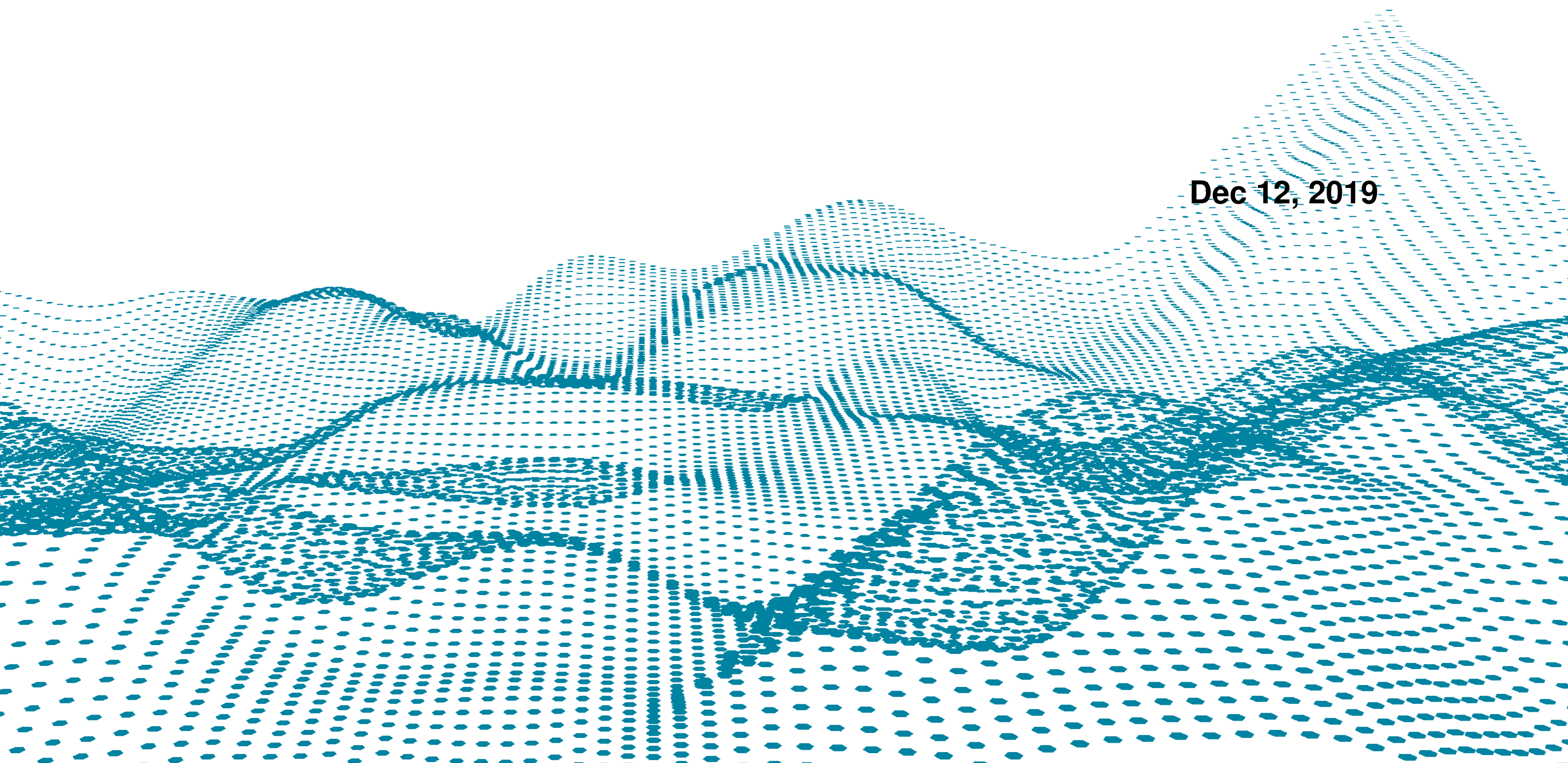
rc_visard

Documentation Revision 1.8.1-2-g3c019b5

Roboception GmbH

Dec 12, 2019



Contents

1 Introduction

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 1.8.1-2-g3c019b5 Dec 12, 2019

Applicable to *rc_visard* firmware 1.8.x

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 3D sensor *rc_visard* provides real-time camera images and disparity images, which are also used to compute depth images and 3D point clouds. Additionally, it provides confidence and error images as quality measures for each image acquisition. The sensor provides self-localization based on image and inertial data. A mobile navigation solution can be established with the optional on-board SLAM module. The *rc_visard* is an IP54-protected sensor that offers an intuitive web and a standardized GenICam interface, making it compatible with all major image processing libraries. The *rc_visard* is offered with two different stereo baselines: The *rc_visard* 65 is optimally suited for mounting on robotic manipulators, whereas the *rc_visard* 160 can be employed as a navigation or as externally-fixed sensor. The *rc_visard*'s intuitive calibration, configuration, and use enable 3D vision for everyone.

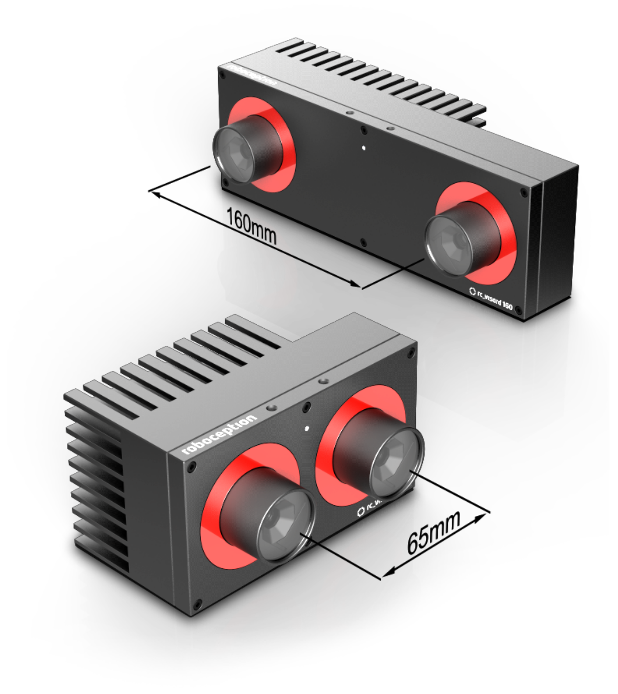


Fig. 1.1.1: *rc_visard* 65 and *rc_visard* 160

The terms "sensor," "*rc_visard* 65," and "*rc_visard* 160" used throughout the manual all refer to the Roboception *rc_visard* family of self-registering cameras. Installation and control for all sensors are exactly the same, and all use the same mounting base.

Note: Unless specified, the information provided in this manual applies to both the *rc_visard* 65 and *rc_visard* 160 versions of the Roboception sensor.

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.

1.2 Warranty

Any changes or modifications not expressly approved by Roboception could void the user's warranty and guarantee rights.

Warning: The *rc_visard* sensor utilizes complex hardware and software technology that may not always function as intended. The purchaser must design its application to ensure that any failure or the *rc_visard* sensor does not cause personal injury, property damage, or other losses.

Warning: Do not attempt to take apart, open, service, or modify the *rc_visard*. Doing so could present the risk of electric shock or other hazard. Any evidence of any attempt to open and/or modify the device, including any peeling, puncturing, or removal of any of the labels, will void the Limited Warranty.

Warning: CAUTION: to comply with the European CE requirement, all cables used to connect this device must be shielded and grounded. Operation with incorrect cables may result in interference with other devices or undesired effects of the product.

Note: This product may not be treated as household waste. By ensuring this product is disposed of correctly, you will help to protect the environment. For more detailed information about the recycling of this product, please contact your local authority, your household waste disposal service provider, or the product's supplier.

1.3 Applicable standards

1.3.1 Interfaces

The *rc_visard* supports the following interface standards:

GEN<i>CAM

The Generic Interface for Cameras standard is the basis for plug & play handling of cameras and devices.

GiGE® VISION

GigE Vision® is an interface standard for transmitting high-speed video and related control data over Ethernet networks.

1.3.2 Approvals

The *rc_visard* has received the following approvals:

CE EC Declaration of Conformity

TÜV SÜD C US certification by TÜV Süd

FC Changes or modifications not expressly approved by the manufacturer could void the user's authority to operate the equipment. This device complies with Part 15 of the FCC rules. Operation is subject to the following two conditions:

1. This device may not cause harmful interference, and
2. this device must accept any interference received, including interference that may cause undesired operation.

1.3.3 Standards

The *rc_visard* has been tested to be in compliance with the following standards:

- AS/NZS CISPR32 : 2015 Information technology equipment, Radio disturbance characteristics, Limits and methods of measurement
- CISPR 32 : 2015 Electromagnetic compatibility of multimedia equipment - Emission requirements
- GB 9254 : 2008 This standard is out of the accreditation scope. Information technology equipment, Radio disturbance characteristics, Limits and methods of measurement
- EN 55032 : 2015 Electromagnetic compatibility of multimedia equipment - Emission requirements
- EN 55024 : 2010 +A1:2015 Information technology equipment, Immunity characteristics, Limits and methods of measurement

- CISPR 24 : 2015 +A1:2015 International special committee on radio interference, Information technology equipment-Immunity characteristics-Limits and methods of measurement
- EN 61000-6-2 : 2005 Electromagnetic compatibility (EMC) Part 6-2:Generic standards - Immunity for industrial environments
- EN 61000-6-3 : 2007+A1:2011 Electromagnetic compatibility (EMC) - Part 6-3: Generic standards - Emission standard for residential, commercial and light-industrial environments

1.4 Glossary

DHCP The Dynamic Host Configuration Protocol (DHCP) is used to automatically assign an *IP* address to a network device. Some DHCP servers only accept known devices. In this case, an administrator needs to configure the DHCP server with the fixed *MAC address* of a device.

DNS

mDNS The Domain Name Server (DNS) manages the host names and *IP* addresses of all network devices. It is responsible for resolving the host name into the IP address for communication with a device. A DNS can be configured to get this information automatically when a device appears on a network or manually by an administrator. In contrast, *multicast DNS* (mDNS) works without a central server by querying all devices on a local network each time a host name needs to be resolved. mDNS is available by default on Linux and Mac operating systems and is used when '.local' is appended to a host name.

GenICam GenICam is a generic standard interface for cameras. It serves as a unified interface around other standards such as *GigE Vision*, Camera Link, USB, etc. See http://genicam.org for more information.

GigE Gigabit Ethernet (GigE) is a networking technology for transmitting data at one gigabit per second.

GigE Vision GigE Vision® is a standard for configuring cameras and transmitting images over a *GigE* network link. See http://gigevision.com for more information.

IMU An Inertial Measurement Unit (IMU) consists of three accelerometers and three gyroscopes that measure the linear accelerations and the turn rates in all three dimensions.

INS An Inertial Navigation System (INS) is a 3D measurement system which uses inertial measurements (accelerations and turn rates) to compute position and orientation information. We refer to our combination of stereo vision and inertial navigation as stereo INS.

IP

IP address The Internet Protocol (IP) is a standard for sending data between devices in a computer network. Every device requires an IP address, which must be unique in the network. The IP address can be configured by *DHCP*, *Link Local*, or manually.

Link Local Link Local is a technology where network devices associate themselves with an *IP address* and check if it is unique in the local network. Link Local can be used if *DHCP* is unavailable and manual IP configuration is not or cannot be done. Link Local is especially useful for connecting a network device directly to a host computer. By default, Windows 10 reverts automatically to Link Local if DHCP is unavailable. Under Linux, Link Local must be enabled manually in the network manager.

MAC address The Media Access Control (MAC) address is a unique, persistent address for networking devices. It is also known as the hardware address of a device. In contrast to the *IP address*, the MAC address is (normally) permanently given to a device and does not change.

NTP The Network Time Protocol (NTP) is a TCP/IP protocol for synchronizing time over a network. Basically a client requests the current time from a server, and uses it to set its own clock.

PTP The Precision Time Protocol (PTP, also known as IEEE1588) is a protocol which enables more precise and robust clock synchronization than with NTP.

SDK A Software Development Kit (SDK) is a collection of software development tools or a collection of software modules.

SGM SGM stands for Semi-Global Matching and is a state-of-the-art stereo matching algorithm which offers brief run times and a great accuracy, especially at object borders, fine structures, and in weakly textured areas.

SLAM SLAM stands for Simultaneous Localization and Mapping and describes the process of creating a map of an unknown environment and simultaneously updating the sensor pose within the map.

UDP The User Datagram Protocol (UDP) is the minimal message-oriented transport layer of the Internet Protocol (*IP*) family. It uses a simple connectionless transmission model with a minimum of protocol mechanism such as integrity verification (via checksum). The *rc_visard* uses UDP for publishing its *estimated dynamical states* (Section 6.3.2) via the *rc_dynamics interface* (Section 8.3). To receive this data, applications may use datagram sockets to bind to the endpoint of the data transmission consisting in a combination of an *IP address* and a service port number such as `192.168.0.100:49500`, which is typically referred to as a *destination* of an rc_dynamics data stream in this documentation.

URI

URL A Uniform Resource Identifier (URI) is a string of characters identifying resources of the *rc_visard*'s REST-API. An example of such a URI is `/nodes/rc_stereocamera/parameters/fps`, which points to the `fps` run-time parameter of the stereo camera component.

A Uniform Resource Locator (URL) additionally specifies the full network location and protocol, i.e., an exemplary URL to locate the above resource would be `https://<rcvisard>/api/v1/nodes/rc_stereocamera/parameters/fps` where `<rcvisard>` refers to the *rc_visard*'s *IP address*.

XYZ+quaternion Format to represent a pose. See *XYZ+quaternion format* (Section 13.1.2) for its definition.

XYZABC format Format to represent a pose. See *XYZABC format* (Section 13.1.1) for its definition.

2 Safety

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

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

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

This manual explains the *rc_visard*'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_visard* in noncompliance with these warnings is inappropriate and may cause injury or damage as well as void the warranty.

Warning:

- The *rc_visard* needs to be properly mounted before use.
- All cable sets need to be secured to the *rc_visard* and the mount.
- Cords must be at most 30 m long.
- An appropriate DC power source must supply power to the *rc_visard*.
- Each *rc_visard* must be connected to a separate power supply.
- The *rc_visard*'s housing must be grounded.
- The *rc_visard*'s and any related equipment's safety guidelines must always be satisfied.
- The *rc_visard* does not fall under the purview of the machinery, low voltage, or medical directives.

Risk assessment and final application:

The *rc_visard* may be used on a robot. Robot, *rc_visard*, and any other equipment used in the final application must be evaluated with a risk assessment. The system integrator's duty is to ensure respect for all local safety measures and regulations. Depending on the application, there may be risks that need additional protection/safety measures.

2.2 Intended use

The *rc_visard* is intended for data acquisition (e.g., images, disparity images, and egomotion) in stationary and mobile robotic applications. The *rc_visard* is intended for installation on a robot, automated machinery, mobile platform, or stationary equipment. It can also be used for data acquisition in other applications.

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

The GigE Vision® industry standard used by the *rc_visard* does not support authentication and encryption. All data from and to the sensor 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_visard* only to a secured internal network.

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

The *rc_visard* may be used only within the scope of its technical specification. Any other use of the sensor 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 Hardware specification

Note: The following hardware specifications are provided here as a general reference; differences with the product might exist.

3.1 Scope of delivery

Standard delivery for an *rc_visard* includes the *rc_visard* sensor and a quickstart guide only. The full manual is available in digital form and is always installed on the sensor, accessible through the *Web GUI* (Section 4.5), and available at http://www.roboception.com/documentation.

Note: The following items are not included in the delivery unless otherwise specified:

- Couplings, adapters, mounts
- Power supply unit, cabling, and fuses
- Network cabling

Please refer to *Accessories* (Section 10) for suggested third-party cable vendors.

A connectivity kit can be purchased for the *rc_visard*. It contains an M12 to RJ45 network cable, 24 V power supply, and a DC plug to M12 power adapter. Please refer to *Accessories* (Section 10) for details.

Note: The connectivity kit is intended only for initial setup, not for permanent installation in industrial environment.

The following picture shows the important parts of the *rc_visard* which are referenced later in the documentation.

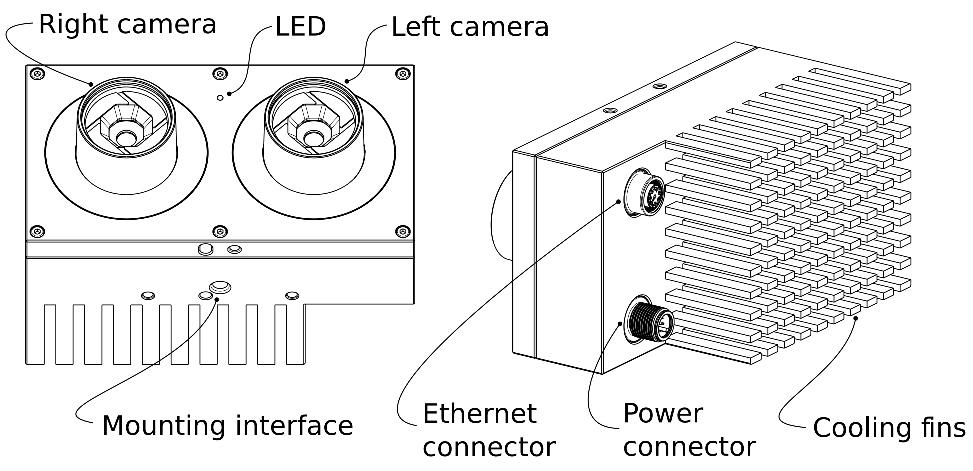


Fig. 3.1.1: Parts description

3.2 Technical specification

The common technical specifications for both *rc_visard* variants are given in Table 3.2.1.

Table 3.2.1: Common technical specifications for both *rc_visard* models

	rc_visard 65 / *rc_visard* 160
Image resolution	1280 x 960 pixel, color or monochrome
Field of view	Horizontal: 61°, Vertical: 48°
IR Cutoff	650 nm
Depth image	640 x 480 pixel (high) @ 3 Hz 320 x 240 pixel (medium) @ 15 Hz 214 x 160 pixel (low) @ 25 Hz
Egomotion	200 Hz, low latency
Computing unit	Nvidia Tegra K1
Power supply	18 V to 30 V
Cooling	Passive

The *rc_visard* 65 and *rc_visard* 160 differ in their baselines, which affects depth range and resolution as well as the sensors' size and weight.

Table 3.2.2: Differing technical specifications for the *rc_visard* variants

	rc_visard 65	*rc_visard* 160
Baseline	65 mm	160 mm
Depth range	0.2 m to infinity	0.5 m to infinity
Depth resolution	0.5 mm @ 0.2 m 15 mm @ 1.0 m	1.5 mm @ 0.5 m 6 mm @ 1.0 m 23 mm @ 2.0 m 50 mm @ 3.0 m
Size (W x H x L)	135 mm x 75 mm x 96 mm	230 mm x 75 mm x 84 mm
Mass	0.68 kg	0.84 kg

The *rc_visard* can be equipped with on-board software modules such as SLAM for additional features. These software modules can be ordered and require a license update.

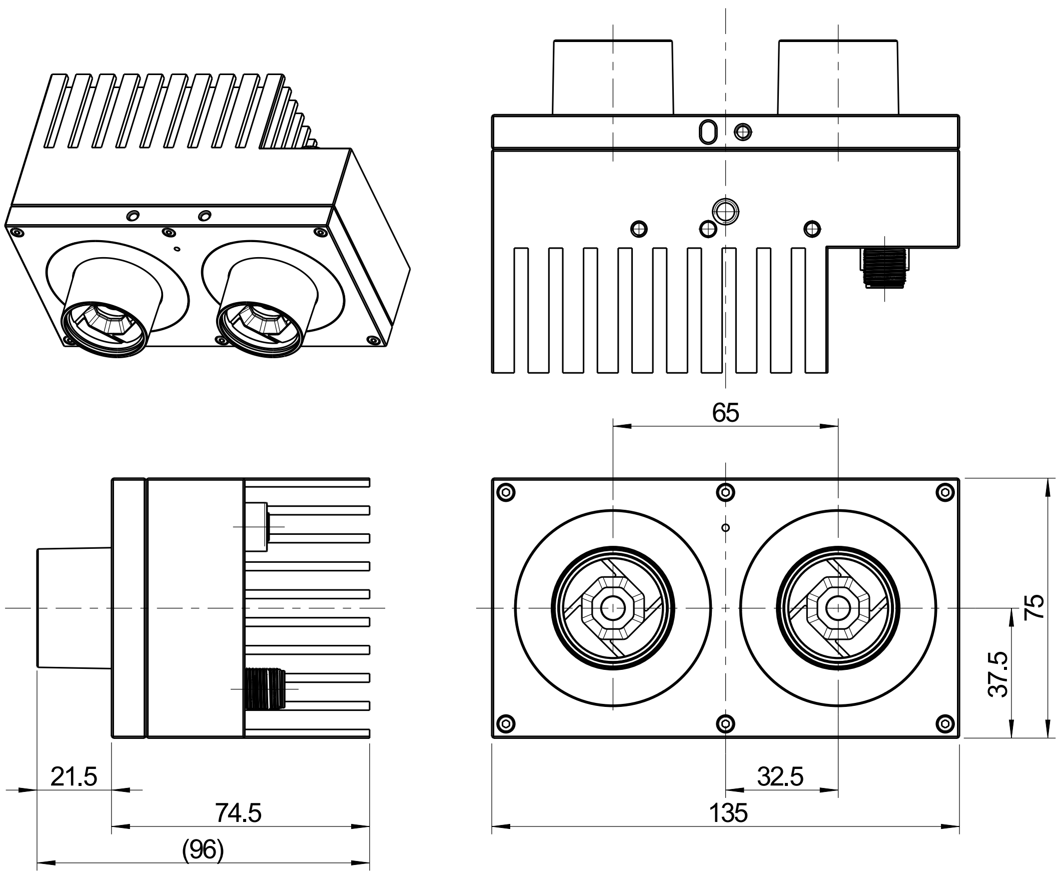


Fig. 3.2.1: Overall dimensions of the *rc_visard* 65

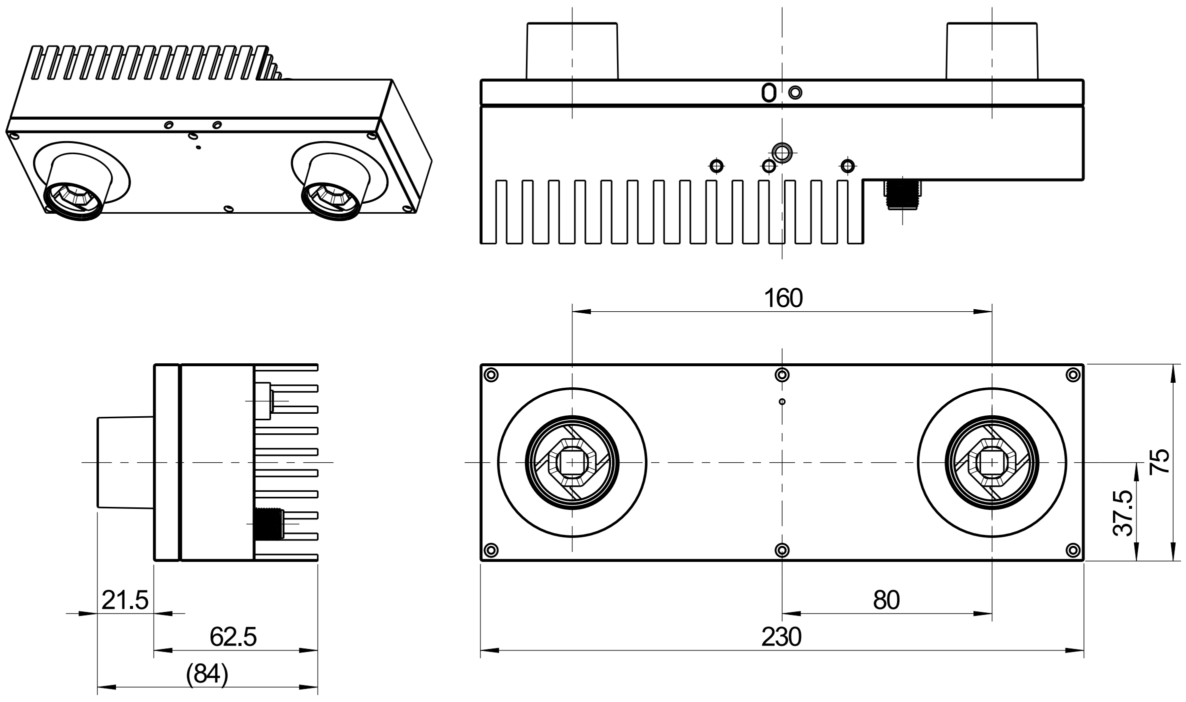


Fig. 3.2.2: Overall dimensions of the *rc_visard* 160

CAD models of the *rc_visard* can be downloaded from http://www.roboception.com/download. The CAD models are provided as-is, with no guarantee of correctness. When a material property of aluminium is assigned (density of $2.76 \frac{\text{g}}{\text{cm}^3}$), the mass properties of the CAD model are within 5% of the product with respect to weight and center of mass, and within 10% with respect to moment of inertia.

3.3 Environmental and operating conditions

The *rc_visard* is designed for industrial applications. Always respect the storage, transport, and operating environmental conditions outlined in Table 3.3.1.

Table 3.3.1: Environmental conditions

	rc_visard 65 / *rc_visard* 160
Storage/Transport temperature	-25 °C to 70 °C
Operating temperature	0 °C to 50 °C
Relative humidity (non condensing)	20 % to 80 %
Vibration	5 g
Shock	50 g
Protection class	IP54
Others	• Free from corrosive liquids or gases • Free from explosive liquids or gases • Free from powerful electromagnetic interference

The *rc_visard* is designed for an operating (surrounding environment) temperature of 0 °C to 50 °C and relies on convective (passive) cooling. Unobstructed airflow, especially around the cooling fins, needs to be ensured during use. The *rc_visard* should only be mounted using the provided mechanical mounting interface, and each part of the housing must remain uncovered. A free space of at least 10 cm extending in all directions from the housing, and sufficient air exchange with the environment is required to ensure adequate cooling. Cooling fins must be free of dirt and other contamination.

The housing temperature depends on the processing load, sensor orientation, and surrounding environmental temperatures. When the sensor's exposed housing surfaces exceed 60°C, the LED at the front will turn from green to red.

Warning: For hand-guided applications, a heat-insulated handle should be attached to the sensor to reduce the risk of burn injuries due to skin exposure to surface temperatures exceeding 60°C.

3.4 Power-supply specifications

The *rc_visard* needs to be supplied by a DC voltage source. The *rc_visard*'s standard package doesn't include a DC power supply. The power supply contained in the connectivity kit may be used for initial setup. For permanent installation, it is the customer's responsibility to provide suitable DC power. The sensor is qualified as industrial equipment Class A under EN55011. As such, each *rc_visard* must be connected to a separate power supply. Connection to domestic grid power is only allowed through a power supply certified as EN55011 Class B.

Table 3.4.1: Absolute maximum ratings for power supply

	Min	*Nominal*	*Max*
Supply voltage	18.0 V	24 V	30.0 V
Max power consumption			25 W
Overcurrent protection	Supply must be fuse-protected to a maximum of 2 A		
EMC compliance	Industrial equipment under EN55011 Class A		

Warning: Exceeding maximum power rating values may lead to damage of the *rc_visard*, power supply, and connected equipment.

Warning: A separate power supply must power each *rc_visard*.

Warning: Connection to domestic grid power is allowed through a power supply certified as EN55011 Class B only.

3.5 Wiring

Cables are not provided with the *rc_visard* standard package. It is the customer's responsibility to obtain the proper cabling. *Accessories* (Section 10) provides an overview of suggested components.

Warning: Proper cable management is mandatory. Cabling must always be secured to the *rc_visard* mount with a strain-relief clamp so that no forces due to cable movements are exerted on the *rc_visard*'s M12 connectors. Enough slack needs to be provided to allow for full range of movement of the *rc_visard* without straining the cable. The cable's minimum bend radius needs to be observed.

The *rc_visard* provides an industrial 8-pin A-coded M12 socket connector for Ethernet connectivity and an 8-pin A-coded M12 plug connector for power and GPIO connectivity. Both connectors are located at the back. Their locations (distance from centerlines) are identical for the *rc_visard* 65 and *rc_visard* 160. The location of both connectors on the *rc_visard* 65 is shown as an example in Fig. 3.5.1.

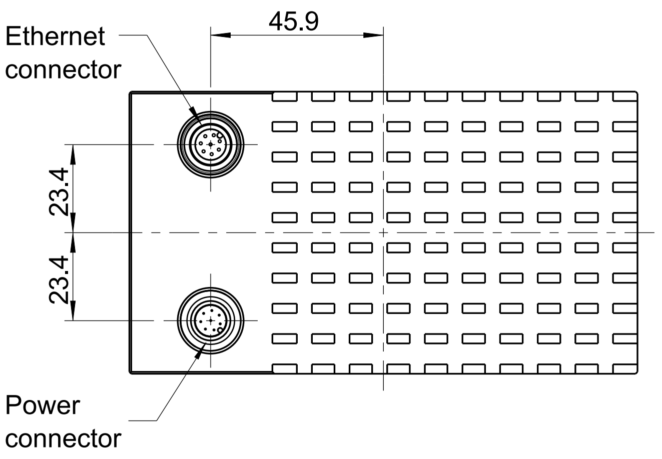
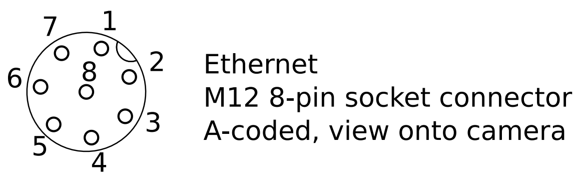


Fig. 3.5.1: Locations of the electrical connections for the *rc_visard* 65, with Ethernet on top and power on the bottom

Connectors are rotated so that standard 90° angled connectors will exit horizontally, away from the camera (away from the cooling fins).



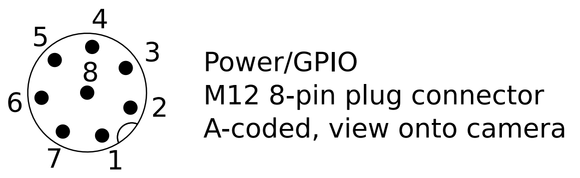


Fig. 3.5.2: Pin positions for power and Ethernet connector

Pin assignments for the Ethernet connector are given in Fig. 3.5.3.

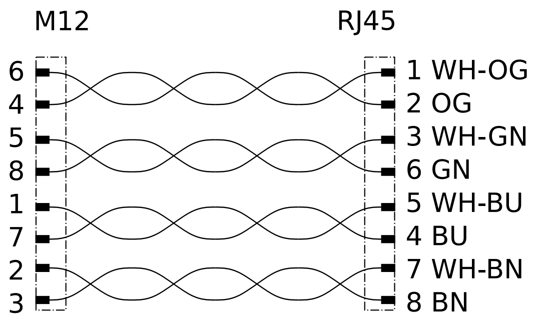


Fig. 3.5.3: Pin assignments for M12 to Ethernet cabling

Pin assignments for the power connector are given in Table 3.5.1.

Table 3.5.1: Pin assignments for the power connector

Pin	*Assignment*
1	GPIO In 2
2	Power
3	GPIO In 1
4	GPIO Gnd
5	GPIO Vcc
6	GPIO Out 1 (image exposure)
7	Gnd
8	GPIO Out 2

GPIOs are decoupled by photocoupler. *GPIO Out 1* by default provides an exposure sync signal with a logic high level for the duration of the image exposure. All GPIOs can be controlled via the optional IOControl component (*IO and Projector Control*, Section 7.2). Pins of unsused GPIOs should be left floating.

Warning: It is especially important that during the boot phase *GPIO In 1* is left floating or remains low. The *rc_visard* will not boot if the pin is high during boot time.

GPIO circuitry and specifications are shown in Fig. 3.5.4. The maximum rated voltage for *GPIO In* and *GPIO Vcc* is 30 V.

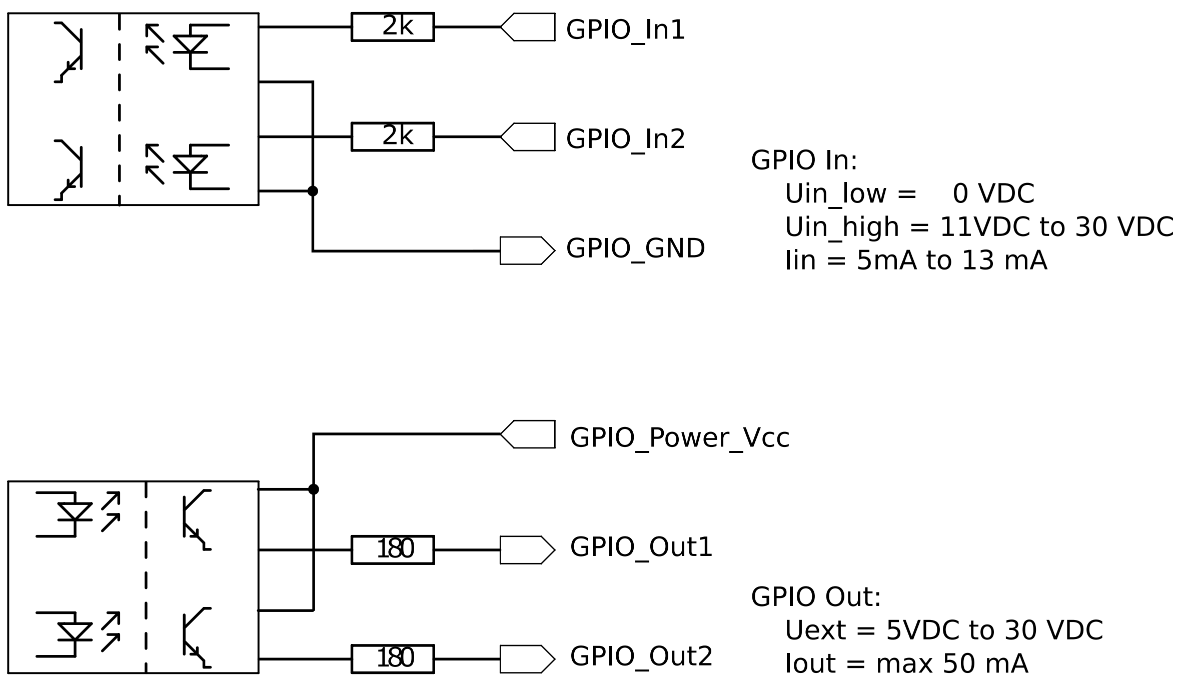


Fig. 3.5.4: GPIO circuitry and specifications – do not connect signals higher than 30 V

Warning: Do not connect signals with voltages higher than 30 V to the *rc_visard*.

3.6 Mechanical interface

The *rc_visard* 65 and *rc_visard* 160 offer identical mounting-point setups at the bottom.

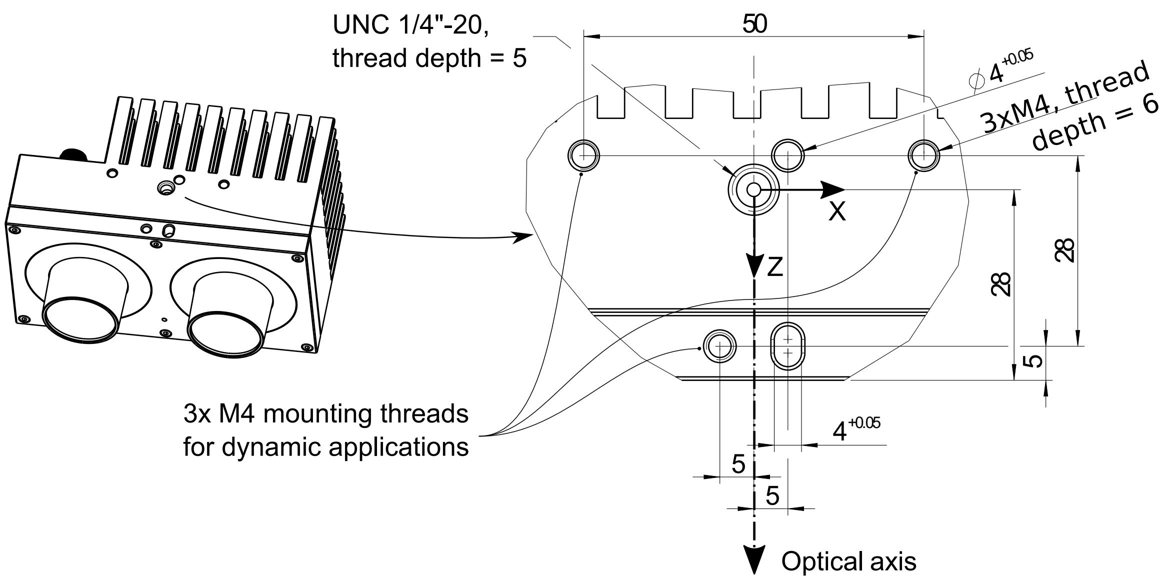


Fig. 3.6.1: Mounting-point for connecting the *rc_visard* to robots or other mountings

For troubleshooting and static applications, the sensor may be mounted using the standardized tripod thread (UNC 1/4"-20) indicated at the coordinate-frame origin. For dynamic applications such as mounting on a robotic arm, the sensor must be mounted with three M4 (metric standard) 8.8 machine screws tightened to 2.5 Nm and secured with a medium-strength threadlocking adhesive such as Loctite 243. Maximum thread depth is 6 mm. The two 4 mm diameter holes may be used for positioning pins (ISO 2338 4 m6) to ensure precise repositioning of the sensor.

Warning: For dynamic applications, the *rc_visard* must be mounted with three M4 8.8 machine screws tightened to 2.5 Nm torque and secured with threadlocking adhesive. Do not use high-strength bolts. The engaged thread depth must be at least 5 mm.

3.7 Coordinate frames

The *rc_visard*'s coordinate-frame origin is defined as the exit pupil of the left camera lens. This frame is called sensor coordinate frame or camera coordinate frame. An approximate location for the *rc_visard* 65 is shown in the next image.

The mounting-point frame for both *rc_visard* devices is defined to be at the bottom, centered in the tripod thread, with orientation identical to that of the sensor's coordinate frame. Fig. 3.7.1 shows approximate offsets.

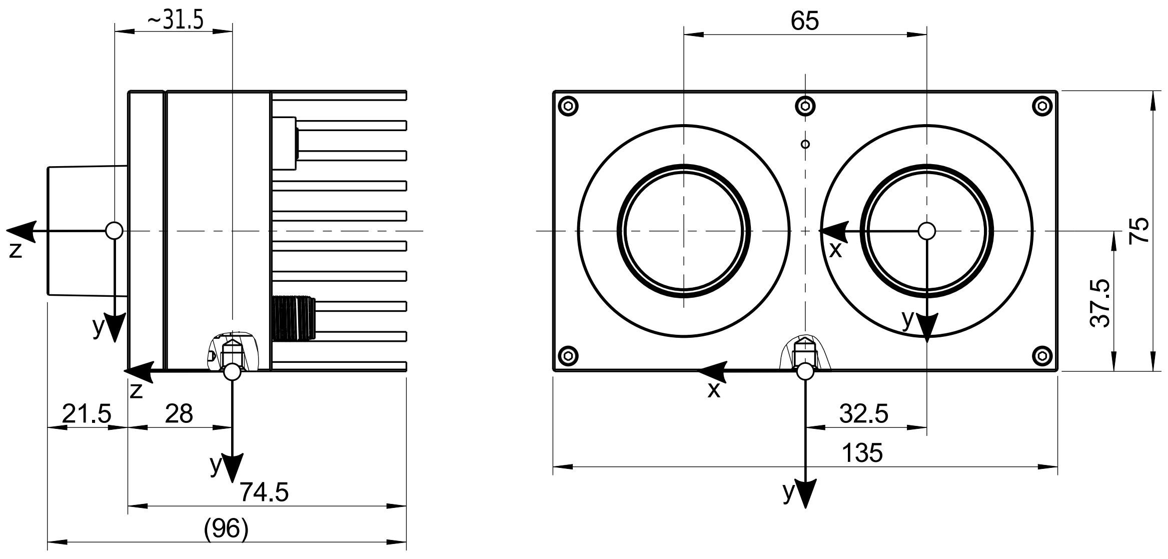


Fig. 3.7.1: Approximate location of sensor/camera coordinate frame (inside left lens) and mounting-point frame (at tripod thread) for the *rc_visard* 65

Approximate locations of sensor/camera coordinate frame and mounting-point frame for the *rc_visard* 160 are shown in Fig. 3.7.2.

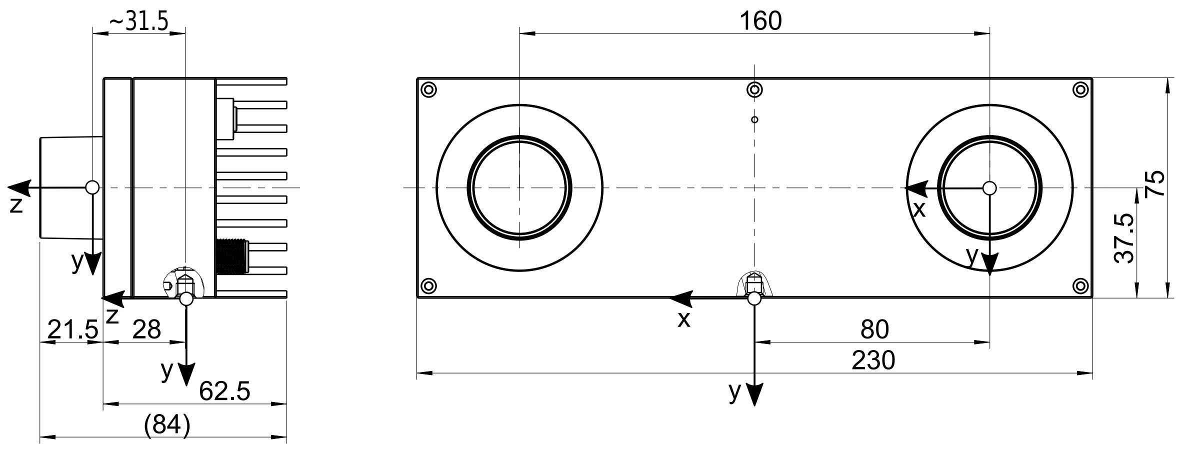


Fig. 3.7.2: Approximate locations of sensor/camera coordinate frame (inside left lens) and mounting-point frame (at tripod thread) for the *rc_visard* 160

Note: The correct offset between the sensor/camera frame and a robot coordinate frame can be calibrated through the *hand-eye-calibration procedure* (Section 6.7).

4 Installation

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

4.1 Installation and configuration

The *rc_visard* offers a Gigabit Ethernet interface for connecting the device to a computer network. All communications to and from the device are performed via this interface. The *rc_visard* has an on-board computing resource that requires booting time after powering up the device.

4.2 Power up

Note: Always fully connect and tighten the M12 power connector on the *rc_visard* *before* turning on the power supply.

After connecting the *rc_visard* to the power, the LED on the front of the device should immediately illuminate. During the device's boot process, the LED will change color and will eventually turn green. This signals that all processes are up and running.

If the network is not plugged in or the network is not properly configured, then the LED will flash red every 5 seconds. In this case, the device's network configuration should be verified. See *LED colors* (Section 11.1) for more information on the LED color codes.

4.3 Network configuration

The *rc_visard* requires an Internet Protocol (*IP*) address for communication with other network devices. The IP address must be unique in the local network, and can be set automatically or manually.

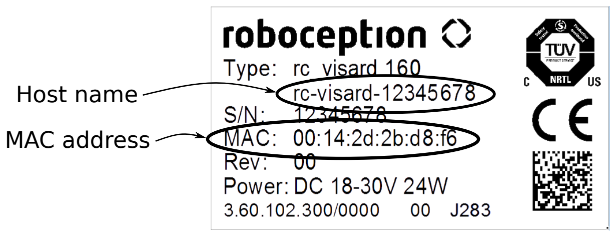


Fig. 4.3.1: Label on the *rc_visard*

4.3.1 Automatic configuration (factory default)

The Dynamic Host Configuration Protocol (*DHCP*) is preferred for setting an IP address. If DHCP is active on the *rc_visard*, which is the factory default, then the device tries to contact a DHCP server at startup and every time the network cable is plugged in. If a DHCP server is available on the network, then the IP address is automatically configured.

In some networks, the DHCP server is configured so that it only accepts known devices. In this case, the Media Access Control address (*MAC address*), which is printed on the sensor label, needs to be configured in the DHCP server. At the same time, the sensor's host name can also be set in the Domain Name Server (*DNS*). The host name is defined as `rc-visard-<serial number>`, which is also printed on the sensor. Both MAC address and host name should be sent to the network administrator for configuration.

If the *rc_visard* cannot contact a DHCP server for about 15 seconds after startup or after plugging in the network cable, it will try to assign itself a unique IP address. This process is called *Link Local*. This option is especially useful for connecting the *rc_visard* directly to a computer. The computer must be configured for Link Local as well. Link Local might already be configured as a standard fallback option, as it is under Windows 10. Other operating systems such as Linux require Link Local to be explicitly configured in their network managers.

4.3.2 Manual configuration

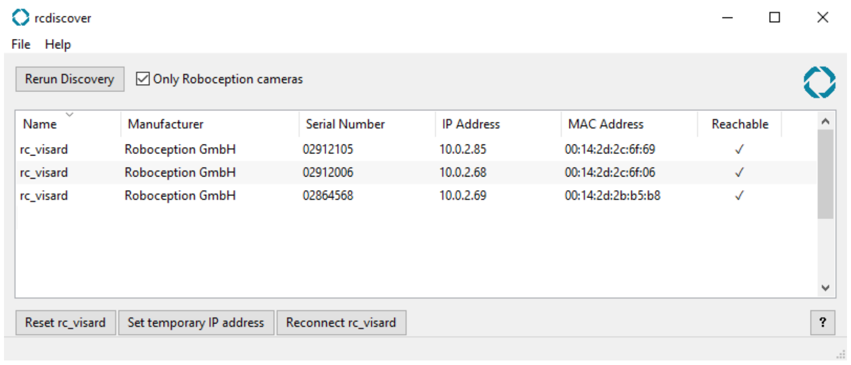
Specifying a persistent IP address manually might be useful in come cases. This is done via the sensor's standard *GigE Vision*® 2.0 interface, and requires a configuration tool to be installed on the host computer. We recommend using the IpConfigTool that is part of the Baumer GAPI SDK. The *SDK* can be downloaded free of charge for Windows and Linux from http://www.baumer.com.

After the configuration tool starts, it scans for all available GigE Vision® sensors on the network. All *rc_visard* devices can be uniquely identified by their serial number and MAC address, which are both printed on the device. If the device cannot be found, it can also be connected directly to the computer for configuration (see *Automatic configuration (factory default)*, Section 4.3.1).

Warning: The IP address must be unique and within the local network's range of valid addresses. Furthermore, the subnet mask must match the local network; otherwise, the *rc_visard* may become inaccessible. This can be avoided by using automatic configuration as explained in *Automatic configuration (factory default)* (Section 4.3.1).

4.4 Discovery of *rc_visard* devices

Devices that are powered up and connected to the local network or directly to a computer (see *Network configuration*, Section 4.3) can be found using the standard GigE Vision® discovery mechanism. Roboception offers the open-source tool `rcdiscover-gui`, which can be downloaded free of charge from http://www.roboception.com/download for Windows and Linux. The tool's Windows version consists of a single executable for Windows 7 and Windows 10, which can be executed without installation. For Linux an installation package is available for Ubuntu 14.04 and 16.04. At startup, all available *rc_visard* devices are listed with their names, serial numbers, current IP addresses, and unique MAC addresses. The discovery tool finds all devices reachable by global broadcasts. Misconfigured devices that are located in different subnets than the computer may also be listed. An icon in the discovery tool indicates whether devices are actually reachable via a web browser.



☑ Only Roboception cameras

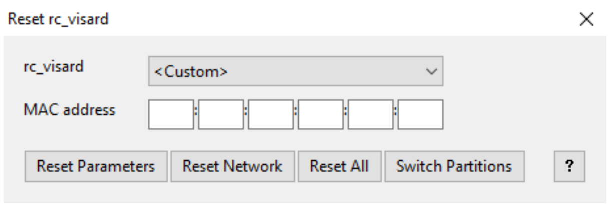
Name	Manufacturer	Serial Number	IP Address	MAC Address	Reachable
rc_visard	Roboception GmbH	02912105	10.0.2.85	00:14:2d:2c:6f:69	✓
rc_visard	Roboception GmbH	02912006	10.0.2.68	00:14:2d:2c:6f:06	✓
rc_visard	Roboception GmbH	02864568	10.0.2.69	00:14:2d:2b:b5:b8	✓

Fig. 4.4.1: `rcdiscover-gui` tool for finding connected *rc_visard* devices

After successful discovery, a double click on the device row opens the *Web GUI* (Section 4.5) of the device in the operating system's default web browser. Mozilla Firefox is recommended as web browser.

4.4.1 Resetting configuration

A misconfigured device can be reset by using the *Reset rc_visard* button in the discovery tool. The reset mechanism is only available for two minutes after device startup. Thus, the *rc_visard* may require rebooting before being able to reset the device.



rc_visard <Custom>

MAC address

Fig. 4.4.2: Reset dialog of the `rcdiscover-gui` tool

If the discovery tool still successfully detects the the misconfigured *rc_visard*, then the latter can be selected from the *rc-visard* drop-down menu. Otherwise, the *rc_visard*'s MAC address, which is printed on the device label, can be entered manually into the designated fields.

One of four options can be chosen after entering the MAC address:

- *Reset Parameters*: Reset all *rc_visard* parameters, such as frame rate, that are configurable via *Web GUI* (Section 4.5).
- *Reset Network*: Reset network settings and user-defined name.
- *Reset All*: Reset the *rc_visard* parameters as well as network settings and user-defined name.
- *Switch Partitions*: Allows a rollback to be performed as described in *Restoring the previous firmware version* (Section 9.4).

A white status LED followed by a device reboot indicates a successful reset. If no reaction is noticeable, the two minutes time slot may have elapsed, requiring another reboot.

Note: The reset mechanism is only available for the first two minutes after startup.

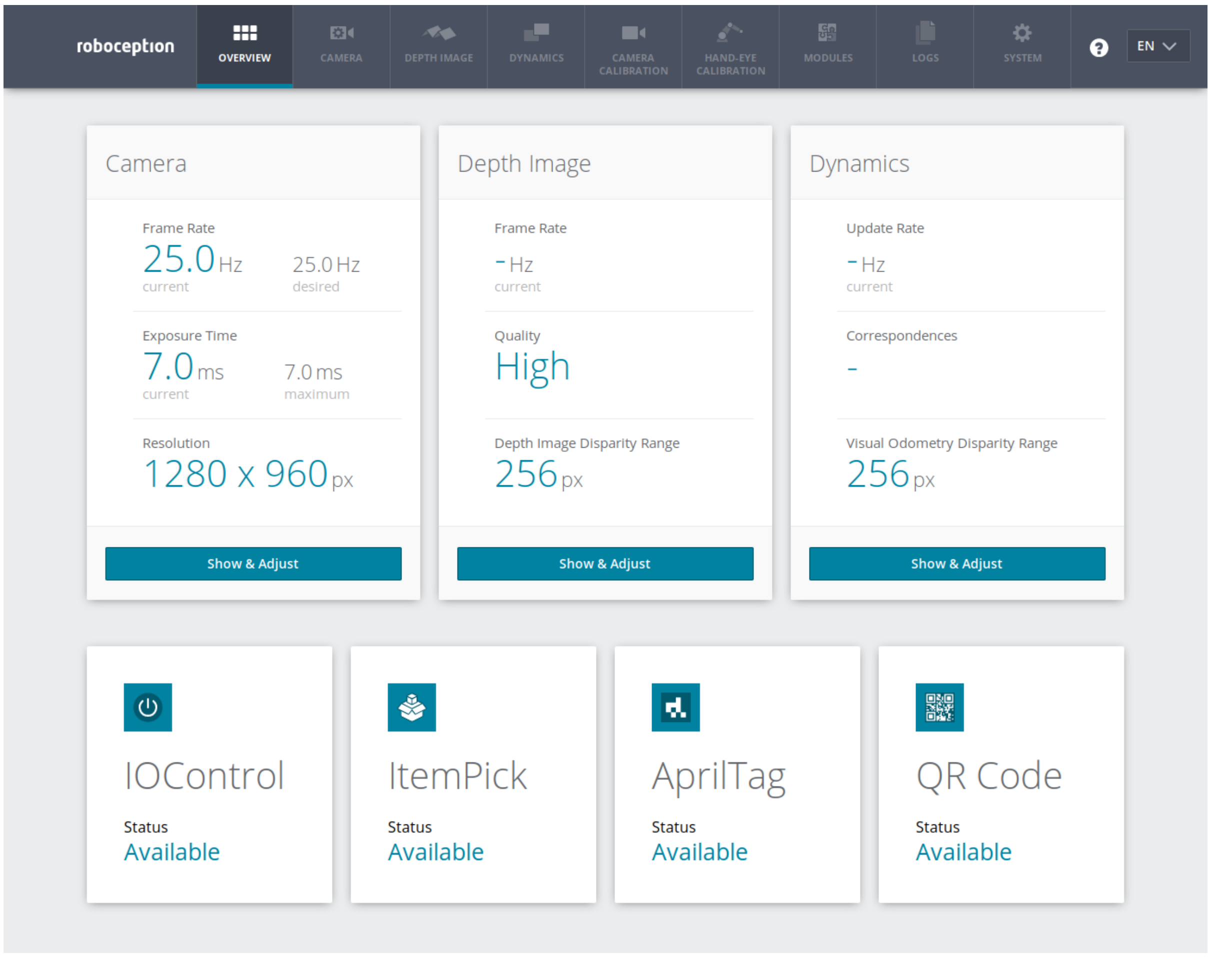
4.5 Web GUI

The *rc_visard*'s Web GUI can be used to test, calibrate, and configure on-board processing. It can be accessed from any web browser, such as Firefox, Google Chrome, or Microsoft Edge, via the sensor's IP address. The easiest way to access the Web GUI is to simply double click on the desired device using the `rcdiscover-gui` tool as explained in *Discovery of rc_visard devices* (Section 4.4).

Alternatively, some network environments automatically configure the unique host name of the *rc_visard* in their Domain Name Server (*DNS*). In this case, the Web GUI can also be accessed directly using the *URL* `http://rc-visard-<serial-number>` by replacing `<serial-number>` with the serial number printed on the device label.

For Linux and Mac operating systems, this even works without DNS via the multicast Domain Name System (*mDNS*), which is automatically used if `.local` is appended to the host name. Thus, the URL simply becomes `http://rc-visard-<serial-number>.local`.

The Web GUI's overview page gives the most important information about on-board processing.



Camera

Frame Rate
25.0 Hz current 25.0 Hz desired

Exposure Time
7.0 ms current 7.0 ms maximum

Resolution
1280 x 960 px

Depth Image

Frame Rate
- Hz current

Quality
High

Depth Image Disparity Range
256 px

Dynamics

Update Rate
- Hz current

Correspondences
-

Visual Odometry Disparity Range
256 px

IOControl
Status
Available

ItemPick
Status
Available

AprilTag
Status
Available

QR Code
Status
Available

Fig. 4.5.1: Overview page of the *rc_visard*'s Web GUI

The page's top row permits access to the individual *rc_visard* modules.

- The *Camera* module shows a live stream of the device's left and right rectified images. The frame rate can be reduced to save bandwidth when streaming to a GigE Vision® client. Furthermore, exposure can be set manually or automatically. See *Parameters* (Section 6.1.3) for more information.
- The *Depth Image* module shows a live stream of the left rectified, depth, and confidence images. The page contains various settings for depth-image computation and filtering. See *Parameters* (Section 6.2.4) for more information.

- The *Dynamics* module shows the location and movement of image features that are used to compute the *rc_visard*'s egomotion. Settings include the number of corners and features that should be used. See *Parameters* (Section 6.4.1) for more information.
- The *Camera Calibration* module permits the camera to be checked for proper calibration. In rare cases when the camera is no longer sufficiently calibrated, calibration also can be performed using this module. See *Camera calibration* (Section 6.6) for more information.
- The *Hand-Eye-Calibration* module allows the computation of the static transformation between the *rc_visard* and a coordinate system known in the robot system. This can be the flange coordinate system of a robotic arm if the *rc_visard* is attached to the flange. Alternatively, the *rc_visard* may be mounted statically in the robot environment and calibrated to any other static frame known in the robot system. See *Hand-eye calibration* (Section 6.7) for more information.
- The *Logs* module permits access to the log files on the *rc_visard*. The log files are typically checked if errors are suspected.
- The *System* module permits the firmware or the license file to be updated and provides some general information about the device.

Changed parameters are not persistent and will be lost when restarting the *rc_visard* unless they are saved by pressing the *Save* button before leaving the corresponding page.

Further information on all parameters in the Web GUI can be obtained by pressing the *Info* button next to each parameter.

5 The *rc_visard* in a nutshell

The *rc_visard* is a self-registering 3D camera. It provides rectified camera, disparity, confidence, and error images, which enable the viewed scene's depth values along with their uncertainties to be computed. Furthermore, the motion of visual features in the images is combined with acceleration and turn-rate measurements at a high rate, which enables the sensor to provide real-time estimates of its current pose, velocity, and acceleration.

5.1 Stereo vision

The *rc_visard* is based on *stereo vision* using the *SGM* (*Semi-Global Matching*) method. 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. 5.1.1.

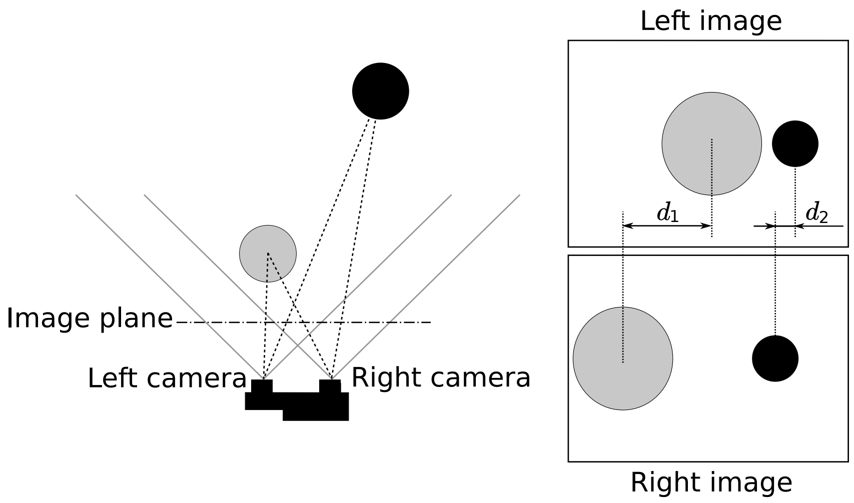


Fig. 5.1.1: Sketch of the stereo-vision principle: The more distant object (black) exhibits a smaller disparity d_2 than that of the close object (gray), d_1.

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. The *rc_visard* can thus work indoors and outdoors and multiple *rc_visard* 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 *SGM* stereo matching method used provides the best trade-off between runtime and accuracy, even for fine structures.

For stereo matching, the position and orientation of the left and right cameras relative to each other has to be known with very high accuracy. This is achieved by calibration. The *rc_visard*'s cameras are pre-calibrated during production. However, if the *rc_visard* has been decalibrated, during transport for example, then the user has to recalibrate the stereo camera.

The following *rc_visard* software components are required to compute 3D information:

- *Stereo camera*: This component is responsible for capturing synchronized stereo image pairs and transforming them into images approaching those taken by an ideal stereo camera (rectification) (Section 6.1).
- *Stereo matching*: This component computes disparities for the rectified stereo camera pair using *SGM* (Section 6.2).
- *Camera calibration*: This component enables the user to recalibrate the *rc_visard*'s stereo camera (Section 6.6).

5.2 Sensor dynamics

In addition to providing 3D information about the scene, the *rc_visard* can also estimate its *egomotion* or *dynamic state* in real time. This comprises its current pose, i.e., its position and orientation relative to a reference coordinate system or reference frame, as well as its velocity and acceleration. Measurements from stereo visual odometry (SVO) and the integrated Inertial Measurement Unit (*IMU*) are fused to compute this information. This combination is called a Visual Inertial Navigation System (V*INS*).

Visual odometry observes the motion of characteristic points in the camera images to estimate the camera motion. Object points are projected on different pixels in the camera image depending on the camera's viewing position. Each point's 3D coordinates can also be computed using stereo matching between the point positions in the left and right camera images. Thus, for two different viewing positions A and B, two sets of corresponding 3D points are computed. Assuming a static environment, the motion that transforms one set of points into the other is the camera's motion. The principle is illustrated for a simplified 2D case in Fig. 5.2.1.

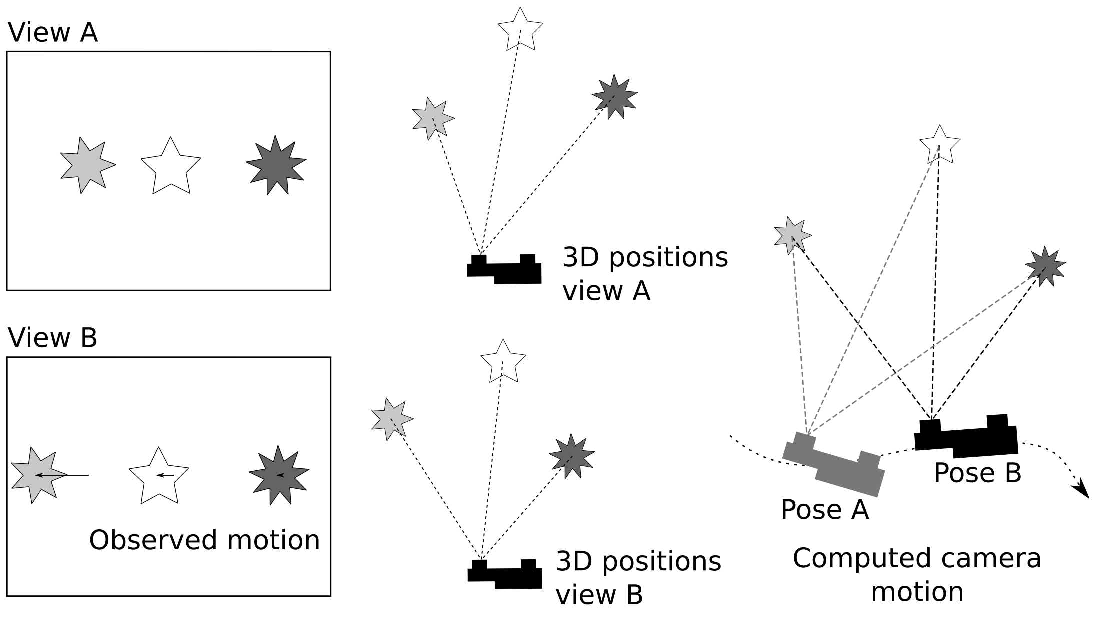


Fig. 5.2.1: Simplified sketch of the stereo visual odometry principle for 2D motions: Camera motion is computed from the observed motion of characteristic image points.

Since visual odometry relies on image-data quality, motion estimates deteriorate when the images are blurred or are poorly illuminated. Furthermore, visual odometry's frame rate is too low for control applications. That's why the *rc_visard* has an integrated Inertial Measurement Unit (IMU), which measures the accelerations and angular velocities that occur when the *rc_visard* moves. It also measures acceleration due to gravity, which gives global orientation in the vertical direction. Further, IMU measurements have a high rate of 200 Hz. The *rc_visard*'s linear velocity, position, and orientation can be computed by integrating the IMU measurements. However, the integration results suffer from increasing drift over time. The *rc_visard* thus fuses accurate, but low-frequency and sometimes volatile visual odometry measurements with reliable high-rate IMU measurements to provide an accurate, robust, high-frequency estimate of the *rc_visard*'s current position, orientation, velocity, and acceleration, which can be used in a control loop.

In addition to the stereo camera component and the calibration component, pose-estimate computations require the following *rc_visard* software components:

- *Sensor dynamics*: This component handles starting, stopping, and streaming of the estimates for the individual components (Section 6.3).
  - *Visual odometry*: This component computes a motion estimate from the camera images (Section 6.4).
  - *Stereo INS*: This component fuses the motion estimates from visual odometry with the measurements from the integrated IMU to provide real-time pose estimates at a high frequency (Section 6.5).
  - *SLAM*: This component is optionally available for the *rc_visard* and creates an internal map of the environment, which is used to correct pose errors (Section 7.1).

5.3 Calibration relative to a robot

The *rc_visard* is designed for industrial environments including those featuring robotic applications in which the *rc_visard* is either mounted on a robot or statically in a robot work cell. To use the *rc_visard*'s output, the robot must know where the sensor is located in the robot coordinate frame. To compute the *rc_visard*'s location in the robot coordinate frame, the sensor offers the so-called *Hand-eye calibration software component* (Section 6.7). The calibration routine can be executed either programmatically via the *REST-API interface* or manually via the *Web GUI* (Section 4.5).

6 Software components

The *rc_visard* comes with several on-board software components, which provide camera images, 3D information, and dynamics state estimates, and allow calibration to be performed. Each software component corresponds to a *node* in the *REST-API interface* (Section 8.2). Fig. 6.1 gives an overview of the relationships between the different software components and the data they provide via *rc_visard*'s various *interfaces* (Section 8).

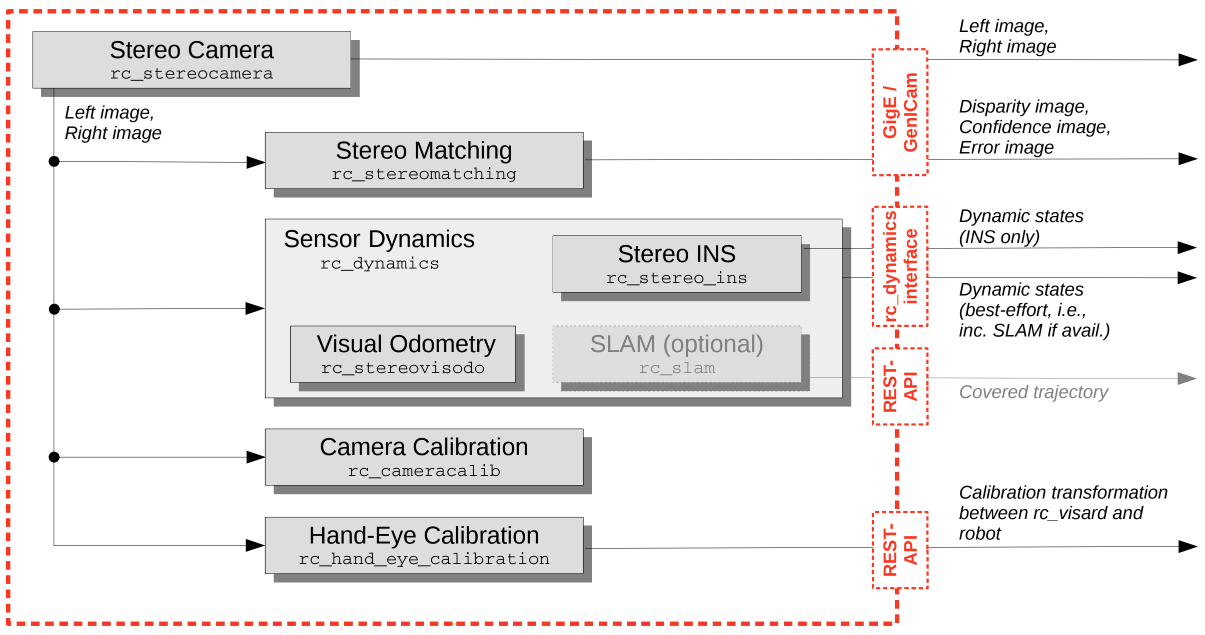


Fig. 6.1: Flowchart of the software components with their node names and the most important outputs

Note: Components marked as *optional* extend the *rc_visard*'s features. Customers can extend the license to purchase additional components.

The *rc_visard*'s on-board software consists of the following components:

- ***Stereo camera*** (**`rc_stereocamera`, Section 6.1**) acquires stereo image pairs and performs planar rectification for using the stereo camera as a measurement device. Images are provided both for further internal processing by other components and for external use as *GenICam image streams*.
- ***Stereo matching*** (**`rc_stereomatching`, Section 6.2**) uses the rectified stereo image pair to compute 3D depth information such as disparity, error, and confidence images. These are provided as GenICam streams, too.
- ***Sensor dynamics*** (**`rc_dynamics`, Section 6.3.**) provides estimates of *rc_visard*'s dynamic state such as its pose, velocity, and acceleration. These states are transmitted as continuous data streams via the *rc_dynamics interface*. For this purpose, the dynamics component manages and fuses data from the following individual subcomponents:

  - *Visual odometry* (**`rc_stereovisodo`, Section 6.4**) estimates the motion of the *rc_visard* device based on the motion of characteristic visual features in the left camera images.
  - *Stereo INS* (**`rc_stereo_ins`, Section 6.5**) combines visual odometry measurements with readings from the on-board Inertial Measurement Unit (IMU) to provide accurate and high-frequency state estimates in real time.
- *Camera calibration* (**`rc_cameracalib`, Section 6.6**) automatically checks and performs the self-calibration of the *rc_visard*'s stereo camera in case it has been decalibrated. It furthermore enables the user to check and perform recalibration manually via the *WEB GUI* (Section 4.5).
- *Hand-eye calibration* (**`rc_hand_eye_calibration`, Section 6.7**) enables the user to calibrate the *rc_visard* with respect to a robot, either via the Web GUI or the REST-API.

6.1 Stereo camera

The stereo camera component contains functionality for acquiring stereo image pairs and performing planar rectification needed to use the stereo camera as a measurement device.

6.1.1 Image acquisition

Acquiring stereo image pairs is the first step toward stereo vision. Since both cameras are equipped with global shutters and their chips are hardware-synchronized, all pixels of both camera images are always exposed at the exactly same time. *GPIO out 1* (Section 3.5) signals the respective exposure time. Additionally, the time in the middle of the image exposure is attached to the images as a timestamp. This timestamp becomes important for dynamic applications in which the *rc_visard* or the scene moves.

Exposure time can be set manually to a fixed value. This is useful in an environment where lighting is controlled so that it is always at the same intensity. The camera is set to auto exposure by default. In this mode, the *rc_visard* chooses the exposure time automatically, up to a user defined maximum. The permitted maximum is meant to limit the motion blur that occurs when taking images while the *rc_visard* or the scene is moving. The maximum exposure time thus depends on the application. If the maximum exposure time is reached, the auto-exposure algorithm uses the gain to increase image brightness. However, larger gain factors also amplify image noise. Thus, the maximum exposure time trades motion blur off against image noise under weak-light conditions.

6.1.2 Planar rectification

Camera parameters such as focal length, lens distortion, and the relationship of the cameras to each other must be exactly known to use the stereo camera as a measuring instrument. The parameters are determined by calibration (see *Camera calibration*, Section 6.6). The *rc_visard* is already calibrated at production time and normally requires no recalibration. The camera parameters describe with great precision all of the stereo-camera system's geometric properties, but the resulting model is complex and difficult to use.

Rectification is the process of remapping the images according to an ideal stereo-camera model. Lens distortion is removed and the images are aligned so that an object point is always projected onto the same image row in both images. The cameras' optical axes become exactly parallel. This means that points at infinite distance are projected onto the same image column in both images. The closer an object point is, the larger is the difference between its image columns in the right and left images. This difference is called disparity.

Mathematically, the object point $P = (P_x, P_y, P_z)$ is projected onto image point $p_l = (p_{lx}, p_{ly}, 1)$ in the left rectified image and onto $p_r = (p_{rx}, p_{ry}, 1)$ in the right rectified image by

$$A = \begin{pmatrix} f & 0 & \frac{w}{2} \\ 0 & f & \frac{h}{2} \\ 0 & 0 & 1 \end{pmatrix}, \qquad T_s = \begin{pmatrix} t \\ 0 \\ 0 \end{pmatrix},$$

$$\begin{aligned} s_1 p_l &= AP, \\ s_2 p_r &= A(P - T_s). \end{aligned}$$

The focal length f is the distance between the common image plane and the optical centers of the left and right cameras. It is measured in pixels. The baseline t is the distance between the optical centers of the two cameras. The image width w and height h are measured in pixels, too. s_1 and s_2 are scale factors ensuring that the third coordinates of the image points p_l and p_r are equal to 1.

The *rc_visard* provides the time-stamped, rectified left and right images over the GenICam interface (see *Chunk data*, Section 8.1.1). Live streams of the images are provided with reduced quality in the *Web GUI* (Section 4.5).

Note: The *rc_visard* 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.

6.1.3 Parameters

The stereo-camera software component is called `rc_stereocamera` and is represented by the *Camera* tab in the *Web GUI* (Section 4.5). The user can change the camera parameters there, or directly via the REST-API (*REST-API interface*, Section 8.2) or GigE Vision (*GigE Vision 2.0/GenICam image interface*, Section 8.1).

Note: Camera parameters cannot be changed via the Web GUI or REST-API if *rc_visard* is used via GigE Vision.

Parameter overview

This component offers the following run-time parameters.

Table 6.1.1: The `rc_stereocamera` component's run-time parameters

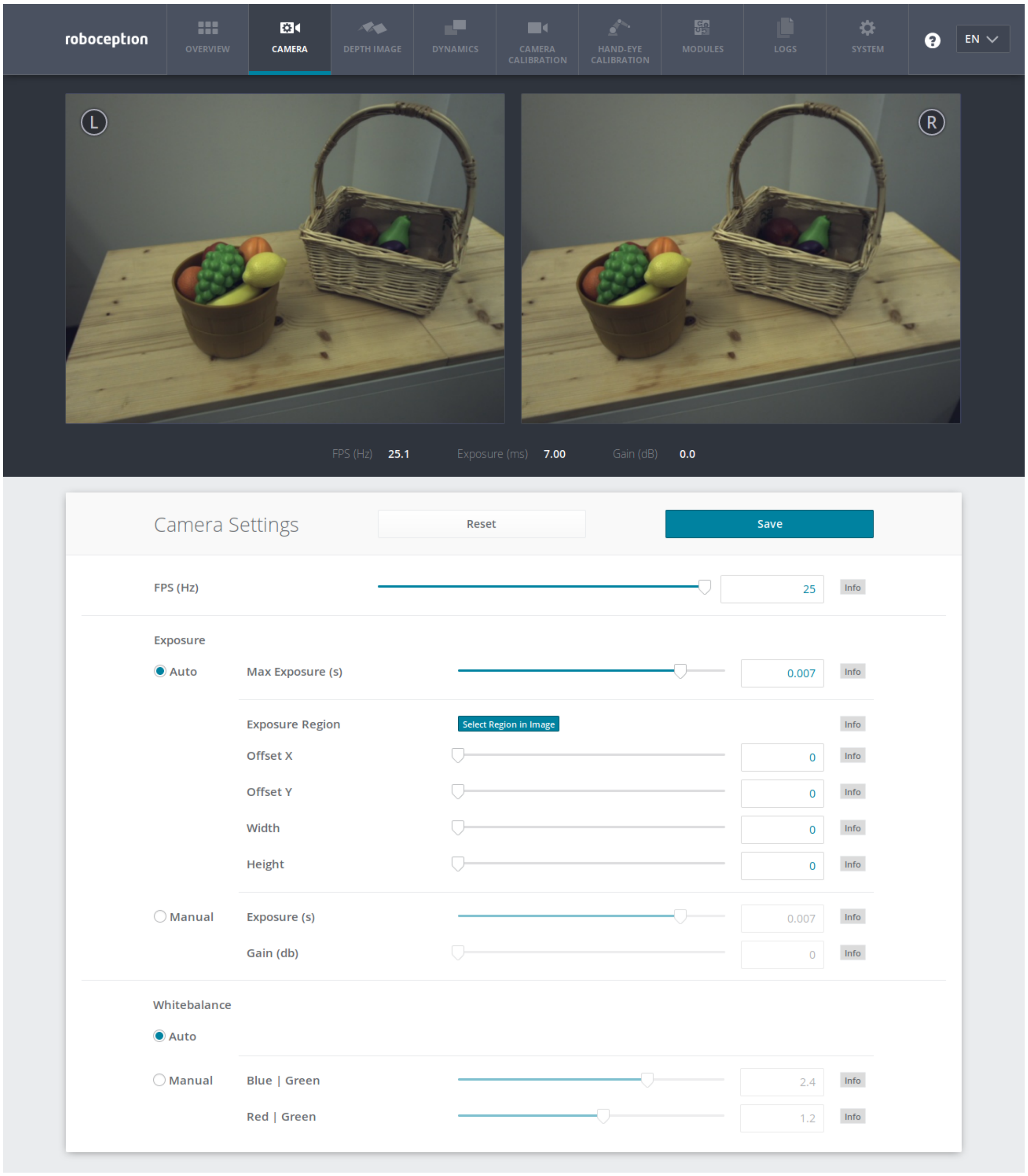
Name	Type	Min	Max	Default	Description
`exp_auto`	bool	False	True	True	Switching between auto and manual exposure
`exp_height`	int32	0	959	0	Height of auto exposure region. 0 for whole image.
`exp_max`	float64	6.6e-05	0.018	0.007	Maximum exposure time in seconds if exp_auto is true
`exp_offset_x`	int32	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	Manual exposure time in seconds if exp_auto is false
`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	Manual gain value in decibel if exp_auto is false
`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 to green balance ratio if wb_auto is false (only for color camera)
`wb_ratio_red`	float64	0.125	8.0	1.2	Red to green balance ratio if wb_auto is false (only for color camera)

This component reports the following status values.

Table 6.1.2: The `rc_stereocamera` component's status values

Name	Description
`baseline`	Stereo baseline t in meters
`color`	0 for monochrome cameras, 1 for color cameras
`exp`	Actual 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`	Actual frame rate of the camera images in Hertz. This value is shown in the Web GUI below the image preview as *FPS (Hz)*.
`gain`	Actual gain factor in decibel. This value is shown in the Web GUI below the image preview as *Gain (dB)*.
`height`	Height of the camera image in pixels
`temp_left`	Temperature of the left camera sensor in degrees Celsius
`temp_right`	Temperature of the right camera sensor in degrees Celsius
`time`	Processing time for image grabbing in seconds
`width`	Width of the camera image in pixels

Description of run-time parameters



FPS (Hz) 25.1 Exposure (ms) 7.00 Gain (dB) 0.0

Camera Settings

FPS (Hz) 25

Exposure

[✓ Auto] Max Exposure (s) 0.007

Exposure Region

Offset X 0

Offset Y 0

Width 0

Height 0

Manual Exposure (s) 0.007

Gain (db) 0

Whitebalance

[✓ Auto]

Manual Blue | Green 2.4

Red | Green 1.2

Fig. 6.1.1: The Web GUI's *Camera* tab

`fps` (*FPS*) 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_visard* delivers images via GigE Vision. Reducing this frequency also reduces the network bandwidth required to transmit the images.

The camera always runs with 25 Hz to ensure proper working of internal modules such as visual odometry that need a constant frame rate. The user frame rate setting is implemented by excluding frames for stereo matching and transmission via GigE Vision to reduce bandwidth as shown in figure Fig. 6.1.2.

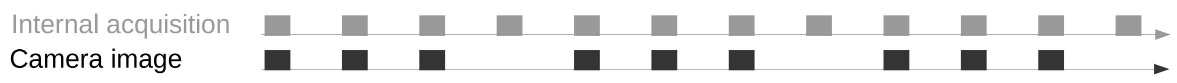


Fig. 6.1.2: Images are internally always captured with 25 Hz. The `fps` parameter determines how many of them are sent as camera images via GigE Vision.

`exp_auto` (*Exposure Auto* or *Manual*) This value can be set to 1 for auto-exposure mode, or to 0 for manual exposure mode. In manual exposure mode, the chosen exposure time is kept, even if the images are overexposed or underexposed. In auto-exposure mode, 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.

`exp_max` (*Max Exposure*) This value is the maximal exposure time in auto-exposure mode in seconds. In the Web GUI, this exposure time can be conveniently entered in milliseconds. 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_visard* 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.

`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`.

`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. In the Web GUI, this exposure time can be entered in milliseconds for convenience.

`gain_value` (*Gain*) 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.

`wb_auto` (*White Balance Auto* or *Manual*) This value can be set to 1 for automatic white balancing or 0 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.

`wb_ratio_red` and `wb_ratio_blue` (*Red | Green* and *Blue | Green*) These values are used to set red to green and blue to green ratios for manual white balance. White balancing is without function for monochrome cameras.

These parameters are also available over the GenICam interface with slightly different names and partly with different units or data types (see *GigE Vision 2.0/GenICam image interface*, Section 8.1).

6.1.4 Services

The stereo camera component offers the following services for persisting and restoring parameter settings.

`save_parameters` (*Save*) With this service call, the stereo camera component's current parameter settings will be made persistent to the *rc_visard*. That is, these values are applied even after reboot.

This service requires no arguments.

This service returns no response.

`reset_defaults` (*Reset*) Restores and applies the default values for this component's parameters ("factory reset").

Warning: The user must be aware that by calling this service, the current parameter settings for the camera component are irrecoverably lost.

This service requires no arguments.

This service returns no response.

6.2 Stereo matching

The stereo matching component uses the rectified stereo-image pair and computes disparity, error, and confidence images.

6.2.1 Computing disparity images

After rectification, the left and right images have the nice property that an object point is projected onto the same pixel row in both images. 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 *rc_visard*. 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_visard* uses *SGM* (*Semi-Global Matching*), which offers brief run times and a 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 6.2.3). 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.

If the *rc_visard* has to work in untextured environments, then 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.

6.2.2 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 *sensor coordinate frame* (Section 3.7) from the disparity image's pixel coordinates p_x, p_y and the disparity value d in pixels:

$$\begin{aligned} P_x &= \frac{p_x \cdot t}{d} \\ P_y &= \frac{p_y \cdot t}{d} \\ P_z &= \frac{f \cdot t}{d}, \end{aligned} \tag{6.2.1}$$

where f is the focal length after rectification in pixels and t is the stereo baseline in meters, which was determined during calibration. These values are also transferred over the GenICam interface (see *Custom GenICam features of the rc_visard*, Section 8.1.1).

Note: The *rc_visard* 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 (6.2.1) assume that the coordinate frame is centered in the middle of the image. The following figure shows the definition of the image coordinate frame.

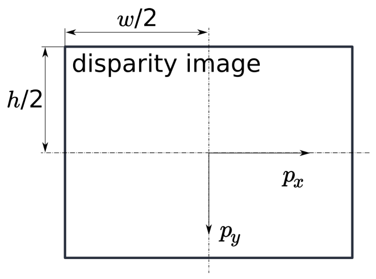


Fig. 6.2.1: The image coordinate frame's origin is defined to be at the image's center – w is the image width and h is the image height.

The same equations, but with the corresponding GenICam parameters are given in *Image stream conversions* :(Section 8.1.3).

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_visard* via GigE Vision and computing depth images and point clouds. See http://www.roboception.com/download.

6.2.3 Confidence and error images

For each disparity image, the *rc_visard* provides an error image and a confidence image, 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.

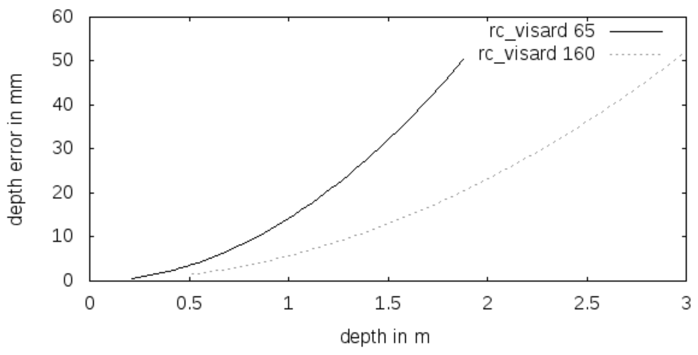
The disparity error d_{eps} (in pixels) can be converted to a depth error z_{eps} (in meters) using the focal length f (in pixels), the baseline t (in meters), and the disparity value d (in pixels) of the same pixel in the disparity image:

$$z_{eps} = \frac{d_{eps} \cdot f \cdot t}{d^2}. \tag{6.2.2}$$

Combining equations (6.2.1) and (6.2.2) allows the depth error to be related to the depth:

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

With the focal length and baselines of both *rc_visard* models and the typical disparity error d_{eps} of 0.5 pixels, the depth error can be visualized as shown below.



The *rc_visard* provides time-stamped disparity, error, and confidence images over the GenICam interface (see *Chunk data*, Section 8.1.1). Live streams of the images are provided with reduced quality in the *Web GUI* (Section 4.5).

6.2.4 Parameters

The stereo matching component is called `rc_stereomatching` in the REST-API and it is represented by the *Depth Image* tab in the *Web GUI* (Section 4.5). The user can change the stereo matching parameters there, or use the REST-API (*REST-API interface*, Section 8.2) or GigE Vision (*GigE Vision 2.0/GenICam image interface*, Section 8.1).

Parameter overview

This component offers the following run-time parameters.

Table 6.2.1: The `rc_stereomatching` component's run-time parameters

Name	Type	Min	Max	Default	Description
`acquisition_mode`	string	-	-	Continuous	S(ingleFrame), (SingleFrame)O(ut1) or C(ontinuous)
`disprange`	int32	32	512	256	Disparity range in pixels
`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
`median`	int32	1	5	1	Window size for median filtering in pixels
`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	F(ull), H(igh), M(edium), or L(ow). 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

This component reports the following status values.

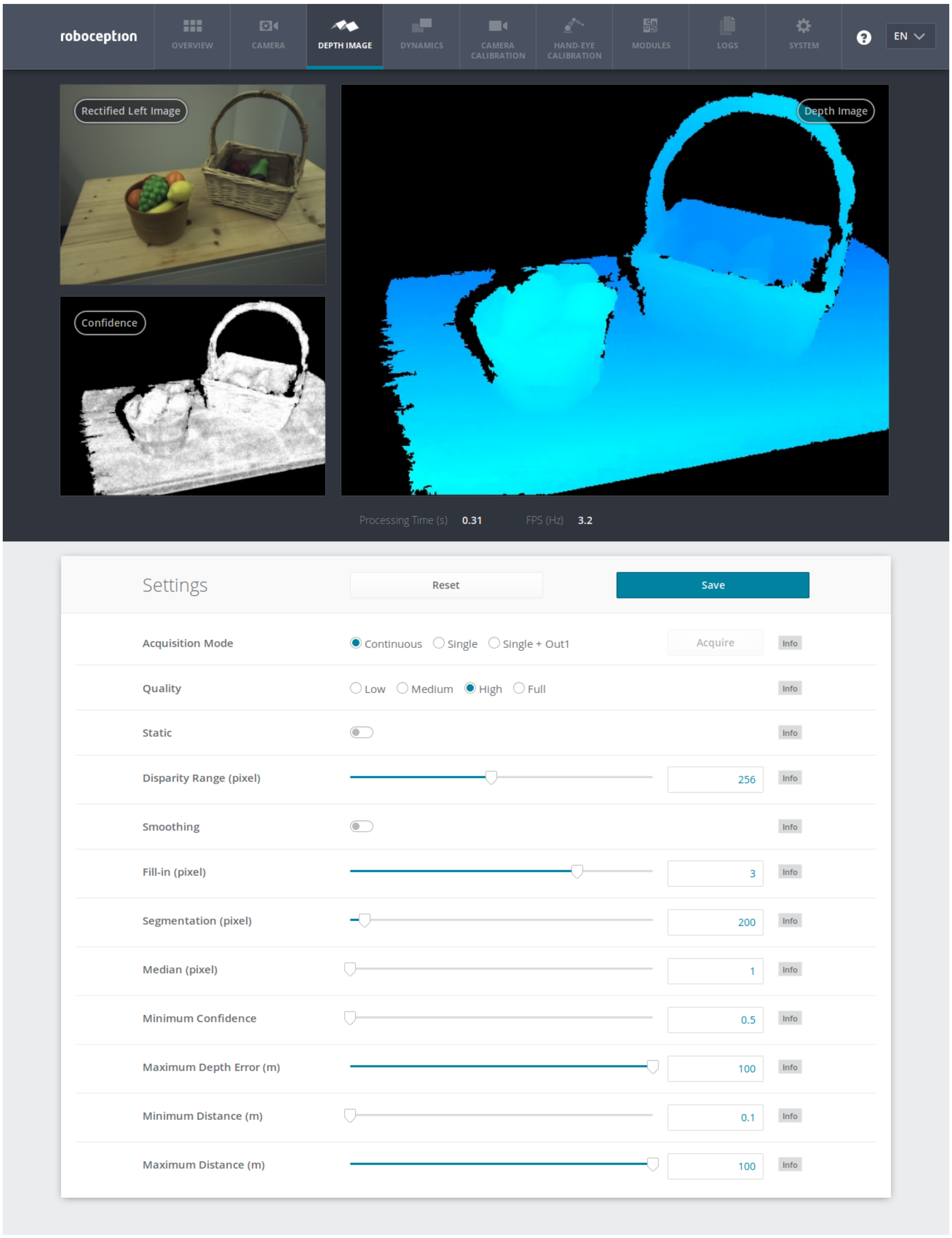
Table 6.2.2: The `rc_stereomatching` component's status values

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)*.
`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

Since SGM stereo matching and post processing run in parallel, the overall processing time for this component is the maximum of `time_matching` and `time_postprocessing`. This time is shown in the Web GUI below the image preview as *Processing Time (s)*.

Description of run-time parameters

Each run-time parameter is represented by a row on the Web GUI's *Depth Image* tab. 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:



Processing Time (s) 0.31 FPS (Hz) 3.2

Settings

Acquisition Mode	[✓ Continuous] Single Single + Out1	
Quality	Low Medium [✓ High] Full	
Static	off	
Disparity Range (pixel)		256
Smoothing	off	
Fill-in (pixel)		3
Segmentation (pixel)		200
Median (pixel)		1
Minimum Confidence		0.5
Maximum Depth Error (m)		100
Minimum Distance (m)		0.1
Maximum Distance (m)		100

Fig. 6.2.2: The Web GUI's *Depth Image* tab

`acquisition_mode` (*Acquisition Mode*) The acquisition mode can be set to *Continuous*, *Single* or *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 7.2). In this mode, `out1_mode` 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_visard*.

`quality` (*Quality*) Disparity images can be computed in different resolutions: high (640 x 480), medium (320 x 240) and low (214 x 160). The lower the resolution, the higher the frame rate of the disparity image. A 25 Hz frame rate can be achieved only at the lowest resolution. 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.

Additionally, full resolution matching with 1280 x 960 pixel is possible with a valid StereoPlus license. If full resolution is selected, the disparity range is internally limited to 128, due to limited on-board memory resources. Due to this limitation, it is not recommended to use the full resolution matching in parallel to the *SLAM* component.

`static_scene` (*Static*) This option averages 8 consecutive camera images before matching. This reduces noise, which improves the stereo matching result. 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.

`disprange` (*Disparity Range*) The disparity range always start at 0 and goes up to the maximum disparity value a pixel in the disparity image can have. Increasing the disparity range results in a smaller minimum distance that can be measured, because larger disparity values mean smaller distances. The disparity range is given in pixels and can be set to a value between 32 pixels and 512 pixels. Since a larger disparity range also means a larger search area for the matching pixel in the right rectified image, the processing time increases with a larger disparity range and the frame rate decreases. The disparity range's value is related to the high quality disparity image with 640 x 480 pixels and does not have to be scaled when a different quality is chosen. Thus, the chosen disparity range gives the same minimum distance for every image-quality option.

If full resolution quality is selected, the disparity range is internally limited to 128, due to limited on-board memory resources.

`smooth` (*Smoothing*) This option activates advanced smoothing of disparity values. It is only available with a valid StereoPlus license.

`fill` (*Fill-in*) This option is used to fill holes in the disparity image by interpolating a plane. Only holes smaller than the segmentation size (see below) are selected for interpolation. The fill-in value is the maximum allowed disparity deviation of any of the hole's border pixels from the interpolation plane. Only if all of its border pixels deviate less than the fill-in value from the plane, a hole will be filled. Larger fill-in values decrease the number of holes, but the interpolated values can have larger errors. The confidence for the interpolated pixels is set to a low value of 0.5. Their error is set to the mean deviation of the hole border pixels from the interpolation plane. A value of 0 effectively switches hole filling off.

`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 640 x 480 pixels 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.

`median` (*Median*) This value gives the window side length in pixels for the median filter, which smoothes the disparity image. Larger values lead to oversmoothing and cost more processing time. A window size of 1 effectively turns this filter off.

`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.

`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 sensor (see *Confidence and error images*, Section 6.2.3).

`mindepth` **(*Minimum Distance*)** The minimum distance is the smallest distance from the sensor 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.

`maxdepth` **(*Maximum Distance*)** The maximum distance is the largest distance from the sensor 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.

The same parameters are also available over the GenICam interface with slightly different names and partly with different data types (see *GigE Vision 2.0/GenICam image interface*, Section 8.1).

6.2.5 Services

The stereo matching component offers the following services for persisting and restoring parameter settings.

`acquisition_trigger` This call signals the module to perform stereo matching of the next available images, if the parameter `acquisition_mode` is set to *SingleFrame*. An error is returned if the `acquisition_mode` is set to *Continuous*.

This service requires no arguments.

This service returns the following response:

```
{
  "return_code": {
    "message": "string",
    "value": "int16"
  }
}
```

Possible return codes are shown below.

Table 6.2.3: 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

`save_parameters` **(*Save*)** With this service call, the stereo matching component's current parameter settings are persisted to the *rc_visard*. That is, these values are applied even after reboot.

This service requires no arguments.

This service returns no response.

`reset_defaults` **(*Reset*)** Restores and applies the default values for this component's parameters ("factory reset").

Warning: The user must be aware that calling this service causes the current parameter settings for the stereo matching component to be irrecoverably lost.

This service requires no arguments.

This service returns no response.

6.3 Sensor dynamics

The dynamics component provides estimates of the sensor state. These include pose, linear velocity, linear acceleration, and rotational rates. The component handles starting and stopping, and streaming of the estimates for individual subcomponents:

- *Visual odometry* (**`rc_stereovisodo`**) estimates the camera's motion from the motion of characteristic image points in the left camera images (Section 6.4).
- *Stereo INS* (**`rc_stereo_ins`**) combines visual odometry measurements with readings from an inertial measurement unit (*IMU*) to provide accurate, high-frequency state estimates in real time (Section 6.5).
- *SLAM* (**`rc_slam`**) performs simultaneous localization and mapping (*SLAM*) for correcting accumulated poses (Section 7.1).

Note: Using *Stereo matching* in parallel to the dynamics component may lead to decreased localization accuracy. See *Visual odometry* for how to avoid this.

6.3.1 Coordinate frames for state estimation

The world coordinate frame for state estimation is defined as follows: The coordinate frame's z-axis points upward and is aligned with the gravity vector. The x-axis is orthogonal to the z-axis and points in the *rc_visard*'s viewing direction at the time when the pose estimation starts. The world frame's origin is located at the origin of the *rc_visard*'s IMU coordinate frame at the instant when state estimation is switched on.

If pose estimation is switched on when the *rc_visard*'s viewing direction parallels the gravity vector (with a tolerance range of 10 degrees), then the world coordinate frame's y-axis is aligned either with the IMU's positive or negative x-axis. In this orientation, the initial alignment of the world coordinate frame is no longer continuous. Thus, special care has to be taken when pose estimation has to be started at such an orientation.

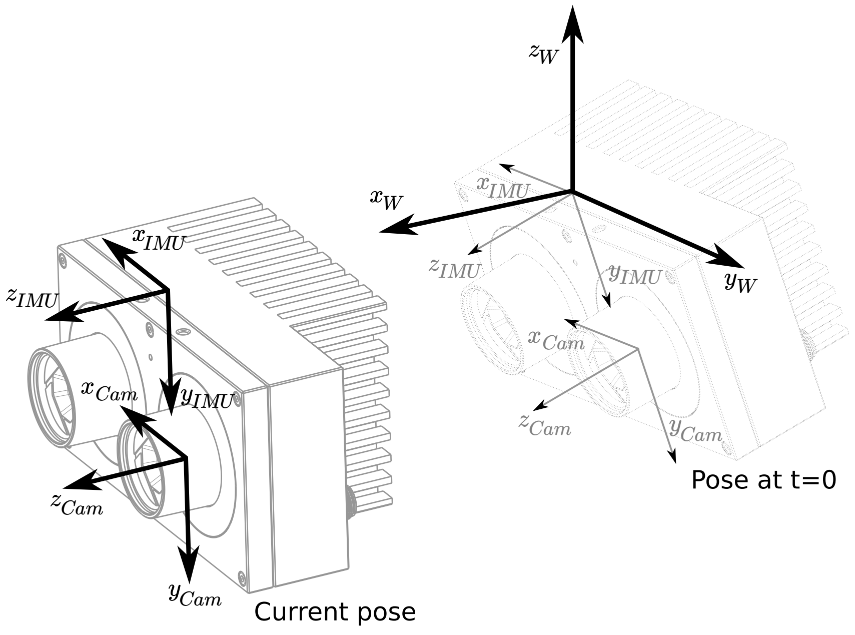


Fig. 6.3.1: Coordinate frames for state estimation. The IMU coordinate frame is inside the *rc_visard*'s housing. The *camera coordinate frame* (Section 3.7) is in the focal point of the left camera.

The transformation between the IMU coordinate frame and the camera/sensor frame is also estimated and provided in the *real-time dynamics stream* over the rc_dynamics interface (see *Interfaces*, Section 8).

Warning: The stereo INS component self-calibrates the IMU during its initialization. It is therefore required that the *rc_visard* is not moving and sufficient texture is visible during startup of the stereo INS component.

6.3.2 Available state estimates

The *rc_visard* provides seven different kinds of timestamped state-estimate data streams via the rc_dynamics interface (see *The rc_dynamics interface*, Section 8.3):

Name	Frequency	Source	Description
pose	25 Hz	best effort	Pose of camera frame, slightly delayed but most accurate
pose_ins	25 Hz	*Stereo INS*	Pose of camera frame, slightly delayed but most accurate
pose_rt	200 Hz	best effort	Pose of camera frame
pose_rt_ins	200 Hz	*Stereo INS*	Pose of camera frame
dynamics	200 Hz	best effort	Pose, velocity and acceleration in IMU frame
dynamics_ins	200 Hz	*Stereo INS*	Pose, velocity and acceleration in IMU frame
imu	200 Hz	*Stereo INS*	Raw IMU data

Best effort here means that if *SLAM* is running, then it contains the loop-closure corrected estimates and is equivalent to the stream from *Stereo INS* when SLAM is not running.

Camera-pose streams (`pose` and `pose_ins`)

The *camera-pose streams* called `pose` and `pose_ins` are provided at 25 Hz with timestamps that correspond to image timestamps. The former stream is the best-effort estimate, combining `rc_slam` and `rc_stereo_ins` if the *SLAM* component is running. If SLAM is not running, then both data streams are equivalent. Pose values are given in world coordinates, and also refer to the *rc_visard*'s camera frame origin (see *Coordinate frames for state estimation*, Section 6.3.1). They are the most accurate estimates, taking all available *rc_visard* information into consideration. They can be used in modeling applications, where camera images, depth images, or point clouds have to be aligned highly accurately with each other. To ensure the greatest possible accuracy, these pose values are delayed until a corresponding visual odometry measurement is available.

Real-time camera-pose streams (`pose_rt` and `pose_rt_ins`)

Two *real-time pose streams* called `pose_rt` and `pose_rt_ins` are provided at the IMU rate of 200 Hz. The former stream is the best-effort estimate, combining `rc_slam` and `rc_stereo_ins` when the SLAM component is running. If SLAM is not running, then both data streams are equivalent. They consist of the pose estimates of the *rc_visard*'s camera frame origin (see *Coordinate frames for state estimation*, Section 6.3.1) in world coordinates. The values given in these streams correspond to the values in the *real-time dynamics streams*, but give the pose of the sensor/camera coordinate frame instead of that of the IMU coordinate frame.

Real-time dynamics streams (`dynamics` and `dynamics_ins`)

Two *real-time dynamics streams* called `dynamics` and `dynamics_ins` are provided at the IMU rate of 200 Hz. The former stream is the best-effort estimate, combining `rc_slam` and `rc_stereo_ins` when the SLAM component is running. If SLAM is not running, then both data streams are equivalent. The estimates can be used for real-time control of a robot. Since the values are provided in real time and visual odometry computation requires some processing time, the latest visual odometry estimate may not be included. Therefore, these estimates are in general slightly less accurate than those in the non-real-time *camera-pose streams* (see above), but are the best estimates available at this instant. The provided dynamics streams contain the *rc_visard*'s

- translation $\mathbf{p} = (x, y, z)^T$ in m,
- rotation $\mathbf{q} = (q_x, q_y, q_z, q_w)^T$ as unit quaternion,

- linear velocities $\mathbf{v} = (v_x, v_y, v_z)^T$ in $\frac{m}{s}$,
- angular velocities $\omega = (\omega_x, \omega_y, \omega_z)^T$ in $\frac{rad}{s}$,
- gravity-compensated linear accelerations $\mathbf{a} = (a_x, a_y, a_z)^T$ in $\frac{m}{s^2}$, and
- transformation from camera to IMU coordinate frame as pose with frame name and parent frame name.

For each component, the stream also provides the name of the coordinate frame in which the values are given. Translation, rotation, and linear velocities are given in the world frame; angular velocities and accelerations are given in the IMU frame (see *Coordinate frames for state estimation*, Section 6.3.1). All values refer to the IMU frame's origin. That means, for example, that linear velocity is the velocity of the IMU frame's origin in the world frame.

Lastly, the stream contains a `possible_jump` flag, which is set to *true* whenever the optional SLAM component (see *SLAM*, Section 7.1) corrects the state estimation after finding a loop closure. The state estimate can jump in this case, which should be considered when the values are used in a control loop. If SLAM is not running, the jump flag can be ignored and will stay *false*.

IMU data stream (`imu`)

The *IMU data stream* called `imu` is provided at the IMU rate of 200 Hz. It consists of the acceleration in x, y, z directions plus the angular velocities around these three axis. The values are calibrated but not bias- and gravity-compensated, and are given in the IMU frame. The transformation between IMU and sensor frame is provided in the *real-time dynamics stream*.

6.3.3 Services

The sensor dynamics component offers the following services for starting dynamics/motion estimation. All services return a numerical code of the entered state. The meaning of the returned state codes and names are given in Table 6.3.1.

Table 6.3.1: Possible states of the sensor dynamics component

State name	Description
IDLE	The component is ready, but idle
WAITING_FOR_INS	Waiting for stereo INS to start up
WAITING_FOR_INS_AND_SLAM	Waiting for stereo INS and SLAM to start up
RUNNING	The stereo INS component is running (SLAM is not running)
WAITING_FOR_SLAM	Waiting for SLAM to start up (stereo INS is running)
RUNNING_WITH_SLAM	Both stereo INS and SLAM are running
STOPPING	Transitional state when going to (or through IDLE)
FATAL	A fatal error has occured (either in stereo INS or SLAM)

`start` Starts the stereo INS component. Transitions from state `IDLE` through `WAITING_FOR_INS` to `RUNNING`.

This service requires no arguments.

This service returns the following response:

```
{
  "accepted": "bool",
  "current_state": "string"
}
```

`start_slam` Starts the SLAM and – if not yet started – the stereo INS component. From state `IDLE`: Transitions through `WAITING_FOR_INS_AND_SLAM` and `WAITING_FOR_SLAM` to `RUNNING_WITH_SLAM`. From state `RUNNING`: Transitions through `WAITING_FOR_SLAM` to `RUNNING_WITH_SLAM`.

This service requires no arguments.

This service returns the following response:

```
{
  "accepted": "bool",
  "current_state": "string"
}
```

stop Stops the stereo INS and – if running – the SLAM components. The trajectory estimate of the SLAM component will still be available. Transitions from state `RUNNING` or `RUNNING_WITH_SLAM` through `STOPPING` to `IDLE`.

This service requires no arguments.

This service returns the following response:

```
{
  "accepted": "bool",
  "current_state": "string"
}
```

stop_slam Stops the SLAM component. Stereo INS will continue to run. The trajectory estimate of the SLAM component will still be available. Transitions from state `RUNNING_WITH_SLAM` to `RUNNING`.

This service requires no arguments.

This service returns the following response:

```
{
  "accepted": "bool",
  "current_state": "string"
}
```

restart Restarts to stereo INS. Equivalent to successive `stop` and `start`.

From state `RUNNING` or `RUNNING_WITH_SLAM`: Transitions through states `STOPPING`, `IDLE` and `WAITING_FOR_INS` to `RUNNING`.

This service requires no arguments.

This service returns the following response:

```
{
  "accepted": "bool",
  "current_state": "string"
}
```

restart_slam Restarts to SLAM mode. Equivalent to successive `stop` and `start_slam`.

From state `RUNNING` or `RUNNING_WITH_SLAM`: Transitions through states `STOPPING`, `IDLE`, `WAITING_FOR_INS_AND_SLAM`, `WAITING_FOR_SLAM` to `RUNNING_WITH_SLAM`.

This service requires no arguments.

This service returns the following response:

```
{
  "accepted": "bool",
  "current_state": "string"
}
```

The following diagram shows the main states and transitions. Intermediate states and the fatal error state are omitted for conceptual clarity.

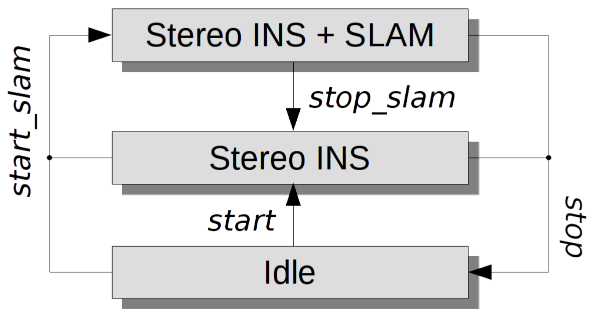


Fig. 6.3.2: Simplified state and transition diagram

These services shall respond quickly. Therefore, for services that cause a state transition the value of the returned `current_state` in general is the first new (intermediate) state that was transitioned to, not the final state. E.g., for the `start` command the returned `current_state` will be WAITING_FOR_INS, not state RUNNING. If the transition does not take place within 0.1 seconds, the current state is returned. See Table 6.3.1 for the meaning of the returned state codes.

Note: The state `FATAL` can only be left by calling `stop`, which performs a transition to the state `IDLE`. The services `restart` and `restart_slam` internally use `stop` and will also work as expected. `start` and `start_slam` only work if the state is `IDLE`, and do nothing if the state is `FATAL`.

Note: The dynamics components can also be started and stopped on the *Dynamics* page of the *Web GUI*.

`get_cam2imu_transform` returns the transformation from camera to IMU coordinate frame. This is equivalent to the `cam2imu_transform` in the *Dynamics message* (Section 8.3.3).

This service requires no arguments.

This service returns the following response:

```
{
  "name": "string",
  "parent": "string",
  "pose": {
    "pose": {
      "orientation": {
        "w": "float64",
        "x": "float64",
        "y": "float64",
        "z": "float64"
      },
      "position": {
        "x": "float64",
        "y": "float64",
        "z": "float64"
      }
    },
    "timestamp": {
      "nsec": "int32",
      "sec": "int32"
    }
  }
}
```

6.4 Visual odometry

Visual odometry is part of the sensor dynamics component. It is used to estimate the camera's motion from the motion of characteristic image points (so-called image features) in left camera images. Image features are computed from image corners, which are image regions with high intensity gradients. Image features are used to look for matches between subsequent images to find correspondences. Their 3D coordinates are computed by stereo matching (independent from the disparity image). The camera's motion is computed from a set of corresponding 3D points between two images. To increase the robustness of visual odometry, correspondences are not only computed to the previous camera image but to a certain number of previous images, which are called *keyframes*. The best result is then chosen.

The visual-odometry frame rate is independent of the user setting in the stereo camera component. It is internally limited to 12 Hz but can be lower, depending on the number of features and keyframes. To ensure good pose-estimation quality, the frame rate should not drop significantly under 10 Hz.

Note: Using *Stereo matching* in parallel to the dynamics component may lead to a decreased frame rate of the visual odometry. In this case, we recommend to decrease the frame rate of the *Stereo camera* (effectively decreasing the frame rate of the depth image computation), to lower the computational load of stereo matching.

The visual odometry component's measurements are not directly accessible on the *rc_visard*. Instead, they are internally fused with measurements from the integrated inertial measurement unit to increase robustness and frame rate and reduce latency. The result of the sensor data fusion is provided in the form of different streams (see *Stereo INS*, Section 6.5).

6.4.1 Parameters

The visual odometry software component is called `rc_stereovisodo` and it is represented by the *Dynamics* tab in the *Web GUI* (Section 4.5). The user can change the visual odometry parameters there, or use the REST-API (*REST-API interface*, Section 8.2).

Parameter overview

This component offers the following run-time parameters.

Table 6.4.1: The `rc_stereovisodo` component's run-time parameters

Name	Type	Min	Max	Default	Description
`disprange`	int32	32	512	256	Disparity range in pixels
`ncorner`	int32	50	4000	500	Number of corners
`nfeature`	int32	50	4000	300	Number of features
`nkey`	int32	1	4	4	Number of keyframes

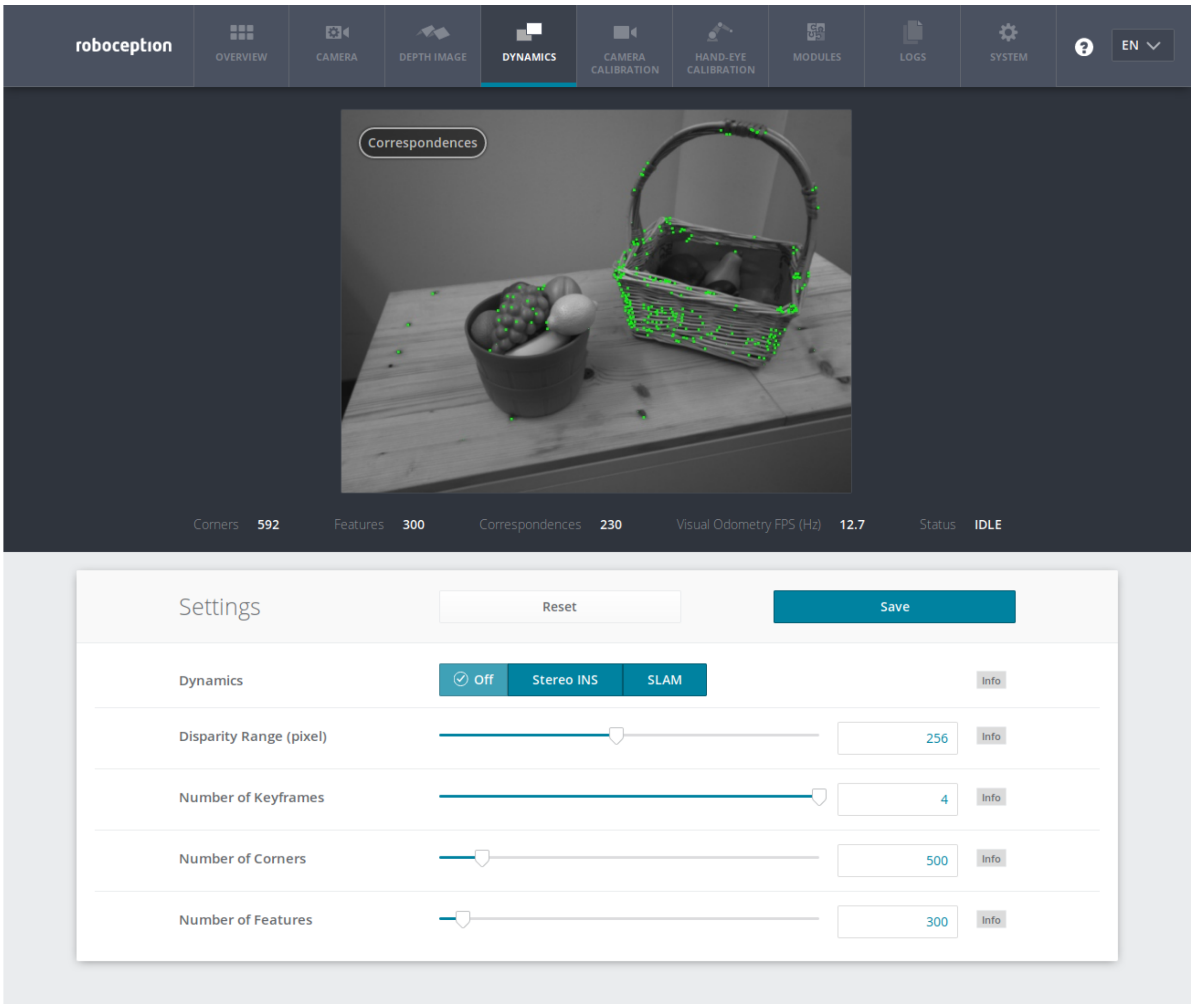
This component reports the following status values.

Table 6.4.2: The `rc_stereovisodo` component's status values

Name	Description
`corner`	Number of detected corners. This value is shown as *Corners* below the image preview in the Web GUI.
`correspondences`	Number of correspondences. This value is shown as *Correspondences* below the image preview in the Web GUI.
`feature`	Number of features. This value is shown as *Features* below the image preview in the Web GUI.
`fps`	Frame rate of the visual odometry in Hertz. This value is shown below the image preview as *Visual Odometry FPS (Hz)* in the Web GUI.
`time_frame`	Processing time in seconds to compute corners and features for each frame
`time_vo`	Processing time in seconds to compute the motion

Description of run-time parameters

Run-time parameters influence the number of features used to compute visual odometry. More features increase the visual odometry's robustness at the expense of more run time, which can reduce the frame rate. Although the resulting state estimate will always have a high frequency due to fusion with IMU measurements, high visual-odometry frame rates are nevertheless desirable, since these measurements are much more accurate than IMU measurements alone. A visual-odometry rate of at least 10 Hz should thus be aimed for. The visual-odometry frame rate is provided as a status parameter and is shown below the camera image on the *Web GUI*'s *Dynamics* page.



Corners 592 Features 300 Correspondences 230 Visual Odometry FPS (Hz) 12.7 Status IDLE

Settings

Dynamics	[✓ Off] Stereo INS SLAM
Disparity Range (pixel)	256
Number of Keyframes	4
Number of Corners	500
Number of Features	300

Fig. 6.4.1: The Web GUI's *Dynamics* tab

The camera image shown on this page depicts image features as small green dots. The bold green dots are the features in the current image for which correspondences could be found in a previous keyframe. Green lines depict the motion of these features relative to the previous keyframe. This visualization should help to find a good set of parameters for visual odometry. The number of correspondences is reported as a status parameter and is shown below the camera image on the *Web GUI*'s *Dynamics* page. For robust visual-odometry measurements, the parameters should be adjusted so that the resulting number of correspondences in the target environment is around at least 50 when the sensor is moving. The correspondence count will be larger when the *rc_visard* is static, and the number will change when the *rc_visard* moves through the environment. Short failures of the visual odometry are tolerated due to the fusion with IMU measurements. Longer failures should be avoided because they lead to large pose uncertainties and can lead to errors in the state estimation.

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

`start` (*Dynamics*) This starts the sensor dynamics estimation components (see *Services*, Section 6.3.3).

`disprange` (*Disparity Range*) The disparity range gives the maximum disparity value for each image feature related to the resolution of the high-quality disparity image (640 x 480 pixels). The disparity range determines the minimum working distance of the visual odometry. When the disparity range is narrow, only more distant features are considered in the visual-odometry estimation. When choosing a broader disparity range, close features can also be used. Broader disparity ranges increase processing time, which can reduce the visual odometry's frame rate.

`nkey` (*Number of Keyframes*) More keyframes can increase the robustness and accuracy of the visual odometry, but they also increase processing time and can decrease the visual-odometry frame rate.

`ncorner` (*Number of Corners*) This value gives the approximate number of corners that will be detected in the left image. Larger numbers make visual odometry more robust and accurate but can lead to lower frame rates of the visual odometry.

`nfeature` (*Number of Features*) This value is the maximum number of features that will be derived from the corners. It is useful to detect more corners and select the best subset as features. Larger numbers make visual odometry more robust and accurate but can lead to lower visual-odometry frame rates. Fewer features might be computed, depending on the scene and movement. The actual number of features is reported below the camera image on the *Web GUI*'s *Dynamics* page.

Note: Increasing the number of keyframes, corners, or features will also increase robustness but will require more computation time and may reduce the frame rate, depending on other components active on the *rc_visard*. The visual-odometry frame rate should be at least 10 Hz.

6.4.2 Services

The visual odometry component offers the following services for persisting and restoring parameter settings. The names of the corresponding Web GUI buttons are added in brackets:

`save_parameters` (*Save*) With this service call, the current parameter settings of the visual odometry component are persisted to the *rc_visard*. That is, these values are applied even after reboot.

This service requires no arguments.

This service returns no response.

`reset_defaults` (*Reset*) Restores and applies the default values for this component's parameters ("factory reset").

Warning: The user must be aware that calling this service causes irrecoverable loss of the visual odometry component's current parameter settings.

This service requires no arguments.

This service returns no response.

This component offers no start or stop services itself, because the *dynamics component* (Section 6.3) starts and stops it.

6.5 Stereo INS

The stereo-vision-aided Inertial Navigation System (*INS*) component is part of the sensor dynamics component. It combines visual-odometry measurements with inertial measurement unit (*IMU*) data and provides robust, low latency, real-time state estimates at a high rate. The IMU consists of three accelerometers and three gyroscopes, which measure accelerations and turn rates in all three dimensions. By fusing IMU and visual-odometry measurements, the state estimate has the same frequency as the IMU (200 Hz) and is very robust even under challenging lighting conditions and for fast motions.

Note: To achieve high-quality pose estimates, it must be ensured that sufficient texture is visible during runtime of the stereo INS component. In case no texture is visible for a longer period of time, the stereo INS component will stop instead of providing highly erroneous data.

6.5.1 Self-Calibration

During startup of the stereo INS component, it will self-calibrate the IMU using the visual-odometry measurements. For the self-calibration to succeed, it is required that

- the *rc_visard* is not moving and
- sufficient texture is visible

during startup of the stereo INS component. Failure to meet these requirements will most likely result in a constant drift of the pose estimates.

6.5.2 Parameters

The stereo INS component's node name is `rc_stereo_ins`.

This component has no run-time parameters.

This component reports the following status values.

Table 6.5.1: The `rc_stereo_ins` component's status values

Name	Description
`freq`	Frequency of the stereo INS process in Hertz. This value is shown as *Update Rate* in the Web GUI *Overview* tab in the *Dynamics* area
`state`	String representing the internal state

6.6 Camera calibration

To use the stereo 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 *Planar rectification*, Section 6.1.2), which is the basis for all other image processing modules. The *rc_visard* is calibrated at production time. Nevertheless, checking calibration and recalibration might be necessary if the *rc_visard* was exposed to strong mechanical impact. The camera calibration component is responsible for checking calibration and recalibrating.

6.6.1 Self-calibration

The camera calibration component automatically runs in self-calibration mode at a low frequency in the background. In this mode, the *rc_visard* observes the alignment of image rows of both rectified images. A mechanical impact, such as one caused by dropping the *rc_visard*, might result in a misalignment. If a significant misalignment is detected, then it is automatically corrected. After each reboot and after each correction, the current self-calibration offset is reported in the camera component's log file (see *Downloading log files*, Section 9.7) as:

> *"rc_stereocalib: Current self-calibration offset is 0.00, update counter is 0"*

The update counter is incremented after each automatic correction. It is reset to 0 after manual recalibration of the *rc_visard*.

Under normal conditions, such as the absence of mechanical impact on the *rc_visard*, self-calibration should never occur. Self-calibration allows the *rc_visard* to work normally even after misalignment is detected, since it is automatically corrected. Nevertheless, checking camera calibration manually is recommended if the update counter is not 0.

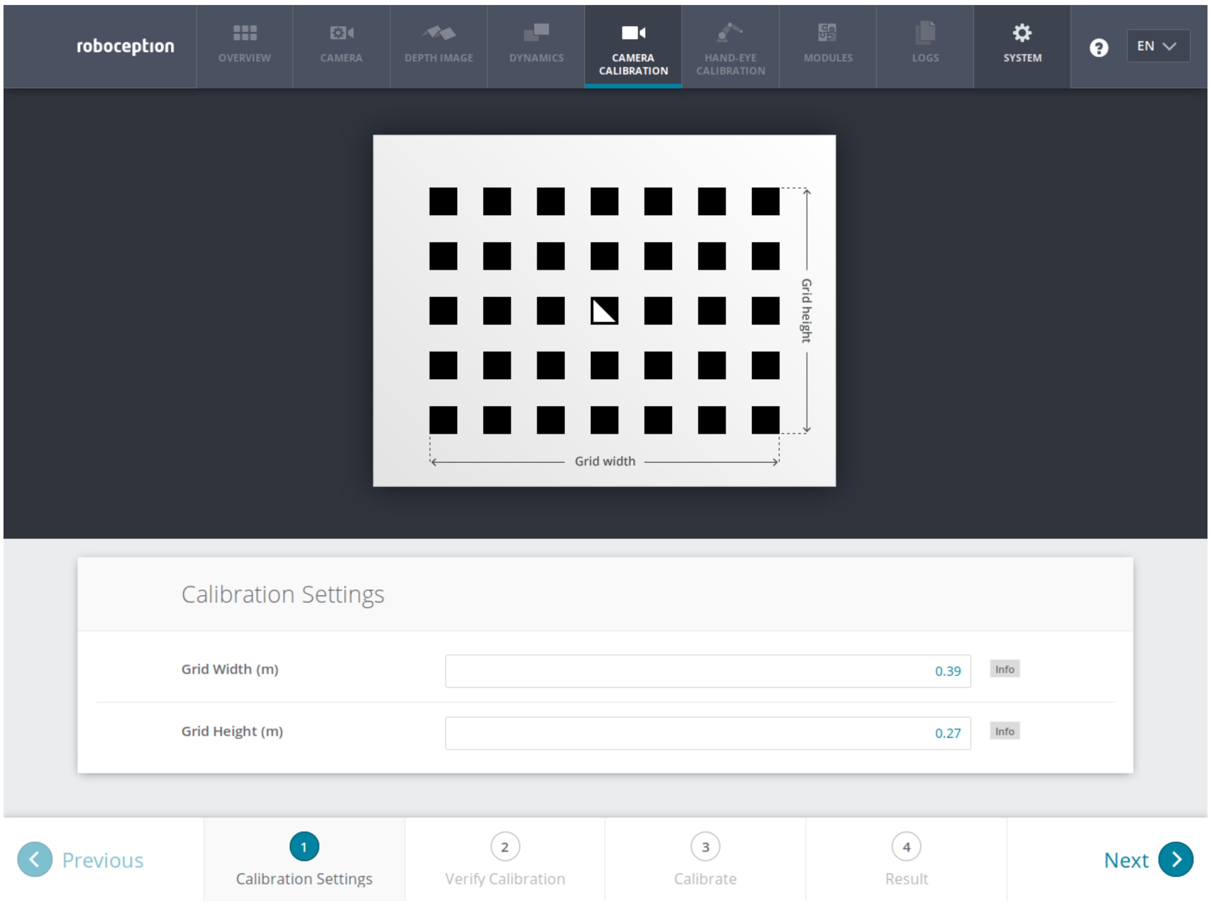
6.6.2 Calibration process

Manual calibration can be done through the Web GUI's *Camera Calibration* tab. This tab provides a wizard to guide the user through the calibration process.

Note: Camera calibration is normally unnecessary since the *rc_visard* is calibrated at production time. Therefore, calibration is only required after strong mechanical impacts, such as occur when dropping the *rc_visard*.

Step 1: Calibration settings

The quality of camera calibration heavily depends on the quality of the calibration grid. Calibration grids for the *rc_visard* can be obtained from Roboception.



Calibration Settings

Grid Width (m) 0.39

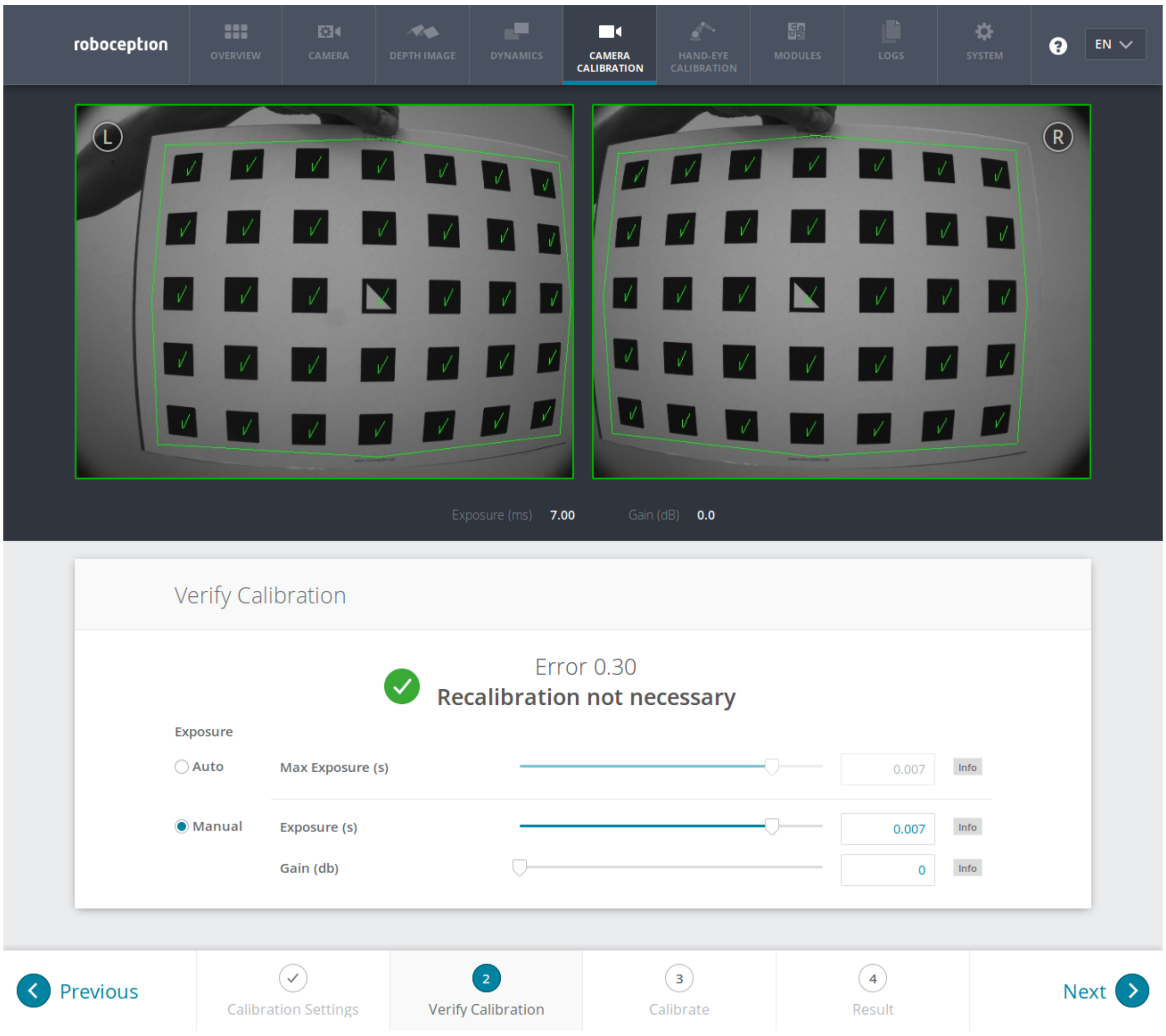
Grid Height (m) 0.27

Fig. 6.6.1: Calibration settings

The *Camera calibration* component has to be selected in the *Web GUI* (Section 4.5) to verify or perform camera calibration. In the first step, the width and height of the grid must be specified as shown in the screenshot above. The *Next* button proceeds to the next step.

Step 2: Verify calibration

In the second 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. Make sure that all black squares of the grid are completely visible and not occluded. 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. After the grid is detected, the calibration error is automatically computed, and the result is displayed on the screen.



Exposure (ms) 7.00 Gain (dB) 0.0

Verify Calibration

Error 0.30
Recalibration not necessary

Exposure

Auto — Max Exposure (s) 0.007 Info

Manual (selected) — Exposure (s) 0.007 Info

Gain (db) 0 Info

Fig. 6.6.2: Verification of calibration

Some of the squares not being detected, or being detected only briefly might indicate a low-quality or damaged calibration grid, or bad lighting conditions.

Note: To compute a meaningful calibration error, the grid should be held as closely 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.

The typical calibration error is around 0.3 pixels. If the error is less than 0.4 to 0.5 pixels, 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.