities within a patch. The smaller this value, the more clusters will be split apart.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`clustering_max_surface_rmse` (*Maximum Surface RMSE*, Only for ItemPick)

is the maximum root-mean-square error (RMSE) in meters of points belonging to a surface.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

`grasp_filter_orientation_threshold` (*Grasp Orientation Threshold*)

is the maximum deviation of the TCP's z axis at the grasp point from the z axis of the TCP's preferred orientation in degrees. Only grasp points which are within this threshold are returned. When set to zero, any deviations are valid.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?grasp_filter_orientation_
↪threshold=<value>
```

allow_any_grasp_z_rotation (*Allow Any Grasp Z Rotation*)

If set to true, the returned grasps are no longer forced to have their x axes aligned with the maximum elongation of the graspable ellipse, but can have any rotation around the z axis. The returned `max_suction_surface_length` and `max_suction_surface_width` will be equal and correspond to the shortest diameter of the largest graspable ellipse. This parameter enables the robot to get more options for grasping objects, especially in scenes where collisions can occur. However, in case of `UNKNOWN` item models, since the grasp is no longer aligned with the graspable ellipse, the correct orientation for placing the object must be determined by other means. In case of ItemPickAI, the corresponding item's pose can be used to determine the correct grasp orientation for placement.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

allow_any_grasp_pose (*Allow Any Grasp Pose*)

If set to true, the grasps are no longer forced to be centered on the object and aligned with the major axis of the object, but may be located anywhere on the object where graspable surfaces are found. For this, the segmented object surfaces are clustered using the clustering parameters to find the graspable surfaces of an object. This parameter enables the robot to get more options for grasping objects, especially on concave objects or objects with surface discontinuities. This parameter has no effect when the `UNKNOWN` item model type is used.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?allow_any_grasp_pose=<value>
```

check_collisions_with_point_cloud (*Check Collisions with Point Cloud*)

This parameter is only used when collision checking is enabled by passing a gripper to the `compute_grasps` service call. If `check_collisions_with_point_cloud` is set to true, all grasp points will be checked for collisions between the gripper and a watertight version of the point cloud, and only grasp points at which the gripper would not collide with this point cloud will be returned.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/parameters?check_collisions_with_point_
↪cloud=<value>
```

6.3.4.6 Status values

The `rc_itempick` node reports the following status values:

Table 6.31: The `rc_itempick` node's status values

Name	Description
`data_acquisition_time`	Time in seconds required by the last active service to acquire images
`grasp_computation_time`	Processing time of the last grasp computation in seconds
`last_timestamp_processed`	The timestamp of the last processed dataset
`load_carrier_detection_time`	Processing time of the last load carrier detection in seconds
`processing_time`	Processing time of the last detection (including load carrier detection) in seconds
`state`	The current state of the rc_itempick node

The reported `state` can take one of the following values.

Table 6.32: Possible states of the ItemPick module

State name	Description
IDLE	The module is idle.
RUNNING	The module is running and ready for load carrier detection and grasp computation.
FATAL	A fatal error has occurred.

6.3.4.7 Services

The user can explore and call the `rc_itempick` node's services, e.g. for development and testing, using the *REST-API interface* (Section 7.2) or the *rc_reason_stack Web GUI* (Section 7.1).

The ItemPick module offers the following services.

`compute_grasps`

Triggers the computation of grasping poses for a suction device as described in *Computation of grasps* (Section 6.3.4.2).

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/compute_grasps
```

Request

Required arguments:

> pose_frame: see *Hand-eye calibration* (Section 6.3.4.4).
>
> suction_surface_length: length of the suction device grasping surface.
>
> suction_surface_width: width of the suction device grasping surface.

Potentially required arguments:

> robot_pose: see *Hand-eye calibration* (Section 6.3.4.4).

Optional arguments:

> load_carrier_id: ID of the load carrier which contains the items to be grasped.
>
> load_carrier_compartment: compartment inside the load carrier where to compute grasps (see *Load carrier compartments*, Section 6.5.1.3).
>
> region_of_interest_id: if load_carrier_id is set, ID of the 3D region of interest where to search for the load carriers. Otherwise, ID of the 3D region of interest where to compute grasps.
>
> item_models: list of items to be detected. In case of ItemPick, currently only a single item model of type UNKNOWN with minimum and maximum dimensions is supported, with the minimum dimensions strictly smaller than the maximum dimensions.
>
> In case of ItemPickAI, currently the item model types BAG and CONSUMER_GOODS are supported.
>
> collision_detection: see *Collision checking within other modules* (Section 6.4.2.2).

The definition for the request arguments with corresponding datatypes is:

```
{
  "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": {
```

(continues on next page)

(continued from previous page)

```
        "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`: list of detected load carriers.

`grasps`: sorted list of suction grasps.

`items`: sorted list of items corresponding to the returned grasps. In case of ItemPick, this list is always empty.

In case of ItemPickAI, `items` contains the segmented items of type `BAG` or `CONSUMER_GOODS` with their poses corresponding to the center of the bounding box of the object's visible part and the dimensions of this `bounding_box`.

`timestamp`: timestamp of the image set the detection ran on.

`return_code`: holds possible warnings or error codes and messages.

The definition for the response with corresponding datatypes is:

```
{
  "name": "compute_grasps",
  "response": {
    "grasps": [
      {
```

(continues on next page)

(continued from previous page)

```
        "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": [
      {
        "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",
        "template_id": "string",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "type": "string",
        "uuid": "string",
        "view_name": "string",
        "view_pose_set": "bool",
        "view_uuid": "string"
      }
```

(continues on next page)

(continued from previous page)

```
    ],
    "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

Persistently stores the preferred orientation of the TCP to compute the reachability of the grasps, which is used for filtering and the grasps returned by the `compute_grasps` service (see *Setting the preferred orientation of the TCP*, Section 6.3.4.3).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/set_preferred_
↪orientation
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/set_preferred_orientation
```

Request

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

get_preferred_orientation

Returns the preferred orientation of the TCP to compute the reachability of the grasps, which is used for filtering the grasps returned by the `compute_grasps` service (see *Setting the preferred orientation of the TCP*, Section 6.3.4.3).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/get_preferred_
↪orientation
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/get_preferred_orientation
```

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

```
{
  "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

Persistently stores the sorting strategy for sorting the grasps returned by the `compute_grasps` service (see *Computation of grasps*, Section 6.3.4.2).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/set_sorting_
↪strategies
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/set_sorting_strategies
```

Request

Only one strategy may have a `weight` greater than 0. If all `weight` values are set to 0, the module will use the default sorting strategy.

If the weight for `direction` is set, the `vector` must contain the direction vector and `pose_frame` must be either `camera` or `external`.

If the weight for `distance_to_point` is set, `point` must contain the sorting point and `pose_frame` must be either `camera` or `external`.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
    },
```

(continues on next page)

(continued from previous page)

```
    "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

The definition for the response with corresponding datatypes is:

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

get_sorting_strategies

Returns the sorting strategy for sorting the grasps returned by the `compute-grasps` service (see *Computation of grasps*, Section 6.3.4.2).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_itempick/services/get_sorting_
↪strategies
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/get_sorting_strategies
```

Request

This service has no arguments.

Response

All `weight` values are 0 when the module uses the default sorting strategy.

The definition for the response with corresponding datatypes is:

```
{
  "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

Starts the module. If the command is accepted, the module moves to state `RUNNING`.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/start
```

Request

This service has no arguments.

Response

The `current_state` value in the service response may differ from `RUNNING` if the state transition is still in process when the service returns.

The definition for the response with corresponding datatypes is:

```
{
  "name": "start",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

Stops the module. If the command is accepted, the module moves to state `IDLE`.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_itempick/services/stop
```

Request

This service has no arguments.

Response

The `current_state` value in the service response may differ from `IDLE` if the state transition is still in process when the service returns.

The definition for the response with corresponding datatypes is:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

reset_defaults

Resets all parameters of the module to its default values, as listed in above table. Also resets sorting strategies.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.3.4.8 Return codes

Each service response contains a `return_code`, which consists of a `value` plus an optional `message`. A successful service returns with a `return_code` value of `0`. Negative `return_code` values indicate that the service failed. Positive `return_code` values indicate that the service succeeded with additional information. The smaller value is selected in case a service has multiple `return_code` values, but all messages are appended in the `return_code` message.

The following table contains a list of common codes:

Table 6.33: Return codes of the ItemPick services

Code	Description
0	Success
-1	An invalid argument was provided
-3	An internal timeout occurred, e.g. during box detection if the given dimension range is too large
-4	Data acquisition took longer than allowed
-8	The template has been deleted during detection.
-10	New element could not be added as the maximum storage capacity of load carriers, regions of interest or template has been exceeded
-11	Sensor not connected, not supported or not ready
-200	Fatal internal error
-301	More than one item model provided to the `compute_grasps` service
10	The maximum storage capacity of load carriers, regions of interest or templates has been reached
11	An existent persistent model was overwritten by the call to `set_load_carrier` or `set_region_of_interest`
100	The requested load carriers were not detected in the scene
101	No valid surfaces or grasps were found in the scene
102	The detected load carrier is empty
103	All computed grasps are in collision
112	Rejected detections of one or more clusters, because min_cluster_coverage was not reached.
300	A valid `robot_pose` was provided as argument but it is not required
999	Additional hints for application development

6.3.5 BoxPick

6.3.5.1 Introduction

The BoxPick module provides an out-of-the-box perception solution for robotic pick-and-place applications. It detects rectangular surfaces and determines their position, orientation and size for grasping. With the +Match extension, BoxPick can be used to detect textured rectangles with consistent orientations, such as printed product packaging, labels, brochures or books.

In addition, the module offers:

- A dedicated page on the *rc_reason_stack* *Web GUI* (Section 7.1) for easy setup, configuration, testing, and application tuning.
- The definition of regions of interest to select relevant volumes in the scene (see *RoiDB*, Section 6.5.2).
- A load carrier detection functionality for bin-picking applications (see *LoadCarrier*, Section 6.3.2), to provide grasps for items inside a bin only.
- The definition of compartments inside a load carrier to provide grasps for specific volumes of the bin only.
- Collision checking between the gripper and the load carrier and/or the point cloud.
- Support for static and robot-mounted cameras and optional integration with the *Hand-eye calibration* (Section 6.4.1) module, to provide grasps in the user-configured external reference frame.
- A quality value associated to each suggested grasp and related to the flatness of the grasping surface.
- Selection of a sorting strategy to sort the returned grasps.
- 3D visualization of the detection results with grasp points and gripper animations in the Web GUI.

Note: This module is pipeline specific. Changes to its settings or parameters only affect the respective camera pipeline and have no influence on other pipelines running on the *rc_reason_stack*.

Note: In this chapter, cluster and surface are used as synonyms and identify a set of points (or pixels) with defined geometrical properties.

The module is an optional on-board module of the *rc_reason_stack* and requires a separate BoxPick *license* (Section 8.2) to be purchased. The +Match extension requires an extra license.

6.3.5.2 Detection of items

There are two different types of models for the rectangles to be detected by the BoxPick module.

Per default, BoxPick only supports `item_models` of `type` `RECTANGLE`. With the +Match extension, also item models of `type` `TEXTURED_BOX` can be detected. The detection of the different item model types is described below.

Optionally, further information can be given to the BoxPick module:

- The ID of the load carrier which contains the items to be detected.
- A compartment inside the load carrier where to detect items.
- The ID of the region of interest where to search for the load carriers if a load carrier is set. Otherwise, the ID of the region of interest where to search for the items.
- The current robot pose in case the camera is mounted on the robot and the chosen coordinate frame for the poses is `external` or the chosen region of interest is defined in the external frame.

The returned `pose` of a detected `item` is the pose of the center of the detected rectangle in the desired reference frame (`pose_frame`), with its z axis pointing towards the camera and the x axis aligned with the long side of the item. This pose has a 180° rotation ambiguity around the z axis, which can be resolved by using the +Match extension with a `TEXTURED_BOX` item model. Each detected item includes a `uuid` (Universally Unique Identifier) and the `timestamp` of the oldest image that was used to detect it.

Detection of items of type RECTANGLE

BoxPick supports multiple `item_models` of `type RECTANGLE`. Each item model is defined by its minimum and maximum size, with the minimum dimensions strictly smaller than the maximum dimensions. The dimensions should be given fairly accurately to avoid misdetections, while still considering a certain tolerance to account for possible production variations and measurement inaccuracies.

The detection of the rectangles runs in several steps. First, the point cloud is segmented into preferably plane clusters. Then, straight line segments are detected in the 2D images and projected onto the corresponding clusters. The clusters and the detected lines are visualized in the "Intermediate Result" visualization on the Web GUI's *BoxPick* page. Finally, for each cluster, the set of rectangles best fitting to the detected line segments is extracted.

Detection of items of type TEXTURED_BOX (BoxPick+Match)

With the +Match extension, BoxPick additionally supports `item_models` of `type TEXTURED_BOX`. When this item model type is used, only one item model can be given for each request.

The `TEXTURED_BOX` item model type should be used to detect multiple rectangles that have the same texture, i.e. the same look or print, such as printed product packaging, labels, brochures or books. It is required that for all objects the texture is at the same position with respect to the object geometry. Furthermore, the texture should not be repetitive.

A `TEXTURED_BOX` item is defined by the item's exact `dimensions` `x`, `y` and `z` (only `z` is allowed to be 0) with a tolerance `dimensions_tolerance_m` that indicates, how much the detected dimensions are allowed to deviate from the given dimensions. By default, a tolerance of 0.01 m is assumed. Furthermore, a `template_id` must be given, which will be used to refer to the specified dimensions and the textures of the detected rectangles. Additionally, the maximum possible deformation of the items `max_deformation_m` can be given in meters (default 0.004 m), to account for rigid or more flexible objects.

If a `template_id` is used for the first time, BoxPick will run the detection of rectangles as for the item model type `RECTANGLE`, and use the given dimensions and tolerance to specify the dimensions range. If the z dimension is given in addition to x and y, rectangles with all possible combinations of the three dimensions will be detected. From the detected rectangles, so-called *views* are created, which contain the shape and the image intensity values of the rectangles, and are stored in a newly created template with the given `template_id`. The views are created iteratively: Starting from the detected rectangle with the highest score, a view is created and then used to detect more rectangles with the same texture. Then, all remaining clusters are used to detect further rectangles by the given dimensions range and again a view is created from the best rectangle and used for further detections. Each template can store up to 10 different views, for example corresponding to different types of the same product packaging. Each view will be assigned a unique ID (`view_uuid`) and all rectangle items with a matching texture will be assigned the same `view_uuid`. That also means that all items with the same `view_uuid` will have consistent orientations, because the orientation of each item is aligned with its texture. The views can be displayed, deleted and the orientation of each view can be set via the *Web GUI* (Section 7.1) by clicking on the template or its edit symbol in the template list. Each detected item contains a field `view_pose_set` indicating whether the orientation of the item's view was explicitly set or is still unset at its original random state, which has a 180° ambiguity. Additionally, a user-defined name can be set for each view, that is returned along with the `view_uuid` for all items and allows an easier identification of a specific view. The `type` of a returned item with a `view_uuid` will be `TEXTURED_RECTANGLE`.

If the template with the given `template_id` already exists, the existing views will be used to detect rectangles based on their texture. If additional rectangles are found with matching dimensions, but different texture, new views will be generated and added to the template. When the maximum number of views is reached, views that are matched only rarely will be deleted so that newly generated views can be added to the template and the template is kept up-to-date. To prevent a template from being updated, automatic view updating can be disabled and enabled for each template in the Web GUI by clicking on the template or the edit symbol in the template list. The dimension tolerance and the maximum deformation can also be changed there for each template. The maximum deformation determines the tolerance for the texture matching, representing possible shifts within the texture, e.g. caused by deformations of the object surface. For rigid objects the `max_deformation_m` should be set to a low value in meters to ensure accurate matching.

The template's `dimensions` can only be specified when creating a new template. Once the template is generated, the dimensions cannot be changed and do not need to be given in the detect request. If the dimensions are still given in the request, they must match the existing dimensions in the template. However, the `dimensions_tolerance_m` and `max_deformation_m` can be set differently in every detect request and their values will also be updated in the stored template.

6.3.5.3 Computation of grasps

The BoxPick module offers a service for computing grasps for suction grippers. The gripper is defined by its suction surface length and width.

The grasps are computed on the detected rectangular `items` (see *Detection of items*, Section 6.3.5.2).

Optionally, further information can be given to the module in a grasp computation request:

- The ID of the load carrier which contains the items to be grasped.
- A compartment inside the load carrier where to compute grasps (see *Load carrier compartments*, Section 6.5.1.3).
- The ID of the 3D region of interest where to search for the load carriers if a load carrier is set. Otherwise, the ID of the 3D region of interest where to compute grasps.
- Collision detection information: The ID of the gripper to enable collision checking and optionally a pre-grasp offset to define a pre-grasp position. Details on collision checking are given below in *CollisionCheck* (Section 6.3.5.5).

A grasp provided by the BoxPick module represents the recommended pose of the TCP (Tool Center Point) of the suction gripper. The grasp `type` is always set to `SUCTION`. The computed grasp pose is the center of the biggest ellipse that can be inscribed in each surface. The grasp orientation is a right-handed coordinate system and is defined such that its z axis is normal to the surface pointing inside the object at the grasp position and its x axis is directed along the maximum elongation of the ellipse. Since the x axis can have two possible directions, the one that better fits to the preferred TCP orientation (see *Setting the preferred orientation of the TCP*, Section 6.3.5.4) is selected. If the run-time parameter `allow_any_grasp_z_rotation` is set to true, the x axis will not be forced to be aligned with the maximum elongation of the graspable ellipse, but can have any rotation around the z axis. In this case, the returned grasp will have the orientation that best fits to the preferred TCP orientation and is collision free, if collision checking is enabled.

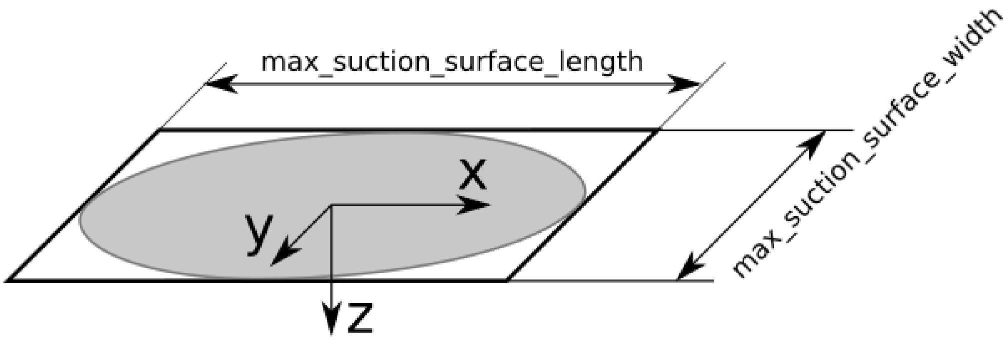

Fig. 6.12: Illustration of a suction grasp with coordinate system and ellipse representing the maximum suction surface

Each grasp includes the dimensions of the maximum suction surface available, modelled as an ellipse of axes `max_suction_surface_length` and `max_suction_surface_width`. The user is enabled to filter grasps by specifying the minimum suction surface required by the suction device in use. If the run-time parameter `allow_any_grasp_z_rotation` is set to true, `max_suction_surface_length` and `max_suction_surface_width` will be equal and correspond to the shortest axis of the largest graspable ellipse.

In the BoxPick module, the grasp position corresponds to the center of the detected rectangle. When BoxPick is called with item models of type `RECTANGLE`, the dimensions of the maximum suction surface available matches the estimated rectangle dimensions. In this case, detected rectangles with missing data or occlusions by other objects for more than 15% of their surface do not get an associated grasp.

When BoxPick is called with item models of type `TEXTURED_BOX`, grasps can also be computed on partly occluded boxes. The maximum suction surface available matches the free surface that is not occluded by other clusters.

Each grasp also includes a `quality` value, which gives an indication of the flatness of the grasping surface. The `quality` value varies between 0 and 1, where higher numbers correspond to a flatter reconstructed surface.

The grasp definition is complemented by a `uuid` (Universally Unique Identifier) and the `timestamp` of the oldest image that was used to compute the grasp.

Grasp sorting is performed based on the selected sorting strategy. The following sorting strategies are available and can be set in the *Web GUI* (Section 7.1) or using the `set_sorting_strategies` service call:

- `gravity`: highest grasp points along the gravity direction are returned first,
- `surface_area`: grasp points with the largest surface area are returned first,
- `direction`: grasp points with the shortest distance along a defined direction `vector` in a given `pose_frame` are returned first.
- `distance_to_point`: grasp points with the shortest or farthest (if `farthest_first` is true) distance from a `point` in a given `pose_frame` are returned first.

If no sorting strategy is set or default sorting is chosen in the Web GUI, sorting is done based on a combination of `gravity` and `surface_area`.

6.3.5.4 Setting the preferred orientation of the TCP

The BoxPick module determines the reachability of grasp points based on the *preferred orientation* of the TCP. The preferred orientation can be set via the `set_preferred_orientation` service or on the BoxPick page in the Web GUI. The resulting direction of the TCP's z axis is used to reject grasps which cannot be reached by the gripper. Furthermore, the preferred orientation is used to select one grasp of several possible symmetries that is best reachable for the robot.

The preferred orientation can be set in the camera coordinate frame or in the external coordinate frame, in case a hand-eye calibration is available. If the preferred orientation is specified in the external coordinate frame and the sensor is robot mounted, the current robot pose has to be given to each object detection call. If no preferred orientation is set, the orientation of the left camera (see Coordinate frames in the rc_visard manual) will be used as the preferred orientation of the TCP.used.

6.3.5.5 Interaction with other modules

Internally, the BoxPick module depends on, and interacts with other on-board modules as listed below.

Note: All changes and configuration updates to these modules will affect the performance of the BoxPick module.

Camera and depth data

The BoxPick module makes internally use of the following data:

- Rectified images from the *Camera module* (`rc_camera`, Section 6.1)
- Disparity, error, and confidence images from the *Stereo matching module* (`rc_stereomatching`, Section 6.2.2), in case a stereo camera is used
- Disparity, error, and confidence images from the *Orbbec module* (`rc_orbbec`, Section 6.2.4), in case an *Orbbec* camera is used
- Disparity, error, and confidence images from the *Zivid module* (`rc_zivid`, Section 6.2.3), in case a *zivid* camera is used

All processed images are guaranteed to be captured after the module trigger time.

IO and Projector Control

In case the *rc_reason_stack* is used in conjunction with an external random dot projector and the *IO and Projector Control* module (`rc_iocontrol`, Section 6.4.4), it is recommended to connect the projector to GPIO Out 1 and set the stereo-camera module's acquisition mode to `SingleFrameOut1` (see *Stereo matching parameters*, Section 6.2.2.1), so that on each image acquisition trigger an image with and without projector pattern is acquired.

Alternatively, the output mode for the GPIO output in use should be set to `ExposureAlternateActive` (see *Description of run-time parameters*, Section 6.4.4.1).

In either case, the *Auto Exposure Mode* `exp_auto_mode` should be set to `AdaptiveOut1` to optimize the exposure of both images.

Hand-eye calibration

In case the camera has been calibrated to a robot, the BoxPick module can automatically provide poses in the robot coordinate frame. For the BoxPick node's *Services* (Section 6.3.5.8), the frame of the output poses can be controlled with the `pose_frame` argument.

Two different `pose_frame` values can be chosen:

1. **Camera frame** (`camera`). All poses provided by the modules are in the camera frame, and no prior knowledge about the pose of the camera in the environment is required. This means that the configured regions of interest and load carriers move with the camera. It is the user's responsibility to update the configured poses if the camera frame moves (e.g. with a robot-mounted camera).

2. **External frame** (external). All poses provided by the modules are in the external frame, configured by the user during the hand-eye calibration process. The module relies on the on-board *Hand-eye calibration module* (Section 6.4.1) to retrieve the sensor mounting (static or robot mounted) and the hand-eye transformation. If the mounting is static, no further information is needed. If the sensor is robot-mounted, the robot_pose is required to transform poses to and from the external frame.

Note: If no hand-eye calibration is available, all pose_frame values should be set to camera.

All pose_frame values that are not camera or external are rejected.

If the sensor is robot-mounted, the current robot_pose has to be provided depending on the value of pose_frame and the definition of the sorting direction or sorting point:

- If pose_frame is set to external, providing the robot pose is obligatory.
- If the sorting direction is defined in external, providing the robot pose is obligatory.
- If the distance-to-point sorting strategy is defined in external, providing the robot pose is obligatory.
- In all other cases, providing the robot pose is optional.

LoadCarrier

The BoxPick module uses the load carrier detection functionality provided by the *LoadCarrier* module (rc_load_carrier, Section 6.3.2), with the run-time parameters specified for this module. However, only one load carrier will be returned and used in case multiple matching load carriers could be found in the scene. In case multiple load carriers of the same type are visible, a 3D region of interest should be set to ensure that always the same load carrier is used for the BoxPick module.

The load carrier is used to filter false detections when BoxPick is triggered with an item model of type TEXTURED_BOX and all three dimensions x, y, z are given. In this case, 3D boxes are created internally by adding the missing dimensions to the detected rectangles and only detections corresponding to boxes which are fully inside the detected load carrier are returned.

CollisionCheck

Collision checking can be easily enabled for grasp computation of the BoxPick module by passing the ID of the used gripper and optionally a pre-grasp offset to the compute_grasps service call. The gripper has to be defined in the GripperDB module (see *Setting a gripper*, Section 6.5.3.2) and details about collision checking are given in *Collision checking within other modules* (Section 6.4.2.2).

If collision checking is enabled, only grasps which are collision free will be returned. However, the visualization images on the *BoxPick* page of the Web GUI also show colliding grasp points as black ellipses.

The CollisionCheck module's run-time parameters affect the collision detection as described in *CollisionCheck Parameters* (Section 6.4.2.3).

6.3.5.6 Parameters

The BoxPick module is called rc_boxpick in the REST-API and is represented in the *Web GUI* (Section 7.1) in the desired pipeline under *Modules* → *BoxPick*. The user can explore and configure the rc_boxpick module's run-time parameters, e.g. for development and testing, using the Web GUI or the *REST-API interface* (Section 7.2).

Parameter overview

Note: The default values in the parameter table below show the values of the *rc_visard*. The values can be different for other sensors.

This module offers the following run-time parameters:

Table 6.34: The `rc_boxpick` module's run-time parameters

Name	Type	Min	Max	Default	Description
`allow_any_grasp_z_rotation`	bool	false	true	false	Whether the grasps are allowed to have arbitrary rotation instead being aligned with the major axis of the graspable ellipse
`allow_untextured_detections`	bool	false	true	false	Whether to return also untextured detections in case a textured box was given
`check_collisions_with_point_cloud`	bool	false	true	false	Whether to check for collisions between gripper and the point cloud
`cluster_max_curvature`	float64	0.005	0.5	0.11	Maximum curvature allowed within one cluster. The smaller this value, the more clusters will be split apart.
`clustering_discontinuity_factor`	float64	0.1	5.0	1.0	Factor used to discriminate depth discontinuities within a patch. The smaller this value, the more clusters will be split apart.
`clustering_max_surface_rmse`	float64	0.0005	0.01	0.004	Maximum root-mean-square error (RMSE) in meters of points belonging to a surface
`grasp_filter_orientation_threshold`	float64	0.0	180.0	45.0	Maximum allowed orientation change between grasp and preferred orientation in degrees
`line_sensitivity`	float64	0.1	1.0	0.1	Sensitivity of the line detector
`manual_line_sensitivity`	bool	false	true	false	Indicates whether the user-defined line sensitivity should be used or the automatic one
`max_grasps`	int32	1	100	5	Maximum number of provided grasps
`min_cluster_coverage`	float64	0.0	0.99	0.0	Gives the minimal ratio of points per cluster that must be covered with detected items.
`mode`	string	-	-	Unconstrained	Mode of the rectangle detection: [Unconstrained, PackedGridLayout, PackedLayers]
`prefer_splits`	bool	false	true	false	Indicates whether rectangles are split into smaller ones when possible

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's *BoxPick* page. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI:

max_grasps (*Maximum Grasps*)

sets the maximum number of provided grasps.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?max_grasps=<value>
```

cluster_max_curvature (*Cluster Maximum Curvature*)

is the maximum curvature allowed within one cluster. The smaller this value, the more clusters will be split apart.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?cluster_max_curvature=<value>
```

clustering_discontinuity_factor (*Discontinuity Factor*)

is the factor used to discriminate depth discontinuities within a patch. The smaller this value, the more clusters will be split apart.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

clustering_max_surface_rmse (*Maximum Surface RMSE*)

is the maximum root-mean-square error (RMSE) in meters of points belonging to a surface.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

`mode` (*Mode*)

determines the mode of the rectangle detection. Possible values are `Unconstrained`, `PackedGridLayout` and `PackedLayers`. In `PackedGridLayout` mode, rectangles of a cluster are detected in a dense grid pattern. In `PackedLayers` mode, boxes are assumed to form layers and box detection will start searching for items at the cluster corners. Use this mode in de-palletizing applications. In `Unconstrained` mode (default), rectangles are detected without posing any constraints on their relative locations or their positions in the segmented cluster. Fig. 6.13 illustrates the modes for different scenarios.

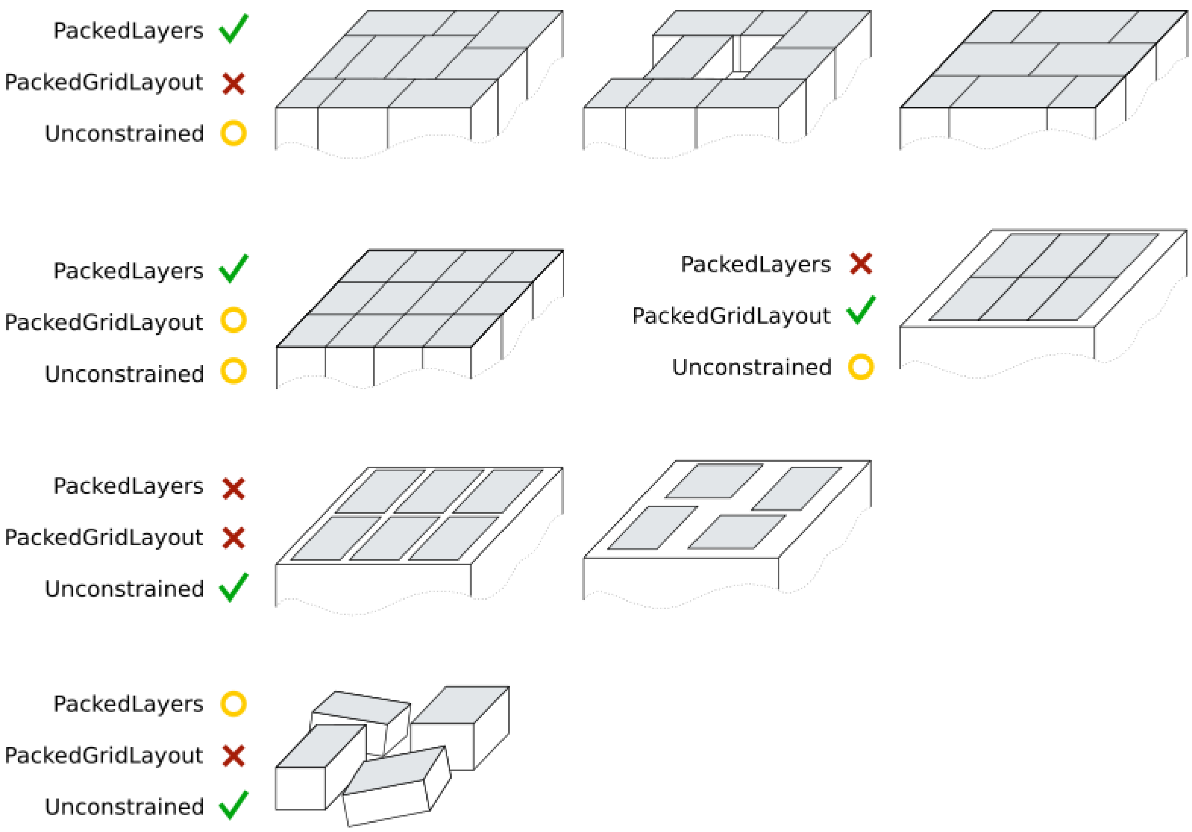

Fig. 6.13: Illustration of appropriate BoxPick modes for different scenes. Modes marked with yellow are applicable but not recommended for the corresponding scene. The gray areas indicate the rectangles to be detected.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?mode=<value>
```

`manual_line_sensitivity` (*Manual Line Sensitivity*)

determines whether the user-defined line sensitivity should be used to extract the lines for rectangle detection. If this parameter is set to true, the user-defined `line_sensitivity` value will be used. If this parameter is set to false, automatic line sensitivity will be used. This parameter should be set to true when automatic line sensitivity does not give enough lines at the box boundaries so that boxes cannot be detected. The detected line segments are visualized in the "Intermediate Result" visualization on the Web GUI's *BoxPick* page.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?manual_line_sensitivity=<value>
```

`line_sensitivity` (*Line Sensitivity*)

determines the line sensitivity for extracting the lines for rectangle detection, if the parameter `manual_line_sensitivity` is set to true. Otherwise, the value of this parameter has no effect on the rectangle detection. Higher values give more line segments, but also increase the runtime of the box detection. This parameter should be increased when boxes cannot be detected because their boundary edges are not detected. The detected line segments are visualized in the "Intermediate Result" visualization on the Web GUI's *BoxPick* page.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?line_sensitivity=<value>
```

`prefer_splits` (*Prefer Splits*)

determines whether rectangles should be split into smaller ones if the smaller ones also match the given item models. This parameter should be set to true for packed box layouts in which the given item models would also match a rectangle of the size of two adjoining boxes. If this parameter is set to false, the larger rectangles will be preferred in these cases.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?prefer_splits=<value>
```

min_cluster_coverage (*Minimum Cluster Coverage*)

determines which ratio of each segmented cluster must be covered with rectangle detections to consider the detections to be valid. If the minimum cluster coverage is not reached for a cluster, no rectangle detections will be returned for this cluster and a warning will be given. This parameter should be used to verify that all items on a layer in a de-palletizing scenario are detected.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?min_cluster_coverage=<value>
```

allow_untextured_detections (Only for BoxPick+Match, *Allow Untextured Detections*)

enables returning all rectangles matching the given template dimensions, even when they cannot be matched to an existing view or when they do not have enough texture to create a new view from them. This parameter is only used when item models of type TEXTURED_BOX are detected. Disabling this parameter leads to faster detections when used with a template for which the automatic view updating is disabled.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

grasp_filter_orientation_threshold (*Grasp Orientation Threshold*)

is the maximum deviation of the TCP's z axis at the grasp point from the z axis of the TCP's preferred orientation in degrees. Only grasp points which are within this threshold are returned. When set to zero, any deviations are valid.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?grasp_filter_orientation_
↪threshold=<value>
```

allow_any_grasp_z_rotation (*Allow Any Grasp Z Rotation*)

If set to true, the returned grasps are no longer forced to have their x axes aligned with the maximum elongation of the graspable ellipse, but can have any rotation around the z axis. The returned max_suction_surface_length and max_suction_surface_width will be equal and correspond to the shortest diameter of the largest graspable ellipse. This parameter enables the robot to get more options for grasping objects, especially in scenes where collisions can occur. However, since the grasp is no longer aligned with the graspable ellipse, the correct orientation for placing the object must be determined from the corresponding item's pose.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/parameters?allow_any_grasp_z_rotation=<value>
```

check_collisions_with_point_cloud (*Check Collisions with Point Cloud*)

This parameter is only used when collision checking is enabled by passing a gripper to the compute_grasps service call. If check_collisions_with_point_cloud is set to true, all grasp points will be checked for collisions between the gripper and a watertight version of the point cloud, and only grasp points at which the gripper would not collide with this point cloud will be returned.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

6.3.5.7 Status values

The rc_boxpick module reports the following status values:

Table 6.35: The `rc_boxpick` module's status values

Name	Description
`data_acquisition_time`	Time in seconds required by the last active service to acquire images
`grasp_computation_time`	Processing time of the last grasp computation in seconds
`last_timestamp_processed`	The timestamp of the last processed dataset
`load_carrier_detection_time`	Processing time of the last load carrier detection in seconds
`processing_time`	Processing time of the last detection (including load carrier detection) in seconds
`state`	The current state of the rc_boxpick node

The reported `state` can take one of the following values.

Table 6.36: Possible states of the BoxPick module

State name	Description
IDLE	The module is idle.
RUNNING	The module is running and ready for load carrier detection and grasp computation.
FATAL	A fatal error has occurred.

6.3.5.8 Services

The user can explore and call the `rc_boxpick` module's services, e.g. for development and testing, using the *REST-API interface* (Section 7.2) or the *rc_reason_stack Web GUI* (Section 7.1).

The BoxPick module offers the following services.

`detect_items`

Triggers the detection of rectangles as described in *Detection of items* (Section 6.3.5.2).

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/detect_items
```

Request

Required arguments:

`pose_frame`: see *Hand-eye calibration* (Section 6.3.5.5).

`item_models`: list of item models to be detected. The type of the item model must be `RECTANGLE` or `TEXTURED_BOX`. For type `RECTANGLE`, `rectangle` must be filled, while for `TEXTURED_BOX`, `textured_box` must be filled. See *Detection of items* (Section 6.3.5.2) for a detailed description of the item model types.

Potentially required arguments:

`robot_pose`: see *Hand-eye calibration* (Section 6.3.5.5).

Optional arguments:

`load_carrier_id`: ID of the load carrier which contains the items to be detected.

`load_carrier_compartment`: compartment inside the load carrier where to detect items (see *Load carrier compartments*, Section 6.5.1.3).

`region_of_interest_id`: if `load_carrier_id` is set, ID of the 3D region of interest where to search for the load carriers. Otherwise, ID of the 3D region of interest where to search for the items.

The definition for the request arguments with corresponding datatypes is:

```
{
  "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",
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      }
    },
    "load_carrier_id": "string",
    "pose_frame": "string",
    "region_of_interest_id": "string",
    "robot_pose": {
      "orientation": {
```

(continues on next page)

(continued from previous page)

```
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

Response

`load_carriers`: list of detected load carriers.

`items`: list of detected rectangles.

`timestamp`: timestamp of the image set the detection ran on.

`return_code`: holds possible warnings or error codes and messages.

The definition for the response with corresponding datatypes is:

```
{
  "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"
        },
        "template_id": "string",
        "timestamp": {
          "nsec": "int32",
          "sec": "int32"
        },
        "type": "string",
        "uuid": "string",
        "view_name": "string",
        "view_pose_set": "bool",
        "view_uuid": "string"
      }
    ],
    "load_carriers": [
      {
```

(continues on next page)

(continued from previous page)


```
        "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"
    }
  }
}
```


compute_grasps

Triggers the detection of rectangles and the computation of grasping poses for the detected rectangles as described in *Computation of grasps* (Section 6.3.5.3).

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/compute_grasps
```

Request

Required arguments:

> `pose_frame`: see *Hand-eye calibration* (Section 6.3.5.5).
>
> `item_models`: list of item models to be detected. The type of the item model must be `RECTANGLE` or `TEXTURED_BOX`. For type `RECTANGLE`, `rectangle` must be filled, while for `TEXTURED_BOX`, `textured_box` must be filled. See *Detection of items* (Section 6.3.5.2) for a detailed description of the item model types.
>
> `suction_surface_length`: length of the suction device grasping surface.
>
> `suction_surface_width`: width of the suction device grasping surface.

Potentially required arguments:

> `robot_pose`: see *Hand-eye calibration* (Section 6.3.5.5).

Optional arguments:

> `load_carrier_id`: ID of the load carrier which contains the items to be grasped.
>
> `load_carrier_compartment`: compartment inside the load carrier where to compute grasps (see *Load carrier compartments*, Section 6.5.1.3).
>
> `region_of_interest_id`: if `load_carrier_id` is set, ID of the 3D region of interest where to search for the load carriers. Otherwise, ID of the 3D region of interest where to compute grasps.
>
> `collision_detection`: see *Collision checking within other modules* (Section 6.4.2.2).

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "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",
```

(continues on next page)

(continued from previous page)

```
            "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`: list of detected load carriers.

`grasps`: sorted list of suction grasps.

`items`: list of detected rectangles corresponding to the returned grasps.

`timestamp`: timestamp of the image set the detection ran on.

`return_code`: holds possible warnings or error codes and messages.

The definition for the response with corresponding datatypes is:

```
{
  "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": {
            "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",
```


(continues on next page)

(continued from previous page)

```
        "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"
    }
  }
}
```

set_preferred_orientation

Persistently stores the preferred orientation of the TCP to compute the reachability of the grasps, which is used for filtering and the grasps returned by the `compute_grasps` service (see *Setting the preferred orientation of the TCP*, Section 6.3.5.4).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/set_preferred_
↪orientation
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/set_preferred_orientation
```

Request

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

get_preferred_orientation

Returns the preferred orientation of the TCP to compute the reachability of the grasps, which is used for filtering the grasps returned by the `compute_grasps` service (see *Setting the preferred orientation of the TCP*, Section 6.3.5.4).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/get_preferred_
↪orientation
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/get_preferred_orientation
```

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

```
{
  "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

Persistently stores the sorting strategy for sorting the grasps returned by the `compute_grasps` service (see *Computation of grasps*, Section 6.3.5.3).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/set_sorting_
↪strategies
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/set_sorting_strategies
```

Request

Only one strategy may have a `weight` greater than 0. If all `weight` values are set to 0, the module will use the default sorting strategy.

If the weight for `direction` is set, the `vector` must contain the direction vector and `pose_frame` must be either `camera` or `external`.

If the weight for `distance_to_point` is set, `point` must contain the sorting point and `pose_frame` must be either `camera` or `external`.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
```

(continues on next page)

(continued from previous page)

```
      },
      "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

The definition for the response with corresponding datatypes is:

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

get_sorting_strategies

Returns the sorting strategy for sorting the grasps returned by the `compute-grasps` service (see *Computation of grasps*, Section 6.3.5.3).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_boxpick/services/get_sorting_
↪strategies
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/get_sorting_strategies
```

Request

This service has no arguments.

Response

All `weight` values are 0 when the module uses the default sorting strategy.

The definition for the response with corresponding datatypes is:

```
{
  "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

Starts the module. If the command is accepted, the module moves to state `RUNNING`.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/start
```

Request

This service has no arguments.

Response

The `current_state` value in the service response may differ from `RUNNING` if the state transition is still in process when the service returns.

The definition for the response with corresponding datatypes is:

```
{
  "name": "start",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

Stops the module. If the command is accepted, the module moves to state `IDLE`.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_boxpick/services/stop
```

Request

This service has no arguments.

Response

The `current_state` value in the service response may differ from `IDLE` if the state transition is still in process when the service returns.

The definition for the response with corresponding datatypes is:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

reset_defaults

Resets all parameters of the module to its default values, as listed in above table. Also resets sorting strategies.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.3.5.9 Return codes

Each service response contains a `return_code`, which consists of a `value` plus an optional `message`. A successful service returns with a `return_code` value of `0`. Negative `return_code` values indicate that the service failed. Positive `return_code` values indicate that the service succeeded with additional information. The smaller value is selected in case a service has multiple `return_code` values, but all messages are appended in the `return_code` message.

The following table contains a list of common codes:

Table 6.37: Return codes of the BoxPick services

Code	Description
0	Success
-1	An invalid argument was provided
-3	An internal timeout occurred, e.g. during box detection if the given dimension range is too large
-4	Data acquisition took longer than allowed
-8	The template has been deleted during detection.
-10	New element could not be added as the maximum storage capacity of load carriers, regions of interest or template has been exceeded
-11	Sensor not connected, not supported or not ready
-200	Fatal internal error
-301	More than one item model provided to the `compute_grasps` service
10	The maximum storage capacity of load carriers, regions of interest or templates has been reached
11	An existent persistent model was overwritten by the call to `set_load_carrier` or `set_region_of_interest`
100	The requested load carriers were not detected in the scene
101	No valid surfaces or grasps were found in the scene
102	The detected load carrier is empty
103	All computed grasps are in collision
112	Rejected detections of one or more clusters, because min_cluster_coverage was not reached.
300	A valid `robot_pose` was provided as argument but it is not required
999	Additional hints for application development

6.3.5.10 BoxPick Template API

BoxPick templates are only available with the +Match extension of BoxPick. For template upload, download, listing and removal, special REST-API endpoints are provided. Templates can also be uploaded, downloaded and removed via the Web GUI. The templates include the dimensions, the views and their poses, if set. Up to 100 templates can be stored persistently on the *rc_reason_stack*.

GET /templates/rc_boxpick

Get list of all rc_boxpick templates.

Template request

```
GET /api/v2/templates/rc_boxpick HTTP/1.1
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "string"
  }
]
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns array of Template)*
- 404 Not Found – node not found

Referenced Data Models

- *Template* (Section 7.2.3)

GET /templates/rc_boxpick/{id}

Get a rc_boxpick template. If the requested content-type is application/octet-stream, the template is returned as file.

Template request

```
GET /api/v2/templates/rc_boxpick/<id> HTTP/1.1
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameters

- **id** (*string*) – id of the template *(required)*

Response Headers

- Content-Type – application/json application/ubjson application/octet-stream

Status Codes

- 200 OK – successful operation *(returns Template)*
- 404 Not Found – node or template not found

Referenced Data Models

- *Template* (Section 7.2.3)

PUT /templates/rc_boxpick/{id}

Create or update a rc_boxpick template.

Template request

```
PUT /api/v2/templates/rc_boxpick/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameters

- **id** (*string*) – id of the template *(required)*

Form Parameters

- **file** – template file *(required)*

Request Headers

- Accept – multipart/form-data application/json

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns Template)*
- 400 Bad Request – Template is not valid or max number of templates reached
- 403 Forbidden – forbidden, e.g. because there is no valid license for this module.
- 404 Not Found – node or template not found
- 413 Request Entity Too Large – Template too large

Referenced Data Models

- *Template* (Section 7.2.3)

DELETE /templates/rc_boxpick/{id}

Remove a rc_boxpick template.

Template request

```
DELETE /api/v2/templates/rc_boxpick/<id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameters

- **id** (*string*) – id of the template *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation
- 403 Forbidden – forbidden, e.g. because there is no valid license for this module.
- 404 Not Found – node or template not found

6.3.6 SilhouetteMatch

6.3.6.1 Introduction

The SilhouetteMatch module is an optional on-board module of the *rc_reason_stack* and requires a separate SilhouetteMatch *license* (Section 8.2) to be purchased.

Note: This module is not available in camera pipelines of type `zivid` or `orbbec`.

The module detects objects by matching a predefined silhouette ("template") to edges in the image.

The SilhouetteMatch module can detect objects in two different scenarios:

> **With calibrated base plane**: The objects are placed on a common base plane, which must be calibrated before the detection, and the objects have significant edges on a common plane that is parallel to the base plane.
>
> **With object plane detection**: The objects can be placed at different, previously unknown heights, if the objects have a planar surface and their outer contours are well visible in the images (e.g. stacked flat objects).

Templates for object detection can be created by uploading a DXF file and specifying the object height. The correct scale and unit of the contours are extracted from the DXF file. If no units are present in the DXF file, the user has to specify which units should be used. When the outer contour of the object in the DXF file is closed, a 3D collision model is created automatically by extruding the contour by the given object height. This model will then be used for collision checking and in 3D visualizations. Uploading a DXF file can be done in the Web GUI via the *+ Create a new Template* button in the *SilhouetteMatch Templates and Grasps* section on the *Modules* → *SilhouetteMatch* or *Database* → *Templates* pages.

Roboception also offers a template generation service on their website (https://roboception.com/en/template-request/), where the user can upload CAD files or recorded data of the objects and request object templates for the SilhouetteMatch module.

The object templates consist of significant edges of each object. These template edges are matched to the edges detected in the left and right camera images, considering the actual size of the objects and their distance from the camera. The poses of the detected objects are returned and can be used for grasping, for example.

The SilhouetteMatch module offers:

- A dedicated page on the *rc_reason_stack* *Web GUI* (Section 7.1) for easy setup, configuration, testing, and application tuning.
- A *REST-API interface* (Section 7.2) and a *KUKA Ethernet KRL Interface* (Section 7.5).
- The definition of 2D regions of interest to select relevant parts of the camera image (see *Setting a region of interest*, Section 6.3.6.3).
- A load carrier detection functionality for bin-picking applications (see *LoadCarrier*, Section 6.3.2), to provide grasps for objects inside a bin only.
- Storing of up to 50 templates.

- The definition of up to 50 grasp points for each template via an interactive visualization in the Web GUI.
- Collision checking between the gripper and the load carrier, the calibrated base plane, other detected objects and/or the point cloud.
- Support for static and robot-mounted cameras and optional integration with the *Hand-eye calibration* (Section 6.4.1) module, to provide grasps in the user-configured external reference frame.
- Selection of a sorting strategy to sort the detected objects and returned grasps.
- 3D visualization of the detection results with grasp points and gripper animations in the Web GUI.

Note: This module is pipeline specific. Changes to its settings or parameters only affect the respective camera pipeline and have no influence on other pipelines running on the *rc_reason_stack*. However, the object templates and grasp points are stored globally. Setting, changing or deleting an object template or its grasps affects all camera pipelines.

Suitable objects

The SilhouetteMatch module is intended for objects which have significant edges on a common plane that is parallel to the plane on which the objects are placed. This applies to flat, nontransparent objects, such as routed, laser-cut or water-cut 2D parts and flat-machined parts. More complex parts can also be detected if there are significant edges on a common plane, e.g. a special pattern printed on a flat surface. The detection works best for objects on a texture-free plane. The color of the base plane should be chosen such that a clear contrast between the objects and the base plane appears in the intensity image.

In case the objects are not placed on a common base plane or the base plane cannot be calibrated beforehand, the objects need to have a planar surface and their outer contour must be well visible in the left and right images. Furthermore, the template for these objects must have a closed outer contour.

Suitable scene

The scene must meet the following conditions to be suitable for the SilhouetteMatch module:

- The objects to be detected must be suitable for the SilhouetteMatch module as described above.
- Only objects belonging to one specific template are visible at a time (unmixed scenario). In case other objects are visible as well, a proper region of interest (ROI) must be set.
- In case a calibrated base plane is used: The offset between the base plane normal and the camera's line of sight does not exceed 10 degrees.
- In case the object planes are detected automatically: The offset between the object's planar surface normal and the camera's line of sight does not exceed 25 degrees.
- The objects are not partially or fully occluded.
- All visible objects are right side up (no flipped objects).
- The object edges to be matched are visible in both, left and right camera images.

6.3.6.2 Base-plane calibration

In case all objects are placed on a common plane that is known beforehand, a base-plane calibration should be performed before triggering a detection. Thereby, the distance and angle of the plane on which the objects are placed is measured and stored persistently on the *rc_reason_stack*.

Separating the detection of the base plane from the actual object detection renders scenarios possible in which the base plane is temporarily occluded. Moreover, it increases performance of the object detection for scenarios where the base plane is fixed for a certain time; thus, it is not necessary to continuously re-detect the base plane.

The base-plane calibration can be performed in three different ways, which will be explained in more detail further down:

- AprilTag based
- Stereo based
- Manual

The base-plane calibration is successful if the normal vector of the estimated base plane is at most 10 degrees offset to the camera's line of sight. If the base-plane calibration is successful, it will be stored persistently on the *rc_reason_stack* until it is removed or a new base-plane calibration is performed.

Note: To avoid privacy issues, the image of the persistently stored base-plane calibration will appear blurred after rebooting the *rc_reason_stack*.

In scenarios where the base plane is not accessible for calibration, a plane parallel to the base plane can be calibrated. Then an `offset` parameter can be used to shift the estimated plane onto the actual base plane where the objects are placed. The `offset` parameter gives the distance in meters by which the estimated plane is shifted towards the camera.

In the REST-API, a plane is defined by a `normal` and a `distance`. `normal` is a normalized 3-vector, specifying the normal of the plane. The normal points away from the camera. `distance` represents the distance of the plane from the camera along the normal. Normal and distance can also be interpreted as a, b, c, and d components of the plane equation, respectively:

$$ax + by + cz + d = 0$$

AprilTag based base-plane calibration

AprilTag detection (ref. *TagDetect*, Section 6.3.3) is used to find AprilTags in the scene and fit a plane through them. At least three AprilTags must be placed on the base plane so that they are visible in the left and right camera images. The tags should be placed such that they are spanning a triangle that is as large as possible. The larger the triangle, the more accurate is the resulting base-plane estimate. Use this method if the base plane is untextured and no external random dot projector is available. This calibration mode is available via the *REST-API interface* (Section 7.2) and the *rc_reason_stack* Web GUI.

Stereo based base-plane calibration

The 3D point cloud computed by the stereo matching module is used to fit a plane through its 3D points. Therefore, the region of interest (ROI) for this method must be set such that only the relevant base plane is included. The `plane_preference` parameter allows to select whether the plane closest to or farthest from the camera should be used as base plane. Selecting the closest plane can be used in scenarios where the base plane is completely occluded by objects or not accessible for calibration. Use this method if the base plane is well textured or you can make use of a random dot projector to project texture on the base plane. This calibration mode is available via the *REST-API interface* (Section 7.2) and the *rc_reason_stack* Web GUI.

Manual base-plane calibration

The base plane can be set manually if its parameters are known, e.g. from previous calibrations. This calibration mode is only available via the *REST-API interface* (Section 7.2) and not the *rc_reason_stack* Web GUI.

6.3.6.3 Setting a region of interest

If objects are to be detected only in part of the camera's field of view, a 2D region of interest (ROI) can be set accordingly as described in *Region of interest* (Section 6.5.2.2).

6.3.6.4 Setting of grasp points

To use SilhouetteMatch directly in a robot application, up to 50 grasp points can be defined for each template. A grasp point represents the desired position and orientation of the robot's TCP (Tool Center Point) to grasp an object as shown in Fig. 6.14.

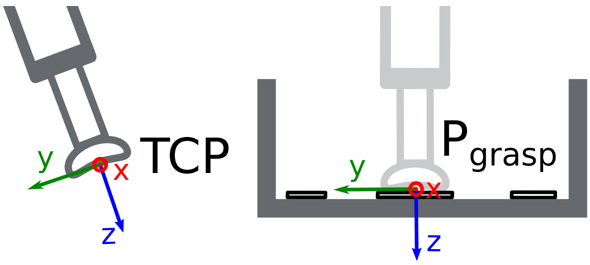

Fig. 6.14: Definition of grasp points with respect to the robot's TCP

Each grasp consists of an `id` which must be unique within all grasps for an object template, the `template_id` representing the template to which the grasp should be attached, and the `pose` in the coordinate frame of the object template. Grasp points can be set via the *REST-API interface* (Section 7.2), or by using the interactive visualization in the Web GUI. Furthermore, a `priority` (spanning -2 for very low to 2 for very high) can be assigned to a grasp. Priorities can facilitate robot applications and shorten response times when the run-time parameter `only_highest_priority_grasps` is set to true. In this case collision checking concludes when grasps with the highest possible priority have been found. Finally, different grasps can be associated with different grippers by specifying a `gripper_id`. These individual grippers are then used for collision checking of the corresponding grasps instead of the gripper defined in the `detect_object` request. If no `gripper_id` is given, the gripper defined in the `detect_object` request will be used for collision checking.

When a grasp is defined on a symmetric object, all grasps symmetric to the defined one will automatically be considered in the SilhouetteMatch module's `detect_object` service call. Symmetric grasps for a given grasp point can be retrieved using the `get_symmetric_grasps` service call and visualized in the Web GUI.

Users can also define replications of grasps around a custom axis. These replications spawn multiple grasps and free users from setting too many grasps manually. The replication origin is defined as a coordinate frame in the object's coordinate frame and the x axis of the replication origin frame corresponds to the replication axis. The grasp is replicated by rotating it around this x axis starting from its original pose. The replication is done in steps of size `step_x_deg` degrees. The range is defined by the minimal and maximal boundaries `min_x_deg` and `max_x_deg`. The minimal (maximal) boundary must be a non-positive (non-negative) number up to (minus) 180 degrees.

Setting grasp points in the Web GUI

The *rc_reason_stack* Web GUI provides an intuitive and interactive way of defining grasp points for object templates. In a first step, the object template has to be uploaded to the *rc_reason_stack*. This can be done in the Web GUI in any pipeline under *Modules* → *SilhouetteMatch* by clicking on *+ Add a new Template* in the *Templates and Grasps* section, or in *Database* → *Templates* in the *SilhouetteMatch Templates and Grasps* section. Once the template upload is complete, a dialog with a 3D visualization of the object template is shown for adding or editing grasp points. The same dialog appears when editing an existing template. If the template contains a collision model or a visualization model, this 3D model is visualized as well.

This dialog provides two ways for setting grasp points:

1. **Adding grasps manually**: By clicking on the + symbol, a new grasp is placed in the object origin. The grasp can be given a unique name which corresponds to its ID. The desired pose of the grasp can be entered in the fields for *Position* and *Roll/Pitch/Yaw* which are given in the coordinate frame of the object template represented by the long x, y and z axes in the visualization. The grasp point can be placed freely with respect to the object template - inside, outside or on the surface. The grasp point and its orientation are visualized in 3D for verification.
2. **Adding grasps interactively**: Grasp points can be added interactively by first clicking on the *Add Grasp* button in the upper right corner of the visualization and then clicking on the desired point on the object template visualization. If the 3D model is displayed, the grasps will be attached to the surface of the 3D model. Otherwise, the grasp is attached to the template surface. The grasp orientation is a right-handed coordinate system and is chosen such that its z axis is perpendicular to the surface pointing inside the template at the grasp position. The position and orientation in the object coordinate frame is displayed on the right. The position and orientation of the grasp can also be changed interactively. In case *Snap to surface* is disabled (default), the grasp can be translated and rotated freely in all three dimensions by clicking on *Move Grasp* in the visualization menu and then dragging the grasp along the appropriate axis to the desired position. The orientation of the grasp can also be changed by rotating the axis with the mouse. In case *Snap to surface* is enabled in the visualization, the grasp can only be moved along the model surface.

Users can also specify a grasp priority by changing the *Priority* slider. A dedicated gripper can be selected in the *Gripper* drop down field.

By activating the *Replication* check box, users can replicate the grasp around a custom axis. The replication axis and the resulting replicated grasps are visualized. The position and orientation of the replication axis relative to the object coordinate frame can be adjusted interactively by clicking on *Move Replication Axis* in the visualization menu and then dragging the axis to the desired position and orientation. The grasps are replicated within the specified rotation range at the selected rotation step size. Users can cycle through a visualization of the replicated grasps by dragging the bar below *Cycle through n replicated grasps* in the *View Options* section of the visualization menu. If a gripper is selected for the grasp or a gripper has been chosen in the visualization menu, the gripper is also shown at the currently selected grasp.

If the object template has symmetries, the grasps which are symmetric to the defined grasps can be displayed along with their replications (if defined) by enabling *... symmetries* in the *View Options* section of the visualization menu. The user can also cycle through a visualization of the symmetric grasps by dragging the bar below *Cycle through n symmetric grasps*.

Setting grasp points via the REST-API

Grasp points can be set via the *REST-API interface* (Section 7.2) using the `set_grasp` or `set_all_grasps` services (see *Internal services*, Section 6.3.6.12). A grasp consists of the `template_id` of the template to which the grasp should be attached, an `id` uniquely identifying the grasp point and the `pose`. The pose is given in the coordinate frame of the object template and consists of a `position` in meters and an `orientation` as quaternion. A dedicated gripper can be specified through setting the `gripper_id` field. The `priority` is specified by an integer value, ranging from -2 for very low, to 2 for very high with a step size of 1. The replication origin is defined as a transformation in the object's coordinate frame and the x axis of the transformation corresponds to the replication axis. The replication range is controlled by the `min_x_deg` and `max_x_deg` fields and the step size `step_x_deg`.

6.3.6.5 Setting the preferred orientation of the TCP

The SilhouetteMatch module determines the reachability of grasp points based on the *preferred orientation* of the TCP. The preferred orientation can be set via the `set_preferred_orientation` service or on the *SilhouetteMatch* page in the Web GUI. The resulting direction of the TCP's z axis is used to reject grasps which cannot be reached by the gripper. Furthermore, the preferred orientation can be used to sort the reachable grasps by setting the corresponding sorting strategy.

The preferred orientation can be set in the camera coordinate frame or in the external coordinate frame, in case a hand-eye calibration is available. If the preferred orientation is specified in the external coordinate frame and the sensor is robot mounted, the current robot pose has to be given to each object detection call. If no preferred orientation is set, the orientation of the left camera (see Coordinate frames in the rc_visard manual) will be used as the preferred orientation of the TCP.

6.3.6.6 Setting the sorting strategies

The objects and grasps returned by the `detect_object` service call are sorted according to a sorting strategy which can be chosen by the user. The following sorting strategies are available and can be set in the *Web GUI* (Section 7.1) or using the `set_sorting_strategies` service call:

- `preferred_orientation`: matches and grasp points with minimal rotation difference of a chosen `axis` (or all axes, when `axis` is empty) with respect to the preferred TCP orientation are returned first,
- `direction`: objects and grasp points with the shortest distance along a defined direction `vector` in a given `pose_frame` are returned first.
- `distance_to_point`: objects and grasp points with the shortest or farthest (if `farthest_first` is true) distance from a `point` in a given `pose_frame` are returned first.

If no sorting strategy is set or default sorting is chosen in the Web GUI, sorting is done based on a combination of `preferred_orientation` and the minimal distance from the camera along the z axis of the preferred orientation of the TCP.

6.3.6.7 Detection of objects

For triggering the object detection, in general, the following information must be provided to the SilhouetteMatch module:

- The template of the object to be detected in the scene.
- The coordinate frame in which the poses of the detected objects shall be returned (ref. *Hand-eye calibration*, Section 6.3.6.8).

Optionally, further information can be given to the SilhouetteMatch module:

- A flag `object_plane_detection` determining whether the surface plane of the objects should be used for the detection instead of the calibrated base plane.
- An offset, in case the calibrated base plane should be used but the objects are not lying on this plane but on a plane parallel to it. The offset is the distance between both planes given in the direction towards the camera. If omitted, an offset of 0 is assumed. It must not be set if `object_plane_detection` is true.
- The ID of the load carrier which contains the objects to be detected.
- The ID of the region of interest where to search for the load carrier if a load carrier is set. Otherwise, the ID of the region of interest where the objects should be detected. If omitted, objects are matched in the whole image.
- The current robot pose in case the camera is mounted on the robot and the chosen coordinate frame for the poses is `external` or the preferred orientation is given in the external frame.
- Collision detection information: The ID of the gripper to enable collision checking and optionally a pre-grasp offset to define a pre-grasp position. Details on collision checking are given below in *CollisionCheck* (Section 6.3.6.8).

In case the `object_plane_detection` flag is not true, objects can only be detected after a successful base-plane calibration. It must be ensured that the position and orientation of the base plane does not change before the detection of objects. Otherwise, the base-plane calibration must be renewed.

When `object_plane_detection` is set to true, a base-plane calibration is not required and an existing base-plane calibration will be ignored. During detection, the scene is clustered into planar surfaces and template matching is performed on each plane whose tilt with respect to the camera's line of sight is less than 25° and whose size is large enough to contain the selected template. When a match is found, its position and orientation are refined using the image edges and the point cloud inside the template's outer contour. For this, it is required that the outer contour of the template is closed and that the object's surface is planar.

On the Web GUI the detection can be tested in the *Try Out* section of the SilhouetteMatch page. Different image streams can be selected to show intermediate results and the final matches.

> The **"Template"** image stream shows the template to be matched in green with the defined grasp points in green (see *Setting of grasp points*, Section 6.3.6.4). The template is warped to the size and tilt matching objects on the calibrated base plane or, in case `object_plane_detection` was used, the highest segmented plane, would have. The corresponding plane is shown in dark blue.
>
> The **"Intermediate Result"** image stream shows the edges of the left image that were used to search for matches in light blue. The chosen region of interest is shown as bold petrol rectangle. A shaded blue area on the left visualizes the region of the left camera image which does not overlap with the right image, and in which no objects can be detected. If `object_plane_detection` was used, this image stream also shows the detected planar clusters in the scene. Clusters that were not used for matching, because they were too small or too tilted, are visualized with a stripe pattern.
>
> The **"Intermediate Result Right"** image stream shows the edges of the right image that were used to search for matches in light blue. The chosen region of interest is shown as bold petrol rectangle. A shaded blue area on the right visualizes the region of the right camera image which does not overlap with the left image, and in which no objects can be detected.
>
> The **"Result"** image shows the detection result. The image edges that were used to refine the match poses are shown in light blue and the matches (`instances`) with the template edges are shown in green. The blue circles are the origins of the detected objects as defined in the template and the green circles are the collision-free grasp points. Colliding grasp points are visualized as red dots and grasp points that were not checked for collisions are drawn in yellow.

The poses of the object origins in the chosen coordinate frame are returned as results in a list of `instances`. In case the calibrated base plane was used for the detection (`object_plane_detection` not set or false), the orientations of the detected objects are aligned with the normal of the base plane. Otherwise, the orientations of the detected objects are aligned with the normal of a plane fitted to the object points in the 3D point cloud.

If the chosen template also has grasp points attached, a list of `grasps` for all objects is returned in addition to the list of detected objects. The grasp poses are given in the desired coordinate frame and the grasps are sorted according to the selected sorting strategy (see *Setting the sorting strategies*, Section 6.3.6.6). There are references between the detected object instances and the grasps via their `uuids`.

In case the templates have a continuous rotational symmetry (e.g. cylindrical objects), all returned object poses will have the same orientation. Furthermore, all grasps symmetric to each grasp point on an object are checked for reachability and collisions, and only the best one according to the given sorting strategy is returned.

For objects with a discrete symmetry (e.g. prismatic objects), all collision-free symmetries of each grasp point which are reachable according to the given preferred TCP orientation are returned, ordered by the given sorting strategy.

The detection results and run times are affected by several run-time parameters which are listed and explained further down. Improper parameters can lead to timeouts of the SilhouetteMatch module's detection process.

6.3.6.8 Interaction with other modules

Internally, the SilhouetteMatch module depends on, and interacts with other on-board modules as listed below.

Note: All changes and configuration updates to these modules will affect the performance of the SilhouetteMatch module.

Camera and depth data

The SilhouetteMatch module makes internally use of the rectified images from the *Camera module* (`rc_camera`, Section 6.1). Thus, the exposure time should be set properly to achieve the optimal performance of the module.

For base-plane calibration in stereo mode, for load carrier detection, for automatic object plane detection and for collision checking with the point cloud, the disparity images from the *Stereo matching module* (`rc_stereomatching`, Section 6.2.2) are used.

For detecting objects with a calibrated base plane, without load carrier and without collision checking with the point cloud, the stereo-matching module should not be run in parallel to the SilhouetteMatch module, because the detection runtime increases.

For best results it is recommended to enable *smoothing* (Section 6.2.2.1) for *Stereo matching module*.

IO and Projector Control

In case the *rc_reason_stack* is used in conjunction with an external random dot projector and the *IO and Projector Control* module (`rc_iocontrol`, Section 6.4.4), the projector should be used for the stereo-based base-plane calibration, for automatic object plane detection and for collision checking with the point cloud.

The projected pattern must not be visible in the left and right camera images during object detection as it interferes with the matching process. Therefore, it is recommended to connect the projector to GPIO Out 1 and set the stereo-camera module's acquisition mode to `SingleFrameOut1` (see *Stereo matching parameters*, Section 6.2.2.1), so that on each image acquisition trigger an image with and without projector pattern is acquired.

Alternatively, the output mode for the GPIO output in use should be set to `ExposureAlternateActive` (see *Description of run-time parameters*, Section 6.4.4.1).

In either case, the *Auto Exposure Mode* `exp_auto_mode` should be set to `AdaptiveOut1` to optimize the exposure of both images.

Hand-eye calibration

In case the camera has been calibrated to a robot, the SilhouetteMatch module can automatically provide poses in the robot coordinate frame. For the SilhouetteMatch node's *Services* (Section 6.3.6.11), the frame of the input and output poses and plane coordinates can be controlled with the `pose_frame` argument.

Two different `pose_frame` values can be chosen:

1. **Camera frame** (`camera`). All poses and plane coordinates provided to and by the module are in the camera frame.
2. **External frame** (`external`). All poses and plane coordinates provided to and by the module are in the external frame, configured by the user during the hand-eye calibration process. The module relies on the on-board *Hand-eye calibration module* (Section 6.4.1) to retrieve the camera mounting (static or robot mounted) and the hand-eye transformation. If the sensor mounting is

static, no further information is needed. If the sensor is robot-mounted, the `robot_pose` is required to transform poses to and from the `external` frame.

All `pose_frame` values that are not `camera` or `external` are rejected.

Note: If no hand-eye calibration is available, all `pose_frame` values should be set to `camera`.

Note: If the hand-eye calibration has changed after base-plane calibration, the base-plane calibration will be marked as invalid and must be renewed.

If the sensor is robot-mounted, the current `robot_pose` has to be provided depending on the value of `pose_frame`, the definition of the preferred TCP orientation and the sorting direction or sorting point:

- If `pose_frame` is set to `external`, providing the robot pose is obligatory.
- If the preferred TCP orientation is defined in `external`, providing the robot pose is obligatory.
- If the sorting direction is defined in `external`, providing the robot pose is obligatory.
- If the distance-to-point sorting strategy is defined in `external`, providing the robot pose is obligatory.
- In all other cases, providing the robot pose is optional.

If the current robot pose is provided during calibration, it is stored persistently on the *rc_reason_stack*. If the updated robot pose is later provided during `get_base_plane_calibration` or `detect_object` as well, the base-plane calibration will be transformed automatically to this new robot pose. This enables the user to change the robot pose (and thus camera position) between base-plane calibration and object detection.

Note: Object detection can only be performed if the limit of 10 degrees angle offset between the base plane normal and the camera's line of sight is not exceeded.

LoadCarrier

The SilhouetteMatch module uses the load carrier detection functionality provided by the *LoadCarrier* module (`rc_load_carrier`, Section 6.3.2), with the run-time parameters specified for this module. However, only one load carrier will be returned and used in case multiple matching load carriers could be found in the scene. In case multiple load carriers of the same type are visible, a region of interest should be set to ensure that always the same load carrier is used for the SilhouetteMatch module.

CollisionCheck

Collision checking can be easily enabled for grasp computation of the SilhouetteMatch module by passing a `collision_detection` argument to the `detect_object` service call. It contains the ID of the used gripper and optionally a pre-grasp offset. The gripper has to be defined in the GripperDB module (see *Setting a gripper*, Section 6.5.3.2) and details about collision checking are given in *Collision checking within other modules* (Section 6.4.2.2).

Alternatively, grasp points can be assigned individual gripper IDs, and collision checking can be enabled for all grasp points with gripper IDs by enabling the run-time parameter `check_collisions`.

In addition to collision checking between the gripper and the detected load carrier, collisions between the gripper and the calibrated base plane will be checked, if the run-time parameter `check_collisions_with_base_plane` is true. If the selected SilhouetteMatch template contains a collision model and the run-time parameter `check_collisions_with_matches` is true, also collisions between the gripper and all other detected objects (not limited to `max_number_of_detected_objects`) will be checked. The object on which the grasp point to be checked is located, is excluded from the collision check.

If the run-time parameter check_collisions_with_point_cloud is true, also collisions between the gripper and a watertight version of the point cloud are checked. If this feature is used with suctions grippers, it should be ensured that the TCP is defined to be outside the gripper geometry, or that the grasp points are defined above the object surface. Otherwise every grasp will result in a collision between the gripper and the point cloud.

If the run-time parameter check_collisions_during_retraction is true and a load carrier and a pre-grasp offset are given, each grasp point will be checked for collisions between the object in the gripper and the load carrier walls during retraction. This collision check is performed along the full linear trajectory from the grasp point back to the pre-grasp position.

If collision checking is enabled, only grasps which are collision free or could not be checked for collisions (e.g. because no gripper was given) will be returned. The visualization images on the *SilhouetteMatch* page of the Web GUI shows collision-free grasps in green, unchecked grasps in yellow and colliding grasp points in red. The detected objects which are considered in the collision check are also visualized with their edges in green.

The CollisionCheck module's run-time parameters affect the collision detection as described in *CollisionCheck Parameters* (Section 6.4.2.3).

6.3.6.9 Parameters

The SilhouetteMatch software module is called rc_silhouettematch in the REST-API and is represented in the *Web GUI* (Section 7.1)in the desired pipeline under *Modules* → *SilhouetteMatch*. The user can explore and configure the rc_silhouettematch module's run-time parameters, e.g. for development and testing, using the Web GUI or the *REST-API interface* (Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.38: The rc_silhouettematch module's run-time parameters

Name	Type	Min	Max	Default	Description
check_collisions	bool	false	true	false	Whether to check for collisions when a gripper is defined for a grasp
check_collisions_during_retraction	bool	false	true	false	Whether to check for collisions between the object in the gripper and the load carrier during retraction
check_collisions_with_base_plane	bool	false	true	true	Whether to check for collisions between gripper and base plane
check_collisions_with_matches	bool	false	true	true	Whether to check for collisions between gripper and detected matches
check_collisions_with_point_cloud	bool	false	true	false	Whether to check for collisions between gripper and the point cloud
edge_sensitivity	float64	0.1	1.0	0.7	Sensitivity of the edge detector
match_max_distance	float64	0.1	10.0	3.0	Maximum allowed distance in pixels between the template and the detected edges in the image
match_percentile	float64	0.7	1.0	0.8	Percentage of template pixels that must be within the maximum distance to successfully match the template
max_number_of_detected_objects	int32	1	20	10	Maximum number of detected objects
only_highest_priority_grasps	bool	false	true	false	Whether to return only the highest priority level grasps
point_cloud_enhancement	string	-	-	Off	Type of enhancement of the point cloud using the base plane: [Off, ReplaceBright]
quality	string	-	-	High	Quality: [Low, Medium, High]

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's SilhouetteMatch page. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI:

max_number_of_detected_objects (*Maximum Object Number*)

This parameter gives the maximum number of objects to detect in the scene. If more than the given number of objects can be detected in the scene, only the objects matching best to the given sorting strategy are returned.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?max_number_of_detected_
↪objects=<value>
```

quality (*Quality*)

Object detection can be performed on images with different resolutions: `High` (full image resolution), `Medium` (half image resolution) and `Low` (quarter image resolution). The lower the resolution, the lower the detection time, but the fewer details of the objects are visible.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

match_max_distance (*Maximum Matching Distance*)

This parameter gives the maximum allowed pixel distance of an image edge pixel from the object edge pixel in the template to be still considered as matching. If the object is not perfectly represented in the template, it might not be detected when this parameter is low. High values, however, might lead to false detections in case of a cluttered scene or the presence of similar objects, and also increase runtime.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?match_max_distance=<value>
```

match_percentile (*Matching Percentile*)

This parameter indicates how strict the matching process should be. The matching percentile is the ratio of template pixels that must be within the Maximum Matching Distance to successfully match the template. The higher this number, the more accurate the match must be to be considered as valid.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?match_percentile=<value>
```

edge_sensitivity (*Edge Sensitivity*)

This parameter influences how many edges are detected in the left and right camera images. The higher this number, the more edges are found in the intensity images. That means, for lower numbers, only the most significant edges are considered for template matching. A large number of edges in the image might increase the detection time. It must be ensured that the edges of the objects to be detected are detected in both, the left and the right camera images.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?edge_sensitivity=<value>
```

`only_highest_priority_grasps` (*Only Highest Priority Grasps*)

If set to true, only grasps with the highest priority will be returned. If collision checking is enabled, only the collision-free grasps among the group of grasps with the highest priority are returned. This can save computation time and reduce the number of grasps to be parsed on the application side.

Without collision checking, only grasps of highest priority are returned.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?only_highest_priority_
↪grasps=<value>
```

`check_collisions` (*Check Collisions*)

If this parameter is enabled, collision checking will be performed for all grasps which have a gripper ID assigned, even when no default gripper is given in the `detect_object` service call. If a load carrier is used, the collision check will always be performed between the gripper and the load carrier. Collision checking with the point cloud and other matches is only performed when the corresponding runtime parameters are enabled.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions=<value>
```

`check_collisions_with_base_plane` (*Check Collisions with Base Plane*)

This parameter is only used when collision checking is enabled by passing a gripper to the `detect_object` service call or by enabling the `check_collisions` run-time parameter. If check_collisions_with_base_plane is set to true, all grasp points will be checked for collisions between the gripper and the calibrated base plane, and only grasp points at which the gripper would not collide with the base plane will be returned.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_with_
↪base_plane=<value>
```

check_collisions_with_matches (*Check Collisions with Matches*)

This parameter is only used when collision checking is enabled by passing a gripper to the detect_object service call or by enabling the check_collisions runtime parameter. If check_collisions_with_matches is set to true, all grasp points will be checked for collisions between the gripper and all other detected objects (not limited to max_number_of_detected_objects), and only grasp points at which the gripper would not collide with any other detected object will be returned.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_with_
↪matches=<value>
```

check_collisions_with_point_cloud (*Check Collisions with Point Cloud*)

This parameter is only used when collision checking is enabled by passing a gripper to the detect_object service call or by enabling the check_collisions runtime parameter. If check_collisions_with_point_cloud set to true, all grasp points will be checked for collisions between the gripper and a watertight version of the point cloud, and only grasp points at which the gripper would not collide with this point cloud will be returned.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_with_
↪point_cloud=<value>
```

`point_cloud_enhancement` (*Enhance with Base Plane*)

This parameter is only considered when `check_collisions_with_point_cloud` is true and the object detection was triggered without `object_plane_detection`. By default, `point_cloud_enhancement` is set to `Off` (*Off*). If `point_cloud_enhancement` is set to `ReplaceBright` (*Replace Bright Image Pixels*), the calibrated base plane will be used to enhance the point cloud that is used for collision checking. For this, the depth values of all bright image pixels inside the image or, if set, the 2D region of interest will be set to the depth of the calibrated base plane. This parameter should be used when dark objects are placed on an untextured bright background, e.g. on a light table.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

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

`check_collisions_during_retraction` (*Check Collisions during Retraction*)

This parameter is only used when collision checking is enabled by passing a gripper to the `detect_object` service call or by enabling the `check_collisions` runtime parameter. When `check_collisions_during_retraction` is enabled and a load carrier and a pre-grasp offset are given, each grasp point will be checked for collisions between the object in the gripper and the load carrier walls during retraction. This collision checking is performed along the full linear trajectory from the grasp point back to the pre-grasp position. Only collision-free grasp points will be returned.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/parameters?check_collisions_during_
↪retraction=<value>
```

6.3.6.10 Status values

This module reports the following status values:

Table 6.39: The `rc_silhouettematch` module's status values

Name	Description
`data_acquisition_time`	Time in seconds required by the last active service to acquire images
`last_timestamp_processed`	The timestamp of the last processed dataset
`load_carrier_detection_time`	Processing time of the last load carrier detection in seconds
`processing_time`	Processing time of the last detection (including load carrier detection) in seconds

6.3.6.11 Services

The user can explore and call the `rc_silhouettematch` module's services, e.g. for development and testing, using the *REST-API interface* (Section 7.2) or the *rc_reason_stack Web GUI* (Section 7.1).

The SilhouetteMatch module offers the following services.

`detect_object`

Triggers an object detection as described in *Detection of objects* (Section 6.3.6.7) and returns the pose of all found object instances.

Details

All images used by the service are guaranteed to be newer than the service trigger time.

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/detect_object
```

Request

Required arguments:

`object_id` in `object_to_detect`: ID of the template which should be detected.

`pose_frame`: see *Hand-eye calibration* (Section 6.3.6.8).

Potentially required arguments:

`robot_pose`: see *Hand-eye calibration* (Section 6.3.6.8).

Optional arguments:

`object_plane_detection`: false if the objects are placed on a calibrated base plane, true if the objects' surfaces are planar and the base plane is unknown or the objects are located on multiple different planes, e.g. stacks.

`offset`: offset in meters by which the base-plane calibration will be shifted towards the camera.

`load_carrier_id`: ID of the load carrier which contains the items to be detected.

`collision_detection`: see *Collision checking within other modules* (Section 6.4.2.2).

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "load_carrier_id": "string",
    "object_plane_detection": "bool",
    "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

The maximum number of returned instances can be controlled with the `max_number_of_detected_objects` parameter.

`object_id`: ID of the detected template.

`instances`: list of detected object instances, ordered according to the chosen sorting strategy.

`grasps`: list of grasps on the detected objects, ordered according to the chosen sorting strategy. The `instance_uuid` gives the reference to the detected object in `instances` this grasp belongs to. The list of returned grasps will be trimmed to the 100 best grasps if more reachable grasps are found. Each grasp contains a flag `collision_checked` and a `gripper_id` (see *Collision checking within other modules*, Section 6.4.2.2).

`load_carriers`: list of detected load carriers.

`timestamp`: timestamp of the image set the detection ran on.

`return_code`: holds possible warnings or error codes and messages.

The definition for the response with corresponding datatypes is:

```
{
  "name": "detect_object",
  "response": {
    "grasps": [
      {
        "collision_checked": "bool",
```

(continues on next page)

(continued from previous page)


```
        "gripper_id": "string",
        "id": "string",
        "instance_uuid": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "priority": "int8",
        "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"
```


(continues on next page)

(continued from previous page)

```
        },
        "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"
      }
    ],
    "object_id": "string",
    "return_code": {
      "message": "string",
      "value": "int16"
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

calibrate_base_plane

Triggers the calibration of the base plane, as described in *Base-plane calibration* (Section 6.3.6.2).

Details

A successful base-plane calibration is stored persistently on the *rc_reason_stack* and returned by this service. The base-plane calibration is persistent over firmware updates and rollbacks.

All images used by the service are guaranteed to be newer than the service trigger time.

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/calibrate_base_plane
```

Request

Required arguments:

> plane_estimation_method: method to use for base-plane calibration. Valid values are STEREO, APRILTAG, MANUAL.
>
> pose_frame: see *Hand-eye calibration* (Section 6.3.6.8).

Potentially required arguments:

> plane if plane_estimation_method is MANUAL: plane that will be set as base-plane calibration.
>
> robot_pose: see *Hand-eye calibration* (Section 6.3.6.8).
>
> region_of_interest_2d_id: ID of the region of interest for base-plane calibration.

Optional arguments:

> offset: offset in meters by which the estimated plane will be shifted towards the camera.
>
> plane_preference in stereo: whether the plane closest to or farthest from the camera should be used as base plane. This option can be set only if plane_estimation_method is STEREO. Valid values are CLOSEST and FARTHEST. If not set, the default is FARTHEST.

The definition for the request arguments with corresponding datatypes is:

```
{
  "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": {
```

(continues on next page)

(continued from previous page)

```
      "plane_preference": "string"
    }
  }
}
```

Response

plane: calibrated base plane.

timestamp: timestamp of the image set the calibration ran on.

return_code: holds possible warnings or error codes and messages.

The definition for the response with corresponding datatypes is:

```
{
  "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

Returns the configured base-plane calibration.

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/get_
↪base_plane_calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_base_plane_calibration
```

Request

Required arguments:

pose_frame: see *Hand-eye calibration* (Section 6.3.6.8).

Potentially required arguments:

robot_pose: see *Hand-eye calibration* (Section 6.3.6.8).

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "pose_frame": "string",
    "robot_pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

Response

The definition for the response with corresponding datatypes is:

```
{
  "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

Deletes the configured base-plane calibration.

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/delete_
↪base_plane_calibration
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/delete_base_plane_
↪calibration
```

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

set_preferred_orientation

Persistently stores the preferred orientation of the TCP to compute the reachability of the grasps, which is used for filtering and, optionally, sorting the grasps returned by the `detect_object` service (see *Setting the preferred orientation of the TCP*, Section 6.3.6.5).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/set_
↪preferred_orientation
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_preferred_orientation
```

Request

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

(continues on next page)

(continued from previous page)

```
    }
  }
}
```

get_preferred_orientation

Returns the preferred orientation of the TCP to compute the reachability of the grasps, which is used for filtering and, optionally, sorting the grasps returned by the `detect_object` service (see *Setting the preferred orientation of the TCP*, Section 6.3.6.5).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/get_
↪preferred_orientation
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_preferred_orientation
```

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

```
{
  "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

Persistently stores the sorting strategy for sorting the grasps and detected objects returned by the `detect_object` service (see *Detection of objects*, Section 6.3.6.7).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/set_
↪sorting_strategies
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_sorting_strategies
```

Request

Only one strategy may have a `weight` greater than 0. If all `weight` values are set to 0, the module will use the default sorting strategy.

If the weight for `direction` is set, the `vector` must contain the direction vector and `pose_frame` must be either `camera` or `external`.

If the weight for `distance_to_point` is set, `point` must contain the sorting point and `pose_frame` must be either `camera` or `external`.

If the weight for `preferred_orientation` is set, the `axis` can be set to `x`, `y` or `z` to consider only rotational differences between the respective axes. If `axis` is empty, the full orientation difference will be used for sorting.

The definition for the request arguments with corresponding datatypes is:

```
{
  "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

The definition for the response with corresponding datatypes is:

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

get_sorting_strategies

Returns the sorting strategy for sorting the grasps and detected objects returned by the detect_object service (see *Detection of objects*, Section 6.3.6.7).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/get_
↪sorting_strategies
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_sorting_strategies
```

Request

This service has no arguments.

Response

All weight values are 0 when the module uses the default sorting strategy.

The definition for the response with corresponding datatypes is:

```
{
  "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"
    }
  }
}
```

reset_defaults

Resets all parameters of the module to its default values, as listed in above table. Also resets preferred orientation and sorting strategies. The reset does not apply to templates and base-plane calibration.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.3.6.12 Internal services

The following services for configuring grasps can change in future without notice. Setting, retrieving and deleting grasps is recommend to be done via the Web GUI.

Note: Configuring grasps is global for all templates on the *rc_reason_stack* and affects all camera pipelines.

set_grasp

Persistently stores a grasp for the given object template on the *rc_reason_stack*. All configured grasps are persistent over firmware updates and rollbacks.

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/set_
↪grasp
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_grasp
```

Request

Details for the definition of the `grasp` type are given in *Setting of grasp points* (Section 6.3.6.4).

The definition for the request arguments with corresponding datatypes is:

```
{
  "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"
      },
      "template_id": "string"
    }
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

(continues on next page)

(continued from previous page)

```
  }
}
```

set_all_grasps

Replaces the list of grasps for the given object template on the *rc_reason_stack*.

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/set_all_
↪grasps
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/set_all_grasps
```

Request

Details for the definition of the `grasp` type are given in *Setting of grasp points* (Section 6.3.6.4).

The definition for the request arguments with corresponding datatypes is:

```
{
  "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"
```

(continues on next page)

(continued from previous page)

```
            }
          },
          "step_x_deg": "float64"
        },
        "template_id": "string"
      }
    ],
    "template_id": "string"
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

get_grasps

Returns all configured grasps which have the requested `grasp_ids` and belong to the requested `template_ids`.

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/get_
↪grasps
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_grasps
```

Request

If no `grasp_ids` are provided, all grasps belonging to the requested `template_ids` are returned. If no `template_ids` are provided, all grasps with the requested `grasp_ids` are returned. If neither IDs are provided, all configured grasps are returned.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

```
{
  "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"
        },
        "template_id": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

delete_grasps

Deletes all grasps with the requested `grasp_ids` that belong to the requested `template_ids`.

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/delete_
↪grasps
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/delete_grasps
```

Request

If no `grasp_ids` are provided, all grasps belonging to the requested `template_ids` are deleted. The `template_ids` list must not be empty.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

get_symmetric_grasps

Returns all grasps that are symmetric to the given grasp.

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_silhouettematch/services/get_
↪symmetric_grasps
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_silhouettematch/services/get_symmetric_grasps
```

Request

Details for the definition of the `grasp` type are given in *Setting of grasp points* (Section 6.3.6.4).

The definition for the request arguments with corresponding datatypes is:

```
{
  "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

The first grasp in the returned list is the one that was passed with the service call. If the object template does not have an exact symmetry, only the grasp passed with the service call will be returned. If the object template has a continuous symmetry (e.g. a cylindrical object), only 12 equally spaced sample grasps will be returned.

Details for the definition of the `grasp` type are given in *Setting of grasp points* (Section 6.3.6.4).

The definition for the response with corresponding datatypes is:

```
{
  "name": "get_symmetric_grasps",
  "response": {
    "grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
```

(continues on next page)

(continued from previous page)


```
          "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"
    }
  }
}
```


6.3.6.13 Return codes

Each service response contains a `return_code`, which consists of a `value` plus an optional `message`. A successful service returns with a `return_code` value of `0`. Negative `return_code` values indicate that the service failed. Positive `return_code` values indicate that the service succeeded with additional information.

Table 6.40: Return codes of the SilhouetteMatch module services

Code	Description
0	Success
-1	An invalid argument was provided.
-3	An internal timeout occurred, e.g. during object detection.
-4	Data acquisition took longer than allowed.
-7	Data could not be read or written to persistent storage.
-8	Module is not in a state in which this service can be called. E.g. `detect_object` cannot be called if there is no base-plane calibration.
-10	New element could not be added as the maximum storage capacity of regions of interest or templates has been exceeded.
-100	An internal error occurred.
-101	Detection of the base plane failed.
-102	The hand-eye calibration changed since the last base-plane calibration.
-104	Offset between the base plane normal and the camera's line of sight exceeds 10 degrees.
10	The maximum storage capacity of regions of interest or templates has been reached.
11	An existing element was overwritten.
100	The requested load carrier was not detected in the scene.
101	None of the detected grasps is reachable.
102	The detected load carrier is empty.
103	All detected grasps are in collision.
107	The base plane was not transformed to the current camera pose, e.g. because no robot pose was provided during base-plane calibration.
108	The template is deprecated.
109	The plane for object detection does not fit to the load carrier, e.g. objects are below the load carrier floor.
111	The detection result's pose could not be refined with the point cloud because the template's outer contour is not closed.
113	No gripper was found for collision checking.
114	Collision checking during retraction was skipped, e.g. because no load carrier or no pre-grasp offset were given.
151	The object template has a continuous symmetry.
999	Additional hints for application development

6.3.6.14 Template API

For template upload, download, listing and removal, special REST-API endpoints are provided. Templates can also be uploaded, downloaded and removed via the Web GUI. The templates include the grasp points, if grasp points have been configured. Up to 50 templates can be stored persistently on the *rc_reason_stack*.

GET /templates/rc_silhouettematch

Get list of all rc_silhouettematch templates.

Template request

```
GET /api/v2/templates/rc_silhouettematch HTTP/1.1
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "string"
```

(continues on next page)

(continued from previous page)

```
  }
]
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns array of Template)*
- 404 Not Found – node not found

Referenced Data Models

- *Template* (Section 7.2.3)

GET /templates/rc_silhouettematch/{id}

Get a rc_silhouettematch template. If the requested content-type is application/octet-stream, the template is returned as file.

Template request

```
GET /api/v2/templates/rc_silhouettematch/<id> HTTP/1.1
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameters

- **id** (*string*) – id of the template *(required)*

Response Headers

- Content-Type – application/json application/ubjson application/octet-stream

Status Codes

- 200 OK – successful operation *(returns Template)*
- 404 Not Found – node or template not found

Referenced Data Models

- *Template* (Section 7.2.3)

PUT /templates/rc_silhouettematch/{id}

Create or update a rc_silhouettematch template.

Template request

```
PUT /api/v2/templates/rc_silhouettematch/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameters

- **id** (*string*) – id of the template *(required)*

Form Parameters

- **file** – template or dxf file *(required)*
- **object_height** – object height in meters, required when uploading dxf *(optional)*
- **units** – Units for dxf file if not embedded in dxf (one of `mm`, `cm`, `m`, `in`, `ft`) *(optional)*

Request Headers

- Accept – multipart/form-data application/json

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns Template)*
- 400 Bad Request – Template is not valid or max number of templates reached
- 403 Forbidden – forbidden, e.g. because there is no valid license for this module.
- 404 Not Found – node or template not found
- 413 Request Entity Too Large – Template too large

Referenced Data Models

- *Template* (Section 7.2.3)

DELETE /templates/rc_silhouettematch/{id}

Remove a rc_silhouettematch template.

Template request

```
DELETE /api/v2/templates/rc_silhouettematch/<id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameters

- **id** (*string*) – id of the template *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation
- 403 Forbidden – forbidden, e.g. because there is no valid license for this module.

- 404 Not Found – node or template not found

6.3.7 CADMatch

6.3.7.1 Introduction

The CADMatch module is an optional module of the *rc_reason_stack* and requires a separate CADMatch *license* (Section 8.2) to be purchased.

This module provides an out-of-the-box perception solution for 3D object detection and grasping. CADMatch targets the detection of 3D objects based on a CAD template for picking with a general gripper. The objects can be located in a bin or placed arbitrarily in the field of view of the camera.

For the CADMatch module to work, special object templates are required for each type of object to be detected. Please get in touch with the Roboception support (*Contact*, Section 10) to order a template for your CAD file.

Note: This module is pipeline specific. Changes to its settings or parameters only affect the respective camera pipeline and have no influence on other pipelines running on the *rc_reason_stack*. However, the object templates, grasp points and pose priors are stored globally. Setting, changing or deleting an object template, its grasps or pose priors affects all camera pipelines.

The CADMatch module offers:

- A dedicated page on the *rc_reason_stack* *Web GUI* (Section 7.1) for easy setup, configuration, testing, and application tuning.
- A *REST-API interface* (Section 7.2) and a *KUKA Ethernet KRL Interface* (Section 7.5).
- The definition of regions of interest to select relevant volumes in the scene (see *RoiDB*, Section 6.5.2).
- A load carrier detection functionality for bin-picking applications (see *LoadCarrier*, Section 6.3.2), to provide grasps for objects inside a bin only.
- The definition of compartments inside a load carrier to provide grasps for specific volumes of the bin only.
- The option to use user-defined object pose priors.
- Storing of up to 50 templates.
- The definition of up to 100 grasp points for each template via an interactive visualization in the Web GUI.
- Collision checking between the gripper and the load carrier, other detected objects and/or the point cloud.
- Collision checking between the object in the gripper and the load carrier walls during retraction.
- Support for static and robot-mounted cameras and optional integration with the *Hand-eye calibration* (Section 6.4.1) module, to provide grasps in the user-configured external reference frame.
- Selection of a sorting strategy to sort the detected objects and returned grasps.
- 3D visualization of the detection results with grasp points and gripper animations in the Web GUI.

6.3.7.2 Setting of grasp points

The CADMatch module detects 3D objects in a scene based on a CAD template and returns the poses of the object origins. To use CADMatch directly in a robot application, up to 100 grasp points can be defined for each template. A grasp point represents the desired position and orientation of the robot's TCP (Tool Center Point) to grasp an object.

Please consult *Setting of grasp points* (Section 6.3.6.4) for further details.

Setting grasp points in the Web GUI

The *rc_reason_stack* Web GUI provides an intuitive and interactive way of defining grasp points for object templates. In a first step, the object template has to be uploaded to the *rc_reason_stack*. This can be done in the Web GUI in any pipeline under *Modules* → *CADMatch* by clicking on *+ Add a new Template* in the *Templates, Grasps and Pose Priors* section, or in *Database* → *Templates* in the *CADMatch Templates, Grasps and Pose Priors* section. Once the template upload is complete, a dialog with a 3D visualization of the object template is shown for adding or editing grasp points. The same dialog appears when editing an existing template.

More details are given in *Setting grasp points in the Web GUI* (Section 6.3.6.4).

Setting grasp points via the REST-API

Grasp points can be set via the *REST-API interface* (Section 7.2) using the `set_grasp` or `set_all_grasps` services (see *Internal services*, Section 6.3.7.11).

More details are given in *Setting grasp points via the REST-API* (Section 6.3.6.4).

6.3.7.3 Setting of pose priors

The CADMatch module offers the possibility to define prior poses of the objects to be detected. If a pose prior is given, the object detection will use this pose prior and only refine the given pose. This speeds up the detection significantly. A pose prior represents the approximate position and orientation of the object to be detected. The pose can be defined in the camera or the external coordinate frame, if a hand-eye calibration is available.

Each pose prior consists of an `id` which must be unique within all pose priors for an object template, the `template_id` representing the template the pose prior applies to, the `pose` and the `pose_frame` of the prior. Pose priors can be set via the *REST-API interface* (Section 7.2), or by using the interactive visualization in the Web GUI. The Web GUI allows to interactively position the object in the current point cloud. This can be done in the "Pose Priors" tab during editing a template.

Pose priors should be used in applications where the approximate object poses are known beforehand. The *rc_reason_stack* can store up to 50 pose priors per template.

6.3.7.4 Setting the preferred orientation of the TCP

The CADMatch module determines the reachability of grasp points based on the *preferred orientation* of the TCP. The preferred orientation can be set via the `set_preferred_orientation` service call or on the *CADMatch* page in the Web GUI. The resulting direction of the TCP's z axis is used to reject grasps which cannot be reached by the gripper. Furthermore, the preferred orientation can be used to sort the reachable grasps by setting the corresponding sorting strategy.

The preferred orientation can be set in the camera coordinate frame or in the external coordinate frame, in case a hand-eye calibration is available. If the preferred orientation is specified in the external coordinate frame and the sensor is robot mounted, the current robot pose has to be given to each object detection call. If no preferred orientation is set, the orientation of the left camera (see Coordinate frames in the rc_visard manual) will be used as the preferred orientation of the TCP.

6.3.7.5 Setting the sorting strategies

The objects and grasps returned by the `detect_object` service call are sorted according to a sorting strategy which can be chosen by the user. The following sorting strategies are available and can be set in the *Web GUI* (Section 7.1) or using the `set_sorting_strategies` service call:

- gravity: highest matches and grasp points along the gravity direction are returned first,
- match_score: matches with the highest match score and grasp points on objects with the highest match score are returned first,
- preferred_orientation: matches and grasp points with minimal rotation difference of a chosen axis (or all axes, when axis is empty) with respect to the preferred TCP orientation are returned first,
- direction: matches and grasp points with the shortest distance along a defined direction vector in a given pose_frame are returned first.
- distance_to_point: matches and grasp points with the shortest or farthest (if farthest_first is true) distance from a point in a given pose_frame are returned first.

If no sorting strategy is set or default sorting is chosen in the Web GUI, sorting is done based on a combination of match_score and the minimal distance from the camera along the z axis of the preferred orientation of the TCP.

6.3.7.6 Detection of objects

The CADMatch module requires an object template for object detection. This template contains information about the 3D shape of the object and prominent edges that can be visible in the camera images. CADMatch also supports partial object templates, which contain only a specific part of the object that can be detected well, e.g., in case of occlusions. Furthermore, templates can require a pose prior for the detection which is then only refined using the image data.

The object detection is a two-stage process consisting of a prior estimation step and a pose refinement step. First, a pose prior is computed based on the appearance of the object in the camera images. Second, the pose is refined by using the 3D point cloud and edges in the camera image. For this to work, the objects to detect must be visible in both left and right camera images. If pose priors are given, only the pose refinement step is performed based, which decreases runtime significantly.

For triggering the object detection, in general, the following information can be provided to the CADMatch module:

- The template ID of the object to be detected in the scene
- The coordinate frame in which the poses of the detected objects and the grasp points shall be returned (ref. *Hand-eye calibration*, Section 6.3.7.7).

Optionally, further information can be given to the CADMatch module:

- The IDs of the pose priors which approximately match the poses of the objects to be detected. In case a template is used that requires a pose prior, one or more pose prior IDs have to be provided.
- The ID of the load carrier which contains the items to be detected.
- A compartment inside the load carrier where to detect objects (see *Load carrier compartments*, Section 6.5.1.3).
- The ID of the 3D region of interest where to search for the load carriers if a load carrier is set. Otherwise, the ID of the 3D region of interest where to search for the objects.
- The current robot pose in case the camera is mounted on the robot and the chosen coordinate frame for the poses is external, or the preferred orientation is given in the external frame, or the chosen region of interest is defined in the external frame.
- Collision detection information: The ID of the gripper to enable collision checking and optionally a pre-grasp offset to define a pre-grasp position. Details on collision checking are given below in *CollisionCheck* (Section 6.3.7.7).
- Data acquisition mode: The user can choose if a new image dataset is acquired for the detection (default), or if the detection should be performed on the previously used image dataset. This saves data acquisition time, e.g. in case several detections with different templates have to be run on the same image.

On the Web GUI the detection can be tested in the *Try Out* section of the CADMatch module's page.

The detected objects are returned in a list of `matches`, sorted according to the selected sorting strategy (see *Setting the sorting strategies*, Section 6.3.7.5). Each detected object includes a `uuid` (Universally Unique Identifier) and the `timestamp` of the oldest image that was used to detect it. The `pose` of a detected object corresponds to the pose of the origin of the object template used for detection. Furthermore, the matching `score` is given to indicate the quality of the detection.

If the chosen template also has grasp points attached (see *Setting of grasp points*, Section 6.3.7.2), a list of `grasps` for all objects is returned in addition to the list of detected objects. The grasps are sorted according to the selected sorting strategy (see *Setting the sorting strategies*, Section 6.3.7.5). The grasp poses are given in the desired coordinate frame. There are references between the detected objects and the grasps via their `uuids`.

For objects with a discrete symmetry (e.g. prismatic objects), all collision-free symmetries of each grasp point which are reachable according to the given preferred TCP orientation are returned, ordered by the given sorting strategy.

For objects with a continuous symmetry (e.g. cylindrical objects), all grasps symmetric to each grasp point on an object are checked for reachability and collisions, and only the best one according to the given sorting strategy is returned.

The returned matches are visualized with green edges in the *Result* image on the *CADMatch* page of the Web GUI. Matches that were found to be overlapped by other objects or parts of the scene (if `max_object_overlap` is smaller than 1) are visualized with red edges in the result image and the overlapped area is marked by red stripes. Additionally, matches that were filtered out due to low scores, overlaps or the maximum number of matches are visualized in the *Discarded Matches* image.

Note: The first detection call with a new object template takes longer than the following detection calls, because the object template has to be loaded into the CADMatch module first. To avoid this, the `warmup_template` service can be used to load a template so that it is ready when the first detection is triggered.

6.3.7.7 Interaction with other modules

Internally, the CADMatch module depends on, and interacts with other on-board modules as listed below.

Note: All changes and configuration updates to these modules will affect the performance of the CADMatch modules.

Camera and depth data

The CADMatch module makes internally use of the following data:

- Rectified images from the *Camera module* (`rc_camera`, Section 6.1)
- Disparity, error, and confidence images from the *Stereo matching module* (`rc_stereomatching`, Section 6.2.2), in case a stereo camera is used. The `quality` parameter of the stereo matching module must be set to `Medium` or higher (see *Parameters*, Section 6.2.2.1). We recommend `Full` or `High` quality for using CADMatch.
- Disparity, error, and confidence images from the *Orbbec module* (`rc_orbbec`, Section 6.2.4), in case an *Orbbec* camera is used
- Disparity, error, and confidence images from the *Zivid module* (`rc_zivid`, Section 6.2.3), in case a *zivid* camera is used

All processed images are guaranteed to be captured after the module trigger time.

IO and Projector Control

In case the *rc_reason_stack* is used in conjunction with an external random dot projector and the *IO and Projector Control* module (`rc_iocontrol`, Section 6.4.4), it is recommended to connect the projector to GPIO Out 1 and set the stereo-camera module's acquisition mode to `SingleFrameOut1` (see *Stereo matching parameters*, Section 6.2.2.1), so that on each image acquisition trigger an image with and without projector pattern is acquired.

Alternatively, the output mode for the GPIO output in use should be set to `ExposureAlternateActive` (see *Description of run-time parameters*, Section 6.4.4.1).

In either case, the *Auto Exposure Mode* `exp_auto_mode` should be set to `AdaptiveOut1` to optimize the exposure of both images.

Hand-eye calibration

In case the camera has been calibrated to a robot, the CADMatch module can automatically provide poses in the robot coordinate frame. For the CADMatch node's *Services* (Section 6.3.7.10), the frame of the output poses can be controlled with the `pose_frame` argument.

Two different `pose_frame` values can be chosen:

1. **Camera frame** (`camera`). All poses provided by the modules are in the camera frame, and no prior knowledge about the pose of the camera in the environment is required. This means that the configured regions of interest and load carriers move with the camera. It is the user's responsibility to update the configured poses if the camera frame moves (e.g. with a robot-mounted camera).
2. **External frame** (`external`). All poses provided by the modules are in the external frame, configured by the user during the hand-eye calibration process. The module relies on the on-board *Hand-eye calibration module* (Section 6.4.1) to retrieve the sensor mounting (static or robot mounted) and the hand-eye transformation. If the mounting is static, no further information is needed. If the sensor is robot-mounted, the `robot_pose` is required to transform poses to and from the `external` frame.

Note: If no hand-eye calibration is available, all `pose_frame` values should be set to `camera`.

All `pose_frame` values that are not `camera` or `external` are rejected.

If the sensor is robot-mounted, the current `robot_pose` has to be provided depending on the value of `pose_frame`, the definition of the preferred TCP orientation and the sorting direction or sorting point:

- If `pose_frame` is set to `external`, providing the robot pose is obligatory.
- If the preferred TCP orientation is defined in `external`, providing the robot pose is obligatory.
- If the sorting direction is defined in `external`, providing the robot pose is obligatory.
- If the distance-to-point sorting strategy is defined in `external`, providing the robot pose is obligatory.
- In all other cases, providing the robot pose is optional.

LoadCarrier

The CADMatch module uses the load carrier detection functionality provided by the *LoadCarrier* module (`rc_load_carrier`, Section 6.3.2), with the run-time parameters specified for this module. However, only one load carrier will be returned and used in case multiple matching load carriers could be found in the scene. In case multiple load carriers of the same type are visible, a region of interest should be set to ensure that always the same load carrier is used for the CADMatch module.

CollisionCheck

Collision checking can be easily enabled for grasp computation of the CADMatch module by passing a `collision_detection` argument to the `detect_object` service call. It contains the ID of the used gripper and optionally a pre-grasp offset. The gripper has to be defined in the GripperDB module (see *Setting a gripper*, Section 6.5.3.2) and details about collision checking are given in *Collision checking within other modules* (Section 6.4.2.2).

Alternatively, grasp points can be assigned individual gripper IDs, and collision checking can be enabled for all grasp points with gripper IDs by enabling the run-time parameter `check_collisions`.

If the selected CADMatch template contains a collision geometry and the run-time parameter `check_collisions_with_matches` is true, also collisions between the gripper and all other detected objects (not limited to `max_matches`) will be checked. The object on which the grasp point to be checked is located, is excluded from the collision check.

If the run-time parameter `check_collisions_with_point_cloud` is true, also collisions between the gripper and a watertight version of the point cloud are checked. If this feature is used with suctions grippers, it should be ensured that the TCP is defined to be outside the gripper geometry, or that the grasp points are defined above the object surface. Otherwise every grasp will result in a collision between the gripper and the point cloud.

If the run-time parameter `check_collisions_during_retraction` is true and a load carrier and a pre-grasp offset are given, each grasp point will be checked for collisions between the object in the gripper and the load carrier walls during retraction. This collision check is performed along the full linear trajectory from the grasp point back to the pre-grasp position.

If collision checking is enabled, only grasps which are collision free or could not be checked for collisions (e.g. because no gripper was given) will be returned. The result image on top of the *CADMatch* page of the Web GUI also shows collision-free grasps in green, unchecked grasps in yellow and colliding grasp points in red. The detected objects which are considered in the collision check are visualized with their edges in red.

The CollisionCheck module's run-time parameters affect the collision detection as described in *CollisionCheck Parameters* (Section 6.4.2.3).

6.3.7.8 Parameters

The CADMatch module is called `rc_cadmatch` in the REST-API and is represented in the *Web GUI* (Section 7.1) in the desired pipeline under *Modules* → *CADMatch*. The user can explore and configure the `rc_cadmatch` module's run-time parameters, e.g. for development and testing, using the Web GUI or the *REST-API interface* (Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.41: The rc_cadmatch module's run-time parameters

Name	Type	Min	Max	Default	Description
check_collisions	bool	false	true	false	Whether to check for collisions when a gripper is defined for a grasp
check_collisions_during_retraction	bool	false	true	false	Whether to check for collisions between the object in the gripper and the load carrier during retraction
check_collisions_with_matches	bool	false	true	true	Whether to check for collisions between gripper and detected matches
check_collisions_with_point_cloud	bool	false	true	false	Whether to check for collisions between gripper and point cloud
edge_max_distance	float64	0.5	5.0	2.0	Maximum allowed distance in pixels between the template edges and the detected edges in the image
edge_sensitivity	float64	0.05	1.0	0.5	Sensitivity of the edge detector
grasp_filter_orientation_threshold	float64	0.0	180.0	45.0	Maximum allowed orientation change between grasp and preferred orientation in degrees
max_matches	int32	1	30	10	Maximum number of matches
max_object_overlap	float64	0.0	1.0	1.0	Maximum fraction of object that is allowed to be overlapped by something else
min_score	float64	0.05	1.0	0.3	Minimum score for matches
only_highest_priority_grasps	bool	false	true	false	Whether to return only the highest priority level grasps
prior_selection_mode	string	-	-	MatchSorting	Method of selecting priors for refinement [MatchSorting, PriorAccessibility]

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's *CADMatch* page. The name in the Web GUI is given in brackets behind the parameter name and the parameters are listed in the order they appear in the Web GUI:

max_matches (*Maximum Matches*)

is the maximum number of objects to detect.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?max_matches=<value>
```

min_score (*Minimum Score*)

is the minimum detection score after refinement. The higher this value, the better 2D edges and 3D point cloud must match the given template.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?min_score=<value>
```

edge_sensitivity (*Edge Sensitivity*)

is the sensitivity of the edge detector. The higher the value of this parameter, the more edges will be used for pose refinement.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?edge_sensitivity=<value>
```

edge_max_distance (*Maximum Edge Distance*)

is the maximum allowed distance in pixels between the template edges and the detected edges in the image during the refinement step.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?edge_max_distance=<value>
```

grasp_filter_orientation_threshold (*Grasp Orientation Threshold*)

is the maximum deviation of the TCP's z axis at the grasp point from the z axis of the TCP's preferred orientation in degrees. Only grasp points which are within this threshold are returned. When set to zero, any deviations are valid.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?grasp_filter_orientation_
↪threshold=<value>
```

prior_selection_mode (*Prior Selection Mode*)

determines the method that is used to select which of the detected priors (initial pose estimates) will be refined. Available options are `MatchSorting` and `PriorAccessibility`. When `MatchSorting` is chosen, the priors are selected according to the chosen sorting strategy. This is the default mode. When `PriorAccessibility` is chosen, the priors are selected according to their accessibility for grasping. This mode should be used for chaotic scenes with many overlapping objects, e.g. in bin picking.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?prior_selection_mode=<value>
```

max_object_overlap (*Maximum Object Overlap*)

This parameter determines the maximum fraction of a match that is allowed to be overlapped by other objects or scene parts relative to the camera's line of sight. Matches with higher overlap values will be discarded. A value of 1 disables the overlap check. Use this parameter to ensure to only get grasps on objects that are not overlapped by others.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?max_object_overlap=<value>
```

only_highest_priority_grasps (*Only Highest Priority Grasps*)

If set to true, only grasps with the highest priority will be returned. If collision checking is enabled, only the collision-free grasps among the group of grasps with the highest priority are returned. This can save computation time and reduce the number of grasps to be parsed on the application side.

Without collision checking, only grasps of highest priority are returned.

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 (deprecated)

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

check_collisions (*Check Collisions*)

If this parameter is enabled, collision checking will be performed for all grasps which have a gripper ID assigned, even when no default gripper is given in the detect_object service call. If a load carrier is used, the collision check will always be performed between the gripper and the load carrier. Collision checking with the point cloud and other matches is only performed when the corresponding runtime parameters are enabled.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?check_collisions=<value>
```

check_collisions_with_matches (*Check Collisions with Matches*)

This parameter is only used when collision checking is enabled by passing a gripper to the detect_object service call or by enabling the check_collisions runtime parameter. If check_collisions_with_matches is set to true, all grasp points will be checked for collisions between the gripper and all other detected objects (not limited to max_matches), and only grasp points at which the gripper would not collide with any other detected object will be returned.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

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

check_collisions_with_point_cloud (*Check Collisions with Point Cloud*)

This parameter is only used when collision checking is enabled by passing a gripper to the detect_object service call or by enabling the check_collisions runtime parameter. If check_collisions_with_point_cloud is set to true, all grasp points will be checked for collisions between the gripper and a watertight version of the point cloud, and only grasp points at which the gripper would not collide with this point cloud will be returned.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?check_collisions_with_point_
↪cloud=<value>
```

check_collisions_during_retraction (*Check Collisions during Retraction*)

This parameter is only used when collision checking is enabled by passing a gripper to the `detect_object` service call or by enabling the `check_collisions` runtime parameter. When `check_collisions_during_retraction` is enabled and a load carrier and a pre-grasp offset are given, each grasp point will be checked for collisions between the object in the gripper and the load carrier walls during retraction. This collision checking is performed along the full linear trajectory from the grasp point back to the pre-grasp position. Only collision-free grasp points will be returned.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/parameters?check_collisions_during_
↪retraction=<value>
```

6.3.7.9 Status values

The `rc_cadmatch` module reports the following status values:

Table 6.42: The `rc_cadmatch` module's status values

Name	Description
data_acquisition_time	Time in seconds required by the last active service to acquire images
last_timestamp_processed	The timestamp of the last processed dataset
last_request_timestamp	The timestamp of the last detection request
load_carrier_detection_time	Processing time of the last load carrier detection in seconds
object_detection_time	Processing time of the last last object detection in seconds
processing_time	Processing time of the last detection (including load carrier detection) in seconds
state	The current state of the rc_cadmatch node

The reported `state` can take one of the following values.

Table 6.43: Possible states of the CADMatch module

State name	Description
IDLE	The module is idle.
RUNNING	The module is running and ready for load carrier detection and object detection.
FATAL	A fatal error has occurred.

6.3.7.10 Services

The user can explore and call the `rc_cadmatch` module's services, e.g. for development and testing, using the *REST-API interface* (Section 7.2) or the *rc_reason_stack Web GUI* (Section 7.1).

The CADMatch modules offer the following services.

detect_object

Triggers the detection of objects as described in *Detection of objects* (Section 6.3.7.6) based on an object template.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/detect_object
```

Request

Required arguments:

pose_frame: see *Hand-eye calibration* (Section 6.3.7.7).

template_id: the ID of the template to be detected.

Potentially required arguments:

robot_pose: see *Hand-eye calibration* (Section 6.3.7.7).

pose_prior_ids: IDs of the pose priors for the items to be detected. In case the chosen template requires a pose prior for the detection, this argument must be provided.

Optional arguments:

load_carrier_id: ID of the load carrier which contains the items to be detected.

load_carrier_compartment: compartment inside the load carrier where to detect items (see *Load carrier compartments*, Section 6.5.1.3).

region_of_interest_id: if load_carrier_id is set, ID of the 3D region of interest where to search for the load carriers. Otherwise, ID of the 3D region of interest where to search for the objects.

collision_detection: see *Collision checking within other modules* (Section 6.4.2.2).

data_acquisition_mode: if set to CAPTURE_NEW (default), a new image dataset will be used for the detection. If set to USE_LAST the previous dataset will be used for the detection.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "collision_detection": {
      "gripper_id": "string",
      "pre_grasp_offset": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "data_acquisition_mode": "string",
```

(continues on next page)

(continued from previous page)

```
    "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`: list of grasps on the detected objects, ordered according to the chosen sorting strategy. The `match_uuid` gives the reference to the detected object in `matches` this grasp belongs to. The list of returned grasps will be trimmed to the 100 best grasps if more reachable grasps are found. Each grasp contains a flag `collision_checked` and a `gripper_id` (see *Collision checking within other modules*, Section 6.4.2.2).

`load_carriers`: list of detected load carriers.

`matches`: list of detected objects matching the template. The matches are ordered according to the chosen sorting strategy. The `score` indicates how well the object matches the template. The `grasp_uuids` refer to the grasps in `grasps` which are reachable on this object.

`timestamp`: timestamp of the image set the detection ran on.

`return_code`: holds possible warnings or error codes and messages.

The definition for the response with corresponding datatypes is:

```
{
  "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",
        "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",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "rim_ledge": {
          "x": "float64",
```

(continues on next page)

(continued from previous page)

```
          "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",
            "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"
    }
  }
}
```

`set_preferred_orientation`

Persistently stores the preferred orientation of the TCP to compute the reachability of the grasps, which is used for filtering and, optionally, sorting the grasps returned by the `detect_object` service (see *Setting the preferred orientation of the TCP*, Section 6.3.7.4).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/set_preferred_
↪orientation
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_preferred_orientation
```

Request

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "orientation": {
      "w": "float64",
      "x": "float64",
      "y": "float64",
      "z": "float64"
    },
    "pose_frame": "string"
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

get_preferred_orientation

Returns the preferred orientation of the TCP to compute the reachability of the grasps, which is used for filtering and, optionally, sorting the grasps returned by the `detect_object` service (see *Setting the preferred orientation of the TCP*, Section 6.3.7.4).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/get_preferred_
↪orientation
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_preferred_orientation
```

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

```
{
  "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

Persistently stores the sorting strategy for sorting the grasps and matches returned by the `detect_object` service (see *Detection of objects*, Section 6.3.7.6).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/set_sorting_
↪strategies
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_sorting_strategies
```

Request

Only one strategy may have a `weight` greater than 0. If all `weight` values are set to 0, the module will use the default sorting strategy.

If the weight for `direction` is set, the `vector` must contain the direction vector and `pose_frame` must be either `camera` or `external`.

If the weight for `distance_to_point` is set, `point` must contain the sorting point and `pose_frame` must be either `camera` or `external`.

If the weight for `preferred_orientation` is set, the `axis` can be set to `x`, `y` or `z` to consider only rotational differences between the respective axes. If `axis` is empty, the full orientation difference will be used for sorting.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "direction": {
      "pose_frame": "string",
      "vector": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "weight": "float64"
```

(continues on next page)

(continued from previous page)

```
    },
    "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"
    }
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

get_sorting_strategies

Returns the sorting strategy for sorting the grasps and matches returned by the `detect_object` service (see *Detection of objects*, Section 6.3.7.6).

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/get_sorting_
↪strategies
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_sorting_strategies
```

Request

This service has no arguments.

Response

All `weight` values are 0 when the module uses the default sorting strategy.

The definition for the response with corresponding datatypes is:

```
{
  "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

Loads a template so that it is ready when the first detection with this template is triggered. Without using this service, the first detection with a new template takes longer than the following ones, because the template is then loaded at the first detection.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/warmup_template
```

Request

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "template_id": "string"
  }
}
```

The template_id is the ID of the template to be loaded into the CADMatch module.

Response

The definition for the response with corresponding datatypes is:

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

start

Starts the module. If the command is accepted, the module moves to state RUNNING.

Details

The current_state value in the service response may differ from RUNNING if the state transition is still in process when the service returns.

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/start
```

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

```
{
  "name": "start",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

stop

Stops the module. If the command is accepted, the module moves to state IDLE.

Details

The `current_state` value in the service response may differ from `IDLE` if the state transition is still in process when the service returns.

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/stop
```

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

```
{
  "name": "stop",
  "response": {
    "accepted": "bool",
    "current_state": "string"
  }
}
```

reset_defaults

Resets all parameters of the module to its default values, as listed in above table. Also resets preferred orientation and sorting strategies. The reset does not apply to templates.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.3.7.11 Internal services

The following services for configuring grasps and pose priors can change in future without notice. Setting, retrieving and deleting grasps and pose priors is recommended to be done via the Web GUI.

Note: Configuring grasps and pose priors is global for all templates on the *rc_reason_stack* and affects all camera pipelines.

set_grasp

Persistently stores a grasp for the given object template on the *rc_reason_stack*. All configured grasps are persistent over firmware updates and rollbacks.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_grasp
```

Request

Details for the definition of the grasp type are given in *Setting of grasp points* (Section 6.3.7.2).

The definition for the request arguments with corresponding datatypes is:

```
{
  "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"
          },
```

(continues on next page)

(continued from previous page)

```
          "position": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          }
        },
        "step_x_deg": "float64"
      },
      "template_id": "string"
    }
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

set_all_grasps

Replaces the list of grasps for the given object template on the *rc_reason_stack*.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_all_grasps
```

Request

Details for the definition of the `grasp` type are given in *Setting of grasp points* (Section 6.3.7.2).

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "grasps": [
      {
        "gripper_id": "string",
        "id": "string",
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
```

(continues on next page)

(continued from previous page)

```
            "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"
        },
        "template_id": "string"
      }
    ],
    "template_id": "string"
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

get_grasps

Returns all configured grasps which have the requested `grasp_ids` and belong to the requested `template_ids`.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_grasps
```

Request

If no `grasp_ids` are provided, all grasps belonging to the requested `template_ids` are returned. If no `template_ids` are provided, all grasps with the requested `grasp_ids` are returned. If neither IDs are provided, all configured grasps are returned.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

```
{
  "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"
            }
```

(continues on next page)

(continued from previous page)

```
        },
        "step_x_deg": "float64"
      },
      "template_id": "string"
    }
  ],
  "return_code": {
    "message": "string",
    "value": "int16"
  }
 }
}
```

delete_grasps

Deletes all grasps with the requested `grasp_ids` that belong to the requested `template_ids`.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/delete_grasps
```

Request

If no `grasp_ids` are provided, all grasps belonging to the requested `template_ids` are deleted. The `template_ids` list must not be empty.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "grasp_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

get_symmetric_grasps

Returns all grasps that are symmetric to the given grasp.

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/get_symmetric_
↪grasps
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_symmetric_grasps
```

Request

Details for the definition of the grasp type are given in *Setting of grasp points* (Section 6.3.7.2).

The definition for the request arguments with corresponding datatypes is:

```
{
  "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

The first grasp in the returned list is the one that was passed with the service call. If the object template does not have an exact symmetry, only the grasp passed with the service call will be returned. If the object template has a continuous symmetry (e.g. a cylindrical object), only 12 equally spaced sample grasps will be returned.

Details for the definition of the `grasp` type are given in *Setting of grasp points* (Section 6.3.7.2).

The definition for the response with corresponding datatypes is:

```
{
  "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

Persistently stores a pose prior for the given object template on the *rc_reason_stack*. All configured pose priors are persistent over firmware updates and rollbacks.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_pose_prior
```

Request

Details for the definition of the `pose_prior` type are given in *Setting of pose priors* (Section 6.3.7.3).

The definition for the request arguments with corresponding datatypes is:

```
{
  "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

The definition for the response with corresponding datatypes is:

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

set_all_pose_priors

Replaces the list of pose priors for the given object template on the *rc_reason_stack*.

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/set_all_pose_
↪priors
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/set_all_pose_priors
```

Request

Details for the definition of the `pose_prior` type are given in *Setting of pose priors* (Section 6.3.7.3).

The definition for the request arguments with corresponding datatypes is:

```
{
  "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

The definition for the response with corresponding datatypes is:

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

get_pose_priors

Returns all configured pose priors which have the requested `pose_prior_ids` and belong to the requested `template_ids`.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/get_pose_priors
```

Request

If no `pose_prior_ids` are provided, all pose priors belonging to the requested `template_ids` are returned. If no `template_ids` are provided, all pose priors with the requested `pose_prior_ids` are returned. If neither IDs are provided, all configured pose priors are returned.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "pose_prior_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

```
{
  "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

Deletes all pose priors with the requested `pose_prior_ids` that belong to the requested `template_ids`.

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_cadmatch/services/delete_pose_
↪priors
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_cadmatch/services/delete_pose_priors
```

Request

If no `pose_prior_ids` are provided, all pose priors belonging to the requested `template_ids` are deleted. The `template_ids` list must not be empty.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "pose_prior_ids": [
      "string"
    ],
    "template_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

6.3.7.12 Return codes

Each service response contains a `return_code`, which consists of a `value` plus an optional `message`. A successful service returns with a `return_code` value of `0`. Negative `return_code` values indicate that the service failed. Positive `return_code` values indicate that the service succeeded with additional information. The smaller value is selected in case a service has multiple `return_code` values, but all messages are appended in the `return_code` message.

The following table contains a list of common codes:

Table 6.44: Return codes of the CADMatch services

Code	Description
0	Success
-1	An invalid argument was provided.
-2	An internal error occurred.
-3	An internal timeout occurred.
-4	Data acquisition took longer than allowed.
-8	Not applicable, stereo quality must be at least Medium.
-9	No valid license for the module.
-10	New element could not be added as the maximum storage capacity of load carriers or regions of interest has been exceeded.
-11	Sensor not connected, not supported or not ready.
10	The maximum storage capacity of load carriers or regions of interest has been reached.
11	Existing data was overwritten.
100	The requested load carrier was not detected in the scene.
101	None of the detected grasps is reachable.
102	The detected load carrier is empty.
103	All detected grasps are in collision.
106	The list of returned grasps has been trimmed to the 100 best grasps.
110	Hints for setting up the application, e.g. reducing the distance from the camera, setting a region of interest.
113	No gripper was found for collision checking.
114	Collision checking during retraction was skipped, e.g. because no load carrier or no pre-grasp offset were given.
151	The object template has a continuous symmetry.
152	The objects are outside the given region of interest, outside the load carrier or outside the image.
153	No edges could be detected in the camera image. Check the Edge Sensitivity.
999	Additional hints for application development

6.3.7.13 Template API

For template upload, download, listing and removal, special REST-API endpoints are provided. Templates can also be uploaded, downloaded and removed via the Web GUI. The templates include the grasp points and pose priors, if grasp points or pose priors have been configured. Up to 50 templates can be stored persistently on the *rc_reason_stack*.

GET /templates/rc_cadmatch

Get list of all rc_cadmatch templates.

Template request

```
GET /api/v2/templates/rc_cadmatch HTTP/1.1
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "string"
  }
]
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns array of Template)*
- 404 Not Found – node not found

Referenced Data Models

- *Template* (Section 7.2.3)

GET /templates/rc_cadmatch/{id}

Get a rc_cadmatch template. If the requested content-type is application/octet-stream, the template is returned as file.

Template request

```
GET /api/v2/templates/rc_cadmatch/<id> HTTP/1.1
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameters

- **id** (*string*) – id of the template *(required)*

Response Headers

- Content-Type – application/json application/ubjson application/octet-stream

Status Codes

- 200 OK – successful operation *(returns Template)*
- 404 Not Found – node or template not found

Referenced Data Models

- *Template* (Section 7.2.3)

PUT /templates/rc_cadmatch/{id}

Create or update a rc_cadmatch template.

Template request

```
PUT /api/v2/templates/rc_cadmatch/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameters

- **id** (*string*) – id of the template *(required)*

Form Parameters

- **file** – template file *(required)*

Request Headers

- Accept – multipart/form-data application/json

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns Template)*
- 400 Bad Request – Template is not valid or max number of templates reached
- 403 Forbidden – forbidden, e.g. because there is no valid license for this module.
- 404 Not Found – node or template not found
- 413 Request Entity Too Large – Template too large

Referenced Data Models

- *Template* (Section 7.2.3)

DELETE /templates/rc_cadmatch/{id}

Remove a rc_cadmatch template.

Template request

```
DELETE /api/v2/templates/rc_cadmatch/<id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameters

- **id** (*string*) – id of the template *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation
- 403 Forbidden – forbidden, e.g. because there is no valid license for this module.
- 404 Not Found – node or template not found

6.4 Configuration modules

The *rc_reason_stack* provides several configuration modules which enable the user to configure the *rc_reason_stack* for specific applications.

The configuration modules are:

- ***Hand-eye calibration*** **(rc_hand_eye_calibration, Section 6.4.1)** enables the user to calibrate the camera with respect to a robot, either via the Web GUI or the REST-API.
- ***CollisionCheck*** **(rc_collision_check, Section 6.4.2)** provides an easy way to check if a gripper is in collision.

- ***Camera calibration*** (**`rc_stereocalib`, Section 6.4.3**) enables the user to check and perform camera calibration via the *WEB GUI* (Section 7.1).
- ***IO and Projector Control*** (**`rc_iocontrol`, Section 6.4.4**) provides control over the camera's general purpose inputs and outputs with special modes for controlling an external random dot projector.

These modules are pipeline specific, which means that they run inside each camera pipeline. Changes to their settings or parameters only affect the corresponding pipeline and have no influence on the other camera pipelines running on the *rc_reason_stack*.

6.4.1 Hand-eye calibration

For applications, in which the camera is integrated into one or more robot systems, it needs to be calibrated w.r.t. some robot reference frames. For this purpose, the *rc_reason_stack* is shipped with an on-board calibration routine called the *hand-eye calibration* module. It is a base module which is available on every *rc_reason_stack*.

Note: This module is pipeline specific. Changes to its settings or parameters only affect the respective camera pipeline and have no influence on other pipelines running on the *rc_reason_stack*.

Note: The implemented calibration routine is completely agnostic about the user-defined robot frame to which the camera is calibrated. It might be a robot's end-effector (e.g., flange or tool center point) or any point on the robot structure. The method's only requirement is that the pose (i.e., translation and rotation) of this robot frame w.r.t. a user-defined external reference frame (e.g., world or robot mounting point) is exactly observable by the robot controller and can be reported to the calibration module.

The *Calibration routine* (Section 6.4.1.3) itself is an easy-to-use multi-step procedure using a calibration grid which can be obtained from Roboception.

6.4.1.1 Calibration interfaces

The following two interfaces are offered to conduct hand-eye calibration:

1. All services and parameters of this module required to conduct the hand-eye calibration **programmatically** are exposed by the *rc_reason_stack*'s *REST-API interface* (Section 7.2). The respective node name of this module is `rc_hand_eye_calibration` and the respective service calls are documented *Services* (Section 6.4.1.5).

   Note: The described approach requires a network connection between the *rc_reason_stack* and the robot controller to pass robot poses from the controller to the *rc_reason_stack*'s calibration module.

2. For use cases where robot poses cannot be passed programmatically to the *rc_reason_stack*'s hand-eye calibration module, the *Web GUI*'s *Hand-Eye Calibration* page under *Configuration* in the desired pipeline offers a guided process to conduct the calibration routine **manually**.

   Note: During the process, the described approach requires the user to manually enter into the Web GUI robot poses, which need to be accessed from the respective robot-teaching or handheld device.

6.4.1.2 Camera mounting

As illustrated in Fig. 6.15 and Fig. 6.17, two different use cases w.r.t. to the mounting of the camera generally have to be considered:

a. The camera is **mounted on the robot**, i.e., it is mechanically fixed to a robot link (e.g., at its flange or a flange-mounted tool), and hence moves with the robot.

b. The camera is not mounted on the robot but is fixed to a table or other place in the robot's vicinity and remains at a **static** position w.r.t. the robot.

While the general *Calibration routine* (Section 6.4.1.3) is very similar in both use cases, the calibration process's output, i.e., the resulting calibration transform, will be semantically different, and the fixture of the calibration grid will also differ.

Calibration with a robot-mounted camera When calibrating a robot-mounted camera with the robot, the calibration grid has to be secured in a static position w.r.t. the robot, e.g., on a table or some other fixed-base coordinate system as sketched in Fig. 6.15.

Warning: It is extremely important that the calibration grid does not move during step 2 of the *Calibration routine* (Section 6.4.1.3). Securely fixing its position to prevent unintended movements such as those caused by vibrations, moving cables, or the like is therefore strongly recommended.

The result of the calibration (step 3 of the *Calibration routine*, Section 6.4.1.3) is a pose $\mathbf{T}^{\text{robot}}_{\text{camera}}$ describing the (previously unknown) relative positional and rotational transformation from the *camera* frame into the user-selected *robot* frame such that

$$\mathbf{p}_{\text{robot}} = \mathbf{R}^{\text{robot}}_{\text{camera}} \cdot \mathbf{p}_{\text{camera}} + \mathbf{t}^{\text{robot}}_{\text{camera}}\,, \tag{6.1}$$

where $\mathbf{p}_{\text{robot}} = (x, y, z)^T$ is a 3D point with its coordinates expressed in the *robot* frame, $\mathbf{p}_{\text{camera}}$ is the same point represented in the *camera* coordinate frame, and $\mathbf{R}^{\text{robot}}_{\text{camera}}$ as well as $\mathbf{t}^{\text{robot}}_{\text{camera}}$ are the corresponding 3×3 rotation matrix and 3×1 translation vector of the pose $\mathbf{T}^{\text{robot}}_{\text{camera}}$, respectively. In practice, in the calibration result and in the provided robot poses, the rotation is defined by Euler angles or as quaternion instead of a rotation matrix (see *Pose formats*, Section 11.1).

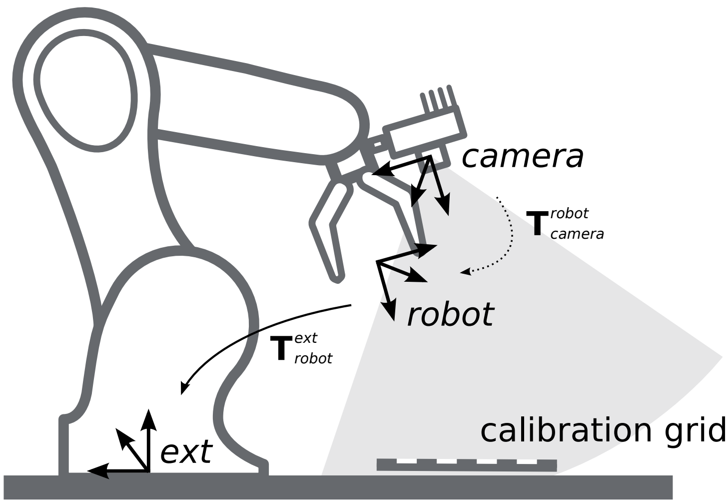

Fig. 6.15: Important frames and transformations for calibrating a camera that is mounted on a general robot. The camera is mounted with a fixed relative position to a user-defined *robot* frame (e.g., flange or TCP). It is important that the pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ of this *robot* frame w.r.t. a user-defined external reference frame *ext* is observable during the calibration routine. The result of the calibration process is the desired calibration transformation $\mathbf{T}^{\text{robot}}_{\text{camera}}$, i.e., the pose of the *camera* frame within the user-defined *robot* frame.

Additional user input is required if the movement of the robot is constrained and the robot can rotate the Tool Center Point (TCP) only around one axis. This is typically the case for robots with four Degrees Of Freedom (4DOF) that are often used for palletizing tasks. In this case, the user must specify which axis of the *robot* frame is the rotation axis of the TCP. Further, the signed offset from the TCP to the *camera* coordinate system along the TCP rotation axis has to be provided. Fig. 6.16 illustrates the situation.

For the *rc_visard* or *rc_visard NG*, the camera coordinate system is located in the optical center of the left camera. The approximate location is given in Coordinate frames in the rc_visard manual.

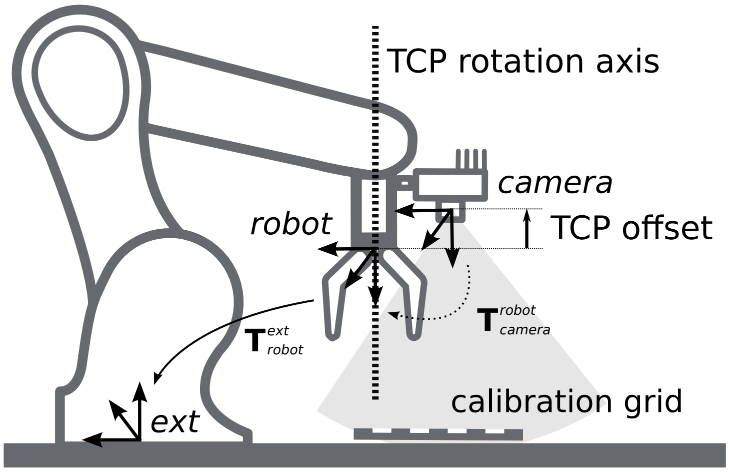

Fig. 6.16: In case of a 4DOF robot, the TCP rotation axis and the offset from the TCP to the camera coordinate system along the TCP rotation axis must be provided. In the illustrated case, this offset is negative.

Calibration with a statically-mounted camera In use cases where the camera is positioned statically w.r.t. the robot, the calibration grid needs to be mounted to the robot as shown for example in Fig. 6.17 and Fig. 6.18.

Note: The hand-eye calibration module is completely agnostic about the exact mounting and positioning of the calibration grid w.r.t. the user-defined *robot* frame. That means, the relative positioning of the calibration grid to that frame neither needs to be known, nor it is relevant for the calibration routine, as shown in Fig. 6.18.

Warning: It is extremely important that the calibration grid is attached securely to the robot such that it does not change its relative position w.r.t. the user-defined *robot* frame during step 2 of the *Calibration routine* (Section 6.4.1.3).

In this use case, the result of the calibration (step 3 of the *Calibration routine*, Section 6.4.1.3) is the pose $\mathbf{T}^{\text{ext}}_{\text{camera}}$ describing the (previously unknown) relative positional and rotational transformation between the *camera* frame and the user-selected external reference frame *ext* such that

$$\mathbf{p}_{\text{ext}} = \mathbf{R}^{\text{ext}}_{\text{camera}} \cdot \mathbf{p}_{\text{camera}} + \mathbf{t}^{\text{ext}}_{\text{camera}} , \qquad (6.2)$$

where $\mathbf{p}_{\text{ext}} = (x, y, z)^T$ is a 3D point with its coordinates expressed in the external reference frame *ext*, $\mathbf{p}_{\text{camera}}$ is the same point represented in the *camera* coordinate frame, and $\mathbf{R}^{\text{ext}}_{\text{camera}}$ as well as $\mathbf{t}^{\text{ext}}_{\text{camera}}$ are the corresponding 3×3 rotation matrix and 3×1 translation vector of the pose $\mathbf{T}^{\text{ext}}_{\text{camera}}$, respectively. In practice, in the calibration result and in the provided robot poses, the rotation is defined by Euler angles or as quaternion instead of a rotation matrix (see *Pose formats*, Section 11.1).

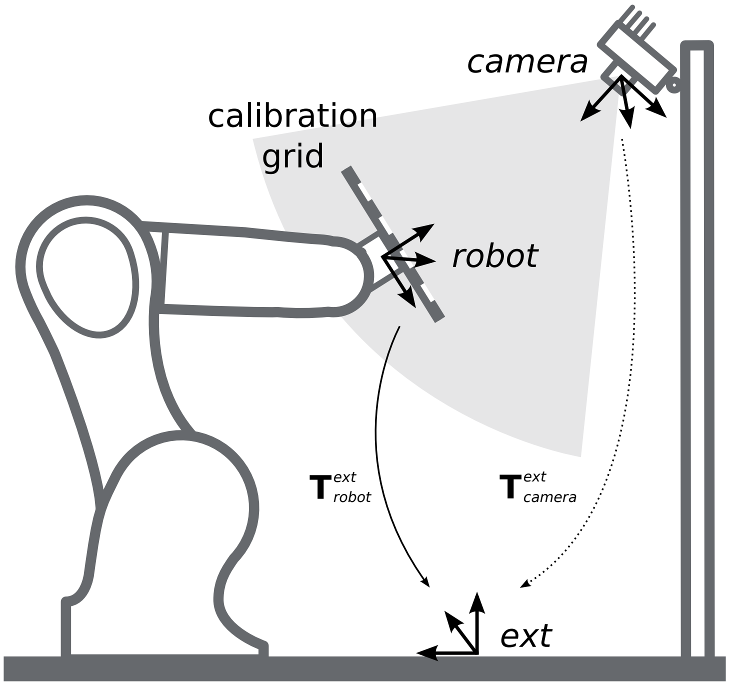

Fig. 6.17: Important frames and transformations for calibrating a statically mounted camera: The latter is mounted with a fixed position relative to a user-defined external reference frame *ext* (e.g., the world coordinate frame or the robot's mounting point). It is important that the pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ of the user-defined *robot* frame w.r.t. this frame is observable during the calibration routine. The result of the calibration process is the desired calibration transformation $\mathbf{T}^{\text{ext}}_{\text{camera}}$, i.e., the pose of the *camera* frame in the user-defined external reference frame *ext*.

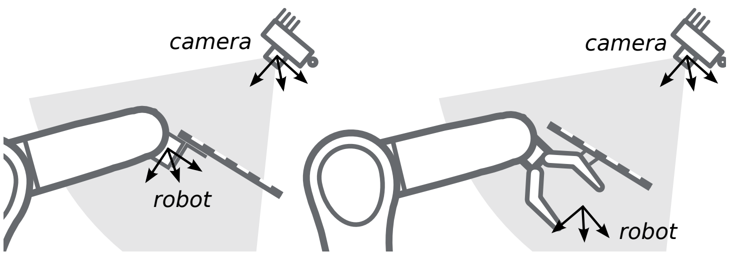

Fig. 6.18: Alternate mounting options for attaching the calibration grid to the robot

Additional user input is required if the movement of the robot is constrained and the robot can rotate the Tool Center Point (TCP) only around one axis. This is typically the case for robots with four Degrees Of Freedom (4DOF) that are often used for palletizing tasks. In this case, the user must specify which axis of the *robot* frame is the rotation axis of the TCP. Further, the signed offset from the TCP to the visible surface of the calibration grid along the TCP rotation axis has to be provided. The grid must be mounted such that the TCP rotation axis is orthogonal to the grid. Fig. 6.19 illustrates the situation.

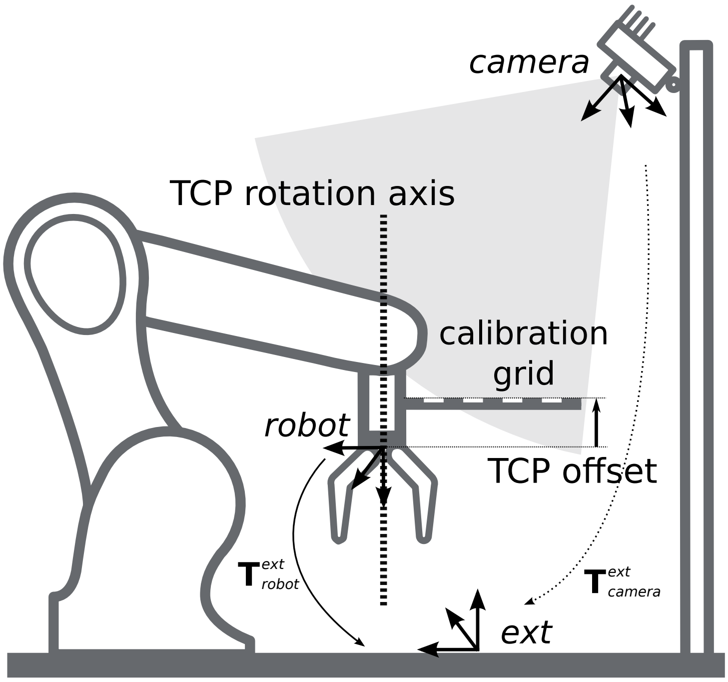

Fig. 6.19: In case of a 4DOF robot, the TCP rotation axis and the offset from the TCP to the visible surface of the grid along the TCP rotation axis must be provided. In the illustrated case, this offset is negative.

6.4.1.3 Calibration routine

The hand-eye calibration can be performed manually using the *Web GUI* (Section 7.1) or programmatically via the *REST-API interface* (Section 7.2). The general calibration routine will be described by following the steps of the hand-eye calibration wizard provided on the Web GUI. This wizard can be found in the *rc_reason_stack*'s Web GUI in the desired pipeline under *Configuration* → *Hand-Eye Calibration*. References to the corresponding REST-API calls are provided at the appropriate places.

Step 1: Hand-Eye Calibration Status

The starting page of the hand-eye calibration wizard shows the current status of the hand-eye calibration. If a hand-eye calibration is saved on the *rc_reason_stack*, the calibration transformation is displayed here (see Fig. 6.20).

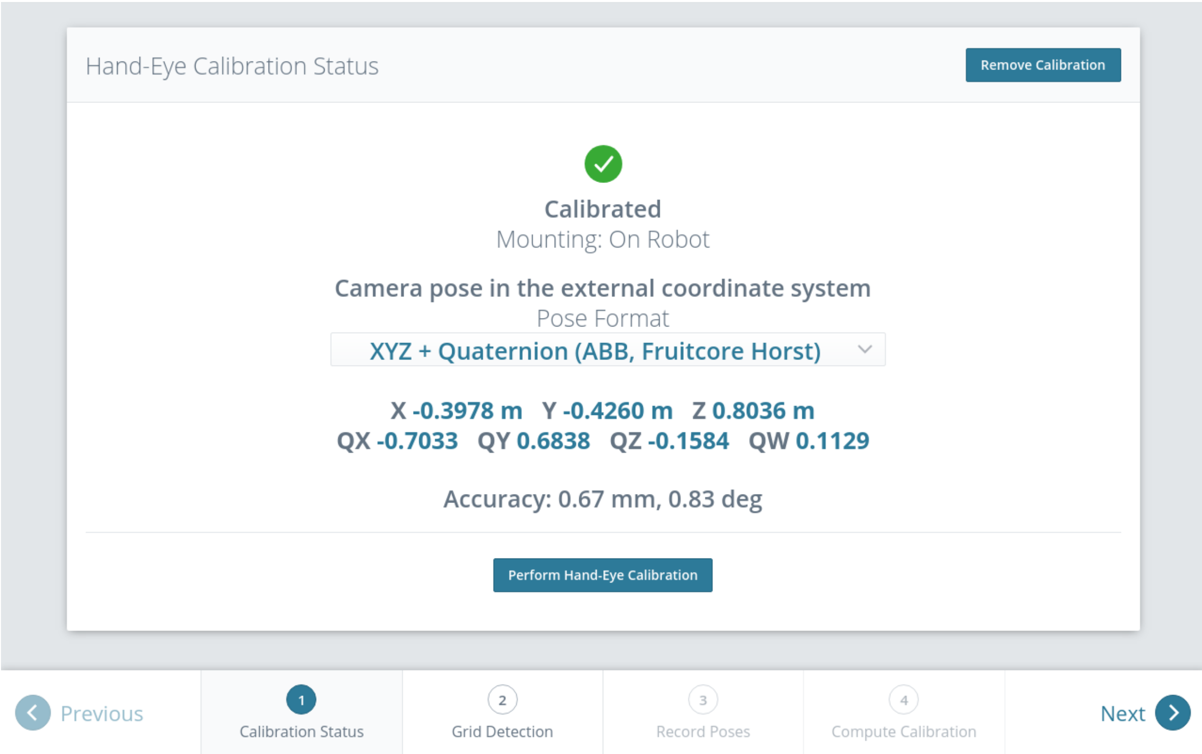

Fig. 6.20: Current status of the hand-eye calibration in case a hand-eye calibration is saved

To query the hand-eye calibration status programmatically, the module's REST-API offers the `get_calibration` service call (see *Services*, Section 6.4.1.5). An existing hand-eye calibration can be removed by pressing *Remove Calibration* or using `remove_calibration` in the REST-API (see *Services*, Section 6.4.1.5).

To start a new hand-eye calibration, click on *Perform Hand-Eye Calibration* or *Next*.

Step 2: Checking Grid Detection

To achieve good calibration results, the images should be well exposed so that the calibration grid can be detected accurately and reliably. In this step, the grid detection can be checked and the camera settings can be adjusted if necessary. In case parts of the calibration grid are overexposed, the respective squares of the calibration grid will be highlighted in red. A successful grid detection is visualized by green check marks on every square of the calibration grid and a thick green border around the grid as shown in Fig. 6.21.

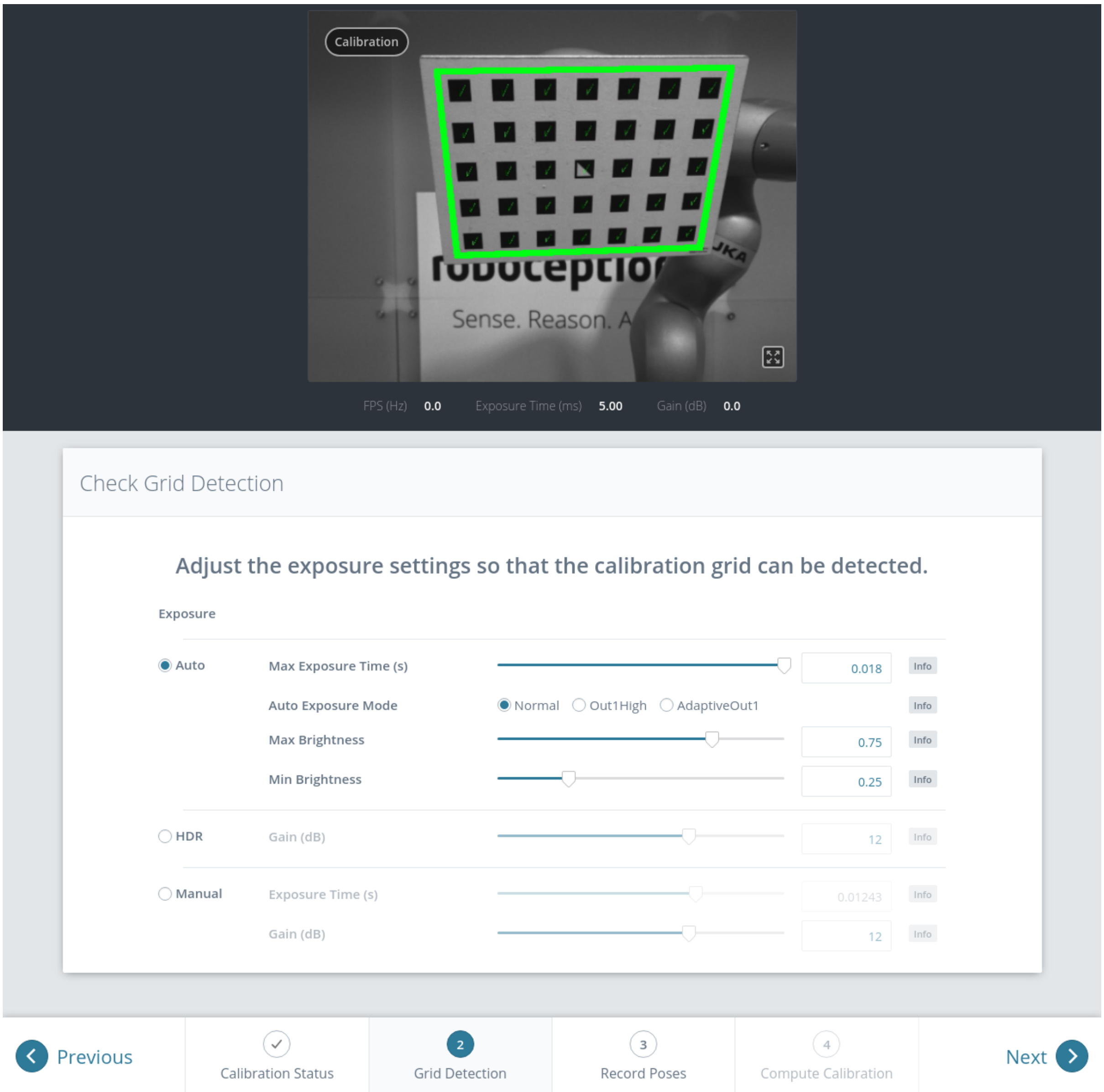

Fig. 6.21: Checking the calibration grid detection

Step 3: Record Poses

In this step, the user records images of the calibration grid at several different robot poses. These poses must each ensure that the calibration grid is completely visible in the left camera image. Furthermore, the robot poses need to be selected properly to achieve a variety of different perspectives for the camera to perceive the calibration grid. Fig. 6.22 shows a schematic recommendation of four different grid positions which should be recorded from a close and a far point of view, resulting in eight images for the calibration.

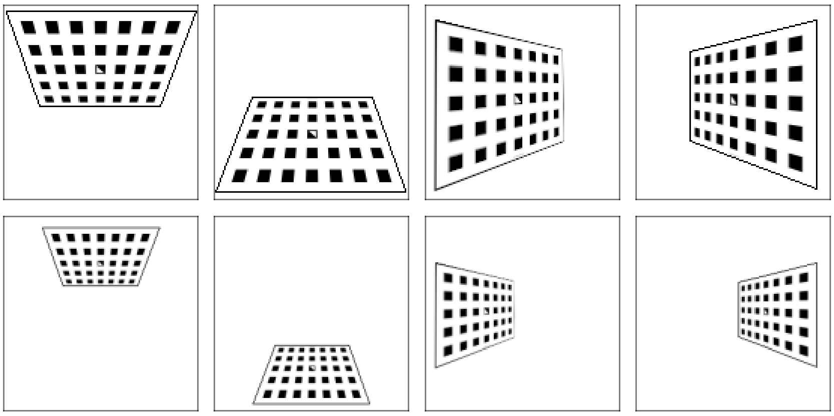

Fig. 6.22: Recommended views on the calibration grid during the calibration procedure. In case of a 4DOF robot, other views have to be chosen, which should be as different as possible.

Warning: Calibration quality, i.e., the accuracy of the calculated calibration result, depends on the calibration-grid views provided. The more diverse the perspectives are, the better is the calibration. Choosing very similar views, i.e., varying the robot pose only slightly before recording a new calibration pose, may lead to inaccurate estimation of the desired calibration transformation.

After the robot reaches each calibration pose, the corresponding pose $\mathbf{T}^{\text{ext}}_{\text{robot}}$ of the user-defined *robot* frame in the user-defined external reference frame *ext* needs to be reported to the hand-eye calibration module. For this purpose, the module offers different *slots* to store the reported poses and the corresponding left camera images. All filled slots will then be used to calculate the desired calibration transformation between the *camera* frame and either the user-defined *robot* frame (robot-mounted camera) or the user-defined external reference frame *ext* (static camera).

In the Web GUI, the user can choose between many different pose formats for providing the calibration poses (see *Pose formats*, Section 11.1). When calibrating using the REST-API, the poses are always given in *XYZ+quaternion*. The Web GUI offers eight slots (*Close View 1*, *Close View 2*, etc.) for the user to fill manually with robot poses. Next to each slot, a figure suggests a respective dedicated viewpoint on the grid. For each slot, the robot should be operated to achieve the suggested view.

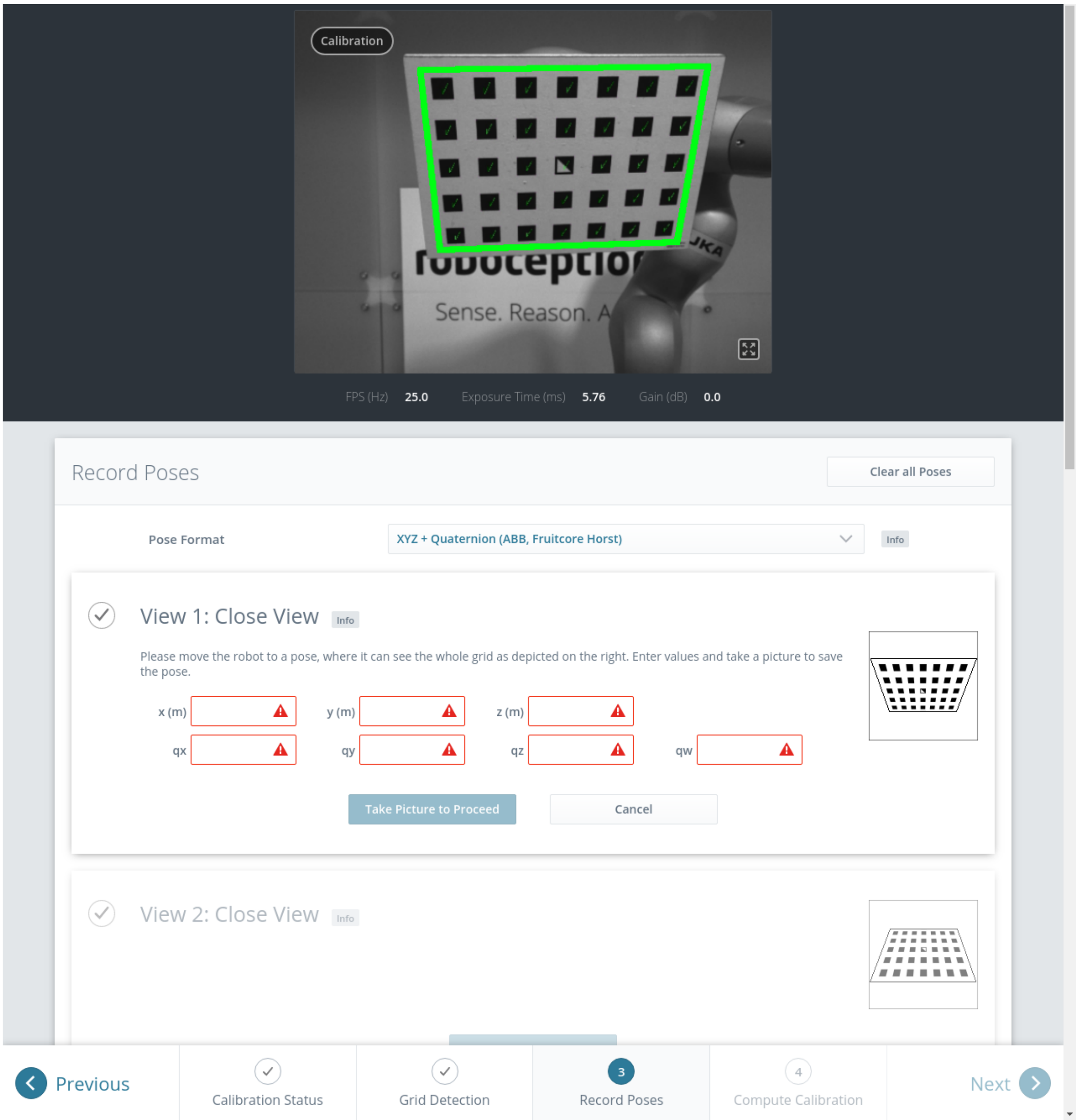

Fig. 6.23: Filling the first slot in the hand-eye calibration process for a statically mounted camera

To record a calibration pose, click on *Set Pose* for the respective slot and enter the *robot* frame's pose into the respective text fields. The pose is then stored with the corresponding camera image by clicking the *Take Picture to Proceed* button. This will save the calibration pose in the respective slot.

To transmit the poses programmatically, the module's REST-API offers the `set_pose` service call (see *Services*, Section 6.4.1.5).

Note: The user's acquisition of robot pose data depends on the robot model and manufacturer – it might be read from a teaching or handheld device, which is shipped with the robot.

Warning: Please be careful to correctly and accurately enter the values; even small variations or typos may lead to calibration-process failure.

The Web GUI displays the currently saved poses (only with slot numbers from 0 to 7) with their camera images and also allows to delete them by clicking *Delete Pose* to remove a single pose, or clicking *Clear all Poses* to remove all poses. In the REST-API the currently stored poses can be retrieved via `get_poses` and removed via `delete_poses` for single poses or `reset_calibration` for removing all poses (see *Services*, Section 6.4.1.5).

When at least four poses are set, the user can continue to the computation of the calibration result by pressing *Next*.

Note: To successfully calculate the hand-eye calibration transformation, at least four different robot calibration poses need to be reported and stored in slots. However, to prevent errors induced by possible inaccurate measurements, at least **eight calibration poses are recommended**.

Step 4: Compute Calibration

Before computing the calibration result, the user has to provide the correct calibration parameters. These include the exact calibration grid dimensions and the sensor mounting type. The Web GUI also offers settings for calibrating 4DOF robots. In this case, the rotation axis, as well as the offset from the TCP to the camera coordinate system (robot-mounted camera) or grid surface (statically mounted camera) must be given. For the REST-API, the respective parameters are listed in *Parameters* (Section 6.4.1.4).

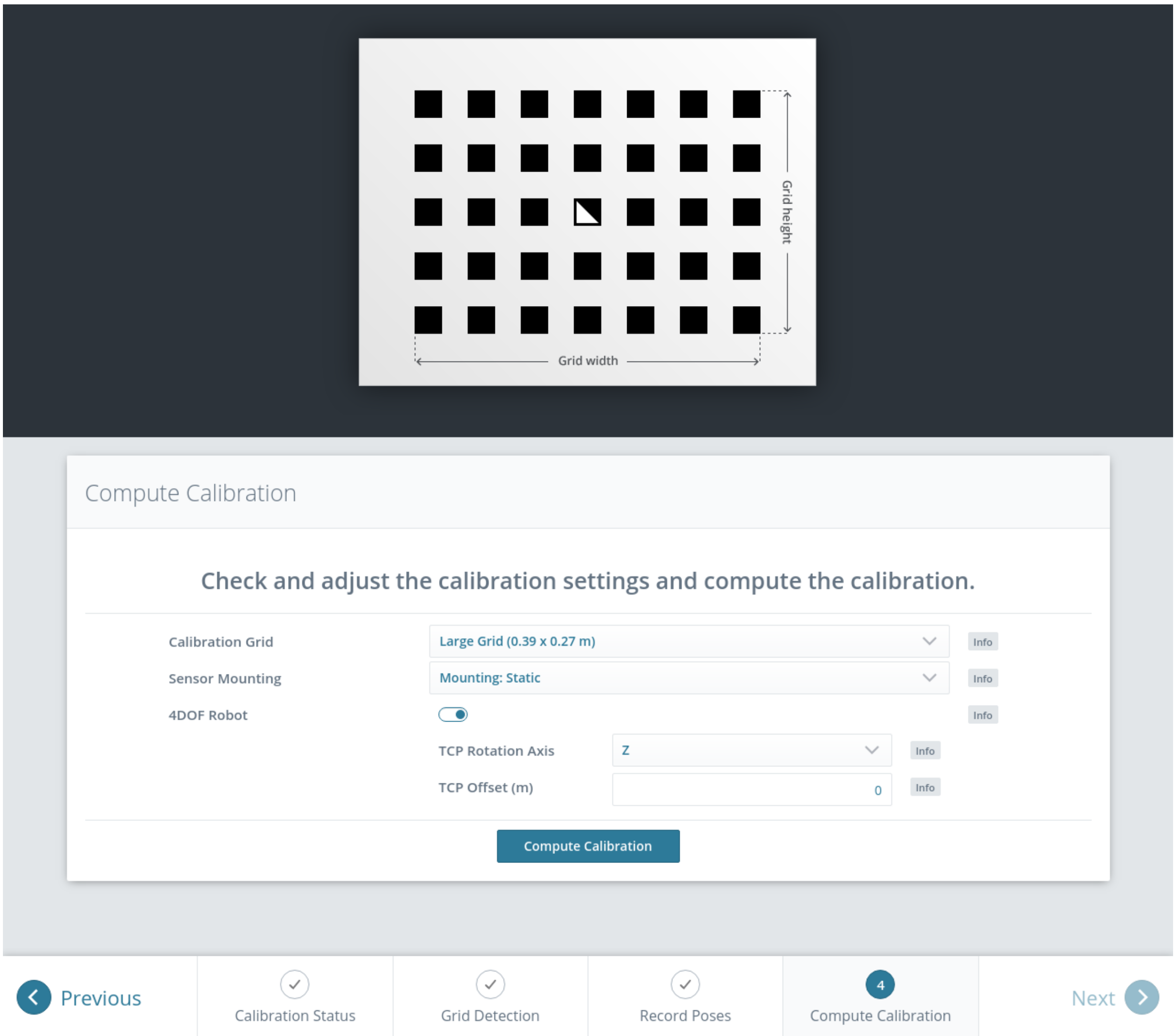

Fig. 6.24: Defining hand-eye calibration parameters and computing the calibration result via the *rc_reason_stack*'s Web GUI

When the parameters are correct, the desired calibration transformation can be computed from the collected poses and camera images by clicking *Compute Calibration*. The REST-API offers this functionality via the `calibrate` service call (see *Services*, Section 6.4.1.5).

Depending on the way the camera is mounted, the calibration result contains the transformation (i.e., the pose) between the *camera* frame and either the user-defined *robot* frame (robot-mounted camera) or the user-defined external reference frame *ext* (statically mounted camera); see *Camera mounting* (Section 6.4.1.2).

To enable users to judge the quality of the resulting calibration transformation, the translational and rotational calibration errors are reported, which are computed from the variance of the calibration result.

If the calibration error is not acceptable, the user can change the calibration parameters and recompute the result, or return to step 3 of the calibration procedure and add more poses or update poses.

To save the calibration result, press *Save Calibration* or use the REST-API `save_calibration` service call (see *Services*, Section 6.4.1.5).

6.4.1.4 Parameters

The hand-eye calibration module is called `rc_hand_eye_calibration` in the REST-API and is represented in the *Web GUI* (Section 7.1) in the desired pipeline under *Configuration* → *Hand-Eye Calibration*. The user can change the calibration parameters there or use the *REST-API interface* (Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.45: The `rc_hand_eye_calibration` module's run-time parameters

Name	Type	Min	Max	Default	Description
`grid_height`	float64	0.0	10.0	0.0	The height of the calibration pattern in meters
`grid_width`	float64	0.0	10.0	0.0	The width of the calibration pattern in meters
`robot_mounted`	bool	false	true	true	Whether the camera is mounted on the robot
`tag_ids`	string	-	-	-	Optional, comma separated list of AprilTag IDs that will be calibrated too
`tcp_offset`	float64	-10.0	10.0	0.0	Offset from TCP along tcp_rotation_axis
`tcp_rotation_axis`	int32	-1	2	-1	-1 for off, 0 for x, 1 for y, 2 for z

Description of run-time parameters

The parameter descriptions are given with the corresponding Web GUI names in brackets.

`grid_width` (*Width*)

Width of the calibration grid in meters. The width should be given with a very high accuracy, preferably with sub-millimeter accuracy.

Via the REST-API, this parameter can be set as follows.

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 (*Height*)

Height of the calibration grid in meters. The height should be given with a very high accuracy, preferably with sub-millimeter accuracy.

Via the REST-API, this parameter can be set as follows.

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 (*Sensor Mounting*)

If set to *true*, the camera is mounted on the robot. If set to *false*, the camera is mounted statically and the calibration grid is mounted on the robot.

Via the REST-API, this parameter can be set as follows.

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*)

The signed offset from the TCP to the camera coordinate system (robot-mounted sensor) or the visible surface of the calibration grid (statically mounted sensor) along the TCP rotation axis in meters. This is required if the robot's movement is constrained and it can rotate its TCP only around one axis (e.g., 4DOF robot).

Via the REST-API, this parameter can be set as follows.

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 Rotation Axis*)

The axis of the *robot* frame around which the robot can rotate its TCP. 0 is used for X, 1 for Y and 2 for the Z axis. This is required if the robot's movement is constrained and it can rotate its TCP only around one axis (e.g., 4DOF robot). -1 means that the robot can rotate its TCP around two independent rotation axes. `tcp_offset` is ignored in this case.

Via the REST-API, this parameter can be set as follows.

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

The REST-API service calls offered to programmatically conduct the hand-eye calibration and to restore this module's parameters are explained below.

get_calibration

returns the hand-eye calibration currently stored on the *rc_reason_stack*.

Details

This service can be called as follows.

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

This service has no arguments.

Response

The field `error` gives the calibration error in pixels which is computed from the translational error `translation_error_meter` and the rotational error `rotation_error_degree`. This value is only given for compatibility with older versions. The translational and rotational errors should be preferred.

Table 6.46: Return codes of the `get_calibration` service call

status	success	Description
0	`true`	returned valid calibration pose
2	`false`	calibration result is not available

The definition for the response with corresponding datatypes is:

```
{
  "name": "get_calibration",
  "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"
  }
}
```

remove_calibration

removes the persistent hand-eye calibration on the *rc_reason_stack*. After this call the `get_calibration` service reports again that no hand-eye calibration is available. This service call will also delete all the stored calibration poses and corresponding camera images in the `slots`.

Details

This service can be called as follows.

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

This service has no arguments.

Response

Table 6.47: Return codes of the `get_calibration` service call

status	success	Description
0	true	removed persistent calibration, device reports as uncalibrated
1	true	no persistent calibration found, device reports as uncalibrated
2	false	could not remove persistent calibration

The definition for the response with corresponding datatypes is:

```
{
  "name": "remove_calibration",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

set_pose

allows to provide a robot pose as calibration pose to the hand-eye calibration routine and records the current image of the calibration grid.

Details

This service can be called as follows.

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

The `slot` argument is used to assign unique numbers to the different calibration poses. The range for `slot` is from 0 to 15. At each instant when `set_pose` is called, an image is recorded. This service call fails if the grid was undetectable in the current image.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
```

(continues on next page)

(continued from previous page)

```
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "slot": "uint32"
  }
}
```

Response

Table 6.48: Return codes of the `set_pose` service call

status	success	Description
1	true	pose stored successfully
3	true	pose stored successfully; collected enough poses for calibration, i.e., ready to calibrate
4	false	calibration grid was not detected, e.g., not fully visible in camera image
8	false	no image data available
12	false	given orientation values are invalid
13	false	invalid slot number

The field `overexposed` indicates if parts of the calibration grid were overexposed in this image.

The definition for the response with corresponding datatypes is:

```
{
  "name": "set_pose",
  "response": {
    "message": "string",
    "overexposed": "bool",
    "status": "int32",
    "success": "bool"
  }
}
```

get_poses

returns the robot poses that are currently stored for the hand-eye calibration routine.

Details

This service can be called as follows.

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

This service has no arguments.

Response

Table 6.49: Return codes of the get_poses service call

status	success	Description
0	true	stored poses are returned
1	true	no calibration pose available

The field overexposed indicates if parts of the calibration grid were overexposed in this image.

The definition for the response with corresponding datatypes is:

```
{
  "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

deletes the calibration poses and corresponding images with the specified slots.

Details

This service can be called as follows.

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

The `slots` argument specifies which calibration poses should be deleted. If no slots are provided, nothing will be deleted.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "slots": [
      "uint32"
    ]
  }
}
```

Response

Table 6.50: Return codes of the `delete_poses` service call

status	success	Description
0	true	poses successfully deleted
1	true	no slots given

The definition for the response with corresponding datatypes is:

```
{
  "name": "delete_poses",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

reset_calibration

deletes all previously provided poses and corresponding images. The last saved calibration result is reloaded. This service might be used to (re-)start the hand-eye calibration from scratch.

Details

This service can be called as follows.

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

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

```
{
  "name": "reset_calibration",
  "response": {
```

(continues on next page)

(continued from previous page)

```
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

calibrate

calculates and returns the hand-eye calibration transformation with the robot poses configured by the `set_pose` service.

Details

`save_calibration` must be called to make the calibration available for other modules via the `get_calibration` service call and to store it persistently.

Note: For calculating the hand-eye calibration transformation at least four robot calibration poses are required (see `set_pose` service). However, eight calibration poses are recommended.

This service can be called as follows.

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

This service has no arguments.

Response

The field `error` gives the calibration error in pixels which is computed from the translational error `translation_error_meter` and the rotational error `rotation_error_degree`. This value is only given for compatibility with older versions. The translational and rotational errors should be preferred.

Table 6.51: Return codes of the `calibrate` service call

status	success	Description
0	true	calibration successful, returned calibration result
1	false	not enough poses to perform calibration
2	false	calibration result is invalid, please verify the input data
3	false	given calibration grid dimensions are not valid
4	false	insufficient rotation, `tcp_offset` and `tcp_rotation_axis` must be specified
5	false	sufficient rotation available, `tcp_rotation_axis` must be set to -1
6	false	poses are not distinct enough from each other

The definition for the response with corresponding datatypes is:

```
{
  "name": "calibrate",
  "response": {
```

(continues on next page)

(continued from previous page)

```
    "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

persistently saves the result of hand-eye calibration to the *rc_reason_stack* and overwrites the existing one. The stored result can be retrieved any time by the `get_calibration` service. This service call will also delete all the stored calibration poses and corresponding camera images in the `slots`.

Details

This service can be called as follows.

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

This service has no arguments.

Response

Table 6.52: Return codes of the `save_calibration` service call

status	success	Description
0	true	calibration saved successfully
1	false	could not save calibration file
2	false	calibration result is not available

The definition for the response with corresponding datatypes is:

```
{
  "name": "save_calibration",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

set_calibration

sets the hand-eye calibration transformation with arguments of this call.

Details

The calibration transformation is expected in the same format as returned by the `calibrate` and `get_calibration` calls. The given calibration information is also stored persistently on the sensor by internally calling `save_calibration`.

This service can be called as follows.

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

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
```

(continues on next page)

(continued from previous page)

```
        "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

Table 6.53: Return codes of the `set_calibration` service call

status	success	Description
0	true	setting the calibration transformation was successful
12	false	given orientation values are invalid

The definition for the response with corresponding datatypes is:

```
{
  "name": "set_calibration",
  "response": {
    "message": "string",
    "status": "int32",
    "success": "bool"
  }
}
```

reset_defaults

restores and applies the default values for this module's parameters ("factory reset"). Does not affect the calibration result itself or any of the `slots` saved during calibration. Only parameters such as the grid dimensions and the mount type will be reset.

Details

This service can be called as follows.

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

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

6.4.2 CollisionCheck

6.4.2.1 Introduction

The CollisionCheck module is an optional on-board module of the *rc_reason_stack* and is licensed with any of the modules *ItemPick* (Section 6.3.4) and *BoxPick* (Section 6.3.5) or *SilhouetteMatch* (Section 6.3.6). Otherwise it requires a separate CollisionCheck *license* (Section 8.2) to be purchased.

The module provides an easy way to check if a gripper is in collision with a load carrier, the point cloud (only in combination with *SilhouetteMatch* (Section 6.3.6)), or other detected objects (only in combination with *SilhouetteMatch* (Section 6.3.6)). It is integrated with the *ItemPick* (Section 6.3.4) and *BoxPick* (Section 6.3.5) and *SilhouetteMatch* (Section 6.3.6) modules, but can be used as standalone product. The models of the grippers for collision checking have to be defined in the *GripperDB* (Section 6.5.3) module.

Warning: Collisions are checked only between the load carrier and the gripper, not the robot itself, the flange, other objects or the item located in the robot gripper. Only if `check_collisions_with_point_cloud` is enabled in the respective detection module, collisions between the gripper and a watertight version of the point cloud will be checked. Only in combination with *SilhouetteMatch* (Section 6.3.6), and only in case the selected template contains a collision geometry and `check_collisions_with_matches` is enabled in the respective detection module, also collisions between the gripper and other *detected* objects will be checked. Collisions with objects that cannot be detected will not be checked.

Note: This module is pipeline specific. Changes to its settings or parameters only affect the respective camera pipeline and have no influence on other pipelines running on the *rc_reason_stack*.

Table 6.54: Specifications of the CollisionCheck module

Collision checking with	detected load carrier, detected objects (only *SilhouetteMatch* (Section 6.3.6)), baseplane (only *SilhouetteMatch*, Section 6.3.6), point cloud (only *SilhouetteMatch* (Section 6.3.6))
Collision checking available in	*ItemPick* (Section 6.3.4) and *BoxPick* (Section 6.3.5), *SilhouetteMatch* (Section 6.3.6)

6.4.2.2 Collision checking

Stand-alone collision checking

The `check_collisions` service call triggers collision checking between the chosen gripper and the provided load carriers for each of the provided grasps. Checking collisions with other objects or the point cloud is not possible with the stand-alone `check_collisions` service. The CollisionCheck module checks if the chosen gripper is in collision with at least one of the load carriers, when the TCP of the gripper is positioned in the grasp position. It is possible to check the collision with multiple load carriers simultaneously. The grasps which are in collision with any of the defined load carriers will be returned as colliding.

The `pre_grasp_offset` can be used for additional collision checking. The pre-grasp offset P_{off} is the offset between the grasp point P_{grasp} and the pre-grasp position P_{pre} in the grasp's coordinate frame (see Fig. 6.25). If the pre-grasp offset is defined, the grasp will be detected as colliding if the gripper is in collision at any point during motion from the pre-grasp position to the grasp position (assuming a linear movement).

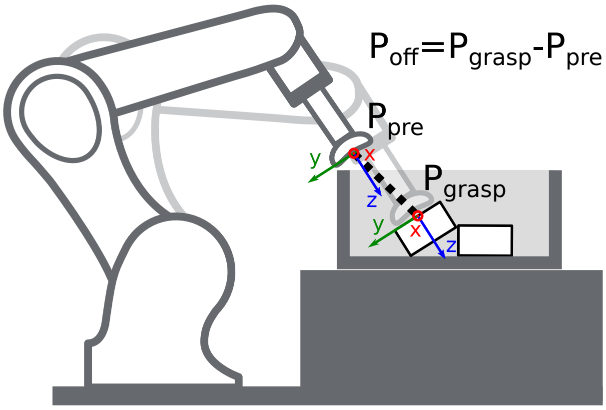

Fig. 6.25: Illustration of the pre-grasp offset parameter for collision checking. In this case, the pre-grasp position as well as the grasp position are collision free. However, the trajectory between these poses would have collisions. Thus, this grasp pose would be marked as colliding.

Collision checking within other modules

Collision checking is integrated in the following modules' services:

- *ItemPick* (Section 6.3.4): `compute_grasps` (see *compute_grasps*, Section 6.3.4.7)
- *BoxPick* (Section 6.3.5): `compute_grasps` (see *compute_grasps*, Section 6.3.5.8)
- *SilhouetteMatch* (Section 6.3.6): `detect_object` (see *detect_object*, Section 6.3.6.11)
- *CADMatch* (Section 6.3.7): `detect_object` (see *detect_object*, Section 6.3.7.10)

Each of these services can take a `collision_detection` argument consisting of the `gripper_id` of the default gripper and the `pre_grasp_offset` as described in the previous section *Stand-alone collision checking* (Section 6.4.2.2). The default gripper given by the `gripper_id` argument is only used for grasp points which do not have an individual gripper ID assigned. When the `collision_detection` argument is given, these services only return the grasps at which the gripper is not in collision or which could not be checked for collisions. When a load carrier ID is provided to these services, collision checking will always be performed between the gripper and the load carrier. Additional collision check features can be enabled depending on the module.

Only for *SilhouetteMatch* (Section 6.3.6), and only in case the selected template contains a collision geometry and `check_collisions_with_matches` is enabled in the respective detection module, grasp points at which the gripper would be in collision with other *detected* objects are also rejected. The object on which the grasp point to be checked is located, is excluded from the collision check.

When a gripper is defined for a grasp point in the object template for *SilhouetteMatch* (Section 6.3.6), then this gripper will be used for collision checking at that specific grasp point instead of the default gripper defined in the `collision_detection` argument of the `detect_object` service (see *Setting of grasp points*, Section 6.3.6.4). The grasps returned by the `detect_object` service contain a flag `collision_checked`, indicating whether the grasp was checked for collisions, and the field `gripper_id`. If `collision_checked` is true, the returned `gripper_id` contains the ID of the gripper that was used for the collision check. That is the ID of the gripper defined for that specific grasp, or, if empty, the gripper that was given in the `collision_detection` argument of the request. If `collision_checked` is false, the returned `gripper_id` is the gripper ID that was defined for that grasp.

In *SilhouetteMatch*, Section 6.3.6, collisions between the gripper and the base plane can be checked, if `check_collisions_with_base_plane` is enabled in SilhouetteMatch.

Collisions between the gripper and a watertight version of the point cloud can be checked if `check_collisions_with_point_cloud` is enabled in the respective module.

> **Warning:** Collisions are checked only between the load carrier and the gripper, not the robot itself, the flange or other objects. Only if `check_collisions_with_point_cloud` is enabled, collisions between the gripper and a watertight version of the point cloud are checked. Only in combination with *SilhouetteMatch* (Section 6.3.6), and only in case the selected template contains a collision geometry and `check_collisions_with_matches` is enabled in the respective detection module, also collisions between the gripper and other *detected* objects are checked. Collisions with objects that cannot be detected will not be checked.
>
> Only in combination with *CADMatch*, Section 6.3.7 and only if `check_collisions_during_retraction` is enabled in CADMatch and a load carrier and a pre-grasp offset are given, collisions between the object in the gripper and the walls of the given load carrier are checked along the linear trajectory from the grasp point to the pre-grasp pose.

The collision-check results are affected by run-time parameters, which are listed and explained further below.

6.4.2.3 Parameters

The CollisionCheck module is called `rc_collision_check` in the REST-API and is represented in the *Web GUI* (Section 7.1) in the desired pipeline under *Configuration* → *CollisionCheck*. The user can explore and configure the `rc_collision_check` module's run-time parameters, e.g. for development and testing, using the Web GUI or the *REST-API interface* (Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.55: The rc_collision_check module's run-time parameters

Name	Type	Min	Max	Default	Description
check_bottom	bool	false	true	true	Whether to enable collision checking with the bottom of the load carrier
check_flange	bool	false	true	true	Whether all grasps with the flange inside the load carrier should be marked as colliding
collision_dist	float64	0.0	0.1	0.01	Minimum distance in meters between any element of the gripper and the load carrier or the base plane (only SilhouetteMatch) for a collision-free grasp

Description of run-time parameters

Each run-time parameter is represented by a row in the Web GUI's *Settings* section in the desired pipeline under *Configuration* → *CollisionCheck*. The name in the Web GUI is given in brackets behind the parameter name:

collision_dist (*Collision Distance*)

Minimal distance in meters between any part of the gripper and the load carrier and/or the base plane (only SilhouetteMatch) for a grasp to be considered collision free.

Note: The collision distance is not applied when checking collisions between the gripper and the point cloud, or the gripper and other detected objects. It is not applied when checking if the flange is inside the load carrier (check_flange), either.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?collision_dist=<value>
```

check_flange (*Check Flange*)

Performs an additional safety check as described in *Robot flange radius* (Section 6.5.3.2). If this parameter is set, all grasps in which any part of the robot's flange is inside the load carrier are marked as colliding.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?check_flange=<value>
```

check_bottom (*Check Bottom*)

When this check is enabled the collisions will be checked not only with the side walls of the load carrier but also with its bottom. It might be necessary to disable this check if the TCP is inside the collision geometry (e.g. is defined inside a suction cup).

The load carrier bottom will always be excluded for the collision check between the object in the gripper and the load carrier during retraction in combination with *CADMatch*, Section 6.3.7 when `check_collisions_during_retraction` is enabled.

Via the REST-API, this parameter can be set as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/parameters?check_bottom=<value>
```

6.4.2.4 Status values

The `rc_collision_check` module reports the following status values:

Table 6.56: The `rc_collision_check` module status values

Name	Description
`last_evaluated_grasps`	Number of evaluated grasps
`last_collision_free_grasps`	Number of collision-free grasps
`collision_check_time`	Collision checking runtime

6.4.2.5 Services

The user can explore and call the `rc_collision_check` module's services, e.g. for development and testing, using *REST-API interface* (Section 7.2) or the *rc_reason_stack Web GUI* (Section 7.1).

The CollisionCheck module offers the following services.

reset_defaults

Resets all parameters of the module to its default values, as listed in above table.

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:

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

check_collisions (deprecated)

Triggers a collision check between a gripper and a load carrier.

Details

This service can be called as follows.

API version 2

```
PUT http://<host>/api/v2/pipelines/<0,1,2,3>/nodes/rc_collision_check/services/check_
↪collisions
```

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/check_collisions
```

Request

Required arguments:

> grasps: list of grasps that should be checked.
>
> load_carriers: list of load carriers against which the collision should be checked. The fields of the load carrier definition are described in *Detection of load carriers* (Section 6.3.2.2). The position frame of the grasps and load carriers has to be the same.
>
> gripper_id: the id of the gripper that is used to check the collisions. The gripper has to be configured beforehand.

Optional arguments:

> pre_grasp_offset: the offset in meters from the grasp position to the pre-grasp position in the grasp frame. If this argument is set, the collisions will not only be checked in the grasp point, but also on the path from the pre-grasp position to the grasp position (assuming a linear movement).

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
```

(continues on next page)

(continued from previous page)

```
          "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": {
        "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`: list of grasps in collision with one or more load carriers.

`collision_free_grasps`: list of collision-free grasps.

`return_code`: holds possible warnings or error codes and messages.

The definition for the response with corresponding datatypes is:


```
{
  "name": "check_collisions",
  "response": {
    "colliding_grasps": [
      {
        "pose": {
          "orientation": {
            "w": "float64",
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
          "position": {
            "x": "float64",
            "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` (deprecated)

Persistently stores a gripper on the *rc_reason_stack*.

API version 2

This service is not available in API version 2. Use *set_gripper* (Section 6.5.3.3) in `rc_gripper_db` instead.

API version 1 (deprecated)

This service can be called as follows.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/set_gripper
```

The definitions of the request and response are the same as described in *set_gripper* (Section 6.5.3.3) in rc_gripper_db.

get_grippers (deprecated)

Returns the configured grippers with the requested gripper_ids.

API version 2

This service is not available in API version 2. Use *get_grippers* (Section 6.5.3.3) in rc_gripper_db instead.

API version 1 (deprecated)

This service can be called as follows.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/get_grippers
```

The definitions of the request and response are the same as described in *get_grippers* (Section 6.5.3.3) in rc_gripper_db.

delete_grippers (deprecated)

Deletes the configured grippers with the requested gripper_ids.

API version 2

This service is not available in API version 2. Use *delete_grippers* (Section 6.5.3.3) in rc_gripper_db instead.

API version 1 (deprecated)

This service can be called as follows.

```
PUT http://<host>/api/v1/nodes/rc_collision_check/services/delete_grippers
```

The definitions of the request and response are the same as described in *delete_grippers* (Section 6.5.3.3) in rc_gripper_db.

6.4.2.6 Return codes

Each service response contains a return_code, which consists of a value plus an optional message. A successful service returns with a return_code value of 0. Negative return_code values indicate that the service failed. Positive return_code values indicate that the service succeeded with additional information. The smaller value is selected in case a service has multiple return_code values, but all messages are appended in the return_code message.

The following table contains a list of common codes:

Table 6.57: Return codes of the CollisionCheck services

Code	Description
0	Success
-1	An invalid argument was provided
-7	Data could not be read or written to persistent storage
-9	No valid license for the module
-10	New gripper could not be added as the maximum storage capacity of grippers has been exceeded
10	The maximum storage capacity of grippers has been reached
11	Existing gripper was overwritten

6.4.3 Camera calibration

The camera calibration module is a base module which is available on every *rc_reason_stack*.

Note: This module is pipeline specific. Changes to its settings or parameters only affect the respective camera pipeline and have no influence on other pipelines running on the *rc_reason_stack*.

To use the camera as measuring instrument, camera parameters such as focal length, lens distortion, and the relationship of the cameras to each other must be exactly known. The parameters are determined by calibration and used for image rectification (see *Rectification*, Section 6.1.1), which is the basis for all other image processing modules.

The camera calibration module is responsible for checking calibration and calibrating.

6.4.3.1 Calibration process

Manual calibration can be done through the *Web GUI* (Section 7.1) in the desired pipeline under *Configuration* → *Camera Calibration*. This page provides a wizard to guide the user through the calibration process.

During calibration, the calibration grid must be detected in different poses. When holding the calibration grid, make sure that all black squares of the grid are completely visible and not occluded in both camera images. A green check mark overlays each correctly detected square. The correct detection of the grid is only possible if all of the black squares are detected. Some of the squares not being detected, or being detected only briefly might indicate bad lighting conditions, or a damaged grid. Squares in overexposed parts of the calibration grid are highlighted in red. In this case, the lighting conditions or exposure setting must be adjusted. A thick green border around the calibration grid indicates that it was detected correctly in both camera images.

Calibration settings

The quality of camera calibration heavily depends on the quality of the calibration grid. Calibration grids can be obtained from Roboception.

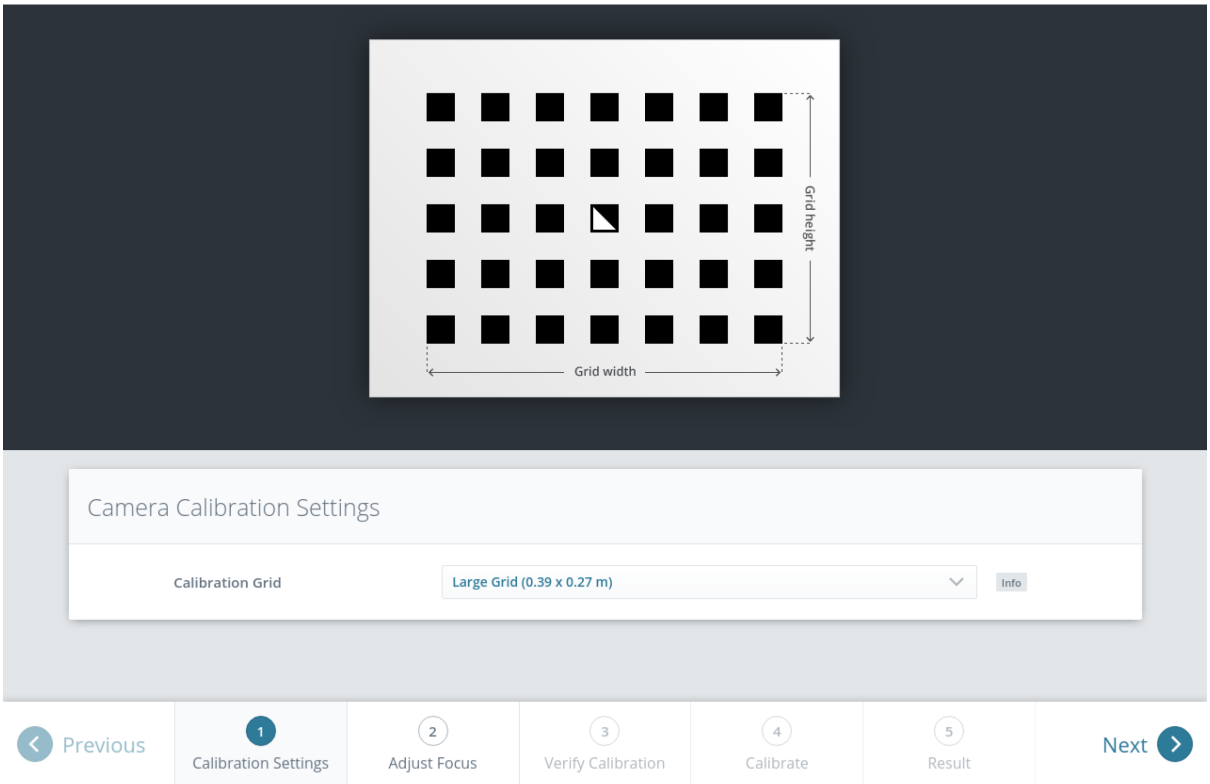

Fig. 6.26: Calibration settings

In the first step, the calibration grid must be specified. The *Next* button proceeds to the next step.

Adjust focus

Note: This step is omitted on *rc_visard* pipelines.

In this step, the focus of the cameras can be adjusted. For this, the grid must be held such that it is simultaneously visible in both cameras. After the grid is detected, the green bars at the right image borders indicate the blur of the image. Adjust the focus of each camera so that the bar in each image is minimal.

Note: While calibrating an *rc_viscore*, the camera exposure settings are temporarily changed to values that allow for easier calibration. The exposure settings can still be changed and will be reset when the calibration is done or cancelled.

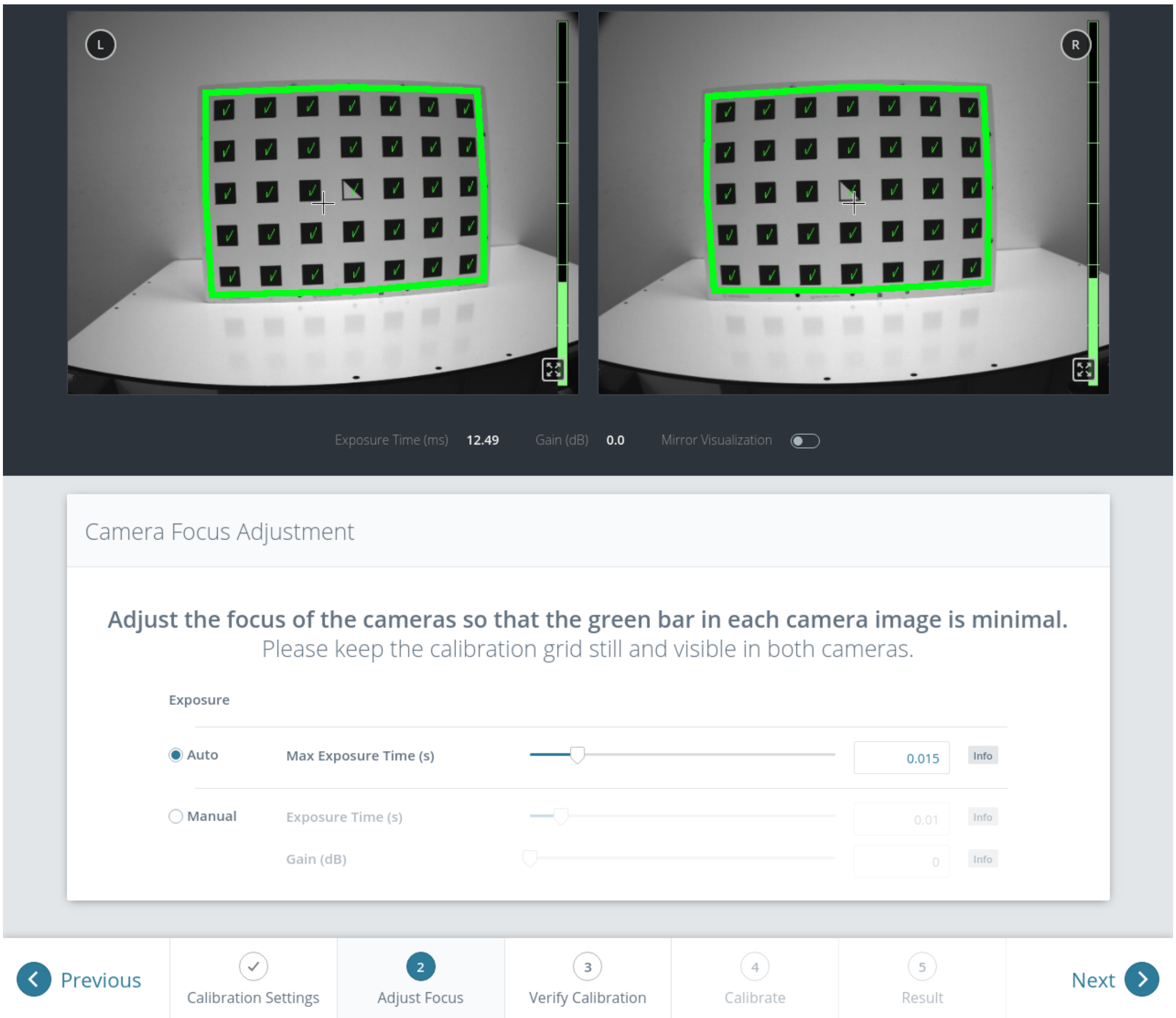

Fig. 6.27: Adjust the focus of each camera

Verify calibration

In the next step, the current calibration can be verified. To perform the verification, the grid must be held such that it is simultaneously visible in both cameras. When the grid is detected, the calibration error is automatically computed and the result is displayed on the screen.

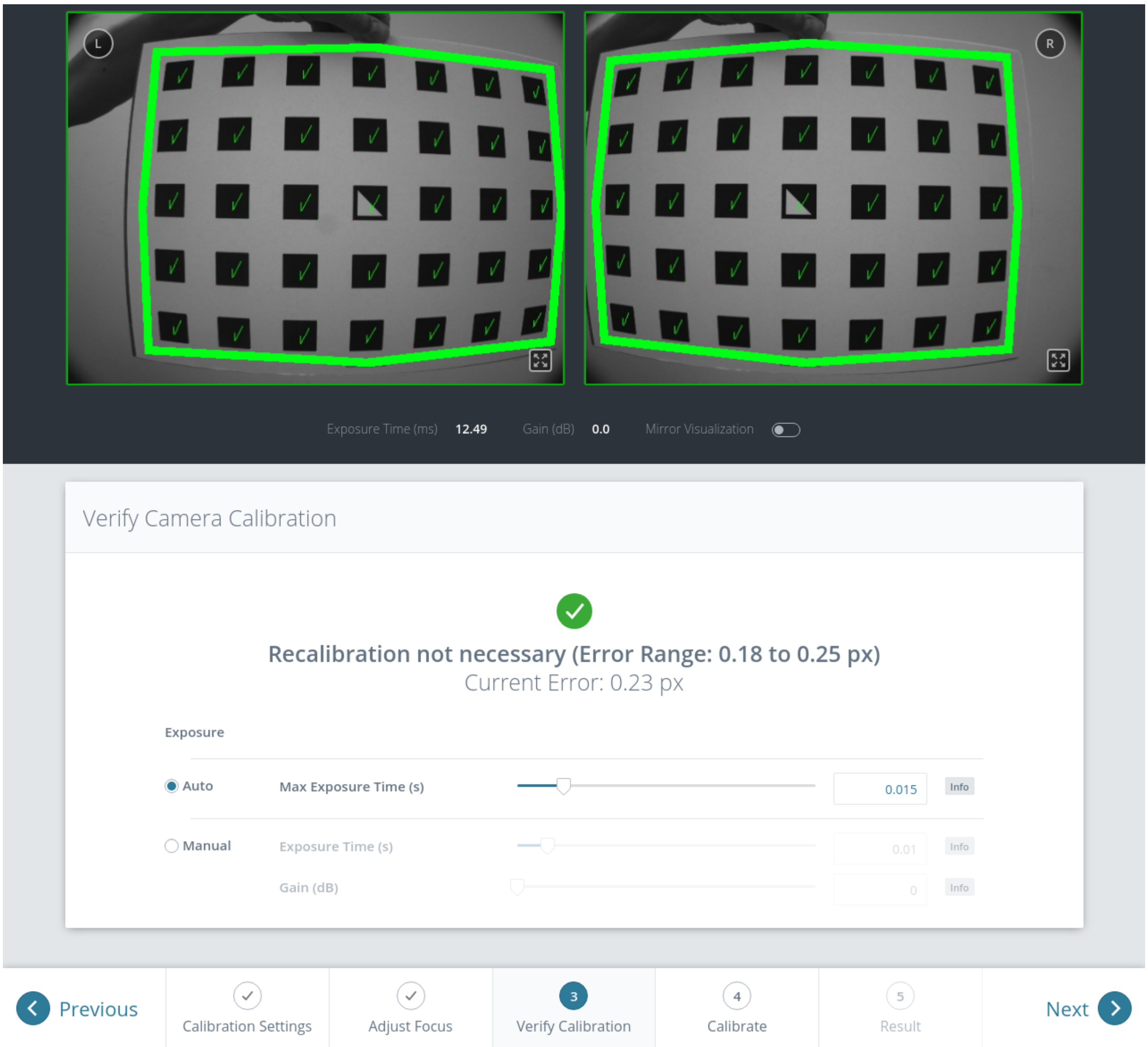

Fig. 6.28: Verification of calibration

Note: To compute a meaningful calibration error, the grid should be held as close as possible to the cameras. If the grid only covers a small section of the camera images, the calibration error will always be less than when the grid covers the full image. For this reason, the minimal and maximal calibration error during verification are shown in addition to the calibration error at the current grid position.

The typical calibration error is below 0.2 pixels. If the error is in this range, then the calibration procedure can be skipped. If the calibration error is greater, the calibration procedure should be performed to guarantee full sensor performance. The button *Next* starts the procedure.

Warning: A large error during verification can be due to miscalibrated cameras, an inaccurate calibration grid, or wrong grid width or height. In case you use a custom calibration grid, please make sure that the grid is accurate and the entered grid width and height are correct. Otherwise, manual calibration will actually decalibrate the cameras!

Calibrate

The camera's exposure time should be set appropriately before starting the calibration. To achieve good calibration results, the images should be well-exposed and motion blur should be avoided. Thus, the maximum auto-exposure time should be as short as possible, but still allow a good exposure. The current exposure time is displayed below the camera images as shown in Fig. 6.30.

Full calibration consists of calibrating each camera individually (monocalibration) and then performing a stereo calibration to determine the relationship between them. In most cases, the intrinsic calibration of each camera does not get corrupted. For this reason, monocalibration is skipped by default during a recalibration, but can be performed by clicking *Perform Monocalibration* in the *Calibrate* tab. This should only be done if the result of the stereo calibration is not satisfactory.

Stereo calibration

During stereo calibration, both cameras are calibrated to each other to find their relative rotation and translation.

The camera images can also be displayed mirrored to simplify the correct positioning of the calibration grid.

First, the grid should be held as close as possible to the camera and very still. It must be fully visible in both images and the cameras should look perpendicularly onto the grid. If the grid is not perpendicular to the line of sight of the cameras, this will be indicated by small green arrows pointing to the expected positions of the grid corners (see Fig. 6.29).

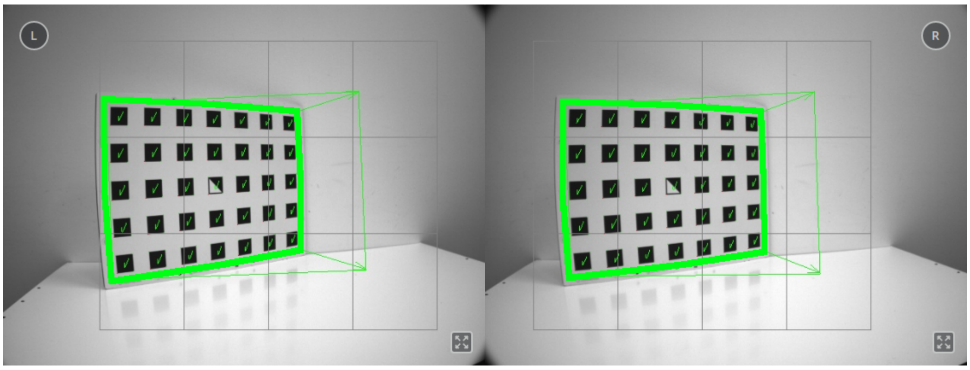

Fig. 6.29: Arrows indicating that the grid is not perpendicular to the camera's line of sight during stereo calibration

The grid must be kept very still for detection. If motion blur occurs, the grid will not be detected. All grid cells that are drawn onto the image have to be covered by the calibration grid. This is visualized by filling the covered cells in green (see Fig. 6.30).

Depending on the camera, the grid has to be held at different positions until all grid cells have been covered and filled in green.

Fig. 6.30: Stereo calibration: Hold the grid as close as possible to fill all visualized cells

Note: If the check marks on the calibration grid all vanish, then either the camera does not look perpendicularly onto the grid, or the grid is too far away from the camera.

Once all grid cells are covered, they disappear and a single far cell is visualized. Now, the grid should be held as far as possible from the cameras, so that the small cell is covered. Arrows will indicate if the grid is still too close to the camera. When the grid is successfully detected at the far pose, the cell is filled in green and the result can be computed (see Fig. 6.31).

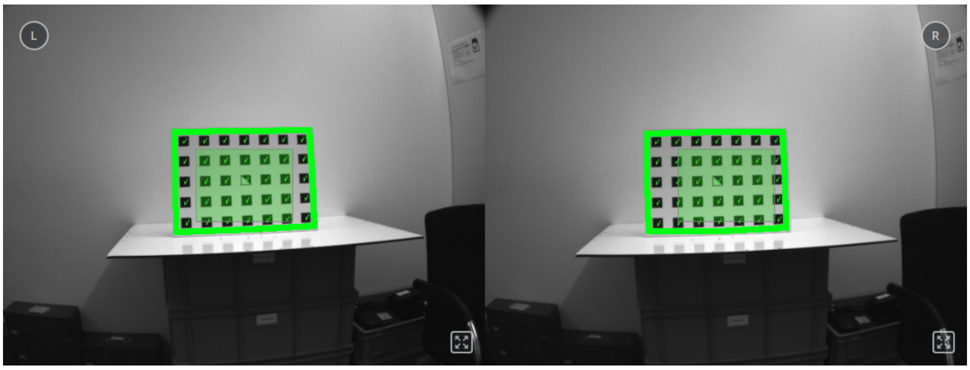

Fig. 6.31: Holding the grid far away during stereo calibration

If stereo calibration yields an unsatisfactory calibration error, then calibration should be repeated with monocalibration (see next Section *Monocalibration*).

Monocalibration

Monocalibration is the intrinsic calibration of each camera individually. Since the intrinsic calibration normally does not get corrupted, the monocalibration should only be performed if the result of stereo calibration is not satisfactory.

Click *Perform Monocalibration* in the *Calibrate* tab to start monocalibration.

For monocalibration, the grid has to be held in certain poses. The arrows from the grid corners to the green areas indicate that all grid corners should be placed inside the green areas. The green areas are called sensitive areas. The *Size of Sensitive Area* slider can control their size to ease calibration. However, please be aware that increasing their size too much may result in slightly lower calibration accuracy.

Holding the grid upside down is a common mistake made during calibration. Spotting this in this case is easy because the green lines from the grid corners into the green areas will cross each other as shown in Fig. 6.32.

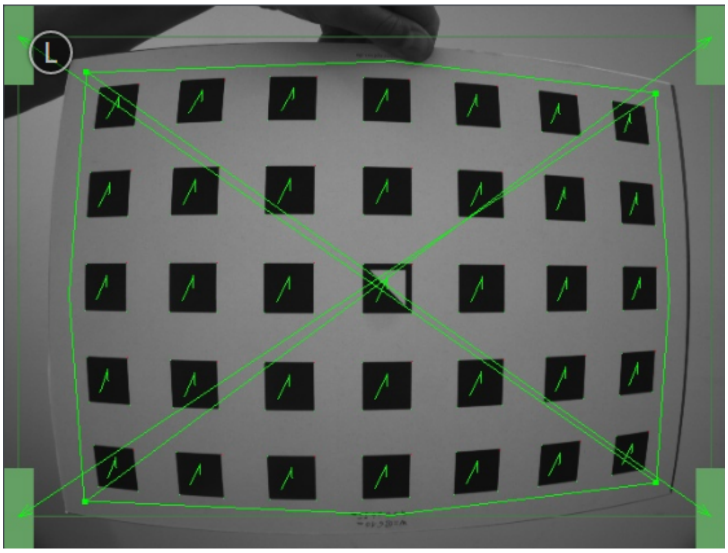

Fig. 6.32: Wrongly holding the grid upside down leads to crossed green lines.

> **Note:** Calibration might appear cumbersome as it involves holding the grid in certain predefined poses. However, these poses are required to ensure an unbiased, high-quality calibration result.

The monocalibration process involves five poses for each camera as shown in Fig. 6.33.

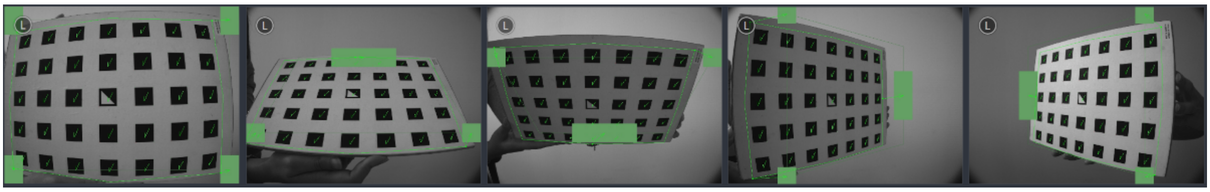

Fig. 6.33: Poses required for monocamera calibration

After the corners or sides of the grid are placed on top of the sensitive areas, the process automatically shows the next pose required. When the process is finished for the left camera, the same procedure is repeated for the right one.

Continue with the guidelines given in the previous Section *Stereo calibration*.

Storing the calibration result

Clicking the *Compute Calibration* button finishes the process and displays the final result. The indicated result is the mean reprojection error of all calibration points. It is given in pixels and typically has a value below 0.2.

Pressing *Save Calibration* applies the calibration and saves it to the device.

Note: The given result is the minimum error left after calibration. The real error is definitely not less than this, but could in theory be larger. This is true for every camera-calibration algorithm and the reason why we enforce holding the grid in very specific poses. Doing so ensures that the real calibration error cannot significantly exceed the reported error.

Warning: If a hand-eye calibration was stored on the *rc_reason_stack* before camera calibration, the hand-eye calibration values could have become invalid. Please repeat the hand-eye calibration procedure.

6.4.3.2 Parameters

The module is called `rc_stereocalib` in the REST-API.

Note: The camera calibration module's available parameters and status values are for internal use only and may change in the future without further notice. Calibration should only be performed through the Web GUI as described above.

6.4.3.3 Services

Note: The camera calibration module's available service calls are for internal use only and may change in the future without further notice. Calibration should only be performed through the Web GUI as described above.

6.4.4 IO and Projector Control

The IOControl module is a base module which is available on every *rc_reason_stack*.

The IOControl module allows reading the status of the general purpose digital inputs and controlling the digital general purpose outputs (GPIOs) of the camera. The outputs can be set to LOW or HIGH, or configured to be HIGH for the exposure time of every image or every second image.

Note: This module is pipeline specific. Changes to its settings or parameters only affect the respective camera pipeline and have no influence on other pipelines running on the *rc_reason_stack*.

The purpose of the IOControl module is the control of an external light source or a projector, which is connected to one of the camera's GPIOs to be synchronized by the image acquisition trigger. In case a pattern projector is used to improve stereo matching, the intensity images also show the projected pattern, which might be a disadvantage for image processing tasks that are based on the intensity image (e.g. edge detection). For this reason, the IOControl module allows setting GPIO outputs to HIGH for the exposure time of every second image, so that intensity images without the projected pattern are also available.

6.4.4.1 Parameters

The IOControl module is called `rc_iocontrol` in the REST-API and is represented in the *Web GUI* (Section 7.1) in the desired pipeline under *Configuration* → *IOControl*.

The user can change the parameters via the Web GUI or the *REST-API interface* (Section 7.2).

Parameter overview

This module offers the following run-time parameters:

Table 6.58: The `rc_iocontrol` module's run-time parameters

Name	Type	Min	Max	Default	Description
`out1_inverted`	bool	false	true	false	Inverting out1
`out1_mode`	string	-	-	Low	Out1 mode: [Low, High, ExposureActive, ExposureAlternateActive]
`out1_ratio`	float64	0.0	1.0	1.0	Ratio of exposure time that Out1 is high in ExposureActive and ExposureAlternateActive mode
`out2_inverted`	bool	false	true	false	Inverting out2
`out2_mode`	string	-	-	Low	Out2 mode: [Low, High, ExposureActive, ExposureAlternateActive]
`out2_ratio`	float64	0.0	1.0	1.0	Ratio of exposure time that Out2 is high in ExposureActive and ExposureAlternateActive mode

Description of run-time parameters

`out1_mode` and `out2_mode` (*Out1* / *Projector* and *Out2*)

The output modes for GPIO Out 1 and Out 2 can be set individually:

`Low` sets the output permanently to LOW. This is the factory default.

`High` sets the output permanently to HIGH.

`ExposureActive` sets the output to HIGH for the exposure time of every image.

ExposureAlternateActive sets the output to HIGH for the exposure time of every second image.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/parameters?<out1_mode|out2_mode>=<value>
```

Fig. 6.34 shows which images are used for stereo matching and transmission via GigE Vision in ExposureActive mode with a user-defined frame rate of 8 Hz.

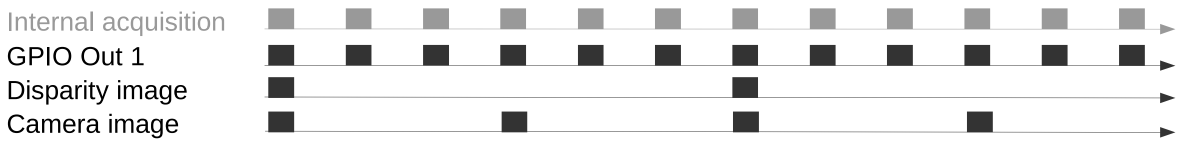

Fig. 6.34: Example of using the ExposureActive mode for GPIO Out 1 with a user-defined frame rate of 8 Hz. The internal image acquisition is always 25 Hz. GPIO Out 1 is HIGH for the exposure time of every image. A disparity image is computed for camera images that are sent out via GigE Vision according to the user-defined frame rate.

The mode ExposureAlternateActive is meant to be used when an external random dot projector is connected to the camera's GPIO Out 1. When setting Out 1 to ExposureAlternateActive, the *stereo matching* (Section 6.2.2) module only uses images with GPIO Out 1 being HIGH, i.e. projector is on. The maximum frame rate that is used for stereo matching is therefore half of the frame rate configured by the user. All modules which make use of the intensity image, like *TagDetect* (Section 6.3.3) and *ItemPick* (Section 6.3.4), use the intensity images with GPIO Out 1 being LOW, i.e. projector is off. Fig. 6.35 shows an example.

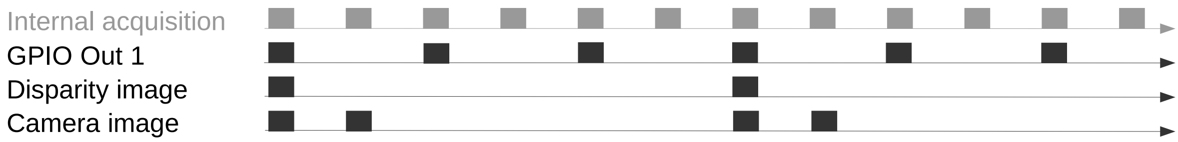

Fig. 6.35: Example of using the ExposureAlternateActive mode for GPIO Out 1 with a user-defined frame rate of 8 Hz. The internal image acquisition is always 25 Hz. GPIO Out 1 is HIGH for the exposure time of every second image. A disparity image is computed for images where Out 1 is HIGH and that are sent out via GigE Vision according to the user-defined frame rate. In ExposureAlternateActive mode, intensity images are always transmitted pairwise: one with GPIO Out 1 HIGH, for which a disparity image might be available, and one with GPIO Out 1 LOW.

Note: In ExposureAlternateActive mode, an intensity image with GPIO Out 1 being HIGH (i.e. with projection) is always 40 ms away from an intensity image with Out 1 being LOW (i.e. without projection), regardless of the user-defined frame rate. This needs to be considered when synchronizing disparity images and camera images without projection in this special mode.

out1_ratio and out2_ratio (*Out1 Ratio* and *Out2 Ratio*)

The output ratios for GPIOs Out 1 and Out 2 determine how much of the image exposure time the corresponding output GPIO should be HIGH, when ExposureActive or ExposureAlternateActive are used. When the ratio is set to 1, the output will be HIGH for full amount of exposure time. In case a projector is connected to the GPIO out1, a lower out1_ratio leads to darker projections.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/parameters?<out1_ratio|out2_ratio>=<value>
```

out1_inverted and out2_inverted (*Invert Out1* and *Invert Out2*)

The `out1_inverted` and `out2_inverted` parameters determine whether the corresponding outputs should be inverted.

Via the REST-API, this parameter can be set as follows.

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 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/parameters?<out1_inverted|out2_inverted>=
↪<value>
```

6.4.4.2 Services

Each service response contains a `return_code`, which consists of a `value` plus an optional `message`. A successful service returns with a `return_code` value of `0`. Negative `return_code` values indicate that the service failed. Positive `return_code` values indicate that the service succeeded with additional information.

The IOControl module offers the following services.

get_io_values

Retrieves the current state of the camera's general purpose inputs and outputs (GPIOs).

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

```
PUT http://<host>/api/v1/nodes/rc_iocontrol/services/get_io_values
```

Request

This service has no arguments.

Response

The returned `timestamp` is the time of measurement.

`input_mask` and `output_mask` are bit masks defining which bits are used for input and output values, respectively.

`values` holds the values of the bits corresponding to input and output as given by the `input_mask` and `output_mask`.

`return_code` holds possible warnings or error codes and messages. Possible `return_code` values are shown below.

Code	Description
0	Success
-2	Internal error
-9	License for *IOControl* is not available

The definition for the response with corresponding datatypes is:

```
{
  "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

Restores and applies the default values for this module's parameters ("factory reset").

Details

This service can be called as follows.

API version 2

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

API version 1 (deprecated)

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

Request

This service has no arguments.

Response

The definition for the response with corresponding datatypes is:


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


6.5 Database modules

The *rc_reason_stack* provides several database modules which enable the user to configure global data which is used in many detection modules, such as load carriers and regions of interest. Via the *REST-API interface* (Section 7.2) the database modules are only available in API version 2.

The database modules are:

- ***LoadCarrierDB*** **(`rc_load_carrier_db`, Section 6.5.1)** allows setting, retrieving and deleting load carriers.
- ***RoiDB*** **(`rc_roi_db` , Section 6.5.2)** allows setting, retrieving and deleting 2D and 3D regions of interest.
- ***GripperDB*** **(`rc_gripper_db`, Section 6.5.3)** allows setting, retrieving and deleting grippers for collision checking.

These modules are global on the *rc_reason_stack*, which means that they run outside the camera pipelines. Changes to their settings or parameters affect all pipelines running on the *rc_reason_stack*.

6.5.1 LoadCarrierDB

6.5.1.1 Introduction

The LoadCarrierDB module (Load carrier database module) allows the global definition of load carriers, which can then be used in many detection modules. The specified load carriers are available for all modules supporting load carriers on the *rc_reason_stack*.

Note: This module is global on the *rc_reason_stack*. Changes to its settings or parameters affect every camera pipeline running on the *rc_reason_stack*.

The LoadCarrierDB module is a base module which is available on every *rc_reason_stack*.

Table 6.59: Specifications of the LoadCarrierDB module

Supported load carrier types	4-sided or 3-sided
Supported rim types	solid rim, stepped rim or ledged rim
Min. load carrier dimensions	0.1 m x 0.1 m x 0.05 m
Max. load carrier dimensions	5 m x 5 m x 5 m
Max. number of load carriers	50
Load carriers available in	*ItemPick* (Section 6.3.4) and *BoxPick* (Section 6.3.5) and *SilhouetteMatch* (Section 6.3.6)
Supported pose types	no pose, orientation prior, exact pose
Supported reference frames	camera, external

6.5.1.2 Load carrier definition

A load carrier (bin) is a container with four walls, a floor and a rectangular rim, which can contain objects. It can be used to limit the volume in which to search for objects or grasp points.

A load carrier is defined by its `outer_dimensions` and `inner_dimensions`. The maximum `outer_dimensions` are 5.0 meters in every dimension.

The origin of the load carrier reference frame is in the center of the load carrier's *outer* box and its z axis is perpendicular to the load carrier's floor pointing outwards (see Fig. 6.36).

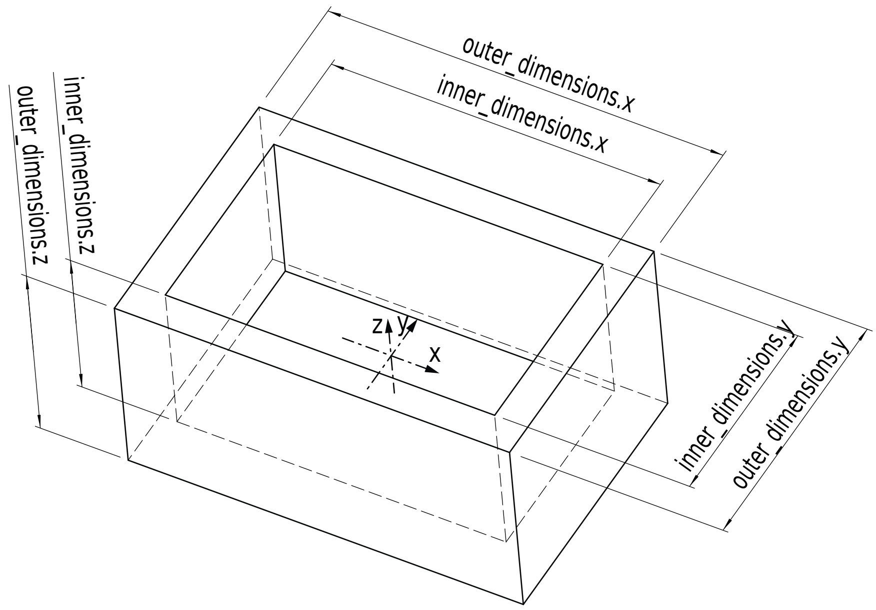

Fig. 6.36: Load carrier with reference frame and inner and outer dimensions

Note: Typically, outer and inner dimensions of a load carrier are available in the specifications of the load carrier manufacturer.

The inner volume of the load carrier is defined by its inner dimensions, but includes a region of 10 cm height above the load carrier, so that also items protruding from the load carrier are considered for detection or grasp computation. Furthermore, an additional `crop_distance` is subtracted from the inner volume in every dimension, which acts as a safety margin and can be configured as run-time parameter in the LoadCarrier module (see *Parameters*, Section 6.3.2.5). Fig. 6.37 visualizes the inner volume of a load carrier. Only points which are inside this volume are considered for detections.

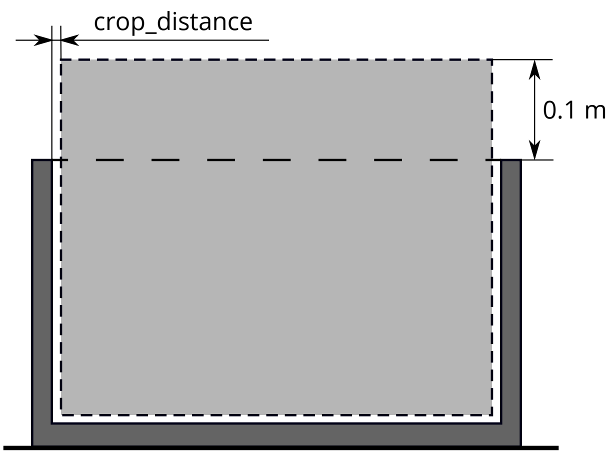

Fig. 6.37: Visualization of the inner volume of a load carrier. Only points which are inside this volume are considered for detections.

Since the load carrier detection is based on the detection of the load carrier's rim, the rim geometry must be specified if it cannot be determined from the difference between outer and inner dimensions. A load carrier with a stepped rim can be defined by setting a `rim_thickness`. The rim thickness gives the thickness of the outer part of the rim in the x and y direction. When a rim thickness is given, an optional `rim_step_height` can also be specified, which gives the height of the step between the outer and the inner part of the rim. When the step height is given, it will also be considered during collision checking (see *CollisionCheck*, Section 6.4.2). Examples of load carriers with stepped rims are shown in Fig. 6.38 A, B. In addition to the `rim_thickness` and `rim_step_height` the `rim_ledge` can be specified for defining load carriers whose inner rim protrudes into the interior of the load carrier, such as pallet cages. The `rim_ledge` gives the thickness of the inner part of the rim in the x and y direction. An example of a load carrier with a ledged rim is shown in Fig. 6.38 C.

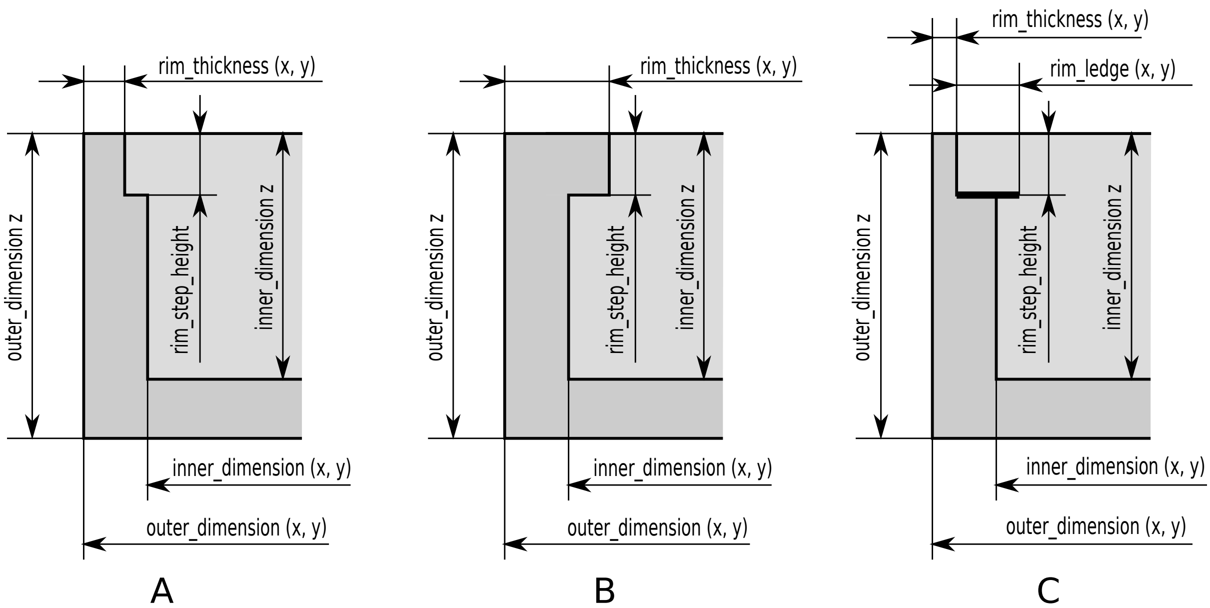

Fig. 6.38: Examples of load carriers with stepped rim (A, B) or ledged rim (C)

The different rim types are applicable to both, standard 4-sided and 3-sided load carriers. For a 3-sided load carrier, the `type` must be `THREE_SIDED`. If the type is set to `STANDARD` or left empty, a 4-sided load carrier is specified. A 3-sided load carrier has one side that is lower than the other three sides. This `height_open_side` is measured from the outer bottom of the load carrier. The open side is at the negative y-axis of the load carrier's coordinate system. Examples of the two load carrier types are given in Fig. 6.39. The height of the lower side is only considered during collision checking and not required for the detection of the load carrier.

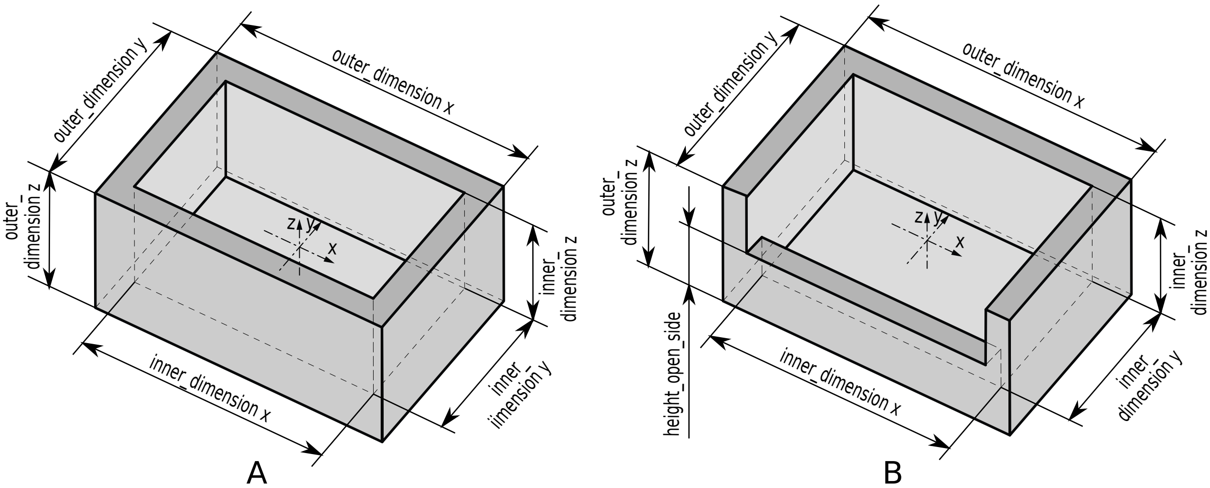

Fig. 6.39: Examples of a standard 4-sided load carrier (A) and a 3-sided load carrier (B)

A load carrier can be specified with a full 3D `pose` consisting of a `position` and an `orientation` quaternion, given in a `pose_frame`. Based on the given `pose_type` this pose is either used as an orientation prior (`pose_type` is `ORIENTATION_PRIOR` or empty), or as the exact pose of the load carrier (`pose_type` is `EXACT_POSE`).

In case the pose serves as orientation prior, the detected load carrier pose is guaranteed to have the minimum rotation with respect to the load carrier's prior pose. This pose type is useful for detecting tilted load carriers and for resolving the orientation ambiguity in the x and y direction caused by the symmetry of the load carrier model.

In case the pose type is set to `EXACT_POSE`, no load carrier detection will be performed on the scene data, but the given pose will be used in exactly the same way as if the load carrier is detected at that pose. This pose type is especially useful in cases where load carriers do not change their positions and/or are hard to detect (e.g. because their rim is too thin or the material is too shiny).

The *rc_reason_stack* can persistently store up to 50 different load carrier models, each one identified by a different `id`. The configuration of a load carrier model is normally performed offline, during the set up the desired application. This can be done via the *REST-API interface* (Section 7.2) or in the *rc_reason_stack* Web GUI.

Note: The configured load carrier models are persistent even over firmware updates and rollbacks.

6.5.1.3 Load carrier compartments

Some detection modules can make use of a `load_carrier_compartment` to further limit the volume for the detection, for example *ItemPick's compute_grasps service* (see 6.3.4.7). A load carrier compartment is a box whose `pose` is defined as the transformation from the load carrier reference frame to the compartment reference frame, which is located in the center of the compartment box (see Fig. 6.40). The load carrier compartment is defined for each detection call separately and is not part of the load carrier definition in the LoadCarrierDB module.

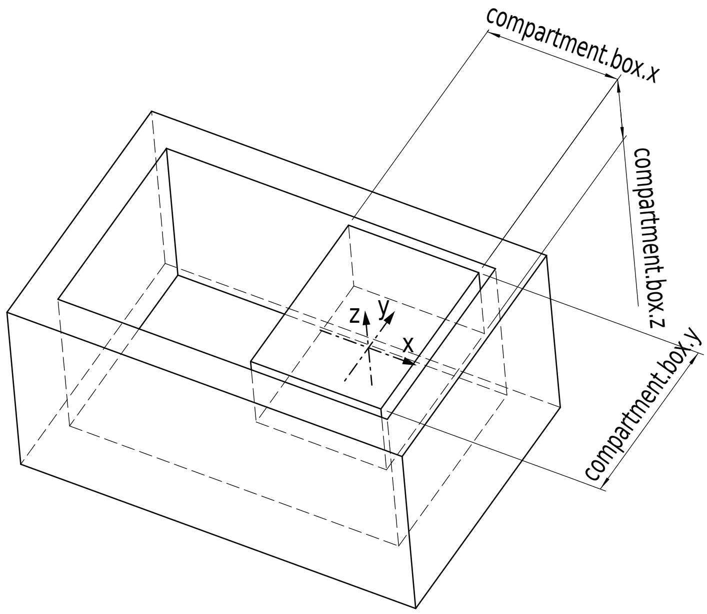

Fig. 6.40: Sample compartment inside a load carrier. The coordinate frame shown in the image is the reference frame of the compartment.

The compartment volume is intersected with the load carrier inner volume to compute the volume for the detection. If this intersection should also contain the 10 cm region above the load carrier, the height of the compartment box must be increased accordingly.

6.5.1.4 Interaction with other modules

Internally, the LoadCarrierDB module depends on, and interacts with other on-board modules as listed below.

Hand-eye calibration

In case the camera has been calibrated to a robot, the load carrier's exact pose or orientation prior can be provided in the robot coordinate frame by setting the corresponding `pose_frame` argument to `external`.

Two different `pose_frame` values can be chosen:

1. **Camera frame** (`camera`). The load carrier pose or orientation prior is provided in the camera frame, and no prior knowledge about the pose of the camera in the environment is required. This means that the configured load carriers move with the camera. It is the user's responsibility to update the configured poses if the camera frame moves (e.g. with a robot-mounted camera).
2. **External frame** (`external`). The load carrier pose or orientation prior is provided in the external frame, configured by the user during the hand-eye calibration process. The module relies on the on-board *Hand-eye calibration module* (Section 6.4.1) to retrieve the sensor mounting (static or robot mounted) and the hand-eye transformation.

Note: If no hand-eye calibration is available, all `pose_frame` values should be set to `camera`.

All `pose_frame` values that are not `camera` or `external` are rejected.

6.5.1.5 Services

The LoadCarrierDB module is called `rc_load_carrier_db` in the REST-API and is represented in the *Web GUI* (Section 7.1) under *Database* → *Load Carriers*. The user can explore and call the LoadCarrierDB module's services, e.g. for development and testing, using the *REST-API interface* (Section 7.2) or the Web GUI.

The LoadCarrierDB module offers the following services.

`set_load_carrier`

Persistently stores a load carrier on the *rc_reason_stack*. All configured load carriers are persistent over firmware updates and rollbacks.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_load_carrier_db/services/set_load_carrier
```

Request

Details for the definition of the `load_carrier` type are given in *Load carrier definition* (Section 6.5.1.2).

The field `type` is optional and accepts `STANDARD` and `THREE_SIDED`.

The field `pose_type` is optional and accepts `NO_POSE`, `EXACT_POSE` and `ORIENTATION_PRIOR`.

The definition for the request arguments with corresponding datatypes is:

```
{
  "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": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "pose_type": "string",
      "rim_ledge": {
        "x": "float64",
```

(continues on next page)

(continued from previous page)

```
        "y": "float64"
      },
      "rim_step_height": "float64",
      "rim_thickness": {
        "x": "float64",
        "y": "float64"
      },
      "type": "string"
    }
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

get_load_carriers

Returns the configured load carriers with the requested `load_carrier_ids`. If no `load_carrier_ids` are provided, all configured load carriers are returned.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_load_carrier_db/services/get_load_carriers
```

Request

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "load_carrier_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

(continues on next page)

(continued from previous page)

```
          "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

Deletes the configured load carriers with the requested `load_carrier_ids`. All load carriers to be deleted must be explicitly stated in `load_carrier_ids`.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_load_carrier_db/services/delete_load_carriers
```

Request

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
```

(continues on next page)

(continued from previous page)

```
    "load_carrier_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

6.5.1.6 Return codes

Each service response contains a `return_code`, which consists of a `value` plus an optional `message`. A successful service returns with a `return_code` value of `0`. Negative `return_code` values indicate that the service failed. Positive `return_code` values indicate that the service succeeded with additional information. The smaller value is selected in case a service has multiple `return_code` values, but all messages are appended in the `return_code` message.

The following table contains a list of common codes:

Table 6.60: Return codes of the LoadCarrierDB module's services

Code	Description
0	Success
-1	An invalid argument was provided
-10	New element could not be added as the maximum storage capacity of load carriers has been exceeded
10	The maximum storage capacity of load carriers has been reached
11	An existent persistent model was overwritten by the call to `set_load_carrier`

6.5.2 RoiDB

6.5.2.1 Introduction

The RoiDB module (region of interest database module) allows the global definition of 2D and 3D regions of interest, which can then be used in many detection modules. The ROIs are available for all modules supporting 2D or 3D ROIs on the *rc_reason_stack*.

Note: This module is global on the *rc_reason_stack*. Changes to its settings or parameters affect every camera pipeline running on the *rc_reason_stack*.

The RoiDB module is a base module which is available on every *rc_reason_stack*.

3D ROIs can be used in *CADMatch* (Section 6.3.7), *ItemPick* (Section 6.3.4) and *BoxPick* (Section 6.3.5). 2D ROIs can be used in *SilhouetteMatch* (Section 6.3.6), and *LoadCarrier* (Section 6.3.2).

Table 6.61: Specifications of the RoiDB module

Supported ROI types	2D, 3D
Supported ROI geometries	2D ROI: rectangle, 3D ROI: box, sphere
Max. number of ROIs	2D: 100, 3D: 100
ROIs available in	2D: *SilhouetteMatch* (Section 6.3.6), *LoadCarrier* (Section 6.3.2), 3D: *CADMatch* (Section 6.3.7), *ItemPick* (Section 6.3.4) and *BoxPick* (Section 6.3.5)
Supported reference frames	camera, external

6.5.2.2 Region of interest

A region of interest (ROI) defines a volume in space (3D region of interest, `region_of_interest`), or a rectangular region in the left camera image (2D region of interest, `region_of_interest_2d`) which is of interest for a specific user-application.

A ROI can narrow the volume where a load carrier is searched for, or select a volume which only contains items to be detected and/or grasped. Processing times can significantly decrease when using a ROI.

3D regions of interest of the following types (`type`) are supported:

- `BOX`, with dimensions `box.x`, `box.y`, `box.z`.
- `SPHERE`, with radius `sphere.radius`.

The user can specify the 3D region of interest `pose` in the `camera` or the `external` coordinate system. `External` can only be chosen if a *Hand-eye calibration* (Section 6.4.1) is available. When the sensor is robot mounted, and the region of interest is defined in the external frame, the current robot pose must be given to every detect service call that uses this region of interest.

A 2D ROI is defined as a rectangular part of the left camera image, and can be set via the *REST-API interface* (Section 7.2) or the *rc_reason_stack* *Web GUI* (Section 7.1) on the page *Regions of Interest* under *Database*. The Web GUI offers an easy-to-use selection tool. Each ROI must have a unique name to address a specific 2D ROI.

In the REST-API, a 2D ROI is defined by the following values:

- `id`: Unique name of the region of interest
- `offset_x`, `offset_y`: offset in pixels along the x-axis and y-axis from the top-left corner of the image, respectively
- `width`, `height`: width and height in pixels

The *rc_reason_stack* can persistently store up to 100 different 3D regions of interest and the same number of 2D regions of interest. The configuration of regions of interest is normally performed offline, during the set up of the desired application. This can be done via the *REST-API interface* (Section 7.2) of RoiDB module, or in the *rc_reason_stack* *Web GUI* (Section 7.1) on the page *Regions of Interest* under *Database*.

Note: The configured regions of interest are persistent even over firmware updates and rollbacks.

6.5.2.3 Interaction with other modules

Internally, the RoiDB module depends on, and interacts with other on-board modules as listed below.

Hand-eye calibration

In case the camera has been calibrated to a robot, the pose of a 3D ROI can be provided in the robot coordinate frame by setting the corresponding `pose_frame` argument.

Two different `pose_frame` values can be chosen:

1. **Camera frame** (`camera`). The ROI pose is provided in the camera frame, and no prior knowledge about the pose of the camera in the environment is required. This means that the configured load carriers move with the camera. It is the user's responsibility to update the configured poses if the camera frame moves (e.g. with a robot-mounted camera).
2. **External frame** (`external`). The ROI pose is provided in the external frame, configured by the user during the hand-eye calibration process. The module relies on the on-board *Hand-eye calibration module* (Section 6.4.1) to retrieve the sensor mounting (static or robot mounted) and the hand-eye transformation.

Note: If no hand-eye calibration is available, all `pose_frame` values should be set to `camera`.

All `pose_frame` values that are not `camera` or `external` are rejected.

6.5.2.4 Services

The RoiDB module is called `rc_roi_db` in the REST-API and is represented in the *Web GUI* (Section 7.1) under *Database* → *Regions of Interest*. The user can explore and call the RoiDB module's services, e.g. for development and testing, using the *REST-API interface* (Section 7.2) or the Web GUI.

The RoiDB module offers the following services.

`set_region_of_interest`

Persistently stores a 3D region of interest on the *rc_reason_stack*. All configured 3D regions of interest are persistent over firmware updates and rollbacks.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/set_region_of_interest
```

Request

Details for the definition of the `region_of_interest` type are given in *Region of interest* (Section 6.5.2.2).

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "region_of_interest": {
      "box": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "id": "string",
      "pose": {
        "orientation": {
          "w": "float64",
          "x": "float64",
          "y": "float64",
```

(continues on next page)

(continued from previous page)

```
          "z": "float64"
        },
        "position": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        }
      },
      "pose_frame": "string",
      "sphere": {
        "radius": "float64"
      },
      "type": "string"
    }
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

set_region_of_interest_2d

Persistently stores a 2D region of interest on the *rc_reason_stack*. All configured 2D regions of interest are persistent over firmware updates and rollbacks.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/set_region_of_interest_2d
```

Request

Details for the definition of the `region_of_interest_2d` type are given in *Region of interest* (Section 6.5.2.2).

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "region_of_interest_2d": {
      "height": "uint32",
      "id": "string",
      "offset_x": "uint32",
      "offset_y": "uint32",
      "width": "uint32"
    }
  }
}
```

Response

The definition for the response with corresponding datatypes is:


```
{
  "name": "set_region_of_interest_2d",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```


get_regions_of_interest

Returns the configured 3D regions of interest with the requested `region_of_interest_ids`.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/get_regions_of_interest
```

Request

If no `region_of_interest_ids` are provided, all configured 3D regions of interest are returned.

The definition for the request arguments with corresponding datatypes is:


```
{
  "args": {
    "region_of_interest_ids": [
      "string"
    ]
  }
}
```


Response

The definition for the response with corresponding datatypes is:


```
{
  "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",
```


(continues on next page)

(continued from previous page)

```
            "y": "float64",
            "z": "float64"
          }
        },
        "pose_frame": "string",
        "sphere": {
          "radius": "float64"
        },
        "type": "string"
      }
    ],
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

get_regions_of_interest_2d

Returns the configured 2D regions of interest with the requested region_of_interest_2d_ids.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/get_regions_of_interest_2d
```

Request

If no region_of_interest_2d_ids are provided, all configured 2D regions of interest are returned.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "region_of_interest_2d_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

```
{
  "name": "get_regions_of_interest_2d",
  "response": {
    "regions_of_interest": [
      {
        "height": "uint32",
        "id": "string",
        "offset_x": "uint32",
        "offset_y": "uint32",
        "width": "uint32"
      }
    ],
    "return_code": {
```

(continues on next page)

(continued from previous page)

```
      "message": "string",
      "value": "int16"
    }
  }
}
```

delete_regions_of_interest

Deletes the configured 3D regions of interest with the requested `region_of_interest_ids`.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/delete_regions_of_interest
```

Request

All regions of interest to be deleted must be explicitly stated in `region_of_interest_ids`.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "region_of_interest_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

delete_regions_of_interest_2d

Deletes the configured 2D regions of interest with the requested `region_of_interest_2d_ids`.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_roi_db/services/delete_regions_of_interest_2d
```

Request

All 2D regions of interest to be deleted must be explicitly stated in `region_of_interest_2d_ids`.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "region_of_interest_2d_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

```
{
  "name": "delete_regions_of_interest_2d",
  "response": {
    "return_code": {
      "message": "string",
      "value": "int16"
    }
  }
}
```

6.5.2.5 Return codes

Each service response contains a `return_code`, which consists of a `value` plus an optional `message`. A successful service returns with a `return_code` value of `0`. Negative `return_code` values indicate that the service failed. Positive `return_code` values indicate that the service succeeded with additional information. The smaller value is selected in case a service has multiple `return_code` values, but all messages are appended in the `return_code` message.

The following table contains a list of common codes:

Table 6.62: Return codes of the RoiDB module's services

Code	Description
0	Success
-1	An invalid argument was provided
-10	New element could not be added as the maximum storage capacity of regions of interest has been exceeded
10	The maximum storage capacity of regions of interest has been reached
11	An existent persistent model was overwritten by the call to `set_region_of_interest` or `set_region_of_interest_2d`

6.5.3 GripperDB

6.5.3.1 Introduction

The GripperDB module (gripper database module) is an optional on-board module of the *rc_reason_stack* and is licensed with any of the modules *ItemPick* (Section 6.3.4) and *BoxPick* (Section 6.3.5) or *SilhouetteMatch* (Section 6.3.6). Otherwise it requires a separate CollisionCheck *license* (Section 8.2) to be purchased.

The module provides services to set, retrieve and delete grippers which can then be used for checking collisions with a load carrier or other detected objects (only in combination with *SilhouetteMatch* (Section 6.3.6)). The specified grippers are available for all modules supporting collision checking on the *rc_reason_stack*.

Note: This module is global on the *rc_reason_stack*. Changes to its settings or parameters affect every camera pipeline running on the *rc_reason_stack*.

Table 6.63: Specifications of the GripperDB module

Max. number of grippers	50
Supported gripper element geometries	Box, Cylinder, CAD Element
Max. number of elements per gripper	15
Collision checking available in	*ItemPick* (Section 6.3.4) and *BoxPick* (Section 6.3.5), *SilhouetteMatch* (Section 6.3.6)

6.5.3.2 Setting a gripper

The gripper is a collision geometry used to determine whether the grasp is in collision with the load carrier. The gripper consists of up to 15 elements connected to each other.

At this point, the gripper can be built of elements of the following types:

- `BOX`, with dimensions `box.x`, `box.y`, `box.z`.
- `CYLINDER`, with radius `cylinder.radius` and height `cylinder.height`.
- `CAD`, with the id `cad.id` of the chosen CAD element.

Additionally, for each gripper the flange radius, and information about the Tool Center Point (TCP) have to be defined.

The configuration of the gripper is normally performed offline during the setup of the desired application. This can be done via the *REST-API interface* (Section 7.2) or the *rc_reason_stack* *Web GUI* (Section 7.1).

Robot flange radius

Collisions are checked only with the gripper, the robot body is not considered. As a safety feature, to prevent collisions between the load carrier and the robot, all grasps having any part of the robot's flange inside the load carrier can be designated as colliding (see Fig. 6.41). This check is based on the defined gripper geometry and the flange radius value. It is optional to use this functionality, and it can be turned on and off with the CollisionCheck module's run-time parameter `check_flange` as described in *Parameter overview* (Section 6.4.2.3).

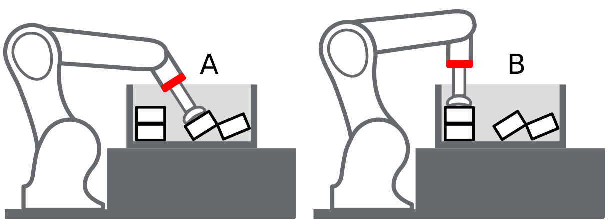

Fig. 6.41: Case A would be marked as collision only if `check_flange` is true, because the robot's flange (red) is inside the load carrier. Case B is collision free independent of `check_flange`.

Uploading gripper CAD elements

A gripper can consist of boxes, cylinders and CAD elements. While boxes and cylinders can be parameterized when the gripper is created, the CAD elements must be uploaded beforehand to be available during gripper creation. A CAD element can be uploaded via the *REST-API interface* (Section 7.2) as described in Section *CAD element API* (Section 6.5.3.5) or via the *rc_reason_stack* *Web GUI* (Section 7.1). Supported file formats are STEP (*.stp, *.step), STL (*.stl), OBJ (*.obj) and PLY (*.ply). The maximum file size to be uploaded is limited to 30 MB. The files are internally converted to PLY and, if necessary, simplified. The CAD elements can be referenced during gripper creation by their ID.

Creating a gripper via the REST-API or the Web GUI

When creating a gripper via the *REST-API interface* (Section 7.2) or the *Web GUI* (Section 7.1), each element of the gripper has a *parent* element, which defines how they are connected. The gripper is always built in the direction from the robot flange to the TCP, and at least one element must have 'flange' as parent. The elements' IDs must be unique and must not be 'tcp' or 'flange'. The pose of the child element has to be given in the coordinate frame of the parent element. The coordinate frame of an element is always in its geometric center. Accordingly, for a child element to be exactly below the parent element, the position of the child element must be computed from the heights of both parent and child element (see Fig. 6.42).

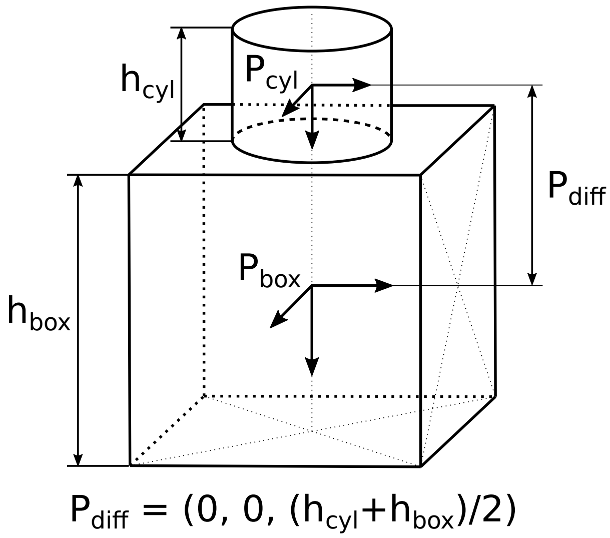

Fig. 6.42: Reference frames for gripper creation via the REST-API and the Web GUI

In case a CAD element is used, the element's origin is defined in the CAD data and is not necessarily located in the center of the element's bounding box.

It is recommended to create a gripper via the Web GUI, because it provides a 3D visualization of the gripper geometry and also allows to automatically attach the child element to the bottom of its parent element, when the corresponding option for this element is activated. In this case, the elements also stay attached when any of their sizes change. Automatic attachment of CAD elements uses the element's bounding box as reference. Automatic attachment is only possible when the child element is not rotated around the x or y axis with respect to its parent.

The reference frame for the first element for the gripper creation is always the center of the robot's flange with the z axis pointing outwards. It is possible to create a gripper with a tree structure, corresponding to multiple elements having the same parent element, as long as they are all connected.

Calculated TCP position

After gripper creation via the `set_gripper` service call, the TCP position in the flange coordinate system is calculated and returned as `tcp_pose_flange`. It is important to check if this value is the same as the robot's true TCP position. When creating a gripper in the Web GUI the current TCP position is always displayed in the 3D gripper visualization.

Creating rotationally asymmetric grippers

For grippers which are not rotationally symmetric around the z axis, it is crucial to ensure that the gripper is properly mounted, so that the representation stored in the GripperDB module corresponds to reality.

6.5.3.3 Services

The GripperDB module is called `rc_gripper_db` in the REST-API and is represented in the *Web GUI* (Section 7.1) under *Database* → *Grippers*. The user can explore and call the GripperDB module's services, e.g. for development and testing, using the *REST-API interface* (Section 7.2) or the Web GUI.

The GripperDB module offers the following services.

`set_gripper`

Persistently stores a gripper on the *rc_reason_stack*. All configured grippers are persistent over firmware updates and rollbacks.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_gripper_db/services/set_gripper
```

Request

Required arguments:

`elements`: list of geometric elements for the gripper. Each element must be of `type` 'CYLINDER' or 'BOX' with the corresponding dimensions in the `cylinder` or `box` field, or of type 'CAD' with the corresponding `id` in the `cad` field. The pose of each element must be given in the coordinate frame of the parent element (see *Setting a gripper* (Section 6.5.3.2) for an explanation of the coordinate frames). The element's `id` must be unique and must not be 'tcp' or 'flange'. The `parent_id` is the ID of the parent element. It can either be 'flange' or it must correspond to another element in list.

`flange_radius`: radius of the flange used in case the `check_flange` run-time parameter is active.

`id`: unique name of the gripper

`tcp_parent_id`: ID of the element on which the TCP is defined

`tcp_pose_parent`: The pose of the TCP with respect to the coordinate frame of the element specified in `tcp_parent_id`.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "elements": [
      {
        "box": {
          "x": "float64",
          "y": "float64",
          "z": "float64"
        },
        "cad": {
          "id": "string"
        },
```

(continues on next page)

(continued from previous page)

```
        "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_parent": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    }
  }
}
```

Response

gripper: returns the gripper as defined in the request with an additional field tcp_pose_flange. This gives the coordinates of the TCP in the flange coordinate frame for comparison with the true settings of the robot's TCP.

return_code: holds possible warnings or error codes and messages.

The definition for the response with corresponding datatypes is:

```
{
  "name": "set_gripper",
  "response": {
    "gripper": {
      "elements": [
        {
          "box": {
            "x": "float64",
            "y": "float64",
            "z": "float64"
          },
```

(continues on next page)

(continued from previous page)

```
          "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",
          "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"
    }
  }
```

(continues on next page)

(continued from previous page)

```
}
```

get_grippers

Returns the configured grippers with the requested gripper_ids.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_gripper_db/services/get_grippers
```

Request

If no gripper_ids are provided, all configured grippers are returned.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "gripper_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

```
{
  "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",
```

(continues on next page)

(continued from previous page)


```
                "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"
    }
  }
}
```


delete_grippers

Deletes the configured grippers with the requested `gripper_ids`.

Details

This service can be called as follows.

```
PUT http://<host>/api/v2/nodes/rc_gripper_db/services/delete_grippers
```

Request

All grippers to be deleted must be explicitly stated in `gripper_ids`.

The definition for the request arguments with corresponding datatypes is:

```
{
  "args": {
    "gripper_ids": [
      "string"
    ]
  }
}
```

Response

The definition for the response with corresponding datatypes is:

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

6.5.3.4 Return codes

Each service response contains a `return_code`, which consists of a `value` plus an optional `message`. A successful service returns with a `return_code` value of `0`. Negative `return_code` values indicate that the service failed. Positive `return_code` values indicate that the service succeeded with additional information. The smaller value is selected in case a service has multiple `return_code` values, but all messages are appended in the `return_code` message.

The following table contains a list of common codes:

Table 6.64: Return codes of the GripperDB services

Code	Description
0	Success
-1	An invalid argument was provided
-7	Data could not be read or written to persistent storage
-9	No valid license for the module
-10	New gripper could not be added as the maximum storage capacity of grippers has been exceeded
10	The maximum storage capacity of grippers has been reached
11	Existing gripper was overwritten

6.5.3.5 CAD element API

For gripper CAD element upload, download, listing and removal, special REST-API endpoints are provided. CAD elements can also be uploaded, downloaded and removed via the Web GUI. Up to 50 CAD elements can be stored persistently on the *rc_reason_stack*.

The maximum file size to be uploaded is limited to 30 MB.

`GET /cad/gripper_elements`

Get list of all CAD gripper elements.

Template request

```
GET /api/v2/cad/gripper_elements HTTP/1.1
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "string"
  }
]
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns array of GripperElement)*
- 404 Not Found – element not found

Referenced Data Models

- *GripperElement* (Section 7.2.3)

GET /cad/gripper_elements/{id}

Get a CAD gripper element. If the requested content-type is application/octet-stream, the gripper element is returned as file.

Template request

```
GET /api/v2/cad/gripper_elements/<id> HTTP/1.1
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameters

- **id** (*string*) – id of the element *(required)*

Response Headers

- Content-Type – application/json application/ubjson application/octet-stream

Status Codes

- 200 OK – successful operation *(returns GripperElement)*
- 404 Not Found – element not found

Referenced Data Models

- *GripperElement* (Section 7.2.3)

PUT /cad/gripper_elements/{id}

Create or update a CAD gripper element.

Template request

```
PUT /api/v2/cad/gripper_elements/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "string"
}
```

Parameters

- **id** (*string*) – id of the element *(required)*

Form Parameters

- **file** – CAD file *(required)*

Request Headers

- Accept – multipart/form-data application/json

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns GripperElement)*
- 400 Bad Request – CAD is not valid or max number of elements reached
- 404 Not Found – element not found
- 413 Request Entity Too Large – File too large

Referenced Data Models

- *GripperElement* (Section 7.2.3)

DELETE /cad/gripper_elements/{id}

Remove a CAD gripper element.

Template request

```
DELETE /api/v2/cad/gripper_elements/<id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameters

- **id** (*string*) – id of the element *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation
- 404 Not Found – element not found

7 Interfaces

The following interfaces are provided for configuring and obtaining data from the *rc_reason_stack*:

- *Web GUI* (Section 7.1)

  Easy-to-use graphical interface to configure the *rc_reason_stack*, do calibrations, view live images, do service calls, visualize results, etc.

- *REST-API interface* (Section 7.2)

  API to configure the *rc_reason_stack*, query status information, do service calls, etc.

- *Generic Robot Interface* (Section 7.3)

  TCP socket communication interface for configuring the *rc_reason_stack* and for service calls.

- *OPC UA interface* (Section 7.4)

  OPC UA interface for configuring the *rc_reason_stack* and for service calls.

- *KUKA Ethernet KRL Interface* (Section 7.5)

  API to configure the *rc_reason_stack* and do service calls from KUKA KSS robots.

- *gRPC image stream interface* (Section 7.6)

  Stream synchronized image sets via gRPC.

7.1 Web GUI

The *rc_reason_stack*'s Web GUI can be used to test, calibrate, and configure the device.

7.1.1 Accessing the Web GUI

The Web GUI of the *rc_reason_stack* can be accessed from any web browser, such as Firefox, Google Chrome, or Microsoft Edge, via the host PC's IP address with the port number 8080:

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

7.1.2 Exploring the Web GUI

The Web GUI's dashboard page gives the most important information about the device and the running camera pipelines.

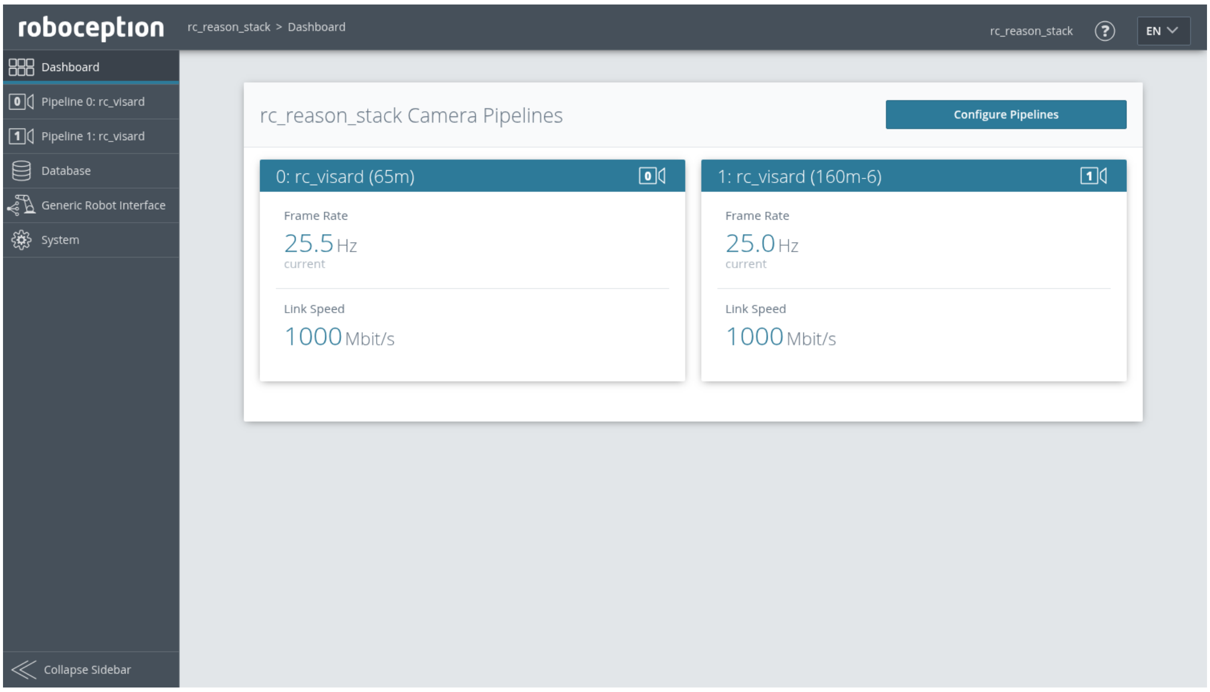

Fig. 7.1: Dashboard page of the *rc_reason_stack*'s Web GUI

The page's side menu permits access to the individual pages of the *rc_reason_stack*'s Web GUI:

Pipeline gives access to the respective camera pipeline and its camera, detection and configuration modules. Each camera pipeline provides an overview page with the most important information about the camera connection and the software modules running in the pipeline.

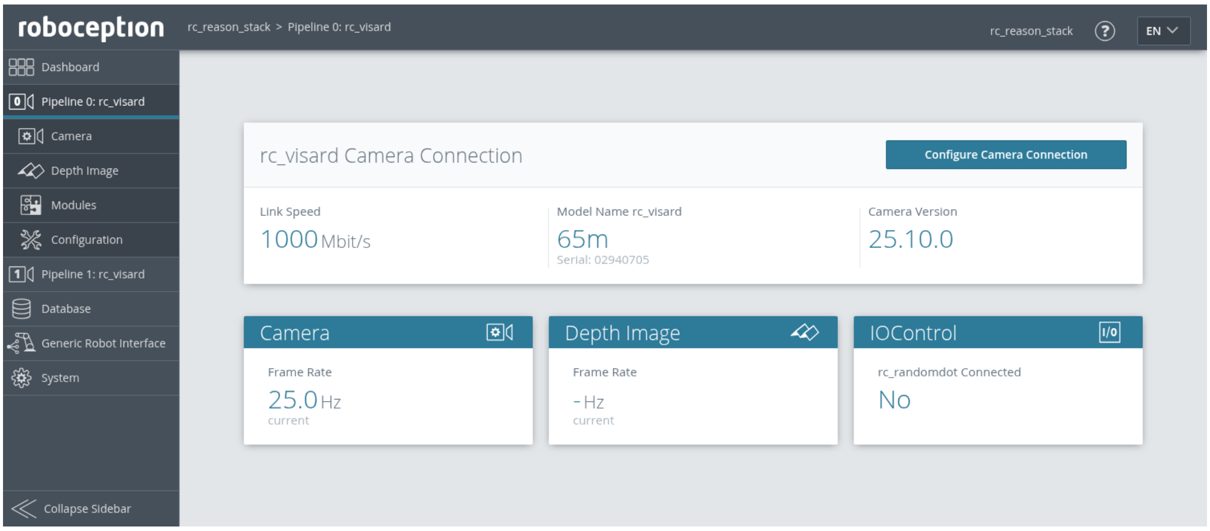

Fig. 7.2: Pipeline overview page of the *rc_reason_stack*'s Web GUI

Each pipeline provides a sub-menu with the individual pages for the modules running in the pipeline:

Camera shows a live stream of the rectified camera images and allows changing camera parameters. See *Camera module* (Section 6.1) for more information.

Depth Image shows a live stream of the left rectified, disparity, and confidence images. The page contains various settings for depth-image computation and filtering. See *3D modules* (Section 6.2) for more information.

Modules gives access to the detection modules of the *rc_reason_stack* (see *Detection & Measure modules*, Section 6.3).

Configuration gives access to the configuration modules of the *rc_reason_stack* (see *Configuration modules*, Section 6.4).

The following modules running outside the pipelines can be accessed in the side menu:

Database gives access to the database modules of the *rc_reason_stack* (see *Database modules*, Section 6.5).

Generic Robot Interface shows the jobs and hand-eye calibration configurations defined for the Generic Robot Interface.

System gives access to general settings, software information and to the log files, and shows license information.

Note: Further information on all parameters in the Web GUI can be obtained by pressing the *Info* button next to each parameter.

7.1.3 Web GUI access control

The Web GUI has a simple mechanism to lock the UI against casual and accidental changes.

When enabling Web GUI access control via the *System* page, you will be asked to set a password. Now the Web GUI is in a locked mode indicated by the lock symbol in the top bar. All pages, camera streams, parameters and detections can be inspected as usual, but changes are not possible.

To temporarily unlock the Web GUI and make changes, click the lock symbol and enter the password. While enabling or disabling Web GUI access control affects anyone accessing this *rc_reason_stack*, the unlocked state is only valid for the browser where it was unlocked and indicated by the open lock symbol. It is automatically locked again after 10 minutes of inactivity.

Web GUI access control can also be disabled again on the *System* page after providing the current password.

Warning: This is not a security feature! It only locks the Web GUI and not the REST-API. It is meant to prevent accidental and casual changes e.g. via a connected screen.

Note: In case the password is lost, this can be disabled via the REST-API *delete ui_lock* (Section 7.2.2.4).

7.1.4 Downloading camera images

The Web GUI provides an easy way to download a snapshot of the current scene as a .tar.gz file by clicking on the camera icon below the image live streams on the *Camera* page. This snapshot contains:

- the rectified camera images in full resolution as .png files,
- a camera parameter file containing the camera matrix, image dimensions, exposure time, gain value and the stereo baseline,
- the current IMU readings as imu.csv file, if available,
- a pipeline_status.json file containing information about all modules running inside the pipelines on the *rc_reason_stack*,
- a backup.json file containing the settings of the *rc_reason_stack* including grippers, load carriers and regions of interest,

- a system_info.json file containing system information about the *rc_reason_stack*.

The filenames contain the timestamps.

7.1.5 Downloading depth images and point clouds

The Web GUI provides an easy way to download the depth data of the current scene as a .tar.gz file by clicking on the camera icon below the image live streams on the *Depth Image* page. This snapshot contains:

- the rectified left and right camera images in full resolution as .png files,
- an image parameter file corresponding to the left image containing the camera matrix, image dimensions, exposure time, gain value and the stereo baseline,
- the disparity, error and confidence images in the resolution corresponding to the currently chosen quality as .png files,
- a disparity parameter file corresponding to the disparity image containing the camera matrix, image dimensions, exposure time, gain value and the stereo baseline, and information about the disparity values (i.e. invalid values, scale, offset),
- the current IMU readings as imu.csv file, if available,
- a pipeline_status.json file containing information about all modules running inside the pipelines on the *rc_reason_stack*,
- a backup.json file containing the settings of the *rc_reason_stack* including grippers, load carriers and regions of interest,
- a system_info.json file containing system information about the *rc_reason_stack*.

The filenames contain the timestamps.

When clicking on the mesh icon below the image live streams on the *Depth Image* page, a snapshot is downloaded which additionally includes a mesh of the point cloud in the current depth quality (resolution) as .ply file.

Note: Downloading a depth snapshot will trigger an acquisition in the same way as clicking on the “Acquire” button on the *Depth Image* page of the Web GUI, and, thus, might affect running applications.

7.2 REST-API interface

The *rc_reason_stack* offers a comprehensive RESTful web interface (REST-API) which any HTTP client or library can access. Whereas most of the provided parameters, services, and functionalities can also be accessed via the user-friendly *Web GUI* (Section 7.1), the REST-API serves rather as a machine-to-machine interface to the *rc_reason_stack*, e.g., to programmatically

- set and get run-time parameters of computation nodes, e.g., of cameras or image processing modules;
- do service calls, e.g., to start and stop individual computational nodes, or to use offered services such as the hand-eye calibration;
- read the current state of the system and individual computational nodes; or
- update the *rc_reason_stack*’s firmware or license.

Note: In the *rc_reason_stack*’s REST-API, a *node* is a computational component that bundles certain algorithmic functionality and offers a holistic interface (parameters, services, current status). Examples for such nodes are the stereo matching node or the hand-eye calibration node.

7.2.1 General API structure

The general **entry point** to the *rc_reason_stack*'s API is http://<host>/api/, where <host> is the IP address of the host PC that runs the *rc_reason_stack* combined with the port 8080, i.e. <host-ip>::8080. Accessing this entry point with a web browser lets the user explore and test the full API during run-time using the *Swagger UI* (Section 7.2.4).

For actual HTTP requests, the **current API version is appended** to the entry point of the API, i.e., http://<host>/api/v2.

All data sent to and received by the REST-API follows the JavaScript Object Notation (JSON). The API is designed to let the user **create, retrieve, modify, and delete** so-called **resources** as listed in *Available resources and requests* (Section 7.2.2) using the HTTP requests below.

Request type	Description
GET	Access one or more resources and return the result as JSON.
PUT	Modify a resource and return the modified resource as JSON.
DELETE	Delete a resource.
POST	Upload file (e.g., license or firmware image).

Depending on the type and the specific request itself, **arguments** to HTTP requests can be transmitted as part of the **path** (URI) to the resource, as **query** string, as **form data**, or in the **body** of the request. The following examples use the command line tool *curl*, which is available for various operating systems. See https://curl.haxx.se.

- Get a node's current status; its name is encoded in the path (URI)

```
curl -X GET 'http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching'
```

- Get values of some of a node's parameters using a query string

```
curl -X GET 'http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters?name=minconf&name=maxdepth'
```

- Set a node's parameter as JSON-encoded text in the body of the request

```
curl -X PUT --header 'Content-Type: application/json' -d '[{"name": "mindepth", "value": 0.1}]' 'http://<host>/api/v2/pipelines/0/nodes/rc_stereomatching/parameters'
```

As for the responses to such requests, some common return codes for the *rc_reason_stack*'s API are:

Status Code	Description
200 OK	The request was successful; the resource is returned as JSON.
400 Bad Request	A required attribute or argument of the API request is missing or invalid.
404 Not Found	A resource could not be accessed; e.g., an ID for a resource could not be found.
403 Forbidden	Access is (temporarily) forbidden; e.g., some parameters are locked while a GigE Vision application is connected.
429 Too many requests	Rate limited due to excessive request frequency.

The following listing shows a sample response to a successful request that accesses information about the `rc_stereomatching` node's `minconf` parameter:

```
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"
}
```

Note: The actual behavior, allowed requests, and specific return codes depend heavily on the specific resource, context, and action. Please refer to the *rc_reason_stack*'s *available resources* (Section 7.2.2) and to each *software module's* (Section 6) parameters and services.

7.2.2 Available resources and requests

The available REST-API resources are structured into the following parts:

- **/nodes** Access the *rc_reason_stack*'s global *Database modules* (Section 6.5) with their run-time status, parameters, and offered services, for storing data used in all camera pipelines and multiple modules, such as load carriers, grippers and regions of interest.
- **/pipelines/<number>/nodes** Access the *rc_reason_stack*'s 3D-camera, detection and configuration *software modules* (Section 6) of the camera pipeline with the specified number, with their run-time status, parameters, and offered services.
- **/pipelines** Access to the status and configuration of the camera pipelines.
- **/templates** Access the object templates on the *rc_reason_stack*.
- **/cad** Access the cad elements, e.g. for grippers, on the *rc_reason_stack*.
- **/presets** Access the 2D and 3D user-defined presets for *zivid* cameras.
- **/system** Access the system state, set network configuration, configure the camera pipeline types, and manage licenses as well as firmware updates.
- **/logs** Access the log files on the *rc_reason_stack*.
- **/generic_robot_interface** Access the job and hec_configs for the Generic Robot Interface on the *rc_reason_stack*.

7.2.2.1 Nodes, parameters, and services

Nodes represent the *rc_reason_stack*'s *software modules* (Section 6), each bundling a certain algorithmic functionality. All available global REST-API database nodes can be listed with their service calls and parameters using

```
curl -X GET http://<host>/api/v2/nodes
```

Information about a specific node (e.g., `rc_load_carrier_db`) can be retrieved using

```
curl -X GET http://<host>/api/v2/nodes/rc_load_carrier_db
```

All available 3D camera, detection and configuration REST-API nodes can be listed with their service calls and parameters using

```
curl -X GET http://<host>/api/v2/pipelines/<pipeline number>/nodes
```

Information about a specific node (e.g., `rc_camera` on camera pipeline 1) can be retrieved using

```
curl -X GET http://<host>/api/v2/pipelines/1/nodes/rc_camera
```

Status: During run-time, each node offers information about its current status. This includes not only the current **processing status** of the module (e.g., `running` or `stale`), but most nodes also offer run-time statistics or read-only parameters, so-called **status values**. As an example, the `rc_camera` values can be retrieved using

```
curl -X GET http://<host>/api/v2/pipelines/<pipeline number>/nodes/rc_camera/status
```

Note: The returned **status values** are specific to individual nodes and are documented in the respective *software module* (Section 6).

Note: The **status values** are only reported when the respective node is in the `running` state.

Parameters: Most nodes expose parameters via the *rc_reason_stack*'s REST-API to allow their run-time behaviors to be changed according to application context or requirements. The REST-API permits to read and write a parameter's value, but also provides further information such as minimum, maximum, and default values.

As an example, the `rc_stereomatching` parameters can be retrieved using

```
curl -X GET http://<host>/api/v2/pipelines/<pipeline number>/nodes/rc_stereomatching/
↪parameters
```

Its `quality` parameter could be set to `Full` using

```
curl -X PUT http://<host>/api/v2/pipelines/<pipeline number>/nodes/rc_stereomatching/
↪parameters?quality=Full
```

or equivalently

```
curl -X PUT --header 'Content-Type: application/json' -d '{ "value": "Full" }' http://<host>
↪/api/v2/pipelines/<pipeline number>/nodes/rc_stereomatching/parameters/quality
```

Note: Run-time parameters are specific to individual nodes and are documented in the respective *software module* (Section 6).

Note: Most of the parameters that nodes offer via the REST-API can be explored and tested via the *rc_reason_stack*'s user-friendly *Web GUI* (Section 7.1).

In addition, each node that offers run-time parameters also features a service to restore the default values for all of its parameters.

Services: Most nodes also offer services that can be called via REST-API, e.g., to restore parameters as discussed above, or to start and stop nodes. As an example, the *services of the hand-eye calibration module* (Section 6.4.1.5) could be listed using

```
curl -X GET http://<host>/api/v2/pipelines/<pipeline number>/nodes/rc_hand_eye_calibration/
↪services
```

A node's service is called by issuing a PUT request for the respective resource and providing the service-specific arguments (see the "args" field of the *Service data model*, Section 7.2.3). As an example, the stereo matching module can be triggered to do an acquisition by:

```
curl -X PUT --header 'Content-Type: application/json' -d '{ "args": {} }' http://<host>/api/
↪v2/pipelines/<pipeline number>/nodes/rc_stereomatching/services/acquisition_trigger
```

Note: The services and corresponding argument data models are specific to individual nodes and are documented in the respective *software module* (Section 6).

The following list includes all REST-API requests regarding the global database nodes' status, parameters, and services calls:

GET /nodes

Get list of all available global nodes.

Template request

```
GET /api/v2/nodes HTTP/1.1
```

Sample response

```
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"
  }
]
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns array of NodeInfo)*

Referenced Data Models

- *NodeInfo* (Section 7.2.3)

GET /nodes/{node}

Get info on a single global node.

Template request

```
GET /api/v2/nodes/<node> HTTP/1.1
```

Sample response

```
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"
}
```

Parameters

- **node** (*string*) – name of the node *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns NodeInfo)*
- 404 Not Found – node not found

Referenced Data Models

- *NodeInfo* (Section 7.2.3)

GET /nodes/{node}/services

Get descriptions of all services a global node offers.

Template request

```
GET /api/v2/nodes/<node>/services HTTP/1.1
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
[
  {
    "args": {},
    "description": "string",
    "name": "string",
    "response": {}
  }
]
```

Parameters

- **node** (*string*) – name of the node *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns array of Service)*
- 404 Not Found – node not found

Referenced Data Models

- *Service* (Section 7.2.3)

GET /nodes/{node}/services/{service}

Get description of a global node's specific service.

Template request

```
GET /api/v2/nodes/<node>/services/<service> HTTP/1.1
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "args": {},
  "description": "string",
  "name": "string",
  "response": {}
}
```

Parameters

- **node** (*string*) – name of the node *(required)*
- **service** (*string*) – name of the service *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns Service)*
- 404 Not Found – node or service not found

Referenced Data Models

- *Service* (Section 7.2.3)

PUT /nodes/{node}/services/{service}

Call a service of a node. The required args and resulting response depend on the specific node and service.

Template request

```
PUT /api/v2/nodes/<node>/services/<service> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Template response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "args": {},
  "description": "string",
  "name": "string",
  "response": {}
}
```

Parameters

- **node** (*string*) – name of the node *(required)*
- **service** (*string*) – name of the service *(required)*

Request JSON Object

- **service args** (*object*) – example args *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – Service call completed *(returns Service)*
- 403 Forbidden – Service call forbidden, e.g. because there is no valid license for this module.
- 404 Not Found – node or service not found

Referenced Data Models

- *Service* (Section 7.2.3)

GET /nodes/{node}/status

Get status of a global node.

Template request

```
GET /api/v2/nodes/<node>/status HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)

```
  "status": "running",
  "timestamp": 1503075030.2335997,
  "values": []
}
```

Parameters

- **node** (*string*) – name of the node *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns NodeStatus)*
- 404 Not Found – node not found

Referenced Data Models

- *NodeStatus* (Section 7.2.3)

The following list includes all REST-API requests regarding the pipeline-specific 3D camera, detection and configuration nodes' status, parameters, and services calls:

GET /pipelines/{pipeline}/nodes

Get list of all available nodes.

Template request

```
GET /api/v2/pipelines/<pipeline>/nodes HTTP/1.1
```

Sample response

```
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",
```

(continues on next page)

(continued from previous page)

```
      "calibrate",
      "get_calibration"
    ],
    "status": "idle"
  },
  {
    "name": "rc_stereomatching",
    "parameters": [
      "quality",
      "seg",
      "fill",
      "minconf",
      "mindepth",
      "maxdepth",
      "maxdeptherr"
    ],
    "services": [
      "reset_defaults"
    ],
    "status": "running"
  }
]
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of 0, 1, 2, 3) *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns array of NodeInfo)*

Referenced Data Models

- *NodeInfo* (Section 7.2.3)

GET /pipelines/{pipeline}/nodes/{node}

Get info on a single node.

Template request

```
GET /api/v2/pipelines/<pipeline>/nodes/<node> HTTP/1.1
```

Sample response

```
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"
}
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of 0, 1, 2, 3) *(required)*
- **node** (*string*) – name of the node *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns NodeInfo)*
- 404 Not Found – node not found

Referenced Data Models

- *NodeInfo* (Section 7.2.3)

GET /pipelines/{pipeline}/nodes/{node}/parameters

Get parameters of a node.

Template request

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/parameters?name=<name> HTTP/1.1
```

Sample response

```
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
  }
]
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of 0, 1, 2, 3) *(required)*
- **node** (*string*) – name of the node *(required)*

Query Parameters

- **name** (*string*) – limit result to parameters with name *(optional)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns array of Parameter)*
- 404 Not Found – node not found

Referenced Data Models

- *Parameter* (Section 7.2.3)

PUT /pipelines/{pipeline}/nodes/{node}/parameters

Update multiple parameters.

Template request

```
PUT /api/v2/pipelines/<pipeline>/nodes/<node>/parameters HTTP/1.1
Accept: application/json application/ubjson

[
  {
    "name": "string",
    "value": {}
  }
]
```

Sample response

```
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
```

(continues on next page)

(continued from previous page)

```
  }
]
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of 0, 1, 2, 3) *(required)*
- **node** (*string*) – name of the node *(required)*

Request JSON Array of Objects

- **parameters** (*ParameterNameValue*) – array of parameters *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns array of Parameter)*
- 400 Bad Request – invalid parameter value
- 403 Forbidden – Parameter update forbidden, e.g. because they are locked by a running GigE Vision application or there is no valid license for this module.
- 404 Not Found – node not found

Referenced Data Models

- *ParameterNameValue* (Section 7.2.3)
- *Parameter* (Section 7.2.3)

GET /pipelines/{pipeline}/nodes/{node}/parameters/{param}

Get a specific parameter of a node.

Template request

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/parameters/<param> HTTP/1.1
```

Sample response

```
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
}
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of 0, 1, 2, 3) *(required)*
- **node** (*string*) – name of the node *(required)*
- **param** (*string*) – name of the parameter *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns Parameter)*
- 404 Not Found – node or parameter not found

Referenced Data Models

- *Parameter* (Section 7.2.3)

PUT /pipelines/{pipeline}/nodes/{node}/parameters/{param}

Update a specific parameter of a node.

Template request

```
PUT /api/v2/pipelines/<pipeline>/nodes/<node>/parameters/<param> HTTP/1.1
Accept: application/json application/ubjson

{
  "value": {}
}
```

Sample response

```
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
}
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of `0`, `1`, `2`, `3`) *(required)*
- **node** (*string*) – name of the node *(required)*
- **param** (*string*) – name of the parameter *(required)*

Request JSON Object

- **parameter** (*ParameterValue*) – parameter to be updated as JSON object *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns Parameter)*
- 400 Bad Request – invalid parameter value
- 403 Forbidden – Parameter update forbidden, e.g. because they are locked by a running GigE Vision application or there is no valid license for this module.

- 404 Not Found – node or parameter not found

Referenced Data Models

- *ParameterValue* (Section 7.2.3)
- *Parameter* (Section 7.2.3)

GET /pipelines/{pipeline}/nodes/{node}/services

Get descriptions of all services a node offers.

Template request

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/services HTTP/1.1
```

Sample response

```
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"
    }
  }
]
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of 0, 1, 2, 3) *(required)*
- **node** (*string*) – name of the node *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns array of Service)*
- 404 Not Found – node not found

Referenced Data Models

- *Service* (Section 7.2.3)

GET /pipelines/{pipeline}/nodes/{node}/services/{service}

Get description of a node's specific service.

Template request

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/services/<service> HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "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"
  }
}
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of 0, 1, 2, 3) *(required)*
- **node** (*string*) – name of the node *(required)*
- **service** (*string*) – name of the service *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns Service)*
- 404 Not Found – node or service not found

Referenced Data Models

- *Service* (Section 7.2.3)

PUT /pipelines/{pipeline}/nodes/{node}/services/{service}

Call a service of a node. The required args and resulting response depend on the specific node and service.

Template request

```
PUT /api/v2/pipelines/<pipeline>/nodes/<node>/services/<service> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "name": "set_pose",
  "response": {
    "message": "Grid detected, pose stored.",
    "status": 1,
    "success": true
  }
}
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of 0, 1, 2, 3) *(required)*
- **node** (*string*) – name of the node *(required)*
- **service** (*string*) – name of the service *(required)*

Request JSON Object

- **service args** (*object*) – example args *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – Service call completed *(returns Service)*
- 403 Forbidden – Service call forbidden, e.g. because there is no valid license for this module.
- 404 Not Found – node or service not found

Referenced Data Models

- *Service* (Section 7.2.3)

GET /pipelines/{pipeline}/nodes/{node}/status

Get status of a node.

Template request

```
GET /api/v2/pipelines/<pipeline>/nodes/<node>/status HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

(continues on next page)

(continued from previous page)


```
  "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",
    "temp_right": "38.2",
    "time": "0.00406513",
    "width": "1280"
  }
}
```


Parameters

- **pipeline** (*string*) – name of the pipeline (one of `0`, `1`, `2`, `3`) *(required)*
- **node** (*string*) – name of the node *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns NodeStatus)*
- 404 Not Found – node not found

Referenced Data Models

- *NodeStatus* (Section 7.2.3)

7.2.2.2 Pipelines

Pipelines represent the *rc_reason_stack*'s camera pipelines.

The following list includes all REST-API requests regarding the camera pipelines' configuration:

GET /pipelines

Get active pipelines

Template request

```
GET /api/v2/pipelines HTTP/1.1
```

Status Codes

- 200 OK – successful operation

GET /pipelines/{pipeline}

Get active pipeline type and status

Template request

```
GET /api/v2/pipelines/<pipeline> HTTP/1.1
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of `0`, `1`, `2`, `3`) *(required)*

Status Codes

- 200 OK – successful operation

GET /system/pipelines

Get pipeline configuration.

Template request

```
GET /api/v2/system/pipelines HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "config": {
    "0": {
      "type": "rc_visard"
    }
  },
  "max_pipelines": 4,
  "pending_changes": false
}
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

GET /system/pipelines/config/{pipeline}

Get configuration for specific pipeline.

Template request

```
GET /api/v2/system/pipelines/config/<pipeline> HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "0": {
    "type": "rc_visard"
  }
}
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of 0, 1, 2, 3) *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

PUT /system/pipelines/config/{pipeline}

Update configuration for specific pipeline.

Template request

```
PUT /api/v2/system/pipelines/config/<pipeline>?type=<type> HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "type": "rc_visard"
}
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of 0, 1, 2, 3) *(required)*

Query Parameters

- **type** (*string*) – pipeline type (one of rc_visard, rc_viscore, blaze, zivid, stereo_ace) *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation
- 400 Bad Request – invalid pipeline name or type

DELETE /system/pipelines/config/{pipeline}

Delete specific pipeline.

Template request

```
DELETE /api/v2/system/pipelines/config/<pipeline> HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "message": "Pipeline 1 deleted"
}
```

Parameters

- **pipeline** (*string*) – name of the pipeline (one of 1, 2, 3) *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation
- 400 Bad Request – invalid pipeline name, e.g. pipeline 0 cannot be deleted

7.2.2.3 UserSpace

UserSpace information including running apps and their published ports can be queried via the userspace endpoint. An app can be of `type` `container` or `compose` (compose stack with potentially multiple containers).

7.2.2.4 System and logs

The following resources and requests expose the *rc_reason_stack*'s system-level API. They enable

- access to log files (system-wide or module-specific)
- access to information about the device and run-time statistics such as date, MAC address, clock-time synchronization status, and available resources;
- management of installed software licenses; and
- the *rc_reason_stack* to be updated with a new firmware image.

GET /logs

Get list of available log files.

Template request

```
GET /api/v2/logs HTTP/1.1
```

Sample response

```
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
  }
]
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns array of LogInfo)*

Referenced Data Models

- *LogInfo* (Section 7.2.3)

GET /logs/{log}

Get a log file. Content type of response depends on parameter 'format'.

Template request

```
GET /api/v2/logs/<log>?format=<format>&limit=<limit> HTTP/1.1
```

Sample response

```
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
}
```

Parameters

- **log** (*string*) – name of the log file *(required)*

Query Parameters

- **format** (*string*) – return log as JSON or raw (one of `json`, `raw`; default: `json`) *(optional)*
- **limit** (*integer*) – limit to last x lines in JSON format (default: `100`) *(optional)*

Response Headers

- Content-Type – text/plain application/json

Status Codes

- 200 OK – successful operation *(returns Log)*
- 404 Not Found – log not found

Referenced Data Models

- *Log* (Section 7.2.3)

GET /system

Get system information on device.

Template request

```
GET /api/v2/system HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "dongle_id": "dinkey:1234",
  "hostname": "rc-cube-00012e96ef39",
  "link_speed": 1000,
  "mac": "00:01:2e:96:ef:39",
  "model_name": "rc_cube S",
  "pipelines": {
    "config": {
      "0": {
        "type": "rc_visard"
      },
      "1": {
        "type": "rc_visard"
      }
    },
    "max_pipelines": 2,
    "pending_changes": false
  },
  "ready": true,
  "reboot_required": false,
  "sensor_interfaces": {
    "sensor0": {
      "link_speed": 2500
    }
  },
  "serial": "00012e96ef39",
  "time": 1649678734.0306993,
  "uptime": 336455.25,
  "userspace": {
    "available": true,
    "enabled": true
  }
}
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns SysInfo)*

Referenced Data Models

- *SysInfo* (Section 7.2.3)

GET /system/backup

Get backup.

Template request

```
GET /api/v2/system/backup?pipelines=<pipelines>&load_carriers=<load_carriers>&regions_of_
↪interest=<regions_of_interest>&grippers=<grippers> HTTP/1.1
```

Query Parameters

- **pipelines** (*boolean*) – backup pipelines with node settings, i.e. parameters and preferred_orientation (default: `True`) *(optional)*
- **load_carriers** (*boolean*) – backup load_carriers (default: `True`) *(optional)*
- **regions_of_interest** (*boolean*) – backup regions_of_interest (default: `True`) *(optional)*
- **grippers** (*boolean*) – backup grippers (default: `True`) *(optional)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

POST /system/backup

Restore backup.

Template request

```
POST /api/v2/system/backup HTTP/1.1
Accept: application/json application/ubjson

{}
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "return_code": {
    "message": "backup restored",
    "value": 0
  },
  "warnings": []
}
```

Request JSON Object

- **backup** (*object*) – backup data as json object *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

GET /system/ca_certificates

Get ca-certificates.

Template request

```
GET /api/v2/system/ca_certificates HTTP/1.1
```

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – successful operation

GET /system/ca_certificates/{id}

Get ca-certificate file or details.

Template request

```
GET /api/v2/system/ca_certificates/<id> HTTP/1.1
```

Parameters

- **id** (*string*) – ID/filename without extension *(required)*

Response Headers

- Content-Type – application/json application/octet-stream

Status Codes

- 200 OK – successful operation
- 404 Not Found – crt file not found

PUT /system/ca_certificates/{id}

Create or update a crt file.

Template request

```
PUT /api/v2/system/ca_certificates/<id> HTTP/1.1
Accept: multipart/form-data application/json
```

Parameters

- **id** (*string*) – ID/filename without extension *(required)*

Form Parameters

- **file** – crt file *(required)*

Request Headers

- Accept – multipart/form-data application/json

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – successful operation
- 400 Bad Request – crt is not valid or max number of elements reached
- 413 Request Entity Too Large – File too large

DELETE /system/ca_certificates/{id}

Remove a crt file.

Template request

```
DELETE /api/v2/system/ca_certificates/<id> HTTP/1.1
Accept: application/json
```

Parameters

- **id** (*string*) – ID/filename without extension *(required)*

Request Headers

- Accept – application/json

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – successful operation
- 404 Not Found – element not found

GET /system/disk_info

Get disk space info

Template request

```
GET /api/v2/system/disk_info HTTP/1.1
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

GET /system/license

Get information about licenses installed on device.

Template request

```
GET /api/v2/system/license HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "components": {
    "hand_eye_calibration": true,
    "rectification": true,
    "stereo": true
  },
  "valid": true
}
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns LicenseInfo)*

Referenced Data Models

- *LicenseInfo* (Section 7.2.3)

POST /system/license

Update license on device with a license file.

Template request

```
POST /api/v2/system/license HTTP/1.1
Accept: multipart/form-data
```

Form Parameters

- **file** – license file *(required)*

Request Headers

- Accept – multipart/form-data

Status Codes

- 200 OK – successful operation
- 400 Bad Request – not a valid license

GET /system/max_power_test

Get last max power test result.

Template request

```
GET /api/v2/system/max_power_test HTTP/1.1
```

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – successful operation

POST /system/max_power_test

Run max power test. Fully load GPU (and CPU) to consume max power for 10 seconds to test the power supply. WARNING: The system might not return a response due to immediate reboot if the power supply is insufficient.

Template request

```
POST /api/v2/system/max_power_test?nocpu=<nocpu> HTTP/1.1
```

Query Parameters

- **nocpu** (*boolean*) – Don't run CPU workers and only load the GPU. *(optional)*

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – Test finished. See return_code for result.
- 400 Bad Request – Test already running.

GET /system/ui_lock

Get UI lock status.

Template request

```
GET /api/v2/system/ui_lock HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": false
}
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation *(returns UILock)*

Referenced Data Models

- *UILock* (Section 7.2.3)

DELETE /system/ui_lock

Remove UI lock.

Template request

```
DELETE /api/v2/system/ui_lock HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": false,
  "valid": false
}
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

POST /system/ui_lock

Verify or set UI lock.

Template request

```
POST /api/v2/system/ui_lock?hash=<hash>&set=<set> HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "enabled": true,
  "valid": true
}
```

Query Parameters

- **hash** (*string*) – hash of the UI lock password *(required)*

- **set** (*boolean*) – set new hash instead of veryfing *(optional)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

7.2.3 Data type definitions

The REST-API defines the following data models, which are used to access or modify *the available resources* (Section 7.2.2) either as required attributes/parameters of the requests or as return types.

GripperElement: CAD gripper element

An object of type GripperElement has the following properties:

- **id** (string) - Unique identifier of the element

Template object

```
{
  "id": "string"
}
```

GripperElement objects are used in the following requests:

- *GET /cad/gripper_elements*
- *GET /cad/gripper_elements/{id}*
- *PUT /cad/gripper_elements/{id}*

ImageInfo: Information about specific firmware image.

An object of type ImageInfo has the following properties:

- **image_version** (string) - image version

Template object

```
{
  "image_version": "string"
}
```

ImageInfo objects are nested in FirmwareInfo.

LicenseComponentConstraint: Constraints on the module version.

An object of type LicenseComponentConstraint has the following properties:

- **max_version** (string) - optional maximum supported version (exclusive)
- **min_version** (string) - optional minimum supported version (inclusive)

Template object

```
{
  "max_version": "string",
  "min_version": "string"
}
```

LicenseComponentConstraint objects are nested in *LicenseConstraints*.

LicenseComponents: List of the licensing status of the individual software modules. The respective flag is true if the module is unlocked with the currently applied software license.

An object of type LicenseComponents has the following properties:

- **hand_eye_calibration** (boolean) - hand-eye calibration module
- **rectification** (boolean) - image rectification module
- **stereo** (boolean) - stereo matching module

Template object

```
{
  "hand_eye_calibration": false,
  "rectification": false,
  "stereo": false
}
```

LicenseComponents objects are nested in *LicenseInfo*.

LicenseConstraints: Version constrains for modules.

An object of type LicenseConstraints has the following properties:

- **image_version** (*LicenseComponentConstraint*) - see description of *LicenseComponentConstraint*

Template object

```
{
  "image_version": {
    "max_version": "string",
    "min_version": "string"
  }
}
```

LicenseConstraints objects are nested in *LicenseInfo*.

LicenseInfo: Information about the currently applied software license on the device.

An object of type LicenseInfo has the following properties:

- **components** (*LicenseComponents*) - see description of *LicenseComponents*
- **components_constraints** (*LicenseConstraints*) - see description of *LicenseConstraints*
- **valid** (boolean) - indicates whether the license is valid or not

Template object

```
{
  "components": {
    "hand_eye_calibration": false,
    "rectification": false,
    "stereo": false
  },
  "components_constraints": {
    "image_version": {
      "max_version": "string",
      "min_version": "string"
    }
  },
  "valid": false
}
```

LicenseInfo objects are used in the following requests:

- *GET /system/license*

Log: Content of a specific log file represented in JSON format.

An object of type Log has the following properties:

- **date** (float) - UNIX time when log was last modified
- **log** (array of *LogEntry*) - the actual log entries
- **name** (string) - name of log file
- **size** (integer) - size of log file in bytes

Template object

```
{
  "date": 0,
  "log": [
    {
      "component": "string",
      "level": "string",
      "message": "string",
      "timestamp": 0
    },
    {
      "component": "string",
      "level": "string",
      "message": "string",
      "timestamp": 0
    }
  ],
  "name": "string",
  "size": 0
}
```

Log objects are used in the following requests:

- *GET /logs/{log}*

LogEntry: Representation of a single log entry in a log file.

An object of type LogEntry has the following properties:

- **component** (string) - module name that created this entry
- **level** (string) - log level (one of `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`)
- **message** (string) - actual log message
- **timestamp** (float) - Unix time of log entry

Template object

```
{
  "component": "string",
  "level": "string",
  "message": "string",
  "timestamp": 0
}
```

LogEntry objects are nested in *Log*.

LogInfo: Information about a specific log file.

An object of type LogInfo has the following properties:

- **date** (float) - UNIX time when log was last modified
- **name** (string) - name of log file
- **size** (integer) - size of log file in bytes

Template object

```
{
  "date": 0,
  "name": "string",
  "size": 0
}
```

LogInfo objects are used in the following requests:

- *GET /logs*

NodeInfo: Description of a computational node running on device.

An object of type NodeInfo has the following properties:

- **name** (string) - name of the node
- **parameters** (array of string) - list of the node's run-time parameters
- **services** (array of string) - list of the services this node offers
- **status** (string) - status of the node (one of `unknown`, `down`, `idle`, `running`)

Template object

```
{
  "name": "string",
  "parameters": [
    "string",
    "string"
  ],
  "services": [
    "string",
    "string"
  ],
  "status": "string"
}
```

NodeInfo objects are used in the following requests:

- *GET /nodes*
- *GET /nodes/{node}*
- *GET /pipelines/{pipeline}/nodes*
- *GET /pipelines/{pipeline}/nodes/{node}*

NodeStatus: Detailed current status of the node including run-time statistics.

An object of type NodeStatus has the following properties:

- **status** (string) - status of the node (one of `unknown`, `down`, `idle`, `running`)
- **timestamp** (float) - Unix time when values were last updated
- **values** (object) - dictionary with current status/statistics of the node

Template object

```
{
  "status": "string",
  "timestamp": 0,
  "values": {}
}
```

NodeStatus objects are used in the following requests:

- *GET /nodes/{node}/status*
- *GET /pipelines/{pipeline}/nodes/{node}/status*

Parameter: Representation of a node's run-time parameter. The parameter's 'value' type (and hence the types of the 'min', 'max' and 'default' fields) can be inferred from the 'type' field and might be one of the built-in primitive data types.

An object of type Parameter has the following properties:

- **default** (type not defined) - the parameter's default value
- **description** (string) - description of the parameter
- **max** (type not defined) - maximum value this parameter can be assigned to
- **min** (type not defined) - minimum value this parameter can be assigned to
- **name** (string) - name of the parameter
- **type** (string) - the parameter's primitive type represented as string (one of `bool`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `int64`, `uint64`, `float32`, `float64`, `string`)
- **value** (type not defined) - the parameter's current value

Template object

```
{
  "default": {},
  "description": "string",
  "max": {},
  "min": {},
  "name": "string",
  "type": "string",
  "value": {}
}
```

Parameter objects are used in the following requests:

- *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: Parameter name and value. The parameter's 'value' type (and hence the types of the 'min', 'max' and 'default' fields) can be inferred from the 'type' field and might be one of the built-in primitive data types.

An object of type ParameterNameValue has the following properties:

- **name** (string) - name of the parameter
- **value** (type not defined) - the parameter's current value

Template object

```
{
  "name": "string",
  "value": {}
}
```

ParameterNameValue objects are used in the following requests:

- *PUT /pipelines/{pipeline}/nodes/{node}/parameters*

ParameterValue: Parameter value. The parameter's 'value' type (and hence the types of the 'min', 'max' and 'default' fields) can be inferred from the 'type' field and might be one of the built-in primitive data types.

An object of type ParameterValue has the following properties:

- **value** (type not defined) - the parameter's current value

Template object

```
{
  "value": {}
}
```

ParameterValue objects are used in the following requests:

- *PUT /pipelines/{pipeline}/nodes/{node}/parameters/{param}*

ProxySettings: HTTP proxy settings for pulling container images and git repos

An object of type ProxySettings has the following properties:

- **http_proxy** (string) - HTTP proxy
- **https_proxy** (string) - HTTPS proxy

Template object

```
{
  "http_proxy": "string",
  "https_proxy": "string"
}
```

Service: Representation of a service that a node offers.

An object of type Service has the following properties:

- **args** (*ServiceArgs*) - see description of *ServiceArgs*
- **description** (string) - short description of this service
- **name** (string) - name of the service
- **response** (*ServiceResponse*) - see description of *ServiceResponse*

Template object

```
{
  "args": {},
  "description": "string",
  "name": "string",
  "response": {}
}
```

Service objects are used in the following requests:

- *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: Arguments required to call a service with. The general representation of these arguments is a (nested) dictionary. The specific content of this dictionary depends on the respective node and service call.

ServiceArgs objects are nested in *Service*.

ServiceResponse: The response returned by the service call. The general representation of this response is a (nested) dictionary. The specific content of this dictionary depends on the respective node and service call.

ServiceResponse objects are nested in *Service*.

SysInfo: System information about the device.

An object of type SysInfo has the following properties:

- **hostname** (string) - Hostname
- **link_speed** (integer) - Ethernet link speed in Mbps
- **mac** (string) - MAC address
- **ready** (boolean) - system is fully booted and ready
- **sensor_interfaces** (object) - Available sensor interfaces with their current link speed
- **serial** (string) - device serial number
- **time** (float) - system time as Unix timestamp
- **ui_lock** (*UILock*) - see description of *UILock*
- **uptime** (float) - system uptime in seconds

Template object

```
{
  "hostname": "string",
  "link_speed": 0,
  "mac": "string",
  "ready": false,
  "sensor_interfaces": {},
  "serial": "string",
  "time": 0,
  "ui_lock": {
    "enabled": false
  },
  "uptime": 0
}
```

SysInfo objects are used in the following requests:

- `GET /system`

Template: Detection template

An object of type Template has the following properties:

- **id** (string) - Unique identifier of the template

Template object

```
{
  "id": "string"
}
```

Template objects are used in the following requests:

- `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.

An object of type UILock has the following properties:

- **enabled** (boolean)

Template object

```
{
  "enabled": false
}
```

UILock objects are nested in *SysInfo*, and are used in the following requests:

- *GET /system/ui_lock*

7.2.4 Swagger UI

The *rc_reason_stack*'s Swagger UI allows developers to easily visualize and interact with the REST-API, e.g., for development and testing. Accessing `http://<host>/api/` or `http://<host>/api/swagger` (the former will automatically be redirected to the latter) opens a visualization of the *rc_reason_stack*'s general API structure including all *available resources and requests* (Section 7.2.2) and offers a simple user interface for exploring all of its features.

> **Note:** Users must be aware that, although the *rc_reason_stack*'s Swagger UI is designed to explore and test the REST-API, it is a fully functional interface. That is, any issued requests are actually processed and particularly `PUT`, `POST`, and `DELETE` requests might change the overall status and/or behavior of the device.

Using this interface, available resources and requests can be explored by clicking on them to uncollapse or recollapse them. The following figure shows an example of how to get a node's current status by filling in the necessary parameters (`pipeline` number and `node` name) and clicking *Execute*. This action results in the Swagger UI showing, amongst others, the actual `curl` command that was executed when issuing the request as well as the response body showing the current status of the requested node in a JSON-formatted string.

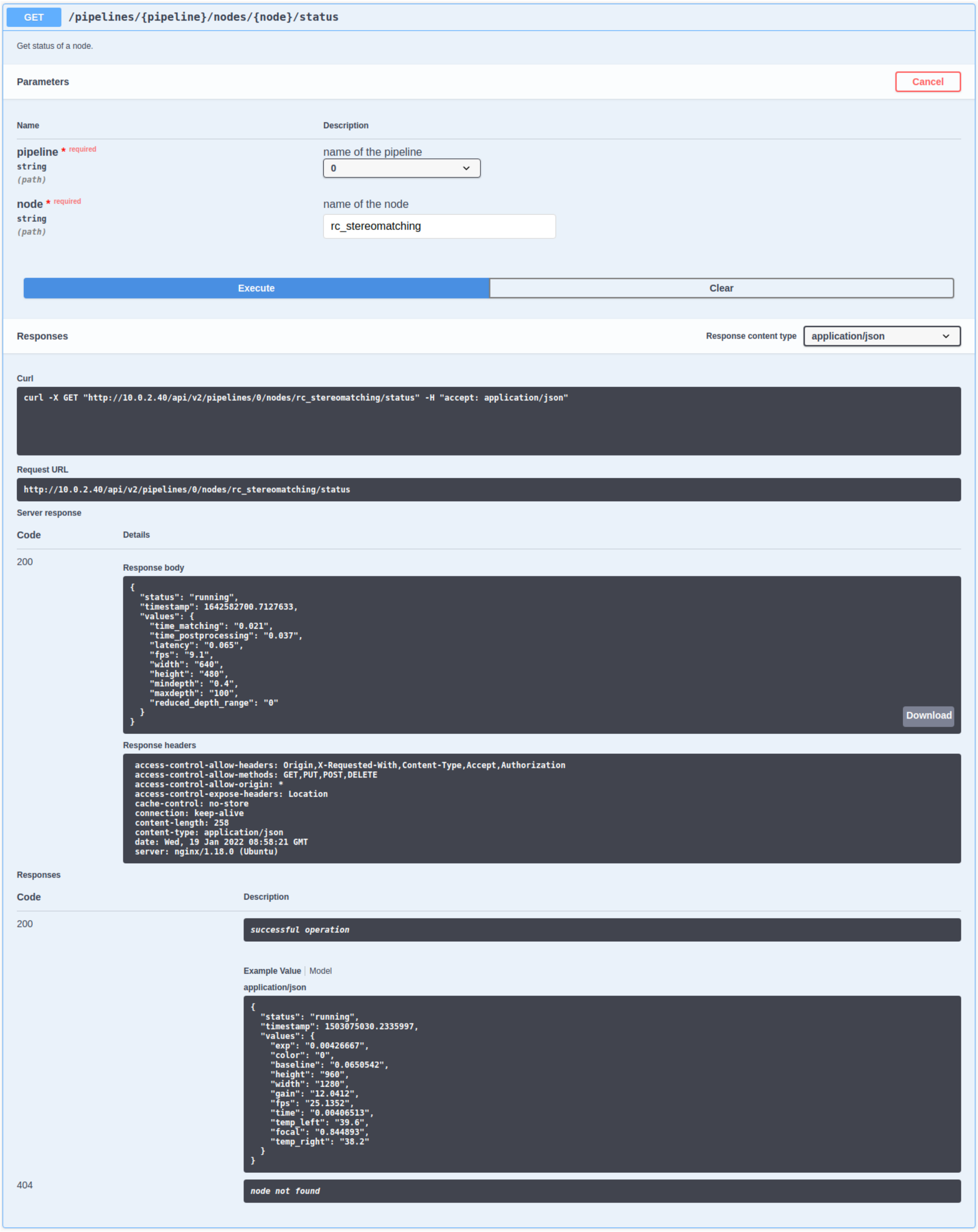

Fig. 7.3: Result of requesting the `rc_stereomatching` node's status

Some actions, such as setting parameters or calling services, require more complex parameters to an HTTP request. The Swagger UI allows developers to explore the attributes required for these actions during run-time, as shown in the next example. In the figure below, the attributes required for the `rc_hand_eye_calibration` node's `set_pose` service are explored by performing a `GET` request on this resource. The response features a full description of the service offered, including all required arguments with their names and types as a JSON-formatted string.

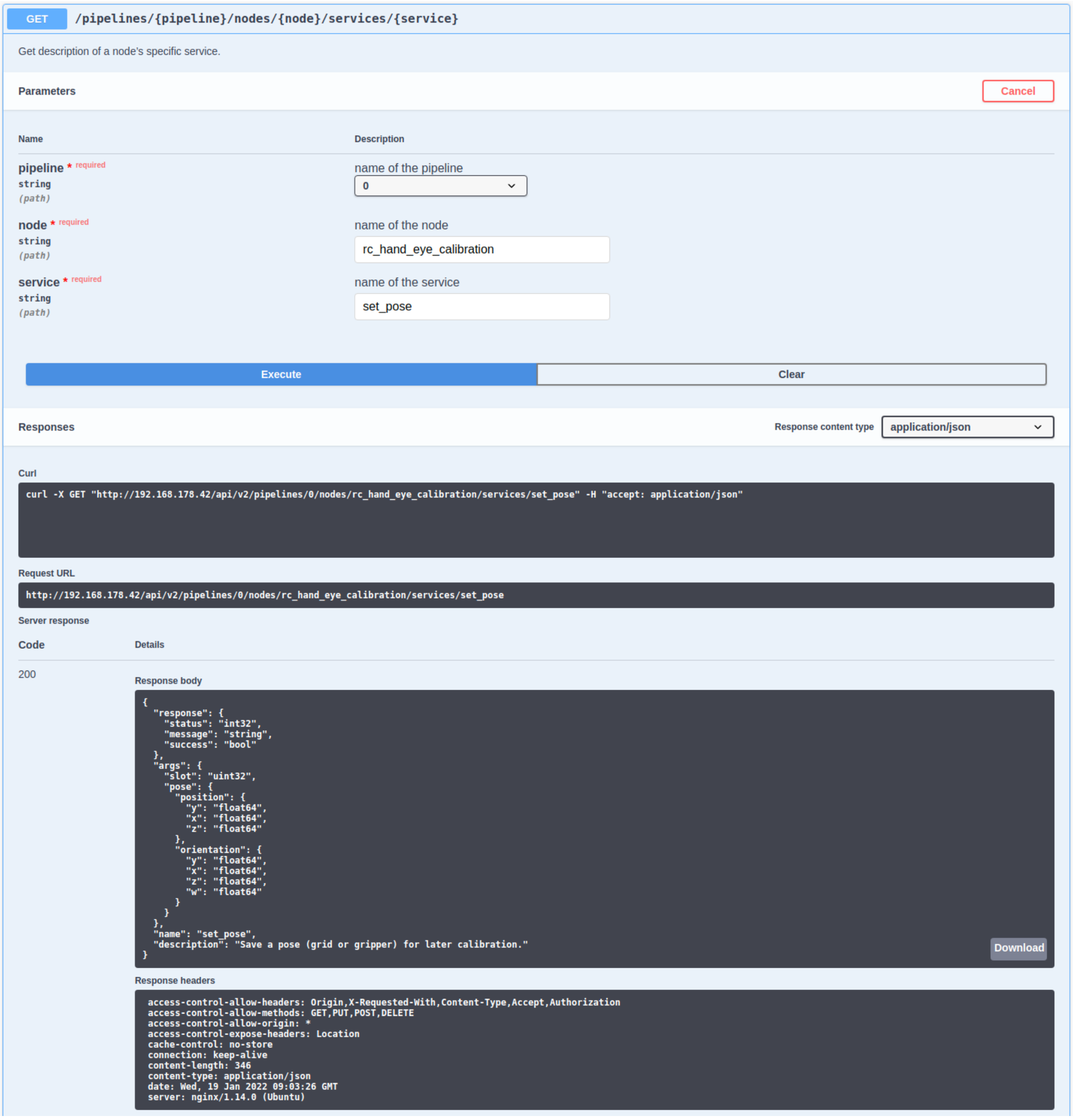

Fig. 7.4: The result of the GET request on the `set_pose` service shows the required arguments for this service call.

Users can easily use this preformatted JSON string as a template for the service arguments to actually call the service:

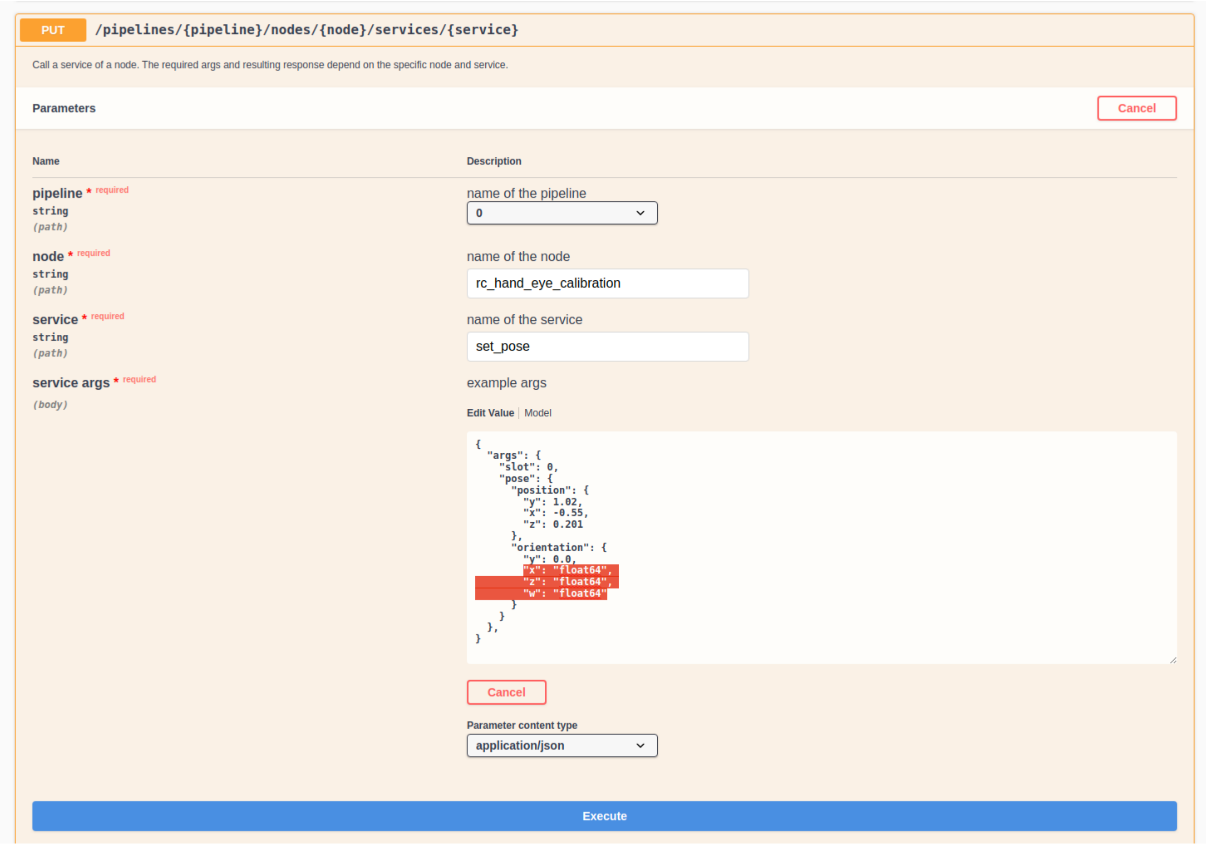

Fig. 7.5: Filling in the arguments of the `set_pose` service request

7.3 Generic Robot Interface

The Generic Robot Interface (GRI) is an integration layer that bridges the *REST-API v2* (Section 7.2) and provides a standardized way to communicate with the software modules using simple TCP socket communication on port 7100. It can be activated via a separate *license* (Section 8.2).

The GRI enables the user to create configurations and save them as numbered jobs. These jobs can be triggered by simple commands from the robot using TCP socket communication. The GRI internally manages the REST-API communication and delivers the selected pose results in a format that can be chosen specifically for the robot.

7.3.1 Job definition

Jobs are pre-configured tasks that can be triggered by the robot application. Each job has a unique ID and contains all the necessary information for a specific operation, e.g. computing grasps for bin picking or changing run-time parameters of a module. Once configured, the robot can execute these jobs using simple socket commands and, if applicable, receive the returned poses.

7.3.1.1 Job Types

The Generic Robot Interface supports three types of jobs:

Pipeline service job (CALL_PIPELINE_SERVICE)

This job calls a service on a specific camera pipeline, e.g. to detect objects or compute grasps, and returns pose data to the robot (e.g. grasp poses).

A pipeline service job consists of:

- `job_type`: the job type `CALL_PIPELINE_SERVICE`
- `name`: name of the job (descriptive name to distinguish jobs)
- `pipeline`: the camera pipeline to be used for the job (e.g. "0")
- `node`: the REST-API name of the pipeline node that should be used (e.g. `rc_load_carrier`)
- `service`: the REST-API name of the service to call
- `args`: the REST-API json arguments to pass to the service
- `selected_return`: the REST-API name of the field to return

A sample pipeline service job definition is:

```
{
   "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"
}
```

The available values for `selected_return` depend on the chosen node and can be e.g. `grasps` or `matches`. Refer to the service definitions of the corresponding module for details about `node`, `service`, `args` and `selected_return`.

Global service job (CALL_GLOBAL_SERVICE)

This job calls a service that is not tied to a specific pipeline, e.g. database services for setting regions of interest or load carriers. Global service jobs do not return any poses.

A global service job consists of:

- `job_type`: the job type `CALL_GLOBAL_SERVICE`
- `name`: name of the job (descriptive name to distinguish jobs)
- `node`: the REST-API name of the global node that should be used (e.g. `rc_load_carrier_db`)
- `service`: the REST-API name of the service to call
- `args`: the REST-API json arguments to pass to the service

A sample global job definition is:

```
{
   "args": {
      "region_of_interest_2d": {
         "id": "2d_roi",
         "width": 526,
         "height": 501,
         "offset_x": 558,
         "offset_y": 307
      }
   },
   "job_type": "CALL_GLOBAL_SERVICE",
   "name": "Set 2D ROI",
   "node": "rc_roi_db",
```

(continues on next page)

(continued from previous page)

```
    "service": "set_region_of_interest_2d"
}
```

Refer to the service definitions of the corresponding module for details about `node`, `service` and `args`.

Parameter setting job (SET_PIPELINE_PARAMETER)

This job sets run-time parameters on pipeline nodes, e.g. for adjusting camera or detection module settings. Parameter setting services do not return any poses.

A parameter setting job consists of:

- `job_type`: the job type `SET_PIPELINE_PARAMETER`
- `name`: name of the job (descriptive name to distinguish jobs)
- `pipeline`: the camera pipeline to be used for the job (e.g. "0")
- `node`: the REST-API name of the pipeline node that should be used (e.g. `rc_stereomatching`)
- `parameters`: the parameters to set as key-value pairs

A sample parameter job definition is:

```
{
    "job_type": "SET_PIPELINE_PARAMETERS",
    "name": "Set Stereo Parameters",
    "node": "rc_stereomatching",
    "parameters": {
        "maxdepth": 2,
        "quality": "High"
    },
    "pipeline": "0"
}
```

Refer to the run-time parameter definitions of the corresponding module for details about `node` and `parameters`.

The jobs can be defined via the Web GUI or via the REST-API (see *Job and HEC_config API*).

7.3.1.2 Primary and related objects

The *primary objects* are the `selected_return` objects, e.g. `grasps`. The *related objects* are then the `items` or `matches` that correspond to the returned grasp. While a primary object *grasp* as exactly one related object *item* or *match*, a primary object *match* can have multiple related objects *grasps*.

7.3.1.3 Execution modes

The Generic Robot Interface supports two execution modes to optimize the robot's cycle time:

- Synchronous Execution: The robot triggers a job and waits for the first result to arrive. This mode should be chosen when results are required immediately.
- Asynchronous Execution: The robot starts a job and can continue with other operations while the job is running in the background. The job status can be queried and results can be retrieved when ready. This mode maximizes efficiency during long detection times.

7.3.2 Hand-Eye Calibration

A hand-eye calibration configuration can be defined for each camera pipeline to allow for programmatic hand-eye calibration using the GRI. Each hand-eye calibration configuration consists of the following information:

- `grid_height`: height of the calibration grid in meters
- `grid_width`: width of the calibration grid in meters
- `robot_mounted`: boolean that determines whether the camera is mounted on the robot
- `tcp_offset`: 0 for 6DOF robots. For 4DOF robots: the signed offset from the TCP to the camera coordinate system (robot-mounted sensor) or the visible surface of the calibration grid (statically mounted sensor) along the TCP rotation axis in meters.
- `tcp_rotation_axis`: -1 for 6DOF robots. For 4DOF robots: determines the axis of the robot frame around which the robot can rotate its TCP (0 is used for X, 1 for Y and 2 for the Z axis).

More detailed information about these settings and the hand-eye calibration in general is given in *Hand-eye calibration*.

The hand-eye calibration configurations can be set via the Web GUI or via the REST-API (see *Job and HEC_config API*).

7.3.3 GRI binary protocol specification

This specification defines the exact on-wire format for client-server messages. A message consists of a fixed 8-byte header and a body whose layout depends on the protocol version. Currently, there is only protocol version 1.

Note: All multi-byte integers are **little-endian**. Types are `uint8` (8-bit unsigned), `int16` (16-bit signed), `int32` (32-bit signed).

7.3.3.1 Message header (8 bytes)

Table 7.1: Message header definition

Field	Type	Size	Description
magic_number	uint32	4	ASCII tag *"GRI0"*, bytes *47 52 49 00* (little-endian)
protocol_version	uint8	1	Protocol version: currently *1*
message_length	uint8	1	Total message size (bytes), incl. header + body
pose_format	uint8	1	Pose data format (see *Pose formats*)
action	uint8	1	Command/action (see *Actions*)

7.3.3.2 Pose formats

The GRI always uses **millimeters** for representing a position. The following tables show different rotation formats that can be chosen to match to the rotation representation of the used robot. The formats are grouped by non-Euler rotation formats, Tait-Bryan-Euler rotation formats (all three axes are used) and proper Euler rotation formats (first and last rotation axis are the same).

Table 7.2: Non-Euler rotation formats

Name	Value	rot_1	rot_2	rot_3	rot_4	Units	Robot Example
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 the following notation primes indicate successive rotations in the intrinsic frame (e.g., Y' = rotation about Y-axis after first rotation). `_B` and `_F` determine the order in which the rotation components are given. `F` stands for *forward*, meaning that the rotation components are given in the same order as the rotation is applied, and `B` stands for *backward*, meaning the rotation components are given in reverse order. `_RAD` and `_DEG` determine whether the rotation components are given in radians or degrees, respectively, if applicable. So the format EULER_ZYX_B_DEG means that the intrinsic rotation order is z-y'-x" (first rotate around the z axis, then rotate around the new y axis, then rotate around the new x axis), the order in which the rotation components are given is backward (so the first rotation element is the angle around the x axis), and the angels are given in degrees.

Table 7.3: Tait-Bryan-Euler rotation formats. Primes indicate successive rotations in the intrinsic frame (e.g., Y’ = rotation about Y-axis after first rotation). **_F (Forward)**: *[1st, 2nd, 3rd]* | **_B (Backward)**: *[3rd, 2nd, 1st]*, **_DEG: degrees** | **_RAD: radian**.

Name	Value	rot_1	rot_2	rot_3	rot_4	Units	Robot Example
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	

Table 7.4: Euler rotation formats. Primes indicate successive rotations in the intrinsic frame (e.g., Y' = rotation about Y-axis after first rotation). **_F (Forward)**: *[1st, 2nd, 3rd]* | **_B (Backward)**: *[3rd, 2nd, 1st]*, **_DEG: degrees | _RAD: radian**.

Name	Value	rot_1	rot_2	rot_3	rot_4	Units	Robot Example
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	48	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	

All pose components (position **and** rotation) are `int32` scaled by **1,000,000**.

- Float to Int: `int = round(float * 1000000)`
- Int to Float: `float = int / 1000000.0`
- Positions in **millimeters** before scaling
- Angles in **degrees/radians** (per format) before scaling
- Quaternions unitless, same scaling
- `rot_4` unused for Euler/axis-angle (set to 0)

7.3.3.3 Actions

The following actions can be sent.

Table 7.5: GRI actions

Name	Value	Description
STATUS	1	Get system readiness; maps readiness to `data_2` (1 or 0)
TRIGGER_JOB_SYNC	2	Execute job synchronously
TRIGGER_JOB_ASYNC	3	Start job asynchronously
GET_JOB_STATUS	4	Query async job status (see *Job status*)
GET_NEXT_POSE	5	Retrieve next available result
GET_RELATED_POSE	6	Retrieve next related pose
HEC_INIT	7	Initialize hand-eye calibration
HEC_SET_POSE	8	Provide/store calibration pose
HEC_CALIBRATE	9	Run calibration and save results

STATUS (1)

Returns system readiness information fetched from the *rc_reason_stack* in `data_2` (1 if ready, 0 if not).

TRIGGER_JOB_SYNC (2)

Runs the job and returns the **first** result immediately; additional results are stored for later retrieval. On success with results, `error_code` will be zero and the pose will be filled. If no results are returned, `error_code` will be `NO_POSES_FOUND` (positive warning). It also reports:

- `data_1` = node's `return_code` value
- `data_2` = number of remaining *primary objects* (ref. *Primary and related objects*)
- `data_3` = number of remaining *related objects* (ref. *Primary and related objects*)

TRIGGER_JOB_ASYNC (3)

Starts the job and returns immediately. The job's status can be polled with `GET_JOB_STATUS` (4) (see *Job status*) and the results can be fetched with `GET_NEXT_POSE` (5), as soon as the job is `DONE`.

GET_JOB_STATUS (4)

Returns the job status. It reports:

- `data_1` = node's `return_code` value
- `data_2` = job status (see table *Job status values*)

Error details flow through `error_code`.

GET_NEXT_POSE (5)

Returns the next result of the *primary object*. It also reports:

- `data_1` = node's `return_code` value
- `data_2` = number of remaining *primary objects* (ref. *Primary and related objects*)
- `data_3` = number of remaining *related objects* (ref. *Primary and related objects*)

When no more primary objects are available, it returns `NO_POSES_FOUND` and resets the job.

GET_RELATED_POSE (6)

Returns the next pose of the *related object* corresponding to the current *primary object*. It also reports:

- `data_1` = node's `return_code` value
- `data_2` = number of remaining *primary objects* (ref. *Primary and related objects*)
- `data_3` = number of remaining *related objects* (ref. *Primary and related objects*)

If no related poses were found, it returns `NO_RELATED_POSES`.

HEC_INIT (7)

This action initializes the hand-eye calibration. It clears any existing calibration data, applies the pipeline's configuration parameters and prepares the system for recording new poses. `data_1` specifies the target pipeline.

HEC_SET_POSE (8)

This action is to be used eight times to record distinct robot poses with visible calibration pattern. The field `data_2` is used to specify the image storage slot (0-7). A previous pose in a slot will be overwritten if a slot is reused. Each pose must provide a different view of the calibration pattern, as described in *Hand-eye calibration*. `data_1` specifies the target pipeline.

HEC_CALIBRATE (9)

This action processes all recorded poses and calculates the transformation between camera and robot. It automatically saves successful calibration results. `data_1` specifies the target pipeline.

7.3.3.4 Job status

The following job status values can be returned.

Table 7.6: Job status values

Name	Value
INACTIVE	1
RUNNING	2
DONE	3
FAILED	4

7.3.3.5 Body definitions

There are different body definitions depending on whether it is a request that is sent or a response that is received. The request body consists of 54 bytes in total and its definition is given in table *Request body definition*.

Table 7.7: Request body definition

Field	Type	Size	Description
header	struct	8	Message header (see *Message header (8 bytes)*)
job_id	uint16	2	Unique job ID from job configuration
pos_x	int32	4	Position X (scaled by 10^6)
pos_y	int32	4	Position Y (scaled by 10^6)
pos_z	int32	4	Position Z (scaled by 10^6)
rot_1	int32	4	Rotation component 1 (scaled by 10^6)
rot_2	int32	4	Rotation component 2 (scaled by 10^6)
rot_3	int32	4	Rotation component 3 (scaled by 10^6)
rot_4	int32	4	Rotation component 4 (scaled by 10^6)
data_1	int32	4	Additional parameter 1
data_2	int32	4	Additional parameter 2
data_3	int32	4	Additional parameter 3
data_4	int32	4	Additional parameter 4

The job ID is the unique identifier from the job configuration. The usage of the fields `data_1...data_4` depends on the action and job. They are set to *0* if unused.

The response body consists of 80 bytes in total. Its definition is given in table *Response body definition*.

Table 7.8: Response body definition

Field	Type	Size	Description
header	struct	8	Protocol header
job_id	uint16	2	Processed job number
error_code	int16	2	GRI result status (severity by sign)
pos_x	int32	4	Position X (scaled by 10^6)
pos_y	int32	4	Position Y (scaled by 10^6)
pos_z	int32	4	Position Z (scaled by 10^6)
rot_1	int32	4	Rotation component 1 (scaled by 10^6)
rot_2	int32	4	Rotation component 2 (scaled by 10^6)
rot_3	int32	4	Rotation component 3 (scaled by 10^6)
rot_4	int32	4	Rotation component 4 (scaled by 10^6)
data_1	int32	4	Node's return code (0 if none)
data_2	int32	4	Additional result 2
data_3	int32	4	Additional result 3
data_4	int32	4	Additional result 4
data_5	int32	4	Additional result 5
data_6	int32	4	Additional result 6
data_7	int32	4	Additional result 7
data_8	int32	4	Additional result 8
data_9	int32	4	Additional result 9
data_10	int32	4	Additional result 10

Note: For `rc_measure`, mean_z is mapped to pos_x/pos_y/pos_z.

7.3.3.6 Error codes and semantics

The `error_code` is `int16` and encodes errors/warnings by sign:

- Negative < *0* = **error** (failure)
- Zero = *0* = **success**
- Positive > *0* = **warning** (success with caveat)

The tables below give the different error codes and are split by sign and sorted.

Success

Name	Value	Description
NO_ERROR	0	Operation successful

Negative error codes

Name	Value	Description
UNKNOWN_ERROR	-1	GRI internal, unspecified
INTERNAL_ERROR	-2	GRI internal system error
API_NOT_REACHABLE	-3	Cannot reach API
API_RESPONSE_ERROR	-4	API returned a negative code
PIPELINE_NOT_AVAILABLE	-5	Processing pipeline unavailable
INVALID_REQUEST_ERROR	-6	Malformed request
INVALID_REQUEST_LENGTH	-7	Wrong message length
INVALID_ACTION	-8	Unsupported action
PROCESSING_TIMEOUT	-9	Operation timed out
UNKNOWN_PROTOCOL_VERSION	-10	Protocol version not supported
WRONG_PROTOCOL_FOR_JOB	-11	Job does not match protocol version
JOB_DOES_NOT_EXIST	-12	Invalid job ID
MISCONFIGURED_JOB	-13	Invalid job configuration
HEC_CONFIG_ERROR	-14	Invalid configuration parameters
HEC_INIT_ERROR	-15	Calibration init failed
HEC_SET_POSE_ERROR	-16	Failed to record pose in specified slot
HEC_CALIBRATE_ERROR	-17	Unable to compute calibration from recorded poses
HEC_INSUFFICIENT_DETECTION	-18	Calibration grid not visible or not detected

Positive codes

Name	Value	Description
NO_POSES_FOUND	1	No results available
NO_RELATED_POSES	2	No related data found
NO_RETURN_SPECIFIED	3	Job configured with no return type
JOB_STILL_RUNNING	4	Async job not complete

Node return code semantics

The modules/nodes may return a `return_code`. This node return code is placed in the response `data_1` field (defaults to 0 if no code). The GRI's primary status is in `error_code` (sign-based semantics).

7.3.4 Integration with a robot

The Generic Robot Interface offers communication on port 7100.

For integrating the GRI communication with a robot, examples for different robot languages are given in https://github.com/roboception/rc_generic_robot_interface_robot.

Different robot platforms can be supported by implementing a TCP socket client following the GRI binary protocol (ref. *GRI binary protocol specification*). This requires a robot controller with TCP/IP support and the ability to pack robot poses into binary messages and to parse binary messages into robot poses.

The implementation steps are as follows:

1. Create TCP socket connection
2. Compose request message:
   - Set message header (8 bytes)
   - Set job ID (2 bytes)
   - Pack position (12 bytes, 3x int32)
   - Pack rotation (16 bytes, 4x int32)
   - Pack additional data (16 bytes, 4x int32)

3. Send request (54 bytes total)
4. Receive response (80 bytes total)
5. Parse response:
    - Message header (8 bytes)
    - Job ID (2 bytes)
    - Error code (2 bytes)
    - Position (12 bytes, 3x int32)
    - Rotation (16 bytes, 4x int32)
    - Additional data (40 bytes, 10x int32)

7.3.5 Job and HEC_config API

The job definitions and the definitions of HEC_configs for hand-eye calibration can be set, retrieved and deleted via the following REST-API endpoints.

GET /generic_robot_interface/hec_configs

Get defined hand-eye calibration configurations

Template request

```
GET /api/v2/generic_robot_interface/hec_configs HTTP/1.1
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "0": {
    "grid_height": 0.18,
    "grid_width": 0.26,
    "robot_mounted": true,
    "tcp_offset": 0,
    "tcp_rotation_axis": -1
  }
}
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

GET /generic_robot_interface/hec_configs/{pipeline}

Get hand-eye calibration configuration for the selected pipeline

Template request

```
GET /api/v2/generic_robot_interface/hec_configs/<pipeline> HTTP/1.1
```

Sample response

```
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
}
```

Parameters

- **pipeline** (*string*) – pipeline of the hand-eye calibration configuration *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

PUT /generic_robot_interface/hec_configs/{pipeline}

Sets a hand-eye calibration configuration for the selected pipeline.

Template request

```
PUT /api/v2/generic_robot_interface/hec_configs/<pipeline> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "return_code": {
    "message": "HEC configuration saved successfully",
    "value": 0
  }
}
```

Parameters

- **pipeline** (*string*) – pipeline of the hand-eye calibration configuration *(required)*

Request JSON Object

- **hand-eye calibration configuration** (*object*) – example args *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

DELETE /generic_robot_interface/hec_configs/{pipeline}

Remove a hand-eye calibration configuration.

Template request

```
DELETE /api/v2/generic_robot_interface/hec_configs/<pipeline> HTTP/1.1
Accept: application/json application/ubjson
```

Parameters

- **pipeline** (*string*) – pipeline of the hand-eye calibration configuration *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation
- 403 Forbidden – forbidden, e.g. because there is no valid license for this module.
- 404 Not Found – hec config for the given pipeline not found

GET /generic_robot_interface/jobs

Get defined jobs

Template request

```
GET /api/v2/generic_robot_interface/jobs HTTP/1.1
```

Sample response

```
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"
  }
}
```

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

GET /generic_robot_interface/jobs/{job_id}

Get selected job definition

Template request

```
GET /api/v2/generic_robot_interface/jobs/<job_id> HTTP/1.1
```

Sample response

```
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"
}
```

Parameters

- **job_id** (*string*) – ID of the job *(required)*

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

PUT /generic_robot_interface/jobs/{job_id}

Sets a job definition for the selected job ID. The required keys depend on the chosen job_type.

Template request

```
PUT /api/v2/generic_robot_interface/jobs/<job_id> HTTP/1.1
Accept: application/json application/ubjson

{}
```

Sample response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "job_id": "1",
  "return_code": {
    "message": "Job configuration updated successfully",
    "value": 0
```

(continues on next page)

(continued from previous page)

```
  }
}
```

Parameters

- **job_id** (*string*) – ID of the job *(required)*

Request JSON Object

- **job definition** (*object*) – example args *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation

DELETE /generic_robot_interface/jobs/{job_id}

Remove a job definition.

Template request

```
DELETE /api/v2/generic_robot_interface/jobs/<job_id> HTTP/1.1
Accept: application/json application/ubjson
```

Parameters

- **job_id** (*string*) – ID of the job *(required)*

Request Headers

- Accept – application/json application/ubjson

Response Headers

- Content-Type – application/json application/ubjson

Status Codes

- 200 OK – successful operation
- 403 Forbidden – forbidden, e.g. because there is no valid license for this module.
- 404 Not Found – job with given id not found

7.4 OPC UA interface

The *rc_reason_stack* also offers an optional OPC UA interface running on TCP port 4840. The OPC UA server can be activated via a separate *license* (Section 8.2).

The OPC UA server provides access to parameters and services of all available software modules analogous to the REST-API. To browse the OPC UA Address Space use e.g. the freely available UAExpert GUI client.

The OPC UA server uses the `DataTypeDefinition` attribute (available in OPC UA version 1.04) for custom datatypes and also uses methods and variable length arrays. Please check if your OPC UA client supports this.

Note: The OPC UA server currently only supports the equivalent of API version 1 (i.e. only camera pipeline 0).

Please contact support@roboception.de if you are interested in using the OPC UA server.

7.5 KUKA Ethernet KRL Interface

The *rc_reason_stack* provides an Ethernet KRL Interface (EKI Bridge), which allows communicating with the *rc_reason_stack* from KUKA KRL via KUKA.EthernetKRL XML.

Note: The component is optional and requires a separate Roboception's EKIBridge *license* (Section 8.2) to be purchased.

Note: The KUKA.EthernetKRL add-on software package version 2.2 or newer must be activated on the robot controller to use this component.

The EKI Bridge can be used to programmatically to

- do service calls, e.g. to start and stop individual computational nodes, or to use offered services such as the hand-eye calibration or the computation of grasp poses;
- set and get run-time parameters of computation nodes, e.g. of the camera, or disparity calculation.

Note: A known limitation of the EKI Bridge is that strings representing valid numbers will be converted to int/float. Hence user-defined names (like ROI IDs, etc.) should always contain at least one letter so they can be used in service call arguments.

7.5.1 Ethernet connection configuration

The EKI Bridge listens on port 7000 for EKI XML messages and transparently bridges the *rc_reason_stack*'s *REST-API v2* (Section 7.2). The received EKI messages are transformed to JSON and forwarded to the *rc_reason_stack*'s REST-API. The response from the REST-API is transformed back to EKI XML.

The EKI Bridge gives access to run-time parameters and offered services of all computational nodes described in *Software modules* (Section 6).

The Ethernet connection to the *rc_reason_stack* on the robot controller is configured using XML configuration files.

The EKI XML configuration files of all nodes running on the *rc_reason_stack* are available for download at:

https://doc.rc-visard.com/latest/en/eki.html#eki-xml-configuration-files

Each node offering run-time parameters has an XML configuration file for setting and getting its parameters. These are named following the scheme <node_name>-parameters.xml. Each node's service has its own XML configuration file. These are named following the scheme <node_name>-<service_name>.xml.

The IP of the host PC running the *rc_reason_stack* needs to be filled into the XML file.

The port is already set to 7000, which corresponds to pipeline 0. This needs to be adjusted if a different pipeline should be used. The port number is 7000 + pipeline number, so 7001 for pipeline 1, etc.

These files must be stored in the directory `C:\KRC\ROBOTER\Config\User\Common\EthernetKRL` of the robot controller and they are read in when a connection is initialized.

As an example, an Ethernet connection to configure the rc_stereomatching parameters is established with the following KRL code.

```
DECL EKI_Status RET
RET = EKI_INIT("rc_stereomatching-parameters")
RET = EKI_Open("rc_stereomatching-parameters")

; ----------- Desired operation -----------

RET = EKI_Close("rc_stereomatching-parameters")
```

Note: The EKI Bridge automatically terminates the connection to the client if the received XML telegram is invalid.

7.5.2 Generic XML structure

For data transmission, the EKI Bridge uses <req> as root XML element (short for request).

The root tag always includes the following elements.

- <node>. This includes a child XML element used by the EKI Bridge to identify the target node. The node name is already included in the XML configuration file.
- <end_of_request>. End of request flag that triggers the request.

The following listing shows the generic XML structure for data transmission.

```
<SEND>
  <XML>
    <ELEMENT Tag="req/node/<node_name>" Type="STRING"/>
    <ELEMENT Tag="req/end_of_request" Type="BOOL"/>
  </XML>
</SEND>
```

For data reception, the EKI Bridge uses <res> as root XML element (short for response). The root tag always includes a <return_code> child element.

```
<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>
```

Note: By default the XML configuration files uses 998 as flag to notify KRL that the response data record has been received. If this value is already in use, it should be changed in the corresponding XML configuration file.

7.5.2.1 Return code

The <return_code> element consists of a value and a message attribute.

As for all other components, a successful request returns with a res/return_code/@value of 0. Negative values indicate that the request failed. The error message is contained in res/return_code/@message. Positive values indicate that the request succeeded with additional information, contained in res/return_code/@message as well.

The following codes can be issued by the EKI Bridge component.

Table 7.9: Return codes of the EKI Bridge component

Code	Description
0	Success
-1	Parsing error in the conversion from XML to JSON
-2	Internal error
-5	Connection error from the REST-API
-9	Missing or invalid license for EKI Bridge component

Note: The EKI Bridge can also return return code values specific to individual nodes. They are documented in the respective *software module* (Section 6).

Note: Due to limitations in KRL, the maximum length of a string returned by the EKI Bridge is 512 characters. All messages larger than this value are truncated.

7.5.3 Services

For the nodes' services, the XML schema is generated from the service's arguments and response in JavaScript Object Notation (JSON) described in *Software modules* (Section 6). The conversion is done transparently, except for the conversion rules described below.

Conversions of poses:

A pose is a JSON object that includes `position` and `orientation` keys.

```
{
  "pose": {
    "position": {
      "x": "float64",
      "y": "float64",
      "z": "float64",
    },
    "orientation": {
      "x": "float64",
      "y": "float64",
      "z": "float64",
      "w": "float64",
    }
  }
}
```

This JSON object is converted to a KRL `FRAME` in the XML message.

```
<pose X="..." Y="..." Z="..." A="..." B="..." C="..."></pose>
```

Positions are converted from meters to millimeters and orientations are converted from quaternions to KUKA ABC (in degrees).

Note: No other unit conversions are included in the EKI Bridge. All dimensions and 3D coordinates that don't belong to a pose are expected and returned in meters.

Arrays:

Arrays are identified by adding the child element `<le>` (short for list element) to the list name. As an example, the JSON object

```
{
  "rectangles": [
    {
      "x": "float64",
      "y": "float64"
    }
  ]
}
```

is converted to the XML fragment

```
<rectangles>
  <le>
    <x>...</x>
    <y>...</y>
  </le>
</rectangles>
```

Use of XML attributes:

All JSON keys whose values are a primitive data type and don't belong to an array are stored in attributes. As an example, the JSON object

```
{
  "item": {
    "uuid": "string",
    "confidence": "float64",
    "rectangle": {
      "x": "float64",
      "y": "float64"
    }
  }
}
```

is converted to the XML fragment

```
<item uuid="..." confidence="...">
  <rectangle x="..." y="...">
  </rectangle>
</item>
```

7.5.3.1 Request XML structure

The <SEND> element in the XML configuration file for a generic service follows the specification below.

```
<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>
```

The <service> element includes a child XML element that is used by the EKI Bridge to identify the target service from the XML telegram. The service name is already included in the configuration file.

The <args> element includes the service arguments and should be configured with EKI_Set<Type> KRL instructions.

As an example, the <SEND> element of the rc_load_carrier_db's get_load_carriers service (see *LoadCarrierDB*, Section 6.5.1) is:

```
<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>
```

The <end_of_request> element allows to have arrays in the request. For configuring an array, the request is split into as many packages as the size of the array. The last telegram contains all tags, including the <end_of_request> flag, while all other telegrams contain one array element each.

As an example, for requesting two load carrier models to the rc_load_carrier_db's get_load_carriers service, the user needs to send two XML messages. The first XML telegram is:

```
<req>
  <args>
    <load_carrier_ids>
      <le>load_carrier1</le>
    </load_carrier_ids>
  </args>
</req>
```

This telegram can be sent from KRL with the EKI_Send command, by specifying the list element as path:

```
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")
```

The second telegram includes all tags and triggers the request to the rc_load_carrier_db node:

```
<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>
```

This telegram can be sent from KRL by specifying req as path for EKI_Send:

```
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 Response XML structure

The <RECEIVE> element in the XML configuration file for a generic service follows the specification below:

```
<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>
```

As an example, the `<RECEIVE>` element of the `rc_april_tag_detect`'s `detect` service (see *TagDetect*, Section 6.3.3) is:

```
<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"/>
    <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>
```

For arrays, the response includes multiple instances of the same XML element. Each element is written into a separate buffer within EKI and can be read from the buffer with KRL instructions. The number of instances can be requested with `EKI_CheckBuffer` and each instance can then be read by calling `EKI_Get<Type>`.

As an example, the tag poses received after a call to the `rc_april_tag_detect`'s `detect` service can be read in KRL using the following code:

```
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")
```

Note: Before each request from EKI to the *rc_reason_stack*, all buffers should be cleared in order to store only the current response in the EKI buffers.

7.5.4 Parameters

All nodes' parameters can be set and queried from the EKI Bridge. The XML configuration file for a generic node follows the specification below:

```
<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"/>
    <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>
```

The request is interpreted as a *get* request if all parameter's `value` attributes are empty. If any `value` attribute is non-empty, it is interpreted as *set* request of the non-empty parameters.

As an example, the current value of all parameters of `rc_stereomatching` can be queried using the XML telegram:

```
<req>
  <node>
    <rc_stereomatching></rc_stereomatching>
  </node>
  <parameters></parameters>
  <end_of_request></end_of_request>
</req>
```

This XML telegram can be sent out with Ethernet KRL using:

```
DECL EKI_STATUS RET
RET = EKI_Send("rc_stereomatching-parameters", "req")
```

The response from the EKI Bridge contains all parameters:

```
<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"/>
```

(continues on next page)

(continued from previous page)

```
  </parameters>
  <return_code message="" value="0"/>
</res>
```

The `quality` parameter of `rc_stereomatching` can be set to `Low` by the XML telegram:

```
<req>
    <node>
      <rc_stereomatching></rc_stereomatching>
    </node>
    <parameters>
      <quality value="Low"></quality>
    </parameters>
    <end_of_request></end_of_request>
</req>
```

This XML telegram can be sent out with Ethernet KRL using:

```
DECL EKI_STATUS RET
RET = EKI_SetString("rc_stereomatching-parameters", "req/parameters/quality/@value",
↪"Low")
RET = EKI_Send("rc_stereomatching-parameters", "req")
```

In this case, only the applied value of `quality` is returned by the EKI Bridge:

```
<res>
  <parameters>
    <quality default="High" max="" min="" value="Low"/>
  </parameters>
  <return_code message="" value="0"/>
</res>
```

7.5.5 Example applications

More detailed robot application examples can be found at https://github.com/roboception/eki_examples.

7.5.6 Troubleshooting

SmartPad error message: Limit of element memory reached

This error may occur if the number of matches exceeds the memory limit.

- Increase BUFFERING and set BUFFSIZE in EKI config files. Adapt these settings to your particular KRC.
- Decrease the 'Maximum Matches' parameter in the detection module
- Even if the total memory limit (BUFFSIZE) of a message is not reached, the KRC might not be able to parse the number of child elements in the XML tree if the BUFFERING limit is too small. For example, if your application proposes 50 different grasps, the BUFFERING limit needs to be 50 too.

7.6 gRPC image stream interface

The gRPC image streaming interface can be used for getting camera images and synchronized sets of images (e.g. left camera image and corresponding disparity image).

gRPC is a remote procedure call system that also supports streaming. It uses Protocol Buffers (see https://developers.google.com/protocol-buffers/) as interface description language and data serialization. For a gRPC introduction and more details please see the official website (https://grpc.io/).

The advantages of the gRPC interface in comparison to GigE Vision are:

- It is simpler to use in own programs than GigE Vision.
- There is gRPC support for a lot of programming languages (see https://grpc.io/).
- The communication is based on TCP instead of UDP and therefore it also works over less stable networks, e.g. WLAN.

The disadvantages of the gRPC interface in comparison to GigE Vision are:

- It does not support changing parameters, but the *REST-API interface* (Section 7.2) can be used for changing parameters.
- It is not a standard vision interface like GigE Vision.

The *rc_reason_stack* provides synchronized image sets via gRPC server side streams on a separate port for each pipeline. The port is 50051 + pipeline number, so 50051 for pipeline 0, 50052 for pipeline 1, etc.

The communication is started by sending an `ImageSetRequest` message to the server. The message contains the information about requested images, i.e. left, right, disparity, confidence and disparity_error images can be enabled separately.

After getting the request, the server starts continuously sending `ImageSet` messages that contain all requested images with all parameters necessary for interpreting the images. The images that are contained in an `ImageSet` message are synchronized, i.e. they are all captured at the same time. The only exception to this rule is if the *out1_mode* (Section 6.4.4.1) is set to `AlternateExposureActive`. In this case, the camera and disparity images are taken 40 ms apart, so that the GPIO Out1 is LOW when the left and right images are taken, and HIGH for the disparity, confidence and error images. This mode is useful when a random dot projector is used, because the projector would be off for capturing the left and right image, and on for the disparity image, which results in undisturbed camera images and a much denser and more accurate disparity image.

Streaming of images is done until the client closes the connection.

7.6.1 gRPC service definition

```
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
```

(continues on next page)

(continued from previous page)

```
  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
  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
}
```

(continues on next page)

(continued from previous page)

```
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
}

service ImageInterface
{
  // A server-to-client streaming RPC.
  rpc StreamImageSets(ImageSetRequest) returns (stream ImageSet) {}
}
```

7.6.1.1 Image stream conversions

The disparity image contains 16 bit unsigned integer values. These values must be multiplied by the `scale` value given in the `DisparityImage` message to get the disparity values d in pixels. To compute the 3D object coordinates from the disparity values, the baseline and the focal length as well as the principal point are required. These parameters are transmitted as `baseline` = t in the `DisparityImage` message, and `focal_length` = f, `principal_point_u` = c_x and `principal_point_v` = c_y in the `ImageData` message. The focal length and principal point depend on the resolution of the camera image and need to be scaled to the resolution of the disparity image. Knowing these values, the pixel coordinates and the disparities can be transformed into 3D object coordinates in the camera coordinate frame using the equations described in *Computing depth images and point clouds* (Section 4.2.2).

Assuming that $d16_{ik}$ is the 16 bit disparity value at column i and row k of a disparity image, the float disparity in pixels d_{ik} is given by

$$d_{ik} = d16_{ik} \cdot \mathrm{scale}$$

The 3D reconstruction in meters can be written as:

$$\begin{aligned} P_x &= (i + 0.5 - c_x)\,\frac{t}{d_{ik}}, \\ P_y &= (k + 0.5 - c_y)\,\frac{t}{d_{ik}}, \\ P_z &= f\,\frac{t}{d_{ik}}. \end{aligned}$$

The confidence image contains 8 bit unsigned integer values. These values have to be divided by 255 to get the confidence as value between 0 an 1.

The error image contains 8 bit unsigned integer values. The error e_{ik} must be multiplied by the `scale` value given in the `DisparityImage` message to get the disparity-error values d_{eps} in pixels. According to the description in *Confidence and error images* (Section 4.2.3), the depth error z_{eps} in meters can be computed as

$$\begin{aligned} d_{ik} &= d16_{ik} \cdot \mathrm{scale}, \\ z_{eps} &= \frac{e_{ik} \cdot \mathrm{scale} \cdot f \cdot t}{(d_{ik})^2}. \end{aligned}$$

For more information about disparity, error, and confidence images, please refer to *Stereo matching module* (Section 6.2.2).

7.6.2 Example client

A simple example C++ client can be found at https://github.com/roboception/grpc_image_client_example.

8 Maintenance

8.1 Creating and restoring backups of settings

The *rc_reason_stack* offers the possibility to download the current settings as backup or for transferring them to a different *rc_visard* or *rc_cube*.

The current settings of the *rc_reason_stack* can be downloaded on the *Web GUI*'s (Section 7.1) *System* page in the *rc_reason_stack Settings* section. They can also be downloaded via the *rc_reason_stack*'s *REST-API interface* (Section 7.2) using the `GET /system/backup` request.

For downloading a backup, the user can choose which settings to include:

- `nodes`: the settings of all modules (parameters, preferred orientations and sorting strategies)
- `load_carriers`: the configured load carriers
- `regions_of_interest`: the configured 2D and 3D regions of interest
- `grippers`: the configured grippers (without the CAD elements)

The returned backup should be stored as a .json file.

The templates of the SilhouetteMatch and CADMatch modules are not included in the backup but can be downloaded manually using the REST-API or the Web GUI (see *Template API*, Section 6.3.6.14 and *Template API*, Section 6.3.7.13).

A backup can be restored to the *rc_reason_stack* on the *Web GUI*'s (Section 7.1) *System* page in the *rc_reason_stack Settings* section by uploading the backup .json file. In the *Web GUI* the settings included in the backup are shown and can be chosen for restore. The corresponding *REST-API interface* (Section 7.2) call is `POST /system/backup`.

Warning: When restoring load carriers, all existing load carriers on the *rc_reason_stack* will get lost and will be replaced by the content of the backup. The same applies to restoring grippers and regions of interest.

When restoring a backup, only the settings which are applicable to the *rc_reason_stack* are restored. Parameters for modules that do not exist on the device or do not have a valid license will be skipped. If a backup can only be restored partially, the user will be notified by warnings.

8.2 Updating the software license

Licenses can be purchased from Roboception for enabling additional features.

8.3 Downloading log files

During operation, the *rc_reason_stack* logs important information, warnings, and errors into files. If the *rc_reason_stack* exhibits unexpected or erroneous behavior, the log files can be used to trace its origin. Log messages can be viewed and filtered using the *Web GUI*'s (Section 7.1) *System* → *Logs* page. If contacting the support (*Contact*, Section 10), the log files are very useful for tracking possible problems. To download them as a .tar.gz file, click on *Download all logs* on the Web GUI's *System* → *Logs* page.

Aside from the Web GUI, the logs are also accessible via the *rc_reason_stack*'s *REST-API interface* (Section 7.2) using the `GET /logs` and `GET /logs/{log}` requests.

9 Troubleshooting

9.1 Camera-image issues

The camera image is too bright

- If the camera is in manual exposure mode, decrease the exposure time, or
- switch to auto-exposure mode.

The camera image is too dark

- If the camera is in manual exposure mode, increase the exposure time, or
- switch to auto-exposure mode.

The camera image is too noisy

Large gain factors cause high-amplitude image noise. To decrease the image noise,

- use an additional light source to increase the scene's light intensity, or
- choose a greater maximal auto-exposure time.

The camera image is out of focus

- Check whether the object is too close to the lens and increase the distance between the object and the lens if it is.
- Check whether the camera lenses are dirty and clean them if they are.
- If none of the above applies, a severe hardware problem might exist. Please *contact support* (Section 10).

The camera image is blurred

Fast motions in combination with long exposure times can cause blur. To reduce motion blur,

- decrease the motion speed of the camera,
- decrease the motion speed of objects in the field of view of the camera, or
- decrease the exposure time of the camera.

The camera image frame rate is too low

- Increase the image frame rate.
- The maximal frame rate of the cameras is 25 Hz.

9.2 Depth/Disparity, error, and confidence image issues

All these guidelines also apply to error and confidence images, because they correspond directly to the disparity image.

The disparity image is too sparse or empty

- Check whether the camera images are well exposed and sharp. Follow the instructions in *Camera-image issues* (Section 9.1) if applicable.
- Check whether the scene has enough texture and install an external pattern projector if required.
- Decrease the *Minimum Distance* (Section 6.2.2.1).
- Increase the *Maximum Distance* (Section 6.2.2.1).
- Check whether the object is too close to the cameras. Consider the different depth ranges of the camera variants.
- Decrease the *Minimum Confidence* (Section 6.2.2.1).
- Increase the *Maximum Depth Error* (Section 6.2.2.1).
- Choose a lesser *Disparity Image Quality* (Section 6.2.2.1). Lower resolution disparity images are generally less sparse.
- Check the cameras' calibration and recalibrate if required (see *Camera calibration*, Section 6.4.3).

The disparity images' frame rate is too low

- Check and increase the frame rate of the camera images. The frame rate of the disparity image cannot be greater than the frame rate of the camera images.
- Choose a lesser *Disparity Image Quality* (Section 6.2.2.1).
- Increase the *Minimum Distance* (Section 6.2.2.1) as much as possible for the application.

The disparity image does not show close objects

- Check whether the object is too close to the cameras. Consider the depth ranges of the camera variants.
- Decrease the *Minimum Distance* (Section 6.2.2.1).

The disparity image does not show distant objects

- Increase the *Maximum Distance* (Section 6.2.2.1).
- Increase the *Maximum Depth Error* (Section 6.2.2.1).
- Decrease the *Minimum Confidence* (Section 6.2.2.1).

The disparity image is too noisy

- Increase the *Segmentation value* (Section 6.2.2.1).
- Increase the *Fill-In value* (Section 6.2.2.1).

The disparity values or the resulting depth values are too inaccurate

- Decrease the distance between the camera and the scene. Depth-measurement error grows quadratically with the distance from the cameras.
- Check whether the scene contains repetitive patterns and remove them if it does. They could cause wrong disparity measurements.

The disparity image is too smooth

- Decrease the *Fill-In value* (Section 6.2.2.1).

The disparity image does not show small structures

- Decrease the *Segmentation value* (Section 6.2.2.1).
- Decrease the *Fill-In value* (Section 6.2.2.1).

10 Contact

10.1 Support

For support issues, please see http://www.roboception.com/support or contact support@roboception.de.

10.2 Downloads

Software SDKs, etc. can be downloaded from http://www.roboception.com/download.

10.3 Address

Roboception GmbH
Kaflerstrasse 2
81241 Munich
Germany

Web: http://www.roboception.com
Email: info@roboception.de
Phone: +49 89 889 50 79-0

11 Appendix

11.1 Pose formats

A pose consists of a translation and rotation. The translation defines the shift along the x, y and z axes. The rotation can be defined in many different ways. The *rc_reason_stack* uses quaternions to define rotations and translations are given in meters. This is called the XYZ+quaternion format. This chapter explains the conversion between different common conventions and the XYZ+quaternion format.

It is quite common to define rotations in 3D by three angles that define rotations around the three coordinate axes. Unfortunately, there are many different ways to do that. The most common conventions are Euler and Cardan angles (also called Tait-Bryan angles). In both conventions, the rotations can be applied to the previously rotated axis (intrinsic rotation) or to the axis of a fixed coordinate system (extrinsic rotation).

We use x, y and z to denote the three coordinate axes. x', y' and z' refer to the axes that have been rotated one time. Similarly, x'', y'' and z'' are the axes after two rotations.

In the (original) Euler angle convention, the first and the third axis are always the same. The rotation order z-x'-z'' means rotating around the z-axis, then around the already rotated x-axis and finally around the (two times) rotated z-axis. In the Cardan angle convention, three different rotation axes are used, e.g. z-y'-x''. Cardan angles are often also just called Euler angles.

For each intrinsic rotation order, there is an equivalent extrinsic rotation order, which is inverted, e.g. the intrinsic rotation order z-y'-x'' is equivalent to the extrinsic rotation order x-y-z.

Rotations around the x, y and z axes can be defined by quaternions as

$$r_x(\alpha) = \begin{pmatrix} \sin\frac{\alpha}{2} \\ 0 \\ 0 \\ \cos\frac{\alpha}{2} \end{pmatrix}, \qquad r_y(\beta) = \begin{pmatrix} 0 \\ \sin\frac{\beta}{2} \\ 0 \\ \cos\frac{\beta}{2} \end{pmatrix}, \qquad r_z(\gamma) = \begin{pmatrix} 0 \\ 0 \\ \sin\frac{\gamma}{2} \\ \cos\frac{\gamma}{2} \end{pmatrix},$$

or by rotation matrices as

$$r_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix},$$
$$r_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix},$$
$$r_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The extrinsic rotation order x-y-z can be computed by multiplying the individual rotations in inverse order, i.e. $r_z(\gamma)r_y(\beta)r_x(\alpha)$.

Based on these definitions, the following sections explain the conversion between common conventions and the XYZ+quaternion format.

Note: Please be aware of units for positions and orientations. *rc_reason_stack* devices always specify positions in meters, while most robot manufacturers use millimeters or inches. Angles are typically specified in degrees, but may sometimes also be given in radians.

11.1.1 Rotation matrix and translation vector

A pose can also be defined by a rotation matrix R and a translation vector T.

$$R = \begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix}, \qquad T = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

The pose transformation can be applied to a point P by

$$P' = RP + T.$$

11.1.1.1 Conversion from rotation matrix to quaternion

The conversion from a rotation matrix (with $det(R) = 1$) to a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ can be done as follows.

$$\begin{aligned} x &= \text{sign}(r_{21} - r_{12})\frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} - r_{11} - r_{22})} \\ y &= \text{sign}(r_{02} - r_{20})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} + r_{11} - r_{22})} \\ z &= \text{sign}(r_{10} - r_{01})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} - r_{11} + r_{22})} \\ w &= \frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} + r_{11} + r_{22})} \end{aligned}$$

The sign operator returns -1 if the argument is negative. Otherwise, 1 is returned. It is used to recover the sign for the square root. The max function ensures that the argument of the square root function is not negative, which can happen in practice due to round-off errors.

11.1.1.2 Conversion from quaternion to rotation matrix

The conversion from a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ with $||q|| = 1$ to a rotation matrix can be done as follows.

$$R = 2\begin{pmatrix} \frac{1}{2} - y^2 - z^2 & xy - zw & xz + yw \\ xy + zw & \frac{1}{2} - x^2 - z^2 & yz - xw \\ xz - yw & yz + xw & \frac{1}{2} - x^2 - y^2 \end{pmatrix}$$

11.1.2 ABB pose format

ABB robots use a position X, Y, Z and a quaternion $Q1$, $Q2$, $Q3$, $Q4$ for describing a pose, similar to *rc_reason_stack* devices. However, the position must be given in millimeters and the quaternion order is as follows:

$q = \begin{pmatrix} x & y & z & w \end{pmatrix} = \begin{pmatrix} Q2 & Q3 & Q4 & Q1 \end{pmatrix}$.

11.1.3 FANUC XYZ-WPR format

The pose format that is used by FANUC robots consists of a position XYZ in millimeters and an orientation WPR that is given by three angles in degrees, with W rotating around x-axis, P rotating around

y-axis and R rotating around z-axis. The rotation order is x-y-z and computed by $r_z(R)r_y(P)r_x(W)$.

11.1.3.1 Conversion from FANUC-WPR to quaternion

The conversion from the WPR angles in degrees to a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ can be done by first converting all angles to radians

$$
\begin{aligned}
W_r &= W \frac{\pi}{180}, \\
P_r &= P \frac{\pi}{180}, \\
R_r &= R \frac{\pi}{180},
\end{aligned}
$$

and then calculating the quaternion with

$$
\begin{aligned}
x &= \cos(R_r/2)\cos(P_r/2)\sin(W_r/2) - \sin(R_r/2)\sin(P_r/2)\cos(W_r/2), \\
y &= \cos(R_r/2)\sin(P_r/2)\cos(W_r/2) + \sin(R_r/2)\cos(P_r/2)\sin(W_r/2), \\
z &= \sin(R_r/2)\cos(P_r/2)\cos(W_r/2) - \cos(R_r/2)\sin(P_r/2)\sin(W_r/2), \\
w &= \cos(R_r/2)\cos(P_r/2)\cos(W_r/2) + \sin(R_r/2)\sin(P_r/2)\sin(W_r/2).
\end{aligned}
$$

11.1.3.2 Conversion from quaternion to FANUC-WPR

The conversion from a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ with $||q|| = 1$ to the WPR angles in degrees can be done as follows.

$$
\begin{aligned}
R &= \mathrm{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi} \\
P &= \mathrm{asin}(2(wy - zx))\frac{180}{\pi} \\
W &= \mathrm{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi}
\end{aligned}
$$

11.1.4 Franka Emika Pose Format

Franka Emika robots use a transformation matrix T to define a pose. A transformation matrix combines a rotation matrix R and a translation vector $t = \begin{pmatrix} x & y & z \end{pmatrix}^T$.

$$
T = \begin{pmatrix}
r_{00} & r_{01} & r_{02} & x \\
r_{10} & r_{11} & r_{12} & y \\
r_{20} & r_{21} & r_{22} & z \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

The pose given by Franka Emika's "Measure Pose" App consists of a translation x, y, z in millimeters and a rotation x, y, z in degrees. The rotation convention is z-y'-x'' (i.e. x-y-z) and is computed by $r_z(z)r_y(y)r_x(x)$.

11.1.4.1 Conversion from transformation matrix to quaternion

The conversion from a rotation matrix (with $det(R) = 1$) to a quaternion $q = \begin{pmatrix} q_x & q_y & q_z & q_w \end{pmatrix}$ can be done as follows:

$$
\begin{aligned}
q_x &= \text{sign}(r_{21} - r_{12})\frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} - r_{11} - r_{22})} \\
q_y &= \text{sign}(r_{02} - r_{20})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} + r_{11} - r_{22})} \\
q_z &= \text{sign}(r_{10} - r_{01})\frac{1}{2}\sqrt{\text{max}(0, 1 - r_{00} - r_{11} + r_{22})} \\
q_w &= \frac{1}{2}\sqrt{\text{max}(0, 1 + r_{00} + r_{11} + r_{22})}
\end{aligned}
$$

The sign operator returns -1 if the argument is negative. Otherwise, 1 is returned. It is used to recover the sign for the square root. The max function ensures that the argument of the square root function is not negative, which can happen in practice due to round-off errors.

11.1.4.2 Conversion from Rotation-XYZ to quaternion

The conversion from the x, y, z angles in degrees to a quaternion $q = \begin{pmatrix} q_x & q_y & q_z & q_w \end{pmatrix}$ can be done by first converting all angles to radians

$$
\begin{aligned}
X_r &= x\frac{\pi}{180}, \\
Y_r &= y\frac{\pi}{180}, \\
Z_r &= z\frac{\pi}{180},
\end{aligned}
$$

and then calculating the quaternion with

$$
\begin{aligned}
q_x &= \cos{(Z_r/2)}\cos{(Y_r/2)}\sin{(X_r/2)} - \sin{(Z_r/2)}\sin{(Y_r/2)}\cos{(X_r/2)}, \\
q_y &= \cos{(Z_r/2)}\sin{(Y_r/2)}\cos{(X_r/2)} + \sin{(Z_r/2)}\cos{(Y_r/2)}\sin{(X_r/2)}, \\
q_z &= \sin{(Z_r/2)}\cos{(Y_r/2)}\cos{(X_r/2)} - \cos{(Z_r/2)}\sin{(Y_r/2)}\sin{(X_r/2)}, \\
q_w &= \cos{(Z_r/2)}\cos{(Y_r/2)}\cos{(X_r/2)} + \sin{(Z_r/2)}\sin{(Y_r/2)}\sin{(X_r/2)}.
\end{aligned}
$$

11.1.4.3 Conversion from quaternion and translation to transformation

The conversion from a quaternion $q = \begin{pmatrix} q_x & q_y & q_z & q_w \end{pmatrix}$ and a translation vector $t = \begin{pmatrix} x & y & z \end{pmatrix}^T$ to a transformation matrix T can be done as follows:

$$
T = \begin{pmatrix}
1 - 2s(q_y^2 + q_z^2) & 2s(q_x q_y - q_z q_w) & 2s(q_x q_z + q_y q_w) & x \\
2s(q_x q_y + q_z q_w) & 1 - 2s(q_x^2 + q_z^2) & 2s(q_y q_z - q_x q_w) & y \\
2s(q_x q_z - q_y q_w) & 2s(q_y q_z + q_x q_w) & 1 - 2s(q_x^2 + q_y^2) & z \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

where $s = ||q||^{-2} = \frac{1}{q_x^2 + q_y^2 + q_z^2 + q_w^2}$ and $s = 1$ if q is a unit quaternion.

11.1.4.4 Conversion from quaternion to Rotation-XYZ

The conversion from a quaternion $q = \begin{pmatrix} q_x & q_y & q_z & q_w \end{pmatrix}$ with $||q|| = 1$ to the x, y, z angles in degrees can be done as follows.

$$
\begin{aligned}
x &= \text{atan}_2(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2))\frac{180}{\pi} \\
y &= \text{asin}(2(q_w q_y - q_z q_x))\frac{180}{\pi} \\
z &= \text{atan}_2(2(q_w q_x + q_y q_z), 1 - 2(q_x^2 + q_y^2))\frac{180}{\pi}
\end{aligned}
$$

11.1.4.5 Pose representation in RaceCom messages and state machines

In RaceCom messages and in state machines a pose is usually defined as one-dimensional array of 16 float values, representing the transformation matrix in column-major order. The indices of the matrix entries below correspond to the array indices

$$
T = \begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix}
$$

11.1.5 Fruitcore HORST pose format

Fruitcore HORST robots use a position in meters and a quaternion with $q_0 = w$, $q_1 = x$, $q_2 = y$ and $q_3 = z$ for describing a pose, like *rc_reason_stack* devices. There is no conversion needed.

11.1.6 Kawasaki XYZ-OAT format

The pose format that is used by Kawasaki robots consists of a position XYZ in millimeters and an orientation OAT that is given by three angles in degrees, with O rotating around z axis, A rotating around the rotated y axis and T rotating around the rotated z axis. The rotation convention is z-y'-z'' (i.e. z-y-z) and computed by $r_z(O)r_y(A)r_z(T)$.

11.1.6.1 Conversion from Kawasaki-OAT to quaternion

The conversion from the OAT angles in degrees to a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ can be done by first converting all angles to radians

$$
\begin{aligned}
O_r &= O\frac{\pi}{180}, \\
A_r &= A\frac{\pi}{180}, \\
T_r &= T\frac{\pi}{180},
\end{aligned}
$$

and then calculating the quaternion with

$$
\begin{aligned}
x &= \cos(O_r/2)\sin(A_r/2)\sin(T_r/2) - \sin(O_r/2)\sin(A_r/2)\cos(T_r/2), \\
y &= \cos(O_r/2)\sin(A_r/2)\cos(T_r/2) + \sin(O_r/2)\sin(A_r/2)\sin(T_r/2), \\
z &= \sin(O_r/2)\cos(A_r/2)\cos(T_r/2) + \cos(O_r/2)\cos(A_r/2)\sin(T_r/2), \\
w &= \cos(O_r/2)\cos(A_r/2)\cos(T_r/2) - \sin(O_r/2)\cos(A_r/2)\sin(T_r/2).
\end{aligned}
$$

11.1.6.2 Conversion from quaternion to Kawasaki-OAT

The conversion from a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ with $||q|| = 1$ to the OAT angles in degrees can be done as follows.

If $x = 0$ **and** $y = 0$ the conversion is

$$\begin{aligned} O &= \text{atan}_2(2(z-w), 2(z+w))\frac{180}{\pi} \\ A &= \text{acos}(w^2+z^2)\frac{180}{\pi} \\ T &= \text{atan}_2(2(z+w), 2(w-z))\frac{180}{\pi} \end{aligned}$$

If $z = 0$ **and** $w = 0$ the conversion is

$$\begin{aligned} O &= \text{atan}_2(2(y-x), 2(x+y))\frac{180}{\pi} \\ A &= \text{acos}(-1.0)\frac{180}{\pi} \\ T &= \text{atan}_2(2(y+x), 2(y-x))\frac{180}{\pi} \end{aligned}$$

In all other cases the conversion is

$$\begin{aligned} O &= \text{atan}_2(2(yz-wx), 2(xz+wy))\frac{180}{\pi} \\ A &= \text{acos}(w^2-x^2-y^2+z^2)\frac{180}{\pi} \\ T &= \text{atan}_2(2(yz+wx), 2(wy-xz))\frac{180}{\pi} \end{aligned}$$

11.1.7 KUKA XYZ-ABC format

KUKA robots use the so called XYZ-ABC format. XYZ is the position in millimeters. ABC are angles in degrees, with A rotating around z axis, B rotating around y axis and C rotating around x axis. The rotation convention is z-y'-x'' (i.e. x-y-z) and computed by $r_z(A)r_y(B)r_x(C)$.

11.1.7.1 Conversion from KUKA-ABC to quaternion

The conversion from the ABC angles in degrees to a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ can be done by first converting all angles to radians

$$\begin{aligned} A_r &= A\frac{\pi}{180}, \\ B_r &= B\frac{\pi}{180}, \\ C_r &= C\frac{\pi}{180}, \end{aligned}$$

and then calculating the quaternion with

$$\begin{aligned} x &= \cos(A_r/2)\cos(B_r/2)\sin(C_r/2) - \sin(A_r/2)\sin(B_r/2)\cos(C_r/2), \\ y &= \cos(A_r/2)\sin(B_r/2)\cos(C_r/2) + \sin(A_r/2)\cos(B_r/2)\sin(C_r/2), \\ z &= \sin(A_r/2)\cos(B_r/2)\cos(C_r/2) - \cos(A_r/2)\sin(B_r/2)\sin(C_r/2), \\ w &= \cos(A_r/2)\cos(B_r/2)\cos(C_r/2) + \sin(A_r/2)\sin(B_r/2)\sin(C_r/2). \end{aligned}$$

11.1.7.2 Conversion from quaternion to KUKA-ABC

The conversion from a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ with $||q|| = 1$ to the ABC angles in degrees can be done as follows.

$$\begin{aligned}
A &= \text{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi} \\
B &= \text{asin}(2(wy - zx))\frac{180}{\pi} \\
C &= \text{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi}
\end{aligned}$$

11.1.8 Mitsubishi XYZ-ABC format

The pose format that is used by Mitsubishi robots is the same as that for KUKA robots (see *KUKA XYZ-ABC format*, Section 11.1.7), except that A is a rotation around x axis and C is a rotation around z axis. Thus, the rotation is computed by $r_z(C)r_y(B)r_x(A)$.

11.1.8.1 Conversion from Mitsubishi-ABC to quaternion

The conversion from the ABC angles in degrees to a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ can be done by first converting all angles to radians

$$\begin{aligned}
A_r &= A\frac{\pi}{180}, \\
B_r &= B\frac{\pi}{180}, \\
C_r &= C\frac{\pi}{180},
\end{aligned}$$

and then calculating the quaternion with

$$\begin{aligned}
x &= \cos{(C_r/2)}\cos{(B_r/2)}\sin{(A_r/2)} - \sin{(C_r/2)}\sin{(B_r/2)}\cos{(A_r/2)}, \\
y &= \cos{(C_r/2)}\sin{(B_r/2)}\cos{(A_r/2)} + \sin{(C_r/2)}\cos{(B_r/2)}\sin{(A_r/2)}, \\
z &= \sin{(C_r/2)}\cos{(B_r/2)}\cos{(A_r/2)} - \cos{(C_r/2)}\sin{(B_r/2)}\sin{(A_r/2)}, \\
w &= \cos{(C_r/2)}\cos{(B_r/2)}\cos{(A_r/2)} + \sin{(C_r/2)}\sin{(B_r/2)}\sin{(A_r/2)}.
\end{aligned}$$

11.1.8.2 Conversion from quaternion to Mitsubishi-ABC

The conversion from a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ with $||q|| = 1$ to the ABC angles in degrees can be done as follows.

$$\begin{aligned}
A &= \text{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi} \\
B &= \text{asin}(2(wy - zx))\frac{180}{\pi} \\
C &= \text{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi}
\end{aligned}$$

11.1.9 Universal Robots pose format

The pose format that is used by Universal Robots consists of a position XYZ in millimeters and an orientation in angle-axis format $V = \begin{pmatrix} RX & RY & RZ \end{pmatrix}^T$. The rotation angle θ in radians is the length of the rotation axis U.

$$V = \begin{pmatrix} RX \\ RY \\ RZ \end{pmatrix} = \begin{pmatrix} \theta u_x \\ \theta u_y \\ \theta u_z \end{pmatrix}$$

V is called a rotation vector.

11.1.9.1 Conversion from angle-axis format to quaternion

The conversion from a rotation vector V to a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ can be done as follows.

We first recover the angle θ in radians from the rotation vector V by

$$\theta = \sqrt{RX^2 + RY^2 + RZ^2}.$$

If $\theta = 0$, then the quaternion is $q = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$, otherwise it is

$$\begin{aligned} x &= RX \frac{\sin(\theta/2)}{\theta}, \\ y &= RY \frac{\sin(\theta/2)}{\theta}, \\ z &= RZ \frac{\sin(\theta/2)}{\theta}, \\ w &= \cos(\theta/2). \end{aligned}$$

11.1.9.2 Conversion from quaternion to angle-axis format

The conversion from a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ with $||q|| = 1$ to a rotation vector in angle-axis form can be done as follows.

We first recover the angle θ in radians from the quaternion by

$$\theta = 2 \cdot \text{acos}(w).$$

If $\theta = 0$, then the rotation vector is $V = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$, otherwise it is

$$\begin{aligned} RX &= \theta \frac{x}{\sqrt{1 - w^2}}, \\ RY &= \theta \frac{y}{\sqrt{1 - w^2}}, \\ RZ &= \theta \frac{z}{\sqrt{1 - w^2}}. \end{aligned}$$

11.1.10 Yaskawa Pose Format

The pose format that is used by Yaskawa robots consists of a position XYZ in millimeters and an orientation that is given by three angles in degrees, with Rx rotating around x-axis, Ry rotating around y-axis and Rz rotating around z-axis. The rotation order is x-y-z and computed by $r_z(Rz)r_y(Ry)r_x(Rx)$.

11.1.10.1 Conversion from Yaskawa Rx, Ry, Rz to quaternion

The conversion from the Rx, Ry, Rz angles in degrees to a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ can be done by first converting all angles to radians

$$X_r = Rx\frac{\pi}{180},$$
$$Y_r = Ry\frac{\pi}{180},$$
$$Z_r = Rz\frac{\pi}{180},$$

and then calculating the quaternion with

$$x = \cos(Z_r/2)\cos(Y_r/2)\sin(X_r/2) - \sin(Z_r/2)\sin(Y_r/2)\cos(X_r/2),$$
$$y = \cos(Z_r/2)\sin(Y_r/2)\cos(X_r/2) + \sin(Z_r/2)\cos(Y_r/2)\sin(X_r/2),$$
$$z = \sin(Z_r/2)\cos(Y_r/2)\cos(X_r/2) - \cos(Z_r/2)\sin(Y_r/2)\sin(X_r/2),$$
$$w = \cos(Z_r/2)\cos(Y_r/2)\cos(X_r/2) + \sin(Z_r/2)\sin(Y_r/2)\sin(X_r/2).$$

11.1.10.2 Conversion from quaternion to Yaskawa Rx, Ry, Rz

The conversion from a quaternion $q = \begin{pmatrix} x & y & z & w \end{pmatrix}$ with $||q|| = 1$ to the Rx, Ry, Rz angles in degrees can be done as follows.

$$Rx = \mathrm{atan}_2(2(wx + yz), 1 - 2(x^2 + y^2))\frac{180}{\pi}$$
$$Ry = \mathrm{asin}(2(wy - zx))\frac{180}{\pi}$$
$$Rz = \mathrm{atan}_2(2(wz + xy), 1 - 2(y^2 + z^2))\frac{180}{\pi}$$

HTTP Routing Table

/cad

GET /cad/gripper_elements, 285
GET /cad/gripper_elements/{id}, 286
PUT /cad/gripper_elements/{id}, 286
DELETE /cad/gripper_elements/{id}, 287

/generic_robot_interface

GET /generic_robot_interface/hec_configs, 340
GET /generic_robot_interface/hec_configs/{pipeline}, 340
GET /generic_robot_interface/jobs, 342
GET /generic_robot_interface/jobs/{job_id}, 343
PUT /generic_robot_interface/hec_configs/{pipeline}, 341
PUT /generic_robot_interface/jobs/{job_id}, 343
DELETE /generic_robot_interface/hec_configs/{pipeline}, 341
DELETE /generic_robot_interface/jobs/{job_id}, 344

/logs

GET /logs, 311
GET /logs/{log}, 311

/nodes

GET /nodes, 295
GET /nodes/{node}, 296
GET /nodes/{node}/services, 296
GET /nodes/{node}/services/{service}, 297
GET /nodes/{node}/status, 298
PUT /nodes/{node}/services/{service}, 297

/pipelines

GET /pipelines, 308
GET /pipelines/{pipeline}, 308
GET /pipelines/{pipeline}/nodes, 299
GET /pipelines/{pipeline}/nodes/{node}, 300
GET /pipelines/{pipeline}/nodes/{node}/parameters, 301
GET /pipelines/{pipeline}/nodes/{node}/parameters/{param}, 303
GET /pipelines/{pipeline}/nodes/{node}/services, 305
GET /pipelines/{pipeline}/nodes/{node}/services/{service}, 306
GET /pipelines/{pipeline}/nodes/{node}/status, 307
PUT /pipelines/{pipeline}/nodes/{node}/parameters, 302
PUT /pipelines/{pipeline}/nodes/{node}/parameters/{param}, 304
PUT /pipelines/{pipeline}/nodes/{node}/services/{service}, 306

/presets

GET /presets/rc_zivid/2d_presets, 49
GET /presets/rc_zivid/2d_presets/{id}, 49
GET /presets/rc_zivid/3d_presets, 63
GET /presets/rc_zivid/3d_presets/{id}, 64
PUT /presets/rc_zivid/2d_presets/{id}, 50
PUT /presets/rc_zivid/3d_presets/{id}, 64
DELETE /presets/rc_zivid/2d_presets/{id}, 50
DELETE /presets/rc_zivid/3d_presets/{id}, 64

/system

GET /system, 312
GET /system/backup, 313
GET /system/ca_certificates, 314
GET /system/ca_certificates/{id}, 315
GET /system/disk_info, 316
GET /system/license, 316
GET /system/max_power_test, 317
GET /system/pipelines, 309
GET /system/pipelines/config/{pipeline}, 309
GET /system/ui_lock, 317
POST /system/backup, 314
POST /system/license, 316
POST /system/max_power_test, 317
POST /system/ui_lock, 318
PUT /system/ca_certificates/{id}, 315
PUT /system/pipelines/config/{pipeline}, 309
DELETE /system/ca_certificates/{id}, 315
DELETE /system/pipelines/config/{pipeline}, 310
DELETE /system/ui_lock, 318

/templates

GET /templates/rc_boxpick, 145
GET /templates/rc_boxpick/{id}, 145
GET /templates/rc_cadmatch, 216
GET /templates/rc_cadmatch/{id}, 217
GET /templates/rc_silhouettematch, 181
GET /templates/rc_silhouettematch/{id}, 182

PUT /templates/rc_boxpick/{id}, 146
PUT /templates/rc_cadmatch/{id}, 217
PUT /templates/rc_silhouettematch/{id}, 182
DELETE /templates/rc_boxpick/{id}, 146
DELETE /templates/rc_cadmatch/{id}, 218
DELETE /templates/rc_silhouettematch/{id}, 183

Index

Symbols

3D coordinates, 16
    disparity image, 16
3D modeling, 16
3D object detection, 184

A

acquisition mode
    disparity image, 53
    stereo ace camera, 33
AdaptiveOut1
    auto exposure mode, 27
AprilTag, **88**
    pose estimation, 91
    re-identification, 92
    return codes, 99
    services, 94
auto
    exposure, 27
auto exposure, 26, 27, 42, 43
auto exposure mode, 27
    AdaptiveOut1, 27
    Normal, 27
    Out1High, 27
    stereo ace camera, 35

B

backup
    settings, 357
base-plane
    SilhouetteMatch, 148
base-plane calibration
    SilhouetteMatch, 148
baseline, 23
bin picking, 99, 119, 184
blue ratio
    stereo ace camera, 38
BoxPick, **119**
    filling level, 74
    grasp, 121
    grasp sorting, 121
    item models, 119
    load carrier, 73, 262
    parameters, 124
    preferred orientation, 122
    RECTANGLE, 120
    region of interest, 270
    return codes, 144
    services, 131
    status, 130
    template api, 144
    template deletion, 144
    template download, 144
    template upload, 144
    texture, 120
    TEXTURED_BOX, 120
    views, 120
brightness
    stereo ace camera, 36, 38

C

CADMatch, **184**
    collision check, 189
    filling level, 74
    grasp points, 184
    load carrier, 73, 262
    object detection, 186
    object template, 184–186
    parameters, 189
    pose priors, 185
    preferred orientation, 185
    region of interest, 270
    return codes, 215
    services, 194
    sorting, 186
    status, 194
    template api, 216
    template deletion, 216
    template download, 216
    template upload, 216
calibration
    camera, 250
    hand-eye calibration, 223
    rectification, 23
calibration grid, 250
camera
    calibration, 250
    frame rate, 25, 47
    gamma, 26, 42
    parameters, 23
    Web GUI, 23
camera calibration
    monocalibration, 256
    parameters, 257
    services, 257
    stereo calibration, 254

camera connection
 installation, 13
camera model, 23
Camera pipelines, **17**, **22**
collision check, 241, 277
CollisionCheck, **241**
 return codes, 249
compartment
 load carrier, 265
confidence, **16**
 minimum, 57
contrast
 Stereo ace camera, 34
contrast mode
 linear, 34
 scurve, 34
 Stereo ace camera, 34
conversions
 gRPC image stream, 355

D

data model
 REST-API, 319
data-type
 REST-API, 319
definition
 load carrier, 263
depth acquisition mode
 zivid, 61
depth error
 maximum, 57
depth image, 15, **16**, 16, 51, 60, 65
 Web GUI, 51
depth measurement, 68
detection
 load carrier, 73
 tag, 87
dimensions
 load carrier, 263
disparity, 14, 15, 23, 51
disparity error, 16
disparity image, 14, 15, 51
 3D coordinates, 16
 acquisition mode, 53
 double_shot, 55
 exposure adaptation timeout, 54
 parameters, 51
 quality, 54
 smooth, 56, 66
 static_scene, 55
 Web GUI, 51
double_shot
 disparity image, 55
download
 images, 23
 log files, 358
 point cloud, 51
 settings, 357

E

eki, 345
error, **16**
 hand-eye calibration, 229
exposure
 auto, 26, 27, 42
 HDR, 26, 42
 manual, 26, 42
 stereo ace camera, 35
exposure adaptation timeout
 disparity image, 54
exposure region, 28, 43
 stereo ace camera, 37
exposure time, 28, 43
 maximum, 27, 43
 stereo ace camera, 37
external reference frame
 hand-eye calibration, 219

F

fill-in, 56, 66
filling level
 BoxPick, 74
 ItemPick, 74
 LoadCarrier, 74
 SilhouetteMatch, 74
focal length, 23
fps, *see* frame rate, *see* frame rate
 stereo ace camera, 34
frame rate
 camera, 25, 47
 stereo ace camera, 34

G

gain
 stereo ace camera, 37
gain factor, 27, 29, 43, 44
gamma
 camera, 26, 42
 stereo ace camera, 34
Generic Robot Interface, 329
grasp computation, 99, 119, 184
green ratio
 stereo ace camera, 38
GRI, 329
gripper CAD element api, 285
gripper CAD element deletion, 285
gripper CAD element download, 285
gripper CAD element upload, 285
GripperDB, **277**
 return codes, 285
gRPC, 352
gRPC image stream
 conversions, 355

H

hand-eye calibration
 calibration, 223

error, 229
external reference frame, 219
mounting, 219
parameters, 229
robot frame, 219
slot, 226

I

image
timestamp, 51
image acquisition mode
zivid, 47
image noise, 27, 43
images
download, 23
inner volume
load carrier, 263
installation, 8
camera connection, 13
ItemPick, **99**
filling level, 74
grasp, 100
grasp sorting, 100
load carrier, 73, 262
preferred orientation, 102
region of interest, 270
return codes, 118
services, 108
status, 108
ItemPickAI, **99**
grasp, 100
grasp sorting, 100
parameters, 104
preferred orientation, 102
return codes, 118
services, 108
status, 108

L

light source preset
stereo ace camera, 38
linear
contrast mode, 34
load carrier
BoxPick, 73, 262
compartment, 265
definition, 263
detection, 73
dimensions, 263
inner volume, 263
ItemPick, 73, 262
orientation prior, 263
pose, 263
rim, 263
SilhouetteMatch, 73, 262
load carrier detection, 73
load carrier model, 262
LoadCarrier, **73**
filling level, 74
parameters, 76
return codes, 86
services, 78
LoadCarrierDB, **262**
return codes, 270
services, 267
log files
download, 358
logs
REST-API, 311

M

manual exposure, 26, 28, 42, 43
max exposure
stereo ace camera, 36
maximum
depth error, 57
exposure time, 27, 43
maximum depth error, 57
maximum distance, 56, 66
Measure, **68**
parameters, 70
return codes, 73
services, 70
minimum
confidence, 57
minimum confidence, 57
minimum distance, 56, 65
monocalibration
camera calibration, 256
motion blur, 27, 43
mounting
hand-eye calibration, 219

N

node
REST-API, 293, 308
Normal
auto exposure mode, 27

O

object detection, 147, 184
OPC UA, 344
orbbec, 65
parameters, 65
orientation prior
load carrier, 263
Out1High
auto exposure mode, 27

P

parameter
REST-API, 294
parameters
camera, 23
camera calibration, 257
disparity image, 51

hand-eye calibration, 229
orbbec, 65
services, 30
stereo ace, 31
zivid, 60
point cloud, 16
download, 51
pose
load carrier, 263
pose estimation
AprilTag, 91
QR code, 91
preset name
zivid, 47, 61

Q

QR Code
return codes, 99
QR code, **88**
pose estimation, 91
re-identification, 92
services, 94
quality
disparity image, 54

R

re-identification
AprilTag, 92
QR code, 92
rectification, 23
red ratio
stereo ace camera, 38
REST-API, 291
data model, 319
data-type, 319
entry point, 292
logs, 311
node, 293, 308
parameter, 294
services, 294
status value, 294
system, 311
UserSpace, 311
version, 292
restore
settings, 357
return codes
AprilTag, 99
BoxPick, 144
CADMatch, 215
CollisionCheck, 249
GripperDB, 285
ItemPick, 118
ItemPickAI, 118
LoadCarrier, 86
LoadCarrierDB, 270
Measure, 73
QR Code, 99
RoiDB, 277
SilhouetteMatch, 180
rim
load carrier, 263
robot frame
hand-eye calibration, 219
ROI, 270
RoiDB, **270**
return codes, 277
services, 272

S

saturation
stereo ace camera, 39
scurve
contrast mode, 34
segmentation, 57, 67
Semi-Global Matching, *see* SGM
services
AprilTag, 94
camera calibration, 257
parameters, 30
QR code, 94
REST-API, 294
stereo ace camera, 40
tag detection, 94
settings
backup, 357
download, 357
restore, 357
upload, 357
SGM, 14, **15**
silhouette, 147
SilhouetteMatch, **147**
base-plane, 148
base-plane calibration, 148
collision check, 155
detection of objects, 152
filling level, 74
grasp points, 150
load carrier, 73, 262
object template, 150
parameters, 156
preferred orientation, 151
region of interest, 150, 270
return codes, 180
services, 162
sorting, 152
status, 161
template api, 181
template deletion, 181
template download, 181
template upload, 181
slot
hand-eye calibration, 226
smooth
disparity image, 56, 66
static_scene

disparity image, 55
status value
REST-API, 294
status values
stereo ace camera, 39
stereo ace
parameters, 31
Stereo ace camera
contrast, 34
contrast mode, 34
stereo ace camera
acquisition mode, 33
auto exposure mode, 35
blue ratio, 38
brightness, 36, 38
exposure, 35
exposure region, 37
exposure time, 37
fps, 34
frame rate, 34
gain, 37
gamma, 34
green ratio, 38
light source preset, 38
max exposure, 36
red ratio, 38
saturation, 39
services, 40
status values, 39
trigger activation, 33
white balance, 38
stereo calibration
camera calibration, 254
stereo camera, 23
stereo matching, 14
Swagger UI, 326
system
REST-API, 311

T

tag detection, **87**
families, 88
pose estimation, 91
re-identification, 92
services, 94
texture, 15
timestamp
image, 51
trigger activation
stereo ace camera, 33

U

upload
settings, 357
UserSpace
REST-API, 311

V

version
REST-API, 292

W

Web GUI, **288**
backup, 357
camera, 23
depth image, 51
disparity image, 51
logs, 358
white balance, 29, 44
stereo ace camera, 38

Z

zivid, 60
depth acquisition mode, 61
image acquisition mode, 47
parameters, 60
preset name, 47, 61

roboception

rc_reason_stack 3D Vision Software Platform

INSTALLATION AND OPERATING MANUAL

Roboception GmbH

Kaflerstrasse 2
81241 Munich   info@roboception.de
Germany   www.roboception.com

Tutorials:	https://tutorials.roboception.com
GitHub:	https://github.com/roboception
Documentation:	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

For customer support, contact

+49 89 889 50 790
(09:00-17:00 CET)   support@roboception.de