RB-Sherpa

User guide manual

Robotnik

Contents

1. Introduction

Welcome to the RB-Sherpa user guide manual. This document describes the main aspects of the robot.

1.1. Overall robot description

The RB-Sherpa is a robotic platform for application development (Outdoor, logistics, indoor transport, etc.) The RB-Sherpa series can execute different applications:

- Freight transport for exteriors and interiors
- Logistics, warehouses and other intra-logistics applications

The mobile platform has Ackermann kinematics based on 4 high power motor wheels and two more motors for the steering. Each wheel integrates a hub brushless motor with gearbox and dc motors with gearbox for the steering (An omnidirectional kinematic is available).

1.2. Intended use and safety warnings

This point describes the safety devices that the RB-Sherpa has implemented, as well as the indicators that indicate the risks that the robot can cause. Furthermore, some situations of specific risk that should be avoided or taken into account for the correct use of the robot are explained in this part.

1.2.1. Safety concept

While reading this guide you may find some warning blocks. These include important information related to common errors or safety concerns that the final user must know.

WARNING: a warning block

"Warning" description includes information on an error and some possible solutions.

FORBIDDEN: a forbidden block

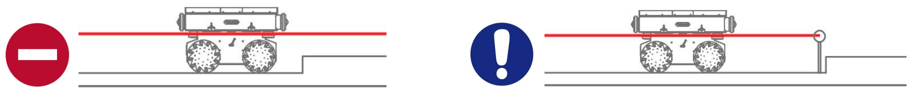
"Forbidden" description includes information about the situation and how avoid it.

MANDATORY: a mandatory block

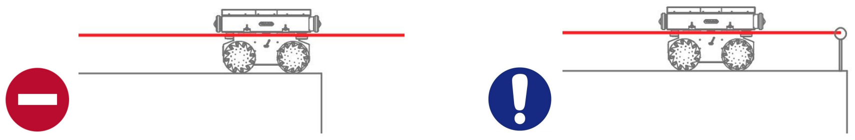
"Mandatory" description includes information to solve forbidden situations.

1.2.2. Specific risk situations

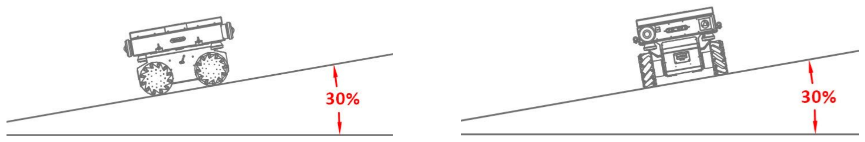
STEPS UP
The robot is capable of climbing steps up to 30 mm in height. Any step of greater height is considered a barrier that can not be overcome by the robot and must be marked by a beacon located at the height of the laser vision. Not marking with beacons, you could damage the robot or injure someone.



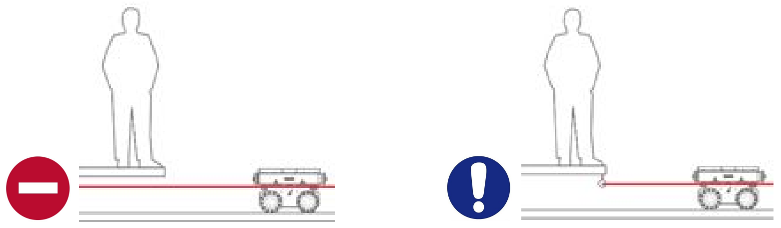
STEPS DOWN
The robot is capable of going steps down of 30 mm maximum height. Any higher step is considered a non-surmountable barrier by the robot. It must be marked by a beacon located at the height of the laser beam. Not marking with beacons, you could damage the robot or injure someone.



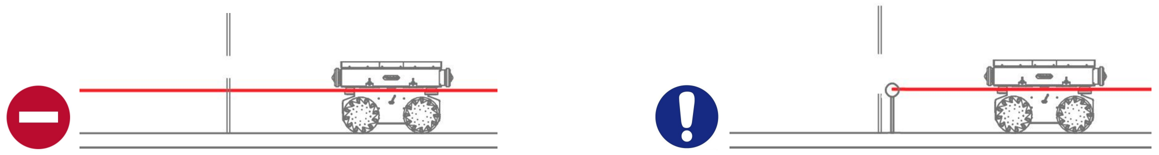
SLOPES
The robot is capable of climbing a maximum slope of 30%. Any greater slope can destabilize the robot, cause its rollover and damage itself or injure someone.



NON-DETECTABLES OBSTACLES
The safety sensors of the platform are not capable of detecting aerial obstacles if they are not correctly marked with beacons at the height of the laser beam. Not marking with beacons, you could damage the robot or injure someone.

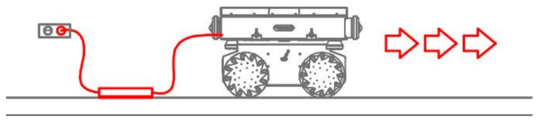


The safety sensors of the platform are not able to detect obstacles like transparent glass. The glasses must be marked with beacons at the height of the laser beam. Not marking with beacons, you could damage the robot or injure someone.



1.2.3. General safety instructions

The robot platform should not be moved when the robot is connected to the manual charger. You could damage the robot.



Do not sprinkle water or oil on the robot or power charging cord.. The exterior cleaning of the robot must be done with pressurized air or dry cloth. Contact with water or oil can cause electric shock or malfunction of the unit.

Do not overload the robot. Check its maximum payload in its technical specification. Incorrect use could damage the robot or injure someone.

Do not drive the robot outdoors when the weather is rainy. You could damage the robot.

Do not attempt to disassemble or modify the robot. You could damage the robot or injure yourself.

Keep the emergency stop pussed if there is an operator close to the robot.

Remember to update maps when the route changes to avoid new fixed obstacles. Not updating maps, you could damage the robot or injure yourself.

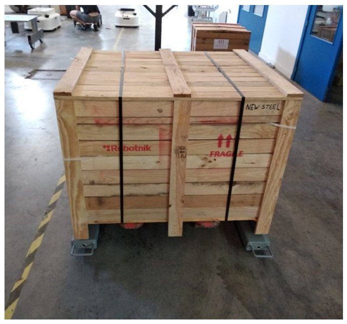
Always use the original charger and plug the power cord firmly into the wall outlet. Incomplete insertion in the wall outlet or the use of another charger could cause the plug to heat up, possibly causing a fire.

Follow these steps if there is evidence of a battery malfunction. Use personal protective equipment, such as gloves, safety glasses and lab coat.

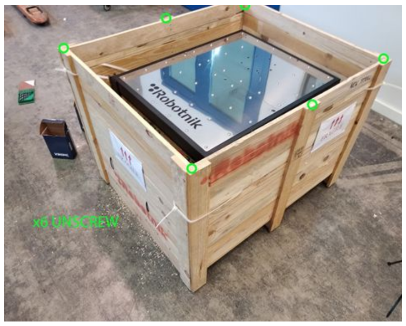
- If batteries are showing evidence of thermal runaway failure, be very cautious because the gases may be flammable and toxic and failure modes can be hazardous.
- Disconnect the battery. Do not disassemble or break it.
- Remove the battery from the robot.
- Place the battery in a metal or other container away from combustibles.

2. Quickstart

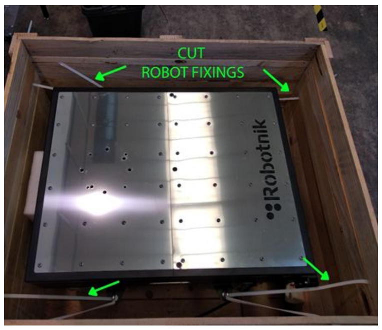
2.1. Unpacking the robot



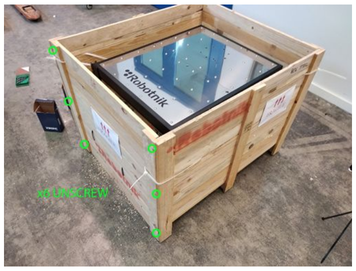
1. Package overview
(Cut external bridles)



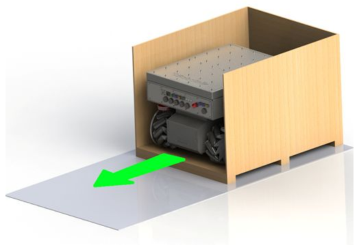
2. Dissasemble top cover
(x6 unscrew)



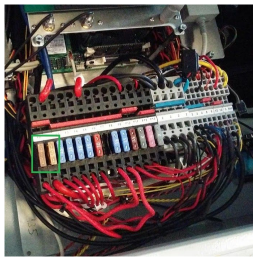
3. Free the robot
(Cut internal bridles)



4. Disassemble front cover
(x6 unscrew)



6. Robot manual extraction
Step 2 - Push the robot out of the box.



9. Install fuses
Open top cover and install fuses F0 - F1 - F2 in the robot.

2.2. Start-up sequence

To turn on the robot you need to interact with the rear panel described in section 3.1.1 and follow the sequence detailed below:

1. Turn the *MAIN POWER SELECTOR (2)* to power on all the components of the robot
2. Press the *CPU POWER* BUTTON (3) to turn on the mobile platform cpu.

After this sequence, the wheels will perform a **homing process**. Then, you will be able to command the robot with the joystick.

With this sequence the default program of the mobile platform will be launched and it can be accessed by typing in a terminal of the robot or through a connection through ssh:

```
screen -r bringup
```

2.3. Pad Teleoperation

If the startup sequence has been executed in the correct order, the pad will light up blue after pressing the start button. Then, the robot will be ready to be teleoperated.

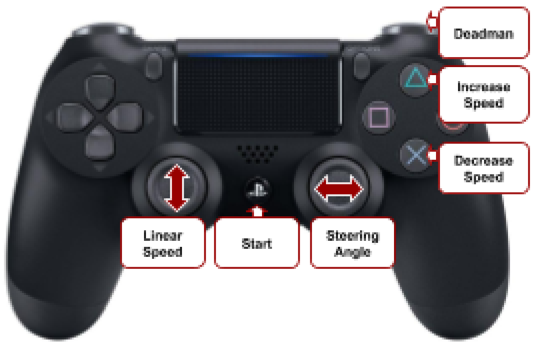


Figure 1. PS4 Pad Operation Mode

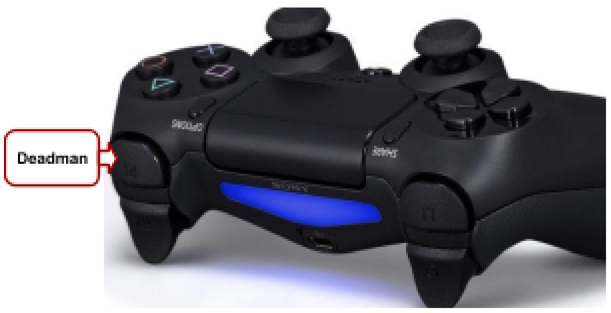


Figure 2. PS4 Pad Operation Mode

NOTES

- If the Bluetooth connection is lost, the robot will detect this situation and will STOP for safety.
- If Deadman button is NOT pressed, the robot will NOT move.
- Sometimes it is necessary to restart the pad to link it again with the robot controller. To restart it, press the “Start” button until the light shuts down and afterwards proceed with the startup sequence.

2.4. Power-off sequence

To power off the robot you need to interact with the rear panel described in section 3.1.1 and follow the sequence detailed below:

1. Press the *CPU POWER* BUTTON (3) to turn off the cpu of the mobile platform.

2. Turn off the *MAIN POWER SELECTOR* (2) to cut the power off.

3. Robot components

This point describes the main parts of the RB-Sherpa mobile robot. Every main piece includes a little description of the mechanical component that composes it.

3.1. External elements

The next figure shows the main external parts of the default robot:

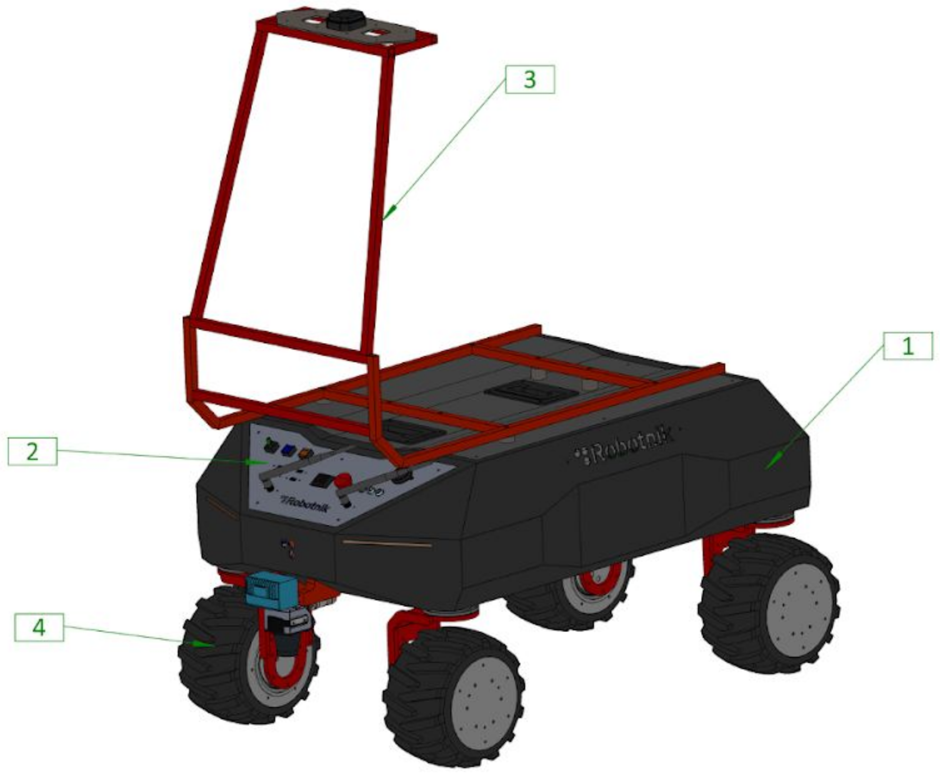


Figure 3. Main external elements of RB-Sherpa

1. Main chassis
2. Front panel
3. Sensor support
4. Traction and stearing wheel

3.1.1. Front panel

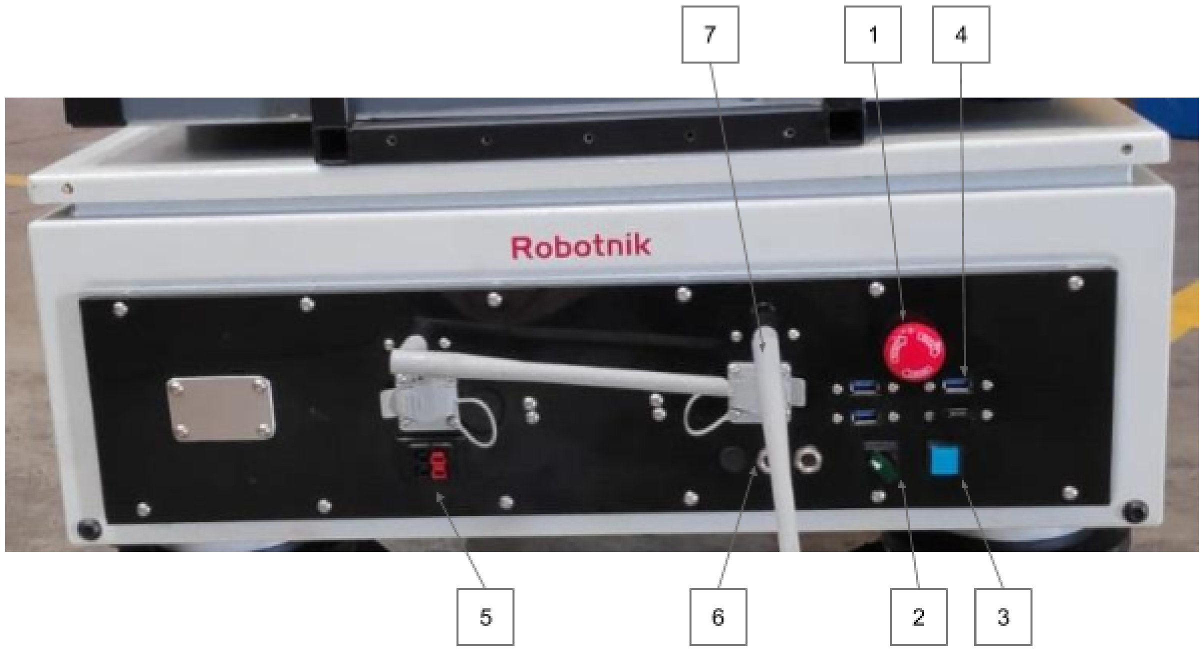


Figure 4 - RB-Sherpa front panel

1. **EMERGENCY STOP BUTTON**: disables the power and stops the robot. To move the robot, the EMERGENCY BUTTON (red) must be pulled out.
2. **MAIN POWER SELECTOR**: powers on/off the whole robot. It has a green light indicator.
3. **CPU POWER BUTTON:** green indicator/switch: turns on/off the main/base cpu.
4. **USB 2.0 PORTS**: two usb port connected to the main/base cpu.
5. **CHARGER CONNECTOR:** to connect the manual battery charger.
6. **POWER CONNECTORS**: 12 VDC, BAT. Intended to power external devices and protected with fuse.
7. **ROUTER ANTENNAS**

3.1.2. Motor wheels

The robot has 4 motor wheels with the same configuration. Each wheel is composed by a motor block and a detachable wheel. The motor block has a 500w 8 poles brushless motor with Hall Effect sensor and a reduction gear box, all of them hold by an aluminum cover. These kinds of motors have a much longer life expectancy and a higher efficiency than brushed motors.

The cable must be kept in good condition, and protected if the external cover is damaged. There are three 48V power wires, two 5VDC power and three Hall Effect signals. If they are short-circuit, the motor and the driver can be damaged.
Also 4 traction motors, the RB-Sherpa has two steering motors, these motors are located in the interior of the chassis. The control of these motor is carried out by means of an encoder and 2 inductive sensors. The rotation range of the wheel is +/- 150º. This rotation range can be decreased and adjusted. Consult the manufacturer to perform this operation.

3.1.7. Other devices

RB-Sherpa is prepared to integrate other kinds of sensors not mentioned, depending on the solution needed:

PTZ Camera. Pan Tilt Zoom Camera is a camera that is capable of remote control of direction and zoom. It is used in teleoperations or to capture images.

3D Laser. These lasers are used to perform robot location actions, detection of obstacles over long distances and analysis of the environment.

Ultrasonic sensor. Ultrasonic sensors measure distances based on transmitting and receiving ultrasonic signals. They can stably detect transparent or complex-shaped targets.

GPS. The GPS is an electronic system that uses satellites to localize the geoposition of the robot outdoors.

There are different sensors not mentioned that can be installed in RB-Sherpa. If you need more specific information about some model or function, contact with Robotnik support department to request it.

3.2. Internal components

The next figure shows the main internal elements of the default robot:

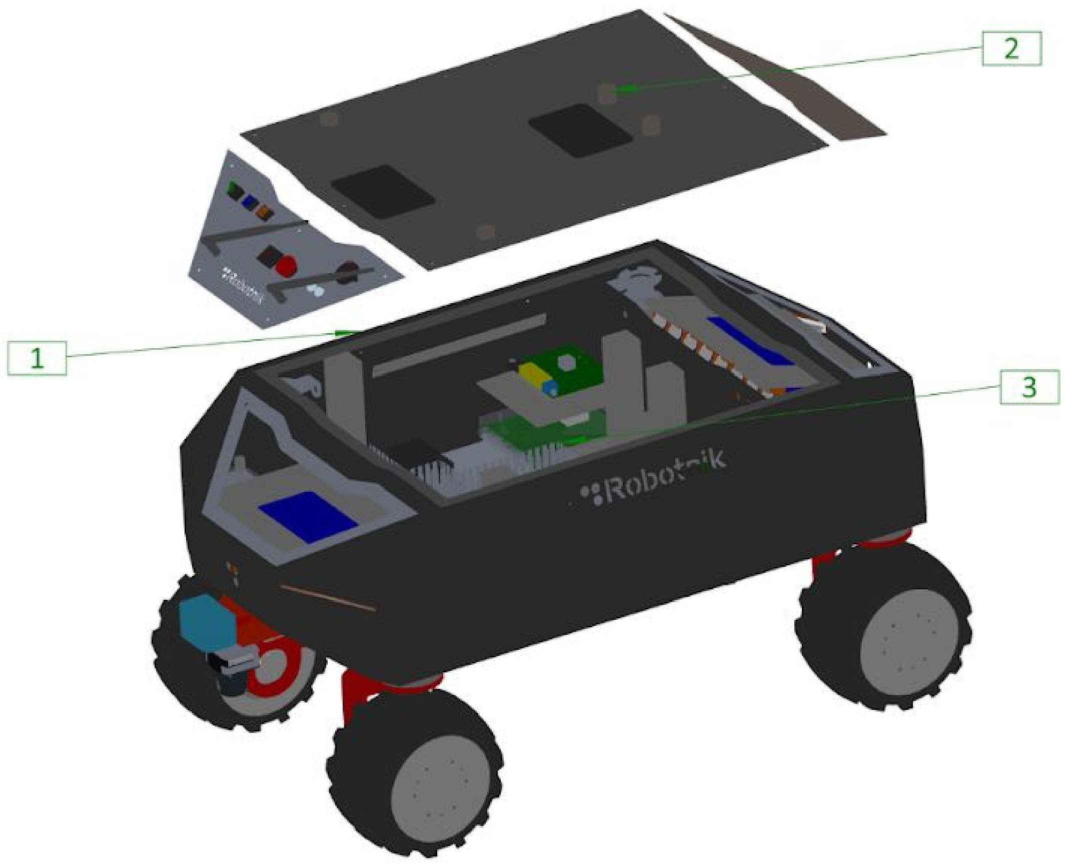
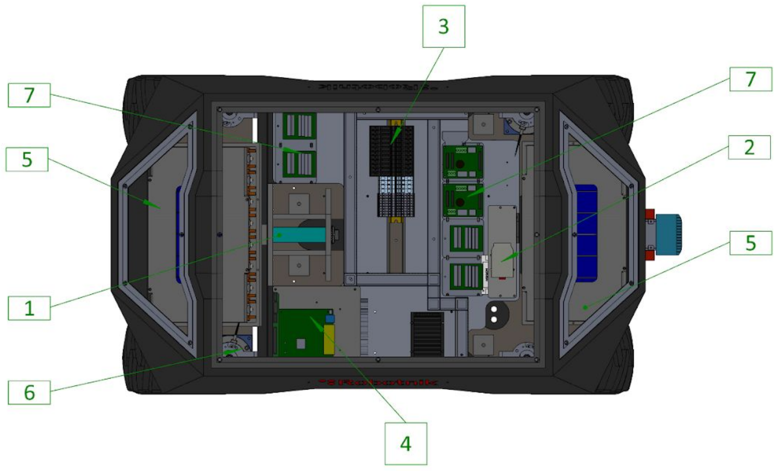


Figure 6. Main internal elements of RB-Sherpa

1. Top gate of electronic box
2. Locks
3. Internal electronic components

3.2.1. Electronic board

The next figure shows the component of the default RB-Sherpa electronic board:



1. IMU
2. PEAK can
3. Terminas, fuses & DCDC
4. Platform CPU & router
5. Platform battery
6. Motor drivers (x6)
7. Steering motors (x2)

Figure 7. Components of RB-Sherpa electronic board

Of all these components, the following parts should be highlighted:

CPU. This CPU is in charge of executing ROS.

Motor driver. There are four drivers which are programmed with specific settings to control each motor wheel of the platform.

IMU. Inertial measurement unit is an electronic device that measures and reports a robot's specific force, angular rate and magnetic field.

There are different models of these components that can be installed in RB-Sherpa. If you need more specific information about these, contact with Robotnik support department to request it.

3.2.2. Battery

RB-Sherpa have installed a 48V@15A LiFePO4 battery pack. It is composed of 3.2V LiFePO4 cells and a protection circuit module. With this battery technology the robot is able to operate between 3 and 10 hours, depending on the robot movements.

The robot circuit is powered when the general switch is ON. The control DC/DC converter, which supplies power to the different devices of control, is powered at the same time.

There are three important values for the levels of the battery (if you check with a tester or via software). These values are:

- Charging voltage: Is the level in the robot during the charging process. Normal values are around 58.4V (16 cells per 3.65 V each cell) Note that after charging, the voltage will drop fast to "Full charge voltage" even without using the battery).
- Full charge voltage: Is the level in the robot with the battery full charge. Normal values are around 53.6V (16 cells per 3.35 V each cell).
- Full discharge voltage: Is the level in the robot with the battery empty. Normal values are around 46.4V (16 cells per 2.9V each cell).

IMPORTANT: The cells can drop up to 2.5V, but it is not recommended. When the battery level is very low, the BMS can shut down easily if any power peak is required.

3.3. Accessories

3.3.1. Dualshock gamepad

The dualshock gamepad is a default robot item and it is used for the manual movements of the robot. Its receiver is located inside the robot and connected to one USB port of the computer.

Figure 8. Dualshock controller

The two joysticks are used for direction, traction and there are important controls like the speed level buttons that select between 10 speed ranges.

If you need more specific information about the DualShock, contact with Robotnik support department and request it.

3.3.2. Manual charger

The manual charger is a default robot device and it is used for charging the robot manually. Depending on the robot battery, the manual charger have different charging power.

Figure 10. Manual charger

For charging the batteries manually, there is a connector at the back panel of the robot where the charger can be connected. It is a direct connection to the battery, so the general ON/OFF switch doesn't affect the charging. It is possible to charge the robot and keep working at the same time.

If you need more specific information about the different charger models, contact with Robotnik support department and request it.

4. Software architecture

4.1. ROS architecture

ROS is an open-source meta-operating system for your robot that provides inter-process message passing services (IPC) in a network.

ROS is also an integrated framework for robots that provides:

- Hardware abstraction layer
- Low level device control
- Robot common functionality (simulation, vision, kinematics, navigation, etc.)
- IPC
- Package and stack management

ROS provides libraries and tools to ease the development of robot software in a multi-computer system.

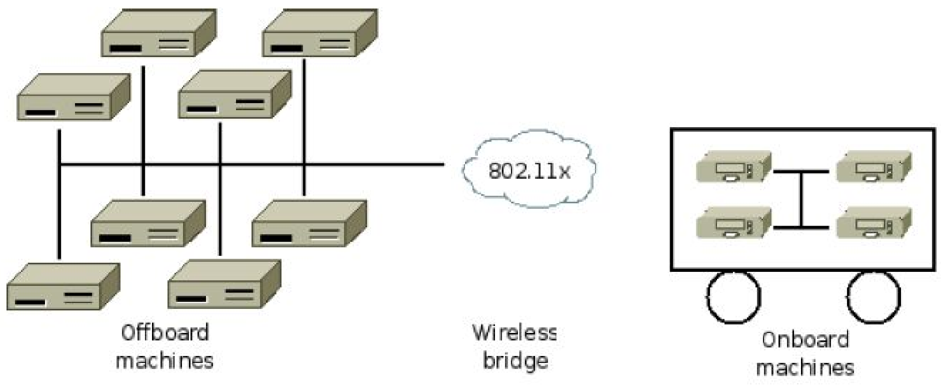


Figure 17. ROS Multi-computer schema

ROS offers a framework to solve common research and development needs of robot systems:

- Cooperation of different research groups
- Proven functionality
- Easy and robust access to robotics hardware

One of the main objectives of ROS is the code reusability. This objective is fulfilled by a large and growing community of developers that share their results worldwide, and by the inclusion of other robot frameworks (ROS integrates Player/Stage, Orocos, etc.) and other open-source components (like Gazebo or Openrave).

ROS integrates additional development tools like rviz (simulation of complete robots and environments with maps), rxgraph (visualization of node interconnection), rosbag (extreme useful data logging utility), etc.

For detailed systems descriptions, tutorials, and a really important number of stacks and packages, please visit www.ros.org.

4.2. Robot packages

This section gives an overview of the packages included in the workspace of the robot (commonly the folder catkin_ws located in the home folder of the robot). The RB-Sherpa has its specific packages but it also uses some of the Summit-XL packages. The packages are grouped in different folders within the workspace:

4.2.1. Imu

It contains the drivers for different types of IMUs and filter-related packages.

4.2.1.1. Imu_manager

Package that manages imu calibration and its state. It will check the current calibration state of the IMU and in case it needs a recalibration, it will run the calibration method. To check calibration and run calibration method, the robot must be stopped. During the calibration, the robot will not be able to move.

4.2.1.2. imu_tools

IMU-related filters and visualizers. The stack contains:

- imu_filter_madgwick: a filter which fuses angular velocities, accelerations, and (optionally) magnetic readings from a generic IMU device into an orientation.Based on the work of http://www.x-io.co.uk/open-source-imu-and-ahrs-algorithms/
- imu_complementary_filter: a filter which fuses angular velocities, accelerations, and (optionally) magnetic readings from a generic IMU device into an orientation quaternion using a novel approach based on a complementary fusion. Based on the work of http://www.mdpi.com/1424-8220/15/8/19302
- rviz_imu_plugin: a plugin for rviz which displays IMU messages

4.2.1.3. Myahrs_driver

This is a driver package for the WITHROBOT's myAHRS+.

4.2.1.4. vectornav

A ROS node for VectorNav INS & GPS devices. This package provides a sensor_msg interface for the VN100, 200, & 300 devices. Simply configure your launch files to point to the serial port of the device and you can use rostopic to quickly get running.

4.2.2. Manipulation

Contains the ROS drivers for the arms and grippers used for manipulation

4.2.2.1. egh_gripper_common

- egh_gripper_description: Contains the Schunk EGH gripper's URDF, meshes, etc.

4.2.2.2. egh_gripper_controller

Package to control the Schunk EGH gripper. At the moment the gripper can only be controlled via the UR's digital input/output interface.

4.2.2.3. universal_robot

This repository contains the bringup launch both for the real arm and simulation, and the moveit package.

4.2.2.4. Universal_Robots_ROS_Driver

This repository contains the new ur_robot_driver and a couple of helper packages, such as:

- controller_stopper: A small external tool that stops and restarts ros-controllers based on the robot's state. This can be helpful when the robot is in a state where it won't accept commands sent from ROS.
- ur_calibration: Package around extracting and converting a robot's factory calibration information to make it usable by the robot_description.
- ur_controllers: Controllers introduced with this driver, such as speed-scaling-aware controllers.
- ur_robot_driver: The actual driver package.

4.2.3. Navigation

4.2.3.1. eband_local_planner

The eband_local_planner package implements a plugin to the base_local_planner for the move_base 2D navigation system. It implements the Elastic Band method on the SE2 manifold.

4.2.3.2. marker_mapping

Package to map visual markers and use them to initialize the robot position with amcl localization

4.2.3.3. poi_manager

A ROS node to manage the points of interest in a map. It reads a list of tagged positions from a YAML file and offers services to obtain the list and update it.

4.2.3.4. robot_localization_utils

The robot_localization_utils package provides nodes and helpers to use robot_localization packages with Robotnik robots and in order to calibrate IMU sensors as the PIXHAWK.

4.2.4. Robot

4.2.4.1. rbkairos_common

Common packages of the RB-KAIROS: URDF description of the RB-KAIROS, platform messages and other files for simulation.

- rbkairos_10_moveit_config: This package contains all the moveit configuration files in order to use the rbkairos with a UR10 arm.

- rbkairos_description: This package contains the URDF and mesh files used to create the description of the robot

- rbkairos_ur5_egh_moveit_config: This package contains all the moveit configuration files in order to use the rbkairos with a UR5 arm along with a egh gripper.

4.2.4.2. robotnik_base_hw

ROS RobotHW component based on ros_control architecture. This component is compatible with most of Robotnik's motor hardware.

4.2.4.3. summit_xl_common

- summit_xl_localization: This package contains several algorithms used in the localization of the SUMMIT-XL robot, both at the local level (odometry) and global level (maps, gps).

- summit_xl_navigation: This package contains the configuration to use the SUMMIT-XL robot with the ROS Navigation stack.

- summit_xl_pad: This package contains the node that subscribes to /joy messages and publishes command messages for the robot platform including speed level control. The node allows you to load different types of joysticks (PS4, PS3, Logitech, Thrustmaster). New models can be easily added by creating new .yaml files.
- summit_xl_perception: This package contains all the launch and configuration files needed to execute the nodes that manage the perception of QR Codes and detection of laser reflectors that allow the robot detect objects of interest such as the charger station.

- summit_xl_robot_local_control: This package contains all the launch and configuration files needed to execute the robot_local_control and launch the generic procedures costumed for summit_xl

4.2.4.4. summit_xl_robot

- summit_xl_bringup: This package contains launch files to start the robot. The most relevant file is *launch/summit_xl_complete.launch*, that is called from the .bashrc at system startup. The file can be configured to start the available devices, usually: lasers, cameras, rgbd devices, imus, gps, etc. Integrates several test launch files to test the components individually. This file defines the robot serial number as a parameter.

- summit_xl_controller: Controller plugin compatible with ros_control architecture. This component is in charge of setting & reading the robot motor joints.

4.2.4.5. rbsherpa_common

- rbsherpa_description: This package contains the URDF and mesh files used to create the description of the robot

- rbsherpa_navigation: This package contains the configuration to use the sherpa robot with the ROS Navigation stack

- rbsherpal_robot_local_control: This package contains all the launch and configuration files needed to execute the robot_local_control and launch the generic procedures costumed for sherpa

- rbsherpa_control: This package contains all the configuration files related to robot's controller

4.2.5. Sensors

4.2.5.1. axis_camera

This ROS package provides an Axis network camera driver, written in Python.

4.2.5.2. Ce30c

This ROS package provides a driver for the Benewake ce30c, which is a solid-state LiDAR.

4.2.5.3. Ira_laser_tools

These nodes are meant to provide some utils for lasers, like listening to different laser scan sources and merge them in a single scan or generate virtual laser scans from a pointcloud.

4.2.5.4. robotnik_sensors

Robotnik standard sensors description.

4.2.5.5. ros_rslidar

ROS driver for RS-Lidar sensor

4.2.5.6. sick_s300_laser

This package implements a driver for the Sick S300 Safety laser scanners.

4.2.5.7. ublox

The ublox package provides support for u-blox (http://www.u-blox.com) GPS receivers.

4.2.6. Simulation

4.2.6.1. Gazebo_ros_pkg

Wrappers, tools and additional API's for using ROS with the Gazebo simulator.

4.2.6.2. rbkairos_sim

This package contains launch files to start the RB-KAIROS in simulation.

4.2.6.3. robotnik_base_hw_sim

This package is intended to simulate the Robotnik base hw common in most of the platforms

4.2.6.4. robotnik_gazebo_models

This package contains models and different environments for Gazebo simulations

4.2.6.5. summit_xl_sim

Packages for the simulation of the Summit XL

4.2.7. Web

4.2.7.1. map_nav_manager

A ROS node to manage processes for performing mapping, navigation and monitoring from a web-based user interface.

4.2.7.2. Robotnik_hmi

A ROS package that launches the different processes to show the information in a web interface.

4.2.7.3. Robotnik_RMS

A ROS node to create a queue of actions to execute a mission.

4.2.7.4. rostful

ROStful is a lightweight web server for making ROS services, topics, and actions available as RESTful web services. It also provides a client proxy to expose a web service locally over ROS.

4.2.7.5. web_video_server

Node to stream camera image through a web video server

4.2.8. Others

4.2.8.1. rcomponent

Basic component that defines the state machine common to most components developed by Robotnik.

4.2.8.2. robotnik_modbus_io

This component is intended to set and read digital I/O from a modbus server.

4.2.8.3. robotnik_msgs

Definition of msgs and services used by some Robotnik's packages

4.2.8.4. system_monitor

This package provides a communication driver for various autopilots with MAVLink communication protocol such as Pixhawk.

5. Configuration

5.1. Users & passwords

In this section you can find all the user and password configurations of the internal devices.

Mobile platform CPU

User	sherpa
Password	R0b0tn1K

Router

User	admin
Password	R0b0tn1K

5.2. Network

The RB-Sherpa has its own WiFi network for all internal components communication and remote control. In the following diagram there are all network connected devices with their IP addresses:

Device	**IP address**
Mobile platform CPU	192.168.0.200
Router	192.168.0.1
Front Laser	192.168.0.10
Benewake	192.168.0.20

5.3. Devices

The devices connected to the robot have two ways of connection: via network or via USB. If they are connected through the robot network, they are configured with a static IP to have them identified (as shown in 5.2. Network). If connected by USB, udev rules are usually defined to identify devices with names. The names and IPs of the devices are defined as environment variables, as specified in the following section.

5.4. Basic robot params

In the home folder of the robot there is a file called *robot_params.env*. All in this file the main configurable parameters of the robot are defined. In the following table there is a brief explanation of all the possibilities.

Parameter	**Description**	**Possible value**
ROBOT_ID	Indicates the name of the robot. This is the name of the namespace under all the nodes will be working. This is	- robot

	also used as the prefix of all the subcomponents.	
ROBOT_XACRO	Indicates the path where the xacro file is; inside the robot folder in robot_description.	*sherpa_standalone.urdf.xacro*
ROBOT_HAS_FRONT_LASER	Indicates if the robot has a front laser.	- true - false
ROBOT_FRONT_LASER_MODEL	Indicates the model of the front laser that the robot is using. The model is the name of the launch file.	- sick_s300 - sick_tim561 - hokuyo_ug01 - hokuyo_ust
ROBOT_FRONT_LASER_PORT	Port of the front laser if its connected using USB.	/dev/ttyFRONT_LASER
ROBOT_FRONT_LASER_IP	IP of the front laser if its connected using Ethernet.	192.168.0.10
ROBOT_HAS_REAR_LASER	Indicates if the robot has a rear laser.	- true - false
ROBOT_FRONT_RGBD_CAMERA_ID	Front camera id to identify in the bus	- #1 (only for orbbec astra) - serial number
ROBOT_HAS_REAR_RGBD_CAMERA	Indicates if the robot has rear camera	- true - false
ROBOT_REAR_RGBD_CAMERA_ID	Rear camera id to identify in the bus	- #1 (only for orbbec astra) - serial number
ROBOT_HAS_ARM	Indicates if the robot has an arm.	- true - false
ROBOT_BASE_HW_CHARGING_CURRENT_OFFSET	Offset applied to the current measured by the sensor to consider whether or not the robot is charging.	Decimal value (i.e: 0.1)

5.5. ROS server

The ROS Master lets the nodes be able to find each other, exchange messages, or invoke services. Connecting a remote computer to the ROS Master of the robot, allows the visualization of all messages that are being published on the server, as well as interaction with the robot through actions and services.

5.5.1. Connecting with a remote PC

Some metapackages of the software have to be present in the remote computer in order to operate the robot remotely. Some of the most useful tools are rviz (ROS Visualization tool) and rqt. To be able to use these in the remote machine, you will need to:

1. Install ROS

2. Create a workspace. More info here. Copy the same src folder as found on the robot.

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
```

3. Download dependencies and compile the stacks:

```
cd ~/catkin_ws
rosdep install --from-path src --ignore-src -y -r
catkin build
```

Once ROS is installed in your machine you will need to configure this machine to be able to connect to the RB-Sherpa ROS_MASTER.

1. Add the robot hostname in your /etc/hosts file:

```
192.168.0.200   Hostname
```

2. Add your computer hostname in the robot /etc/hosts file:

```
YourIp   YourHostname
```

3. Start the RB-Sherpa robot, connect to its wifi network, open a terminal and type:

```
export ROS_MASTER_URI=http://Hostname:11311
cd ~/catkin_ws
source devel/setup.bash
```

4. If everything worked, you should be able to see all the topics published by the robot and the services offered by the robot controller:

```
rostopic list
rosservice list
```

5. You can view the values published by the nodes with rostopic echo, e.g.:

```
rostopic echo /tf
```

At this stage you can have a first look at the robot with

```
rqt
```

and

```
rosrun rviz rviz
```

5.6. Bringup

The main program of the mobile platform is launched by default and accessible by typing in a terminal:

```
screen -r bringup
```

The file with the program is named *summit_xl_complete.launch* can be found in the folder *launch* in the package *summit_xl_bringup*.

This program execution contains the following nodes:

Node	Description
summit_xl_state_robot	Is responsible for publishing the urdf of the robot and its joints.
summit_xl_pad	Is responsible for allowing the remote control of the robot with the ps4 remote.
robotnik_base_hw	This node contains the driver of the robot motors.
summit_xl_control	This node is responsible for controlling the commands to the motors in order to achieve a desired pose.
axis	Executes the program needed to visualize the axis camera in the ros environment.
pixhawk	Controls the imu used.
rl_utils	Provides nodes and helpers to use robot_localization packages with Robotnik robots in order to calibrate the IMU.
map_nav_manager	Publishes the website that contributes to visualize the generated maps and executing nodes.
front_laser_node/rear_laser_node	Provide the scan of the two lasers of the robot.
3d_laser_to_2d	Executes a pointcloud to laserscan node and a laser filter to transform the RGBD camera pointcloud to a 2D laser scan.
robotnik_modbus_io	This node is responsible for communicating with the modbus I/O signals.
safety_module_node	Provides the needed topics and services to manage the safety of the mobile platform.
rgbd_camera	Nodes responsible for the rgbd camera.

5.7. Controller

The motors of the wheels of the RB-Sherpa are controlled through drivers that accept the commands from the ROS software of the robot. This one is called robotnik_base_hw and sends joint speeds to the wheels according to the linear and angular velocity desired for the platform.

5.7.1. robotnik_base_hw

As mentioned in the previous section, the robotnik_base_hw node is launched in the main platform program by default when turning on the CPU.
The configurable parameters of this node are located in the folder *config* in the package *summit_xl_bringup* . The file is named *robot_robotnik_base_hw.yaml* and contains:

Parameter	Description
port	Port where the can driver is in the cpu
diameter_wheel	Wheel diameter
gearbox_ratio	Ratio of the gearbox
motors_encoder	True if the motors have an encoder
motors_encoder_factor	Factor of the encoder used
joint_name	Names of the joints of the wheels of the mobile platform
joint_can_id	Rach driver has its own id that will be related to the joints of the wheels
joint_type	Specification of the type of control used: velocity or position
joint_home_offset	If there is an offset in the initial position it should be specified here
joint_spin	Specifies the turning direction of the motors

WARNING

- These parameters shouldn't be changed during the normal use of the robot because they are configured with the motor drivers in the manufacturing process.
- The incorrect configuration can lead to unexpected behavior

5.8. Mapping

The RB-Sherpa robot is able to map its environment thanks to the lasers it is equipped with. Having the bringup launched, execute:

```
ROS_NAMESPACE=robot roslaunch summit_xl_localization slam_gmapping.launch
```

Once you have the desired map, execute the following command to save it:

```
ROS_NAMESPACE=robot roslaunch summit_xl_localization map_saver.launch
```

It will save your in the user home directory as map.yaml and map.png, but you can custom the destination folder and the name of the map as follows:

```
ROS_NAMESPACE=robot roslaunch summit_xl_localization map_saver.launch
map_name:=my_map destination_folder:=/path/to/map/folder
```

WARNING: The map is empty

- Make sure the rest of the robot software is running. Try moving it with the remote control.
- Check if the laser is publishing data. (*rostopic echo /robot/front_laser/scan*)
- Make sure that slam_gmapping node is subscribing to the right scan topic. (*rostopic info /robot/front_laser/scan*)

If the robot has map_nav_manager, the mapping process can be started through the services provided by the web (7.1. Map Nav Manager).

5.9. Localization

First of all you have to load the map you are going to work with. To do this, execute:

```
ROS_NAMESPACE=robot roslaunch summit_xl_localization map_server.launch
prefix:=robot
```

The available maps can be found in the *summit_xl_localization* package, in the *maps* folder. If you want to change the map that is loaded:

```
ROS_NAMESPACE=robot roslaunch summit_xl_localization map_server.launch
prefix:=robot map_file:=file_to_load
```

WARNING: The map is not loaded

- Check that the map is saved in the *summit_xl_localization* package, in the *maps* folder.
- Make sure that the map filename is correct.
- If you have changed the name of the map file, you will need to edit the map_file.yaml too. This yaml file contains a reference to the pgm file.

Once you have the map loaded, you need to run the localization program:

```
ROS_NAMESPACE=robot roslaunch summit_xl_localization amcl.launch
```

WARNING: The localization is not working

- Check that the map_server is running. (*rosnode list | grep map_server*)
- Check if the laser is publishing data. (*rostopic echo /robot/fornt_laser/scan*)
- Make sure that localization node is subscribing to the right scan topic. (*rostopic info /robot/fornt_laser/scan*)

The localization process can be started through the services provided by the map_nav_manager web (7.1. Map Nav Manager).

5.10. Navigation

RB-Sherpa can perform autonomous moves

5.10.1. Move_base

The move_base package provides an implementation of an action that, given a goal in the map, will attempt to reach it with a mobile base. The move_base node links together a global and local planner to accomplish its global navigation task. The local planner used is the Teb that will be detailed later.

The configurable parameters are located in several files in the *config* folder of the package *summit_xl_navigation*. The main file related with move_base is named *move_base_params.yaml* .

Parameter	Description	Default value
controller_frequency	The rate in Hz at which to run the control loop and send velocity commands to the base.	20
controller_patience	How long the controller will wait in seconds without receiving a valid control before space-clearing operations are performed	15
planner_frequency	The rate in Hz at which to run the global planning loop. If the frequency is set to 0.0, the global planner will only run when a new goal is received or the local planner reports that its path is blocked	0
planner_patience	How long the planner will wait in seconds in an attempt to find a valid plan before space-clearing operations are performed.	5

conservative_reset_dist	The distance away from the robot in meters beyond which obstacles will be cleared from the costmap when attempting to clear space in the map. Note, this parameter is only used when the default recovery behaviors are used for move_base.	0
recovery_behavior_enabled	Whether or not to enable the move_base recovery behaviors to attempt to clear out space.	true
clearing_rotation_allowed	Determines whether or not the robot will attempt an in-place rotation when attempting to clear out space. Note: This parameter is only used when the default recovery behaviors are in use, meaning the user has not set the recovery_behaviors parameter to anything custom.	true
shutdown_costmaps	Determines whether or not to shutdown the costmaps of the node when move_base is in an inactive state	false
oscillation_timeout	How long in seconds to allow for oscillation before executing recovery behaviors. A value of 0.0 corresponds to an infinite timeout.	0
oscillation_distance	How far in meters the robot must move to be considered not to be oscillating. Moving this far resets the timer counting up to the ~oscillation_timeout	0.5
max_planning_retries	How many times to allow for planning retries before executing recovery behaviors. A value of -1.0 corresponds to an infinite retries.	1

WARNING: The navigation is not working

- Check that the map_server is running. (*rosnode list | grep map_server*)
- Check that the localization is running. (rosnode list | grep amcl)

The navigation process can be started through the services provided by the map_nav_manager web (7.1. Map Nav Manager).

5.10.1.1.1 TEB

The initial trajectory generated by a global planner is optimized during runtime w.r.t. minimizing the trajectory execution time (time-optimal objective), separation from obstacles and compliance with kinodynamic constraints such as satisfying maximum velocities and accelerations.

The configurable parameters are stored in summit_xl_navigation/config/:

- Maximum velocities and accelerations
- Minimum obstacle distance
- Goal tolerances

Further information can be found here.

5.11. Perception

This part of the software manages the detection of QR codes and laser reflectors to create frames of interest which the robot can use to perform actions related to them.

Node	Description
ar_locator	**Node that allow to identify qr markers**
two_tag_laser_locator	Node that allow to identify laser reflectors separated by a configured distance.
laser_filter	Node that filters by intensity the readings of the laser to identify correctly the reflectors.

5.11.1 Environment variables

There are some environment variables that allows configuring the perception features of the robot.

SCREEN

Parameter	Description	Default value
ROBOT_RUN_NAVIGATION	Flag to launch nodes used to docking to frames	- true - false

Parameter	Description	Possible value
ROBOT_RUN_AR_LOCATOR	Flag to launch qr detector.	- true - false
ROBOT_RUN_REFLECTOR_LOCATOR	Flag to launch reflector detector.	- true - false
ROBOT_LASER_MODEL_INTENSITY_FILTER	Name of the intensity filter used to detect the reflectors.	- default - hokuyo_utm - hokuyo_ust

6. Basic functionality

This section shows some functionalities of the robot such as teleoperation using the remote controller, as well as the visualization of the status of the motors.

6.1. Robot controller

6.1.1. ROS control

RobotnikBaseHW registers a collection of HardwareInterface joints for all motors to work with the controller manager. To learn more about ROS Control you can visit the following ROS Wiki page.

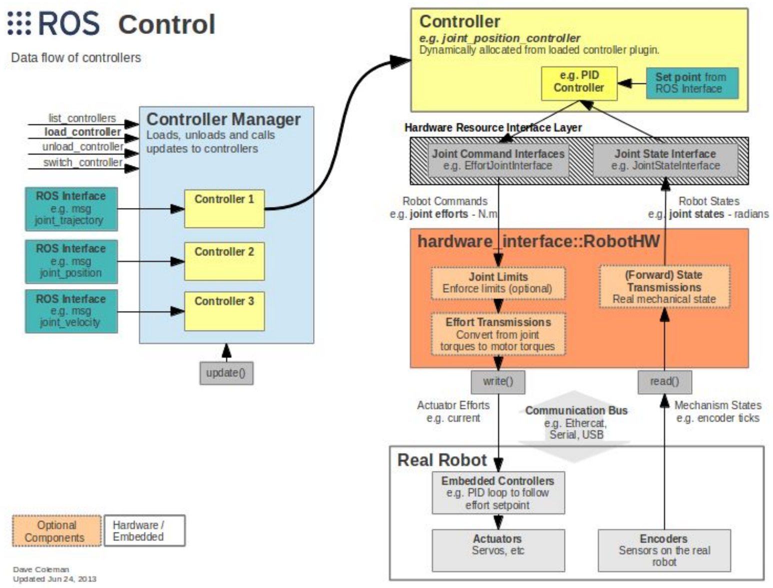

ROS Control
Data flow of controllers
list_controllers
load_controller
unload_controller
switch_controller
Controller Manager
Loads, unloads and calls updates to controllers
ROS Interface e.g. msg joint_trajectory
ROS Interface e.g. msg joint_position
ROS Interface e.g. msg joint_velocity
Controller 1
Controller 2
Controller 3
update()
Controller
e.g. joint_position_controller
Dynamically allocated from loaded controller plugin.
e.g. PID Controller
Set point from ROS Interface
Hardware Resource Interface Layer
Joint Command Interfaces e.g. EffortJointInterface
Joint State Interface e.g. JointStateInterface
Robot Commands e.g. joint efforts - N.m
Robot States e.g. joint states - radians
hardware_interface::RobotHW
Joint Limits Enforce limits (optional)
Effort Transmissions Convert from joint torques to motor torques
(Forward) State Transmissions Real mechanical state
write()
read()
Actuator Efforts e.g. current
Communication Bus e.g. Ethercat, Serial, USB
Mechanism States e.g. encoder ticks
Real Robot
Embedded Controllers e.g. PID loop to follow effort setpoint
Actuators Servos, etc
Encoders Sensors on the real robot
Optional Components
Hardware / Embedded
Dave Coleman
Updated Jun 24, 2013

Figure 21. ROS Control schema

6.1.2. Base hardware

Robotnik Base Hardware provides an interface between ROS Control and the real drivers.

6.1.2.1. Status

The node robotnik_base_hw publishes information in topics about the general state of the robot related to the motors and the battery.

6.1.2.1.1. Drives status

Information about the motor drives can be obtained through the topic *status* of the *robotnik_base_hw* node.

```
rostopic echo /robot/robotnik_base_hw/status
```

The messages published in that topic is structured as follows:

```
string state
string status
string communicationstatus
string statusword
string driveflags
string[] activestatusword
string[] activedriveflags
int32 digitaloutputs
int32 digitalinputs
float32 averagecurrent
float32[] analoginputs
```

Output example

```
name: [robot_front_left_wheel_joint, robot_back_left_wheel_joint,
robot_front_right_wheel_joint, robot_back_right_wheel_joint]
can_id: [1, 2, 3, 4]
motor_status:
  -
```

```
      state: "READY"
      status: "READY"
      communicationstatus: "OPERATIONAL"
      statusword: "0001010001100000"
      driveflags:
"00100000000010000000000000001000000100000011000000000000000000000100011"
      activestatusword: [SW_FAULT, SW_QUICK_STOP, UNKNOWN,
SW_TARGET_REACHED]
      activedriveflags: [SHUNT_ENABLED, UNDER_VOLTAGE, COMMANDED_DISABLE,
USER_UNDER_VOLTAGE, ZERO_VELOCITY,
  AT_COMMAND, USER_QUICKSTOP, UNKNOWN, UNKNOWN]
      digitaloutputs: 0
      digitalinputs: 31
      averagecurrent: 0.0
      analoginputs: []
...
```

6.1.2.1.2. Battery

Information about the battery can be obtained through the topic *battery/data* of the *robotnik_base_hw* node.

```
rostopic echo /robot/robotnik_base_hw/battery/data
```

The message published in that topic is structured as follows:

```
float32 voltage               # in volts
float32 level                 # in %
uint32 time_remaining         # in minutes
uint32 time_charging          # in minutes
bool is_charging              # true when connected
```

Output example:

```
voltage: 51.8464698792
level: 32.0
time_remaining: 197
```

```
time_charging: 0
is_charging: False
```

6.1.2.1.3. Docking status

Information about the docking contact status can be obtained through the topic *battery/docking_status* of the *robotnik_base_hw* node.

```
rostopic echo /robot/robotnik_base_hw/battery/docking_status
```

The message published in that topic is structured as follows:

```
# Modes of operation:
# no docking station contacts
string MODE_DISABLED=disabled
# Unattended relay detection & activation with no inputs/outputs feedback.
Done by the hw
string MODE_AUTO_HW=automatic_hw
# Unattended relay detection & activation with inputs/outputs feedback.
Done by the sw. It can be forced as it was in manual_sw mode
string MODE_AUTO_SW=automatic_sw
# Unattended relay detection & and manual activation of the charging relay
string MODE_MANUAL_SW=manual_sw


string operation_mode

float32 battery_current        # current flow in Amperes
bool contact_relay_status      # shows if there's contact with the charger.
True if there’s contact between the robot and docking station
bool charger_relay_status      # shows if the relay for the charge is active
or not. True if the charger relay is active and the battery charge is
enabled through the contacts
```

Output example:

```
header:
  seq: 17463
  stamp:
    secs: 1567437248
    nsecs: 964432933
  frame_id: ''
status:
  operation_mode: "automatic_sw"
  battery_current: 3.1135559082
  contact_relay_status: False
  charger_relay_status: False
```

6.1.2.2. Operations

The base_hw node has several services available

Set Charger Relay

Switches on/off the charger relay manually.

```
rosservice call /robot/robotnik_base_hw/set_charger_relay "data: false"
success: True
message: "ok"
```

Reset hw

Stop the communication with the motors and start it again.

```
rosservice call /robot/robotnik_base_hw/reset_hw "{}"
```

Send to home

Triggers all motors homing.

```
rosservice call /robot/robotnik_base_hw/send_to_home "{}"
```

Set digital output

Set motor digital output

```
rosservice call /robot/robotnik_base_hw/set_digital_output "output: 0
value: false"
```

6.1.3. Kinematics controller

6.1.3.1. Status

The robot controller has some topics that show information about the status of the robot.

To see if the controller is enabled you can consult the topic *enable*. This topic only shows a boolean value. In some situations the controller is disabled, for example when auto-calibration is performed.

```
rostopic echo /robot/robotnik_base_control/enabled
```

Doing *echo* of the *in_motion* topic you can check if the robot is moving:

```
rostopic echo /robot/robotnik_base_control/in_motion
```

The odometry calculation is also published by the robot controller. This topic uses the standard Odometry msg:

```
rostopic echo /robot/robotnik_base_control/odom
```

6.1.3.2. Operations

The robot controller allows you to enable or disable robot control through a service:

```
rosservice call /robot/robotnik_base_control/enable "value: false"
```

When the robot has control disabled, it will not be able to move using velocity commands.

6.1.3.3. Homing

The robot needs to perform a homing operation every time the robot is powered on in order to know its motor position. This operation is only done by front wheels. Once this procedure is finished the robot can be moved normally.

6.2. Command the base

6.2.1. Twist mux

This node provides a multiplexer for Twist messages. It takes N input twist topics and outputs the messages from a single one. For selecting the topic they are prioritized based on their priority, the messages timeout and M input lock topics that can inhibit one input twist topic. The RB-Sherpa twist_mux configuration is illustrated in the diagram below:

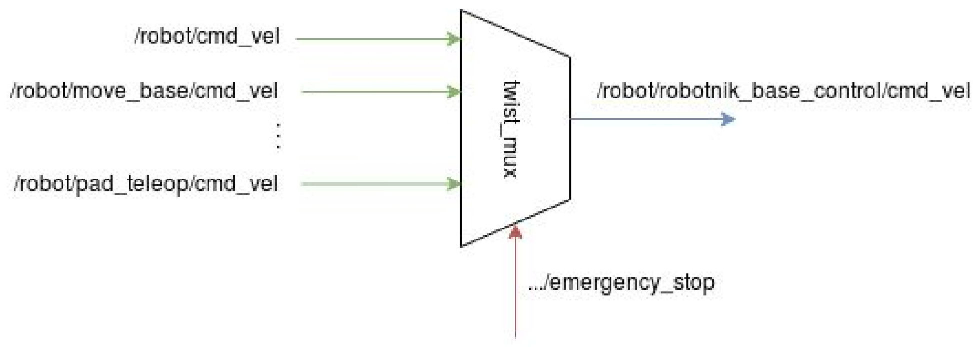


Figure 22. Twist Mux schema

At the end, any speed command goes through the multiplexer and is republished in */robot/robotnik_base_control/cmd_vel*, which is the topic from that the robot controller reads the speed commands. The configuration file of the twist_mux is located in the summit_xl_control package.

6.2.2. Gamepad

When the pad is linked to the robot's PC (by pressing the PS button when using PS4 controller), the data transmitted by the remote control can be read in the Ubuntu system as an input. To check the correct reading of data:

```
jstest /dev/input/jsX
```

⚠️ X can be a number from 0 to 9.

Depending on the number of input devices connected to the robot, the number associated with the controller may change. To avoid trying to read from a device other than the remote pad, a udev rule has been created that allows the controller to be identified as an input device with the name *js_base*. Therefore, with the remote controller connected, the information can also be read running:

```
jstest /dev/input/js_base
```

6.2.2.1. Configuration

The control is provided with a default configuration that works correctly. The configuration is shown earlier in this manual. (2.3. Pad Teleoperation)

In the summit_xl_pad package, in the config folder, you will find the configuration files for the commands. They associate the number of axes and buttons (of the topic */robot/joy*) to the desired actions.

⚠️ If you want to make any changes, it is recommended to back up the default remote control configuration files. Changing this setting does not ensure proper system operation.

6.2.2.2. Battery charge

To charge the pad, connect it using the supplied micro USB cable. The remote controller can be connected to the robot itself (rear panel) or other power system.

6.2.2.3. How to use it

All the information published by the remote control (in */dev/input/js_base*) is published in ROS in the *joy* topic:

```
rostopic echo /robot/joy
```

The summit_xl_pad node takes the information published by the remote pad and publishes speed commands. This commands can be read in *pad_teleop/cmd_vel* topic:

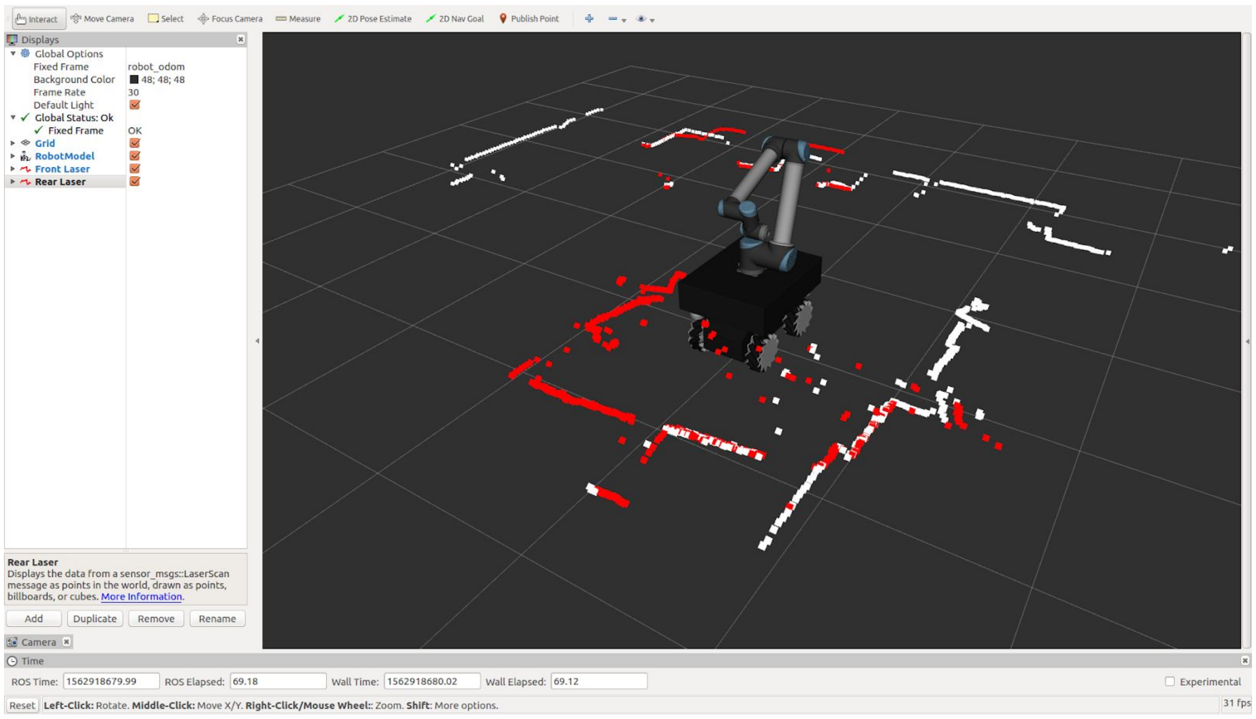
```
rostopic echo /robot/pad_teleop/cmd_vel
```

To send speed commands to the robot, press the deadman button and move the left axis to give linear speed and the right axis to give angular speed.

6.3. RVIZ

RVIZ is a 3D visualization tool for ROS. To launch it:

```
rviz
```



Global Options	
Fixed Frame	robot_odom
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
Front Laser	✓
Rear Laser	✓

Rear Laser
Displays the data from a sensor_msgs::LaserScan message as points in the world, drawn as points, billboards, or cubes. More Information.

ROS Time: 1562918679.99 ROS Elapsed: 69.18 Wall Time: 1562918680.02 Wall Elapsed: 69.12 ☐ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options. 31 fps

Figure 23. RVIZ

RVIZ allows us to visualize the data provided by the sensors and to command the robot using plugins. For more information about RVIZ please check ROS Wiki.

7. Advanced functionality

This section will show some advanced functionalities of the robot, such as tele-operation and mapping through the user interface, arm movement through ROS or using the UR interface itself, etc.

7.1. Human Machine Interface (HMI)

The HMI is an internal web application running locally on the robot. You can access this website through the browser using the url http://robot_ip/robotnik_hmi. If you are connected to the robot's router then you should access like this: http://192.168.0.200/robotnik_hmi

This application is a tool to monitor the robot status, control and perform localization and navigation actions in a simpler way.

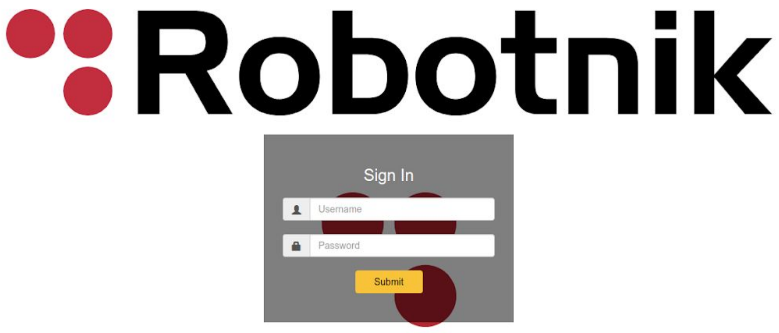


Figure 23. HMI Log in

To login, use "admin" as user and "4dm1N" as password. It is recommended to change the password when accessed for the first time.

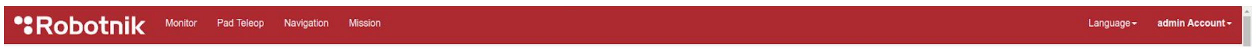


Figure 24. HMI Tab bars

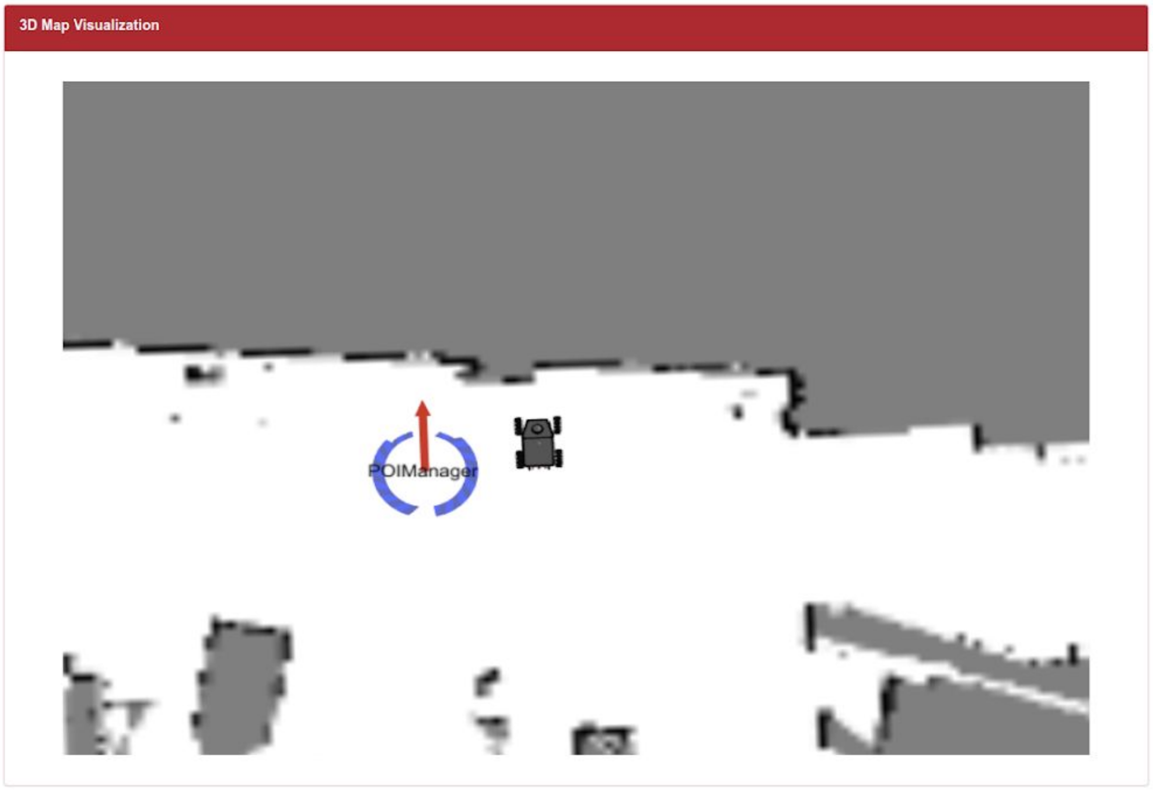
The web has a toolbar to change between the robot monitor (monitor), control the robot through a joystick connected to the computer (Pad Teleop), send or save points to the map (Navigation) and create, delete or start missions (Mission). To the right of the toolbar, you can select the language of the web page or administrate the current account.

7.1.1 Monitor

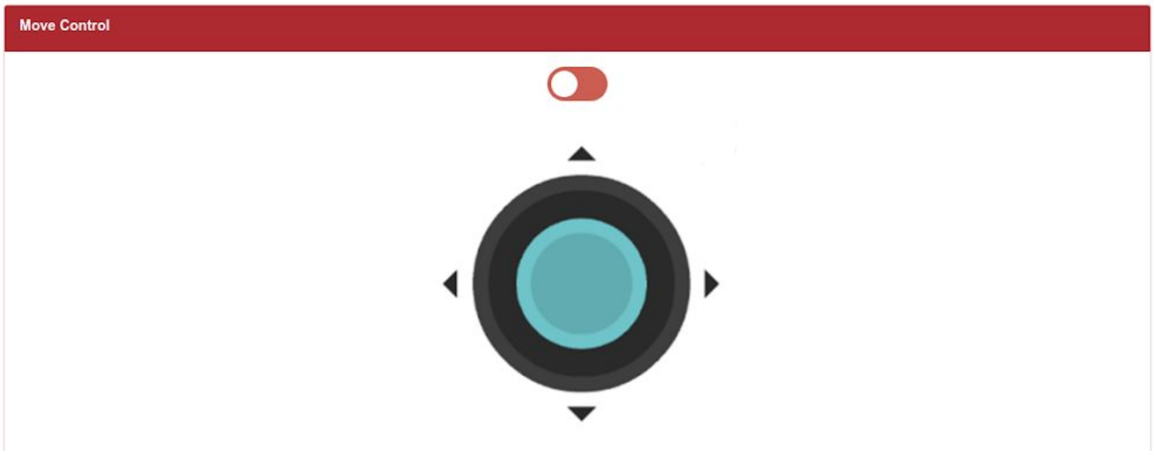
The monitor shows information about the robot, this information is separated into two groups:

7.1.1.1 Control panel

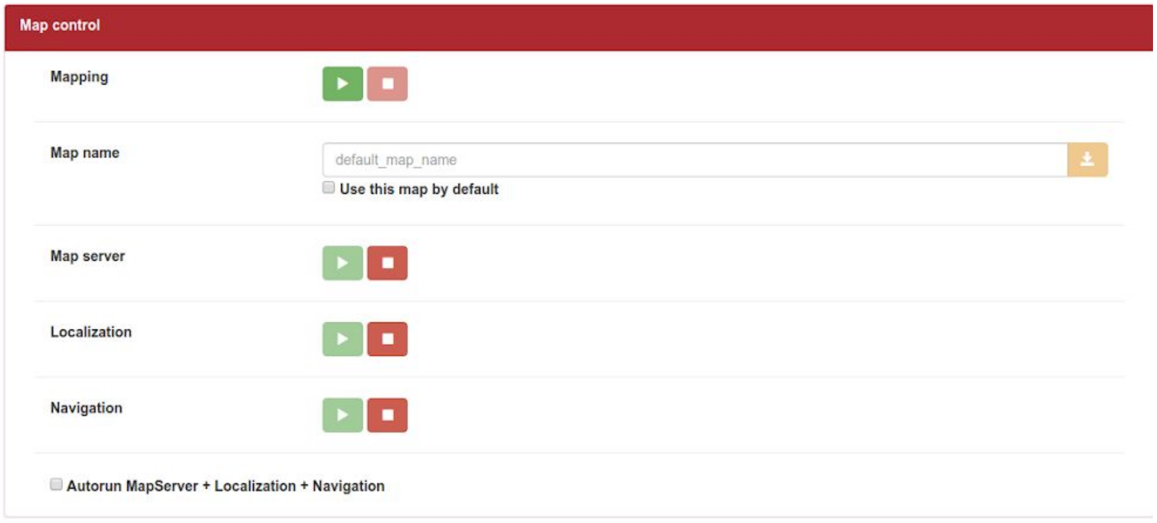
1. **3D Map Visualization**: In this section you can see the position of the robot inside the map. The markers or points of interest (POI Manager) can be rotated or moved through the map by clicking on it and moving it. If you press the right button of the map on the POI Manager, a contextual menu appears with some options that allows you to move the robot, localize, create points of interest, etc.



2. **Move Control**: This joystick allows you to control the robot through the map interface



3. **Map control**: Control panel to start/stop navigation, localization or mapping procedure.



3.1. **Mapping**: If mapping is activated on the panel of 3D Map Visualization, the robot will start to build the map by using the defined SLAM algorithm (gmapping by default). Once you have the desired map, you can save it. To do this, you must enter the name of the map and press the save button. You can set this map as the default by checking the checkbox before saving.

⚠ Mapping is not compatible with Map Server nor Localization modules.

3.2. **Map Server**: After saving a map, it is possible to load it in order to localize the robot. To start the map server, set the name in the “Map name” field and press the start button.

⚠ Map Server is not compatible with the Mapping module.

3.3. **Localization**: Start or stop the localization module. Once started, it is necessary to initialize the robot position by using the web interactive marker.

⚠ Localization is not compatible with the Mapping module. Localization requires Map Server running.

3.4. **Navigation**: You can send the robot to any point of the map using the POI Manager or the POI markers.

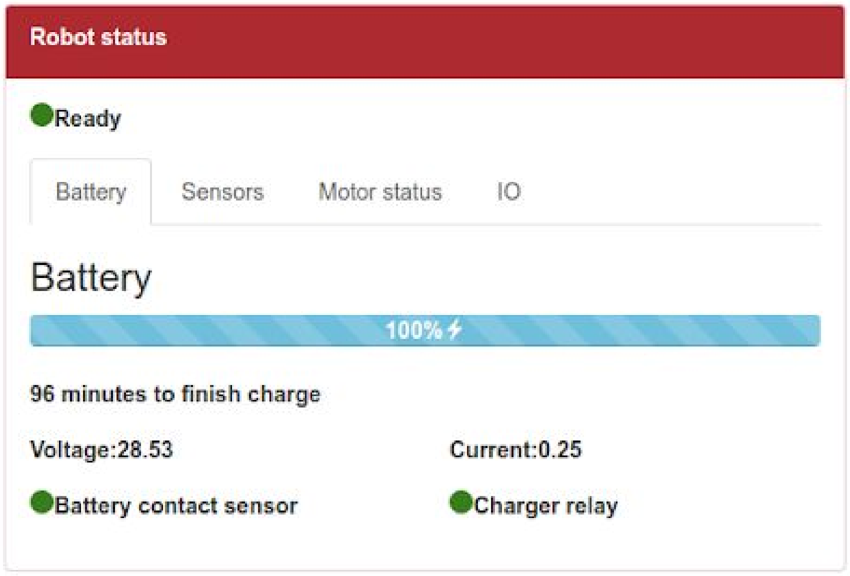
⚠ Navigation requires Mapping or Localization+Map Server modules running.

You can make the Map Server, Localization and Navigation start automatically with the complete launch of map_nav_manager by checking the Autorun checkbox.

4. **Output map Control**: Allows you to see information about launched navigation nodes in the robot.

7.1.1.2 Status summary

1. **Robot status**: Shows if the robot is ready to be operated.
   1.1. **Battery**: Shows battery information as battery level, remaining minutes, voltage, current and if the battery is detecting the contact with the docking station through "Battery contact sensor" indicator. Also shows if the charger relay is enabled (and the contact pins are connected to the battery).



Battery

100%

96 minutes to finish charge

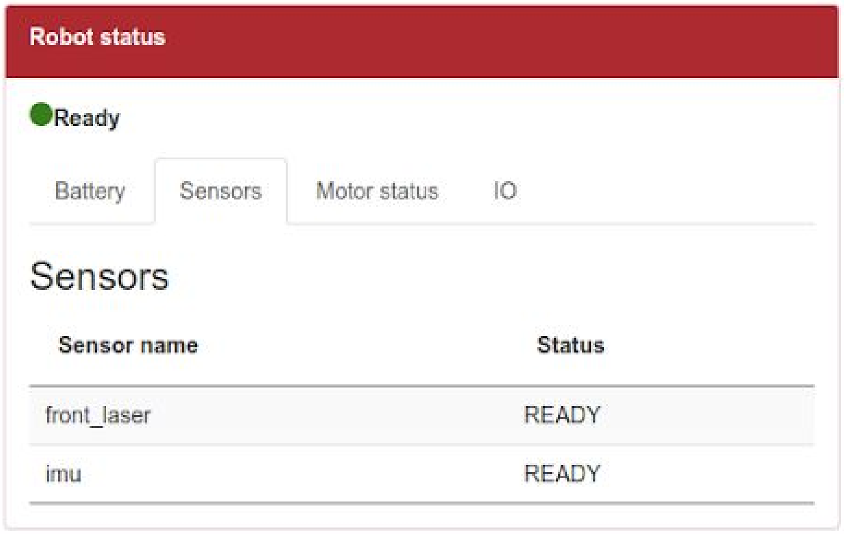
Voltage:28.53

Current:0.25

● Battery contact sensor

● Charger relay

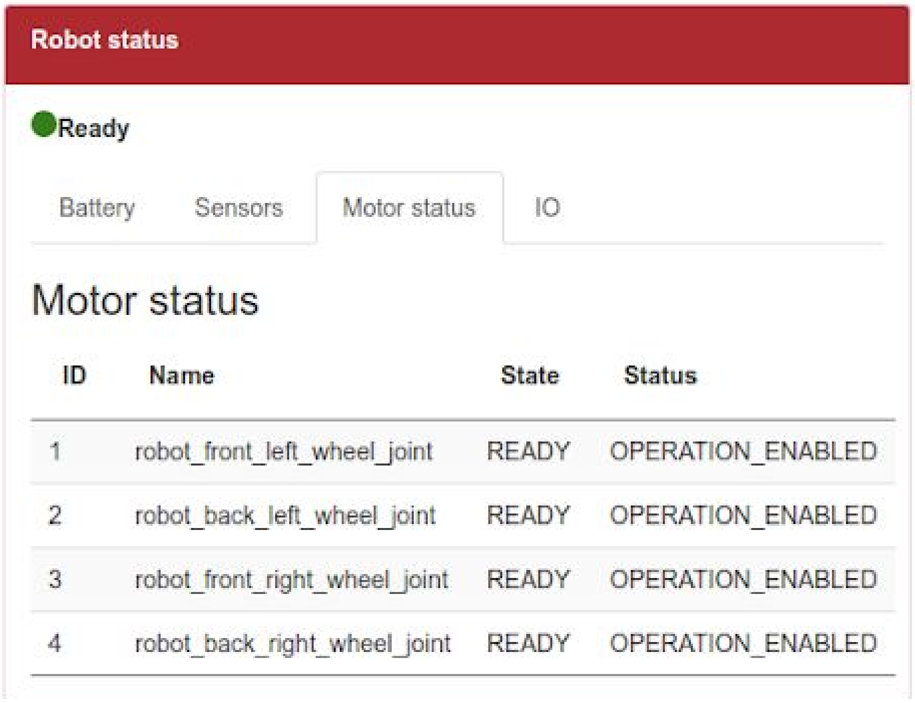
   1.2. **Sensors**: Shows the status of the sensors in the robot.



Sensors

Sensor name	Status
front_laser	READY
imu	READY

1.3. **Motor Status**: Shows the status of the motors. If everything is working in the right way, the state should be in ready state. If something goes wrong or the emergency stop is pressed, the indicators will not be ready.

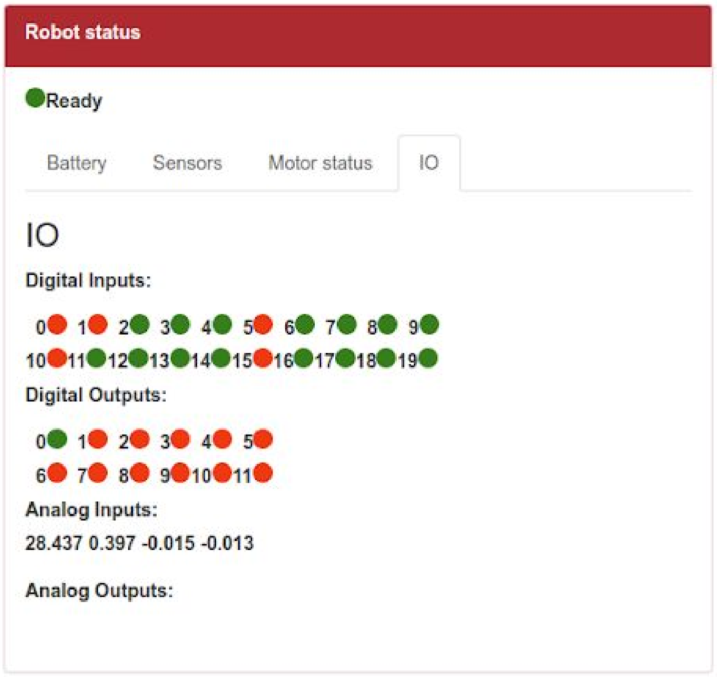

Robot status

🟢Ready

Motor status

ID	Name	State	Status
1	robot_front_left_wheel_joint	READY	OPERATION_ENABLED
2	robot_back_left_wheel_joint	READY	OPERATION_ENABLED
3	robot_front_right_wheel_joint	READY	OPERATION_ENABLED
4	robot_back_right_wheel_joint	READY	OPERATION_ENABLED

1.4. **IO**: Shows the raw information got by the IO sensors of the robot.


Robot status

🟢Ready

IO

Digital Inputs:

0🔴 1🔴 2🟢 3🟢 4🟢 5🔴 6🟢 7🟢 8🟢 9🟢
10🔴11🟢12🟢13🟢14🟢15🔴16🟢17🟢18🟢19🟢

Digital Outputs:

0🟢 1🔴 2🔴 3🔴 4🔴 5🔴
6🔴 7🔴 8🔴 9🔴10🔴11🔴

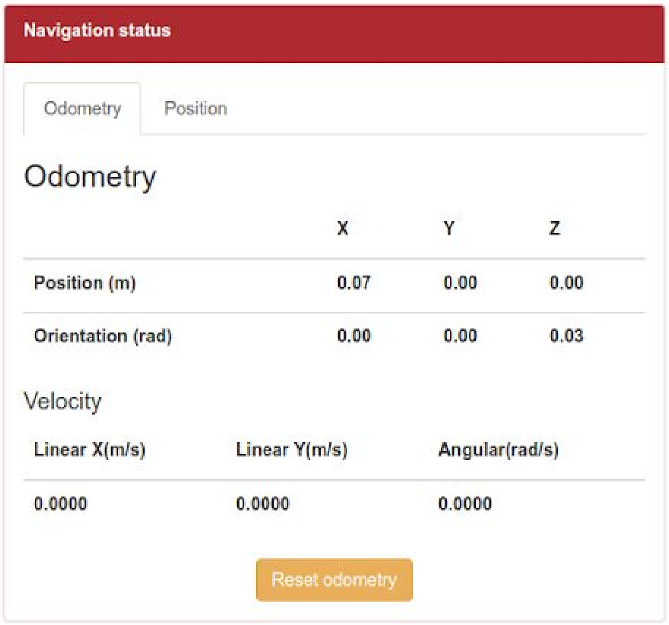
Analog Inputs:

28.437 0.397 -0.015 -0.013

Analog Outputs:

2. **Navigation status**: Shows information about the robot position.

   2.1.1. **Odometry**: Shows information about the robot odometry such as position or velocity.



Navigation status

Odometry | Position

Odometry

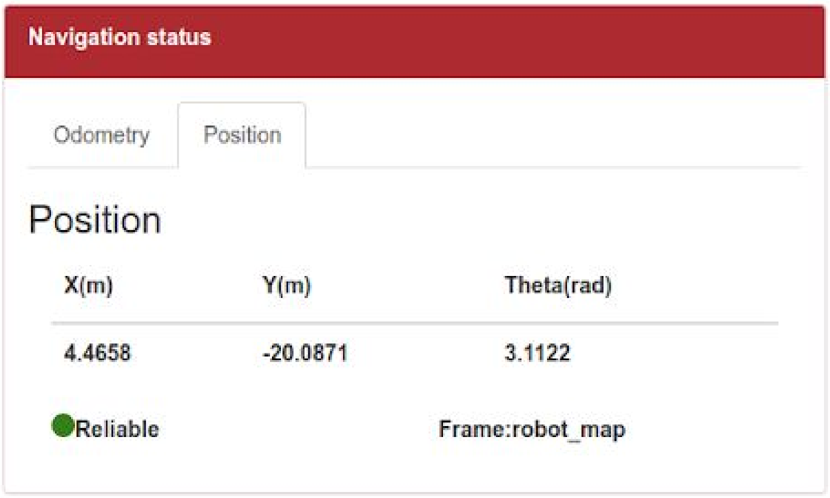
	X	Y	Z
Position (m)	0.07	0.00	0.00
Orientation (rad)	0.00	0.00	0.03

Velocity

Linear X(m/s)	Linear Y(m/s)	Angular(rad/s)
0.0000	0.0000	0.0000

Reset odometry

   2.1.2. **Position**: Shows information about the position of the robot, relative to the frame, and if the position is reliable (when using AMCL localization).



Navigation status

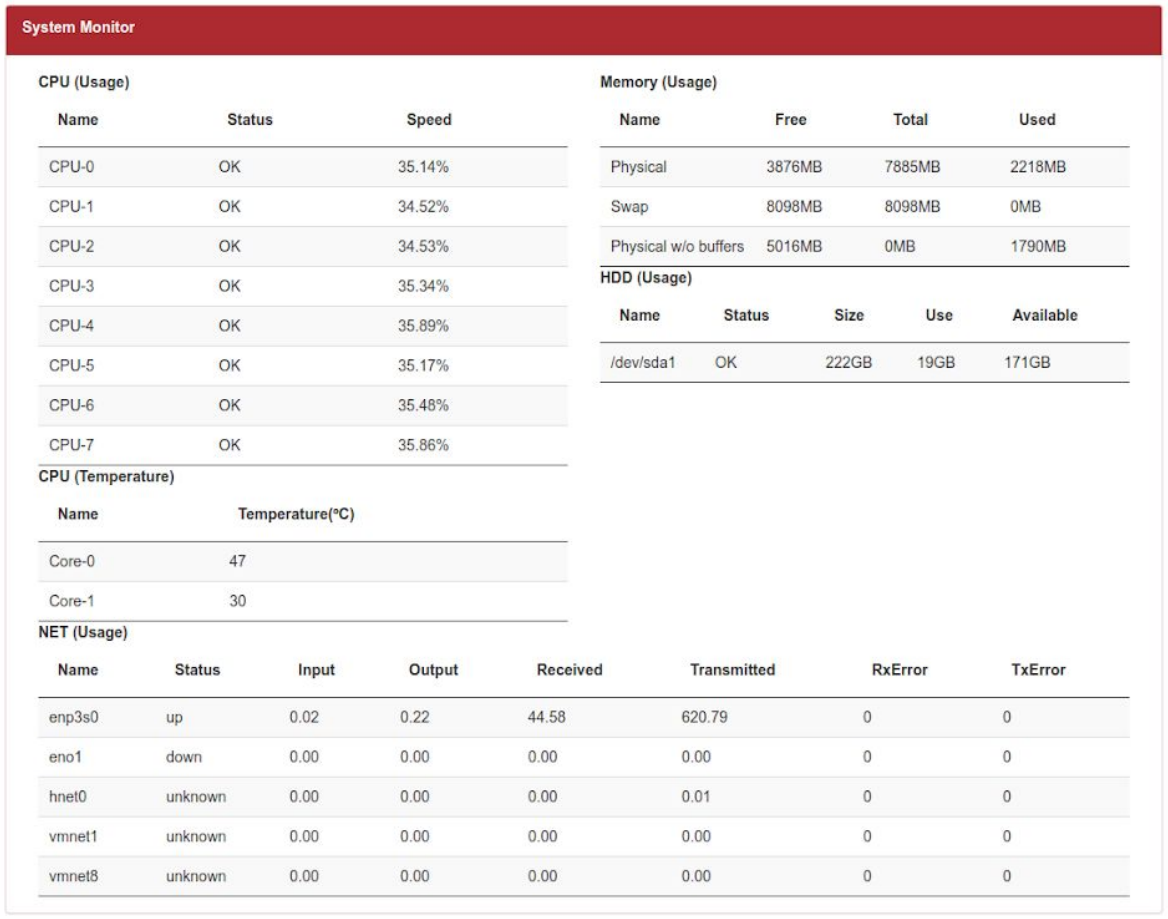
Odometry | Position

Position

X(m)	Y(m)	Theta(rad)
4.4658	-20.0871	3.1122

●Reliable Frame:robot_map

3. **System Monitor**: Shows general information from the computer of the robot.



System Monitor

CPU (Usage)

Name	Status	Speed
CPU-0	OK	35.14%
CPU-1	OK	34.52%
CPU-2	OK	34.53%
CPU-3	OK	35.34%
CPU-4	OK	35.89%
CPU-5	OK	35.17%
CPU-6	OK	35.48%
CPU-7	OK	35.86%

Memory (Usage)

Name	Free	Total	Used
Physical	3876MB	7885MB	2218MB
Swap	8098MB	8098MB	0MB
Physical w/o buffers	5016MB	0MB	1790MB

HDD (Usage)

Name	Status	Size	Use	Available
/dev/sda1	OK	222GB	19GB	171GB

CPU (Temperature)

Name	Temperature(ºC)
Core-0	47
Core-1	30

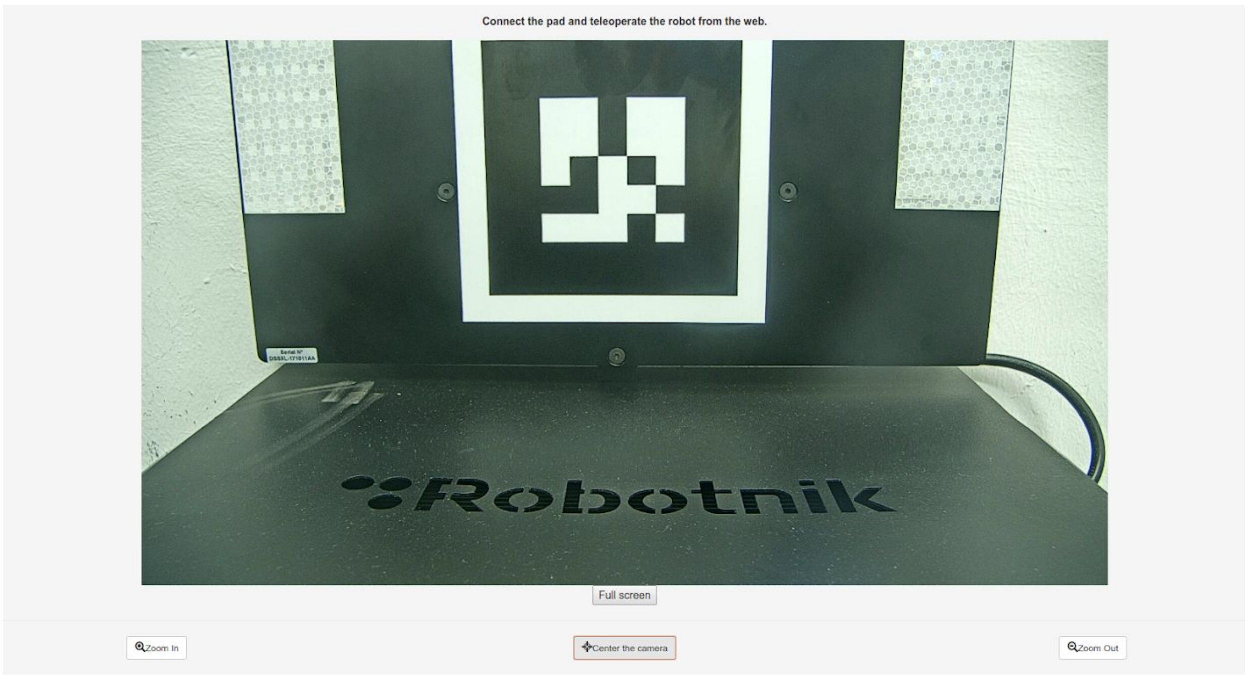
NET (Usage)

Name	Status	Input	Output	Received	Transmitted	RxError	TxError
enp3s0	up	0.02	0.22	44.58	620.79	0	0
eno1	down	0.00	0.00	0.00	0.00	0	0
hnet0	unknown	0.00	0.00	0.00	0.01	0	0
vmnet1	unknown	0.00	0.00	0.00	0.00	0	0
vmnet8	unknown	0.00	0.00	0.00	0.00	0	0

Figure 24. Map nav manager interface (II)

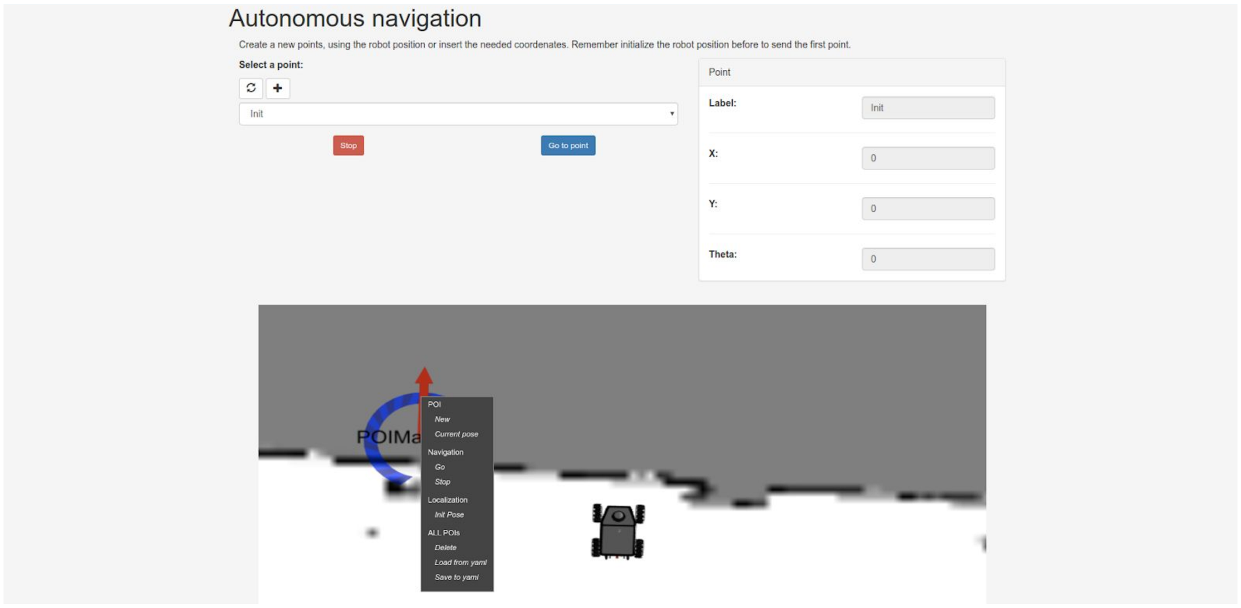
7.1.2 Pad Teleop

In this option, the robot can be controlled using a joystick connected to the computer and visualizing the video. If the robot has an axis camera, the direction of the camera can be controlled by clicking in the video view and moving the mouse.


Connect the pad and teleoperate the robot from the web.

7.1.2 Navigation

In this option, the pose of the robot can be initialized, send the robot to a point on the map and save the points in the map.


Autonomous navigation

Create a new points, using the robot position or insert the needed coordenates. Remember initialize the robot position before to send the first point.

Select a point:

Init

Point

Label: Init

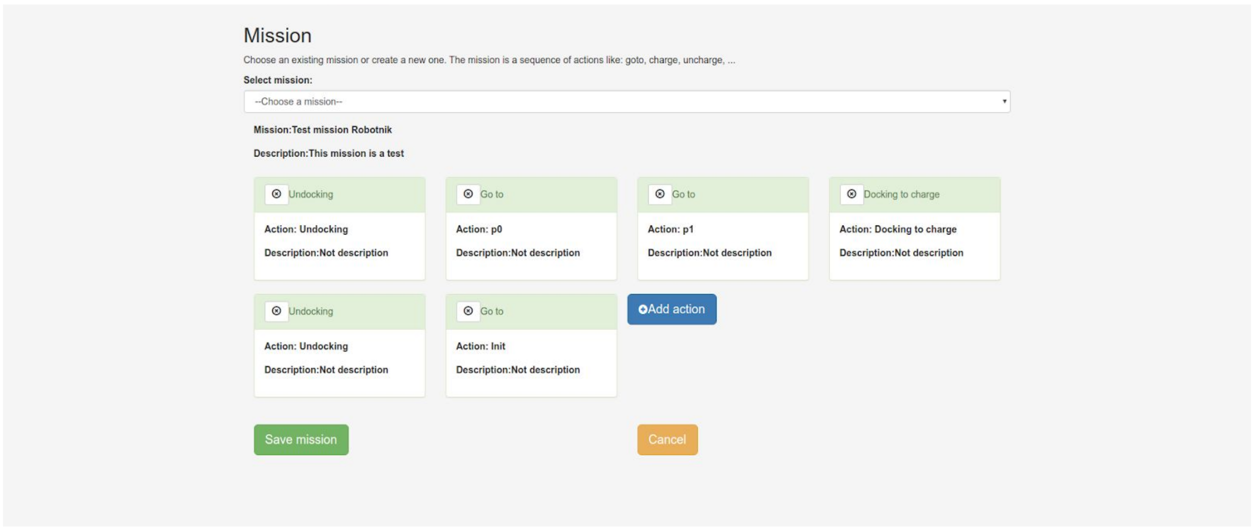
X: 0

Y: 0

Theta: 0

7.1.3 Mission

In this option, the missions can be created and assigned to a robot.



Mission

Choose an existing mission or create a new one. The mission is a sequence of actions like: goto, charge, uncharge, ...

Select mission:

--Choose a mission--

Mission:Test mission Robotnik

Description:This mission is a test

Undocking
Action: Undocking
Description:Not description

Go to
Action: p0
Description:Not description

Go to
Action: p1
Description:Not description

Docking to charge
Action: Docking to charge
Description:Not description

Undocking
Action: Undocking
Description:Not description

Go to
Action: Init
Description:Not description

7.2. Goto

You can send positions relative to the map to send the robot there. There are many ways to do this:

Actionlib:

```
rosrun actionlib axclient.py
/robot/move_base
```

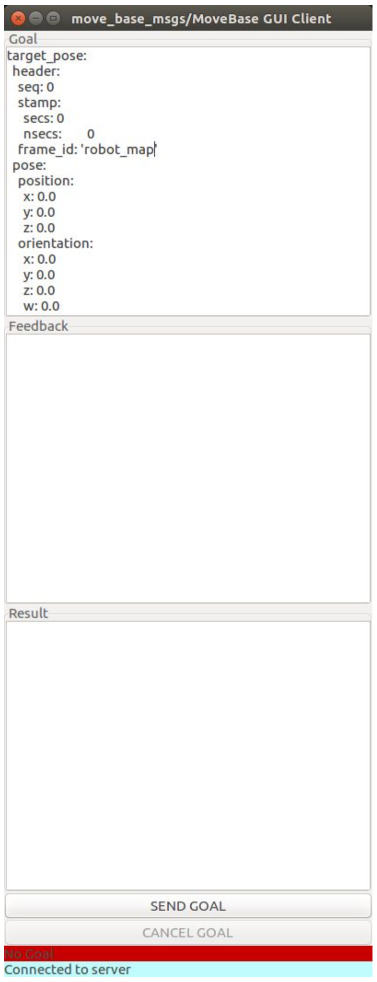
Command line

```
rostopic pub /robot/move_base/goal
TAB TAB
```

Map_nav_manager

Explained in section 7.1.2 Navigation

The behaviour of the robot when navigating depends on the parameters explained in 5.10.1. Move_base.



```
target_pose:
 header:
  seq: 0
  stamp:
   secs: 0
   nsecs:      0
  frame_id: 'robot_map'
 pose:
  position:
   x: 0.0
   y: 0.0
   z: 0.0
  orientation:
   x: 0.0
   y: 0.0
   z: 0.0
   w: 0.0
```

Figure 25. Move base axclient

7.3. Battery docking

The rbsherpa platform is able to dock to the charger thanks to the relative movements described earlier in this manual. (5.10.2.1. Docking) .
A QR marker, as shown below, is normally used to identify the charger station.

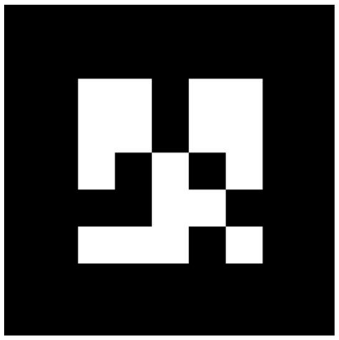


Figure 26. Docking QR

To dock the charger station, the *dock_frame* of the action should be *robot_docking_station_contact* and the *robot_dock_frame* should be *robot_base_docking_contact*.

7.4. Robot local control

Robotnik robot local control is a software that manages the overall state and behaviour of a robot. This is done by monitoring the health status of several ROS components, and proceeding according to this status and the actions the robot is required to perform.

It also provides a set of components that allow the execution of procedures. A procedure is identified by an id, which is set by the robot_local_control and returned to the one who added the procedure. After a procedure is added, its state can be queried using its id.

7.4.1. Configuration

The package is already configured with the correct parameters and they can be found in the package *summit_xl_robot_local_control*. However, there are some parameters that you can tune according to your needs and they can be found in the '*navigation.yaml*' file. These parameters and their meaning is explained in the following sections.

7.4.2. Procedures

This package contains a set of procedures. A procedure is an action that is performed by the robot. This action can be composed of one or multiple different actions.
These procedures have common services:

- **cancel**: It cancels the current procedure
- **query_state**: It asks the state of the current procedure

```
rosservice call
/robot/robot_local_control/NavigationComponent/Component/cancel "header:
id: -1
priority: 0
stamp:
  secs: 0
  nsecs: 0
name: ''"
```

7.4.2.1 GOTO

This component connects to an instance of ROS Move Base to send navigation goals to it.

7.4.2.1.1 Parameters

There are some parameters that the user can modify to adjust the behaviour of the procedure in order to perform a custom movement. These parameters are listed below:

Parameter	Description	Default value
desired_freq	Desired frequency of the node.	5.0
action_namespace	Namespace of the move_base server	move_base
global_frame	Global frame used to perform move_base goals	robot_map
base_frame	Base frame used to perform move_base goals	robot_base_footprint
has_safety_laser	Flag to indicate if the robot has safety laser. If it has, it will change the safety area before performing the move_base goal	false
default_yaw_tolerance		
yaw_toleramce	Move base goal yaw tolerance	0.1
xy_tolerance	Move base goal x and y tolerance	0.15
clear_costmaps_before_send_goal	Flag to clear costmap every time that a procedure is added	false
local_planner_namespace	Namespace of the local planner used by move_base	TEBLocalPlanner

7.4.2.1.2 ADD REQUEST

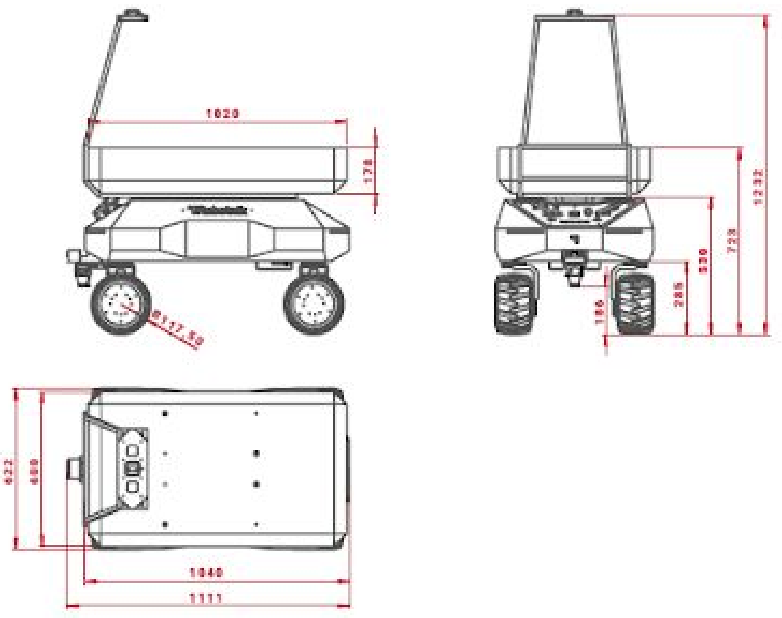
In order to initiate the charge procedure you have to call the following service:

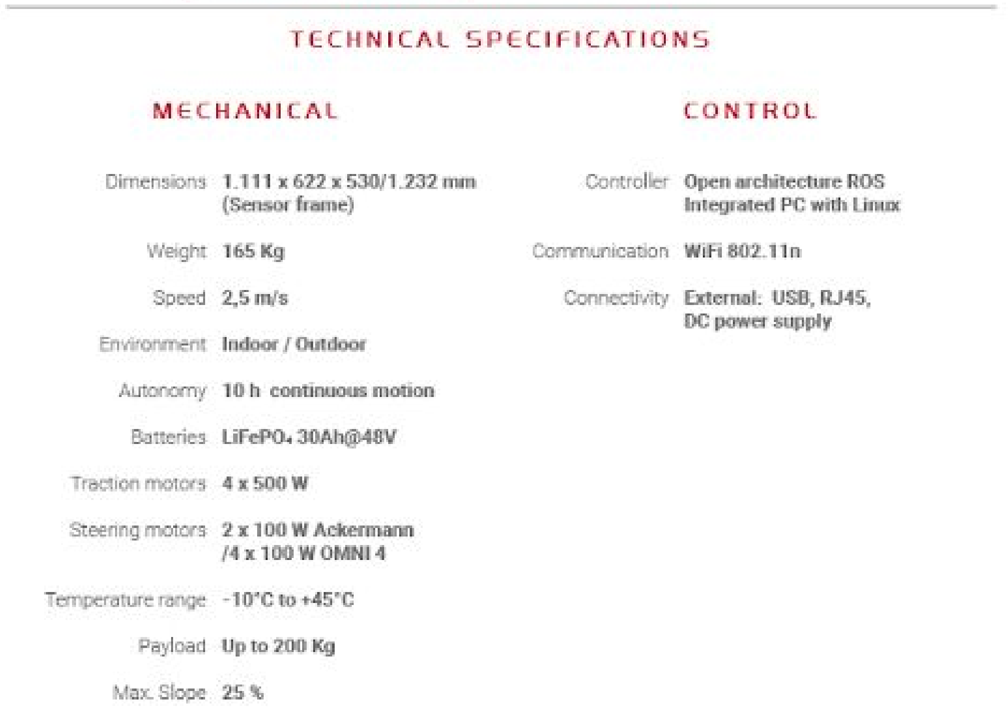
```
rosservice call
/robot/robot_local_control/NavigationComponent/GoToComponent/add
"procedure:
  goals:
  - header:
      seq: 0
      stamp:
      secs: 0
      nsecs: 0
      frame_id: 'robot_map'
      pose:
        x: 0.0
        y: 0.0
        theta: 0.0
  max_velocities:
  - linear_x: 0.0
    linear_y: 0.0
    angular_z: 0.0"
```

The frame_id parameter specifies the map frame where the robot is going to navigate. Then, the pose within the map that the robot should reach and the maximum velocity allowed to perform this action (0.0 velocity means that it uses the maximum velocity established in move_base configuration).

8. Technical specifications

8.1. RB-Sherpa





TECHNICAL SPECIFICATIONS

MECHANICAL

Dimensions	1.111 x 622 x 530/1.232 mm (Sensor frame)
Weight	165 Kg
Speed	2,5 m/s
Environment	Indoor / Outdoor
Autonomy	10 h continuous motion
Batteries	LiFePO4 30Ah@48V
Traction motors	4 x 500 W
Steering motors	2 x 100 W Ackermann /4 x 100 W OMNI 4
Temperature range	-10°C to +45°C
Payload	Up to 200 Kg
Max. Slope	25 %

CONTROL

Controller	Open architecture ROS Integrated PC with Linux
Communication	WiFi 802.11n
Connectivity	External: USB, RJ45, DC power supply

9. Maintenance

Before starting any maintenance task, switch off the power to manipulate the wheels or any other mechanical part, or if he has to access the electrical cabinets.

The following table summarizes all the elements that need maintenance and the periodicity of this maintenance.

	Often	**Every 6 month**	**Observations**
Screws	Check they are not loosen.		
Wheels		Visual control of the wheel.	Replace them when the mecanum wheel rollers or rubber tire need it.
Outer wires		Visual control of the wear.	If wear appears, temporarily protect them with a shrink tube until they are replaced.
Emergency stop button	Check regularly that the emergency stop works properly.		Change the button if it does not work well.
Battery	Control Batteries Voltage, don't let the batteries get fully discharged	Check battery autonomy	Recharge ASAP if fully discharged
Optical sensors	Clean the optics of the sensors regularly to avoid false obstacles.		

If you have any doubt about the robot maintenance, contact with Robotnik support department and request it. Below, it is explained more information about the maintenance of robot components.

9.1. Motor wheel

The most important thing to check on the motor wheels is the cable located inside the robot. This cable must be kept in good condition, and protected if the external cover is damaged. There are three 48V power wires, two 5VDC power and three Hall Effect signals. If they are short-circuit, the motor and the driver can be damaged. To access it, open the rear gate (explained in the point “9.3. Battery”.

9.1.1. Wheels disassembly instructions

Wheels are attached to the hub via 6 hexagonal head screws, as the next figure shows.

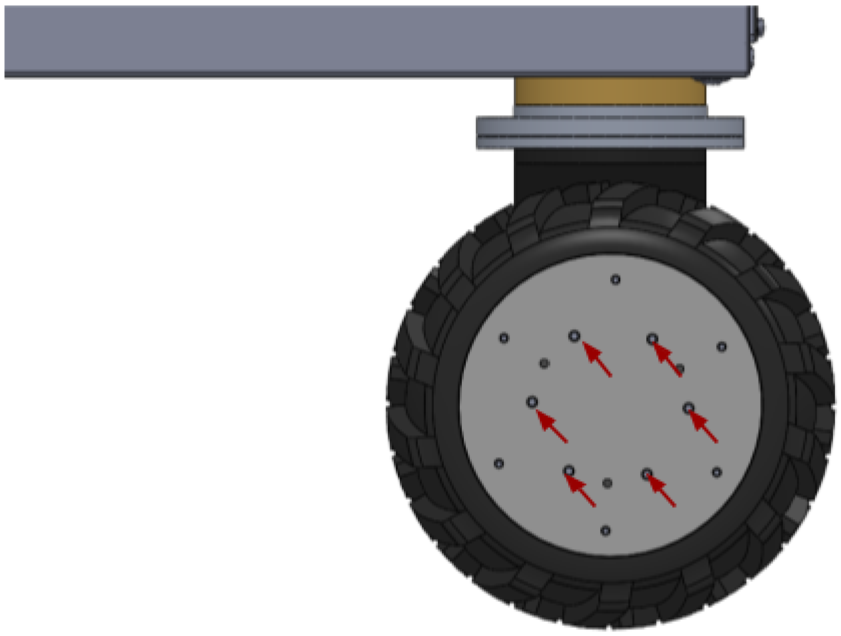


Figure 29. Wheel screws for disassembly

In the following table you could find the tightening torque four wheels depending on the situation. These are the recommended value to avoid damages in the wheels.

The robot can be assembled with **rubber or mecanum wheels**.

MECANUM WHEELS REPLACEMENT
Be careful! Each type of wheel uses different screws (rubber or mecanum). Do not use the screws of one wheel on the other, it could damage the motor.

9.1.2. Motor drivers

The drivers are programmed at Robotnik with specific settings for each motor. The serial identifier is the default one (63), but each driver has its own CAN bus identifier (1, 2, 3 and 4). DO NOT change them from one motor to another. To reduce power consumption, if the velocity is zero, only the rear wheels brake, the front ones are free.

The computer sends CAN messages to move the robot, and they are different from the left side (1 & 2) and right side (3 & 4). Driver 1 is the only one with the Can bus resistor installed.

9.2. Electronic components

To access the electronic board you have to unscrew the screw located at the top .

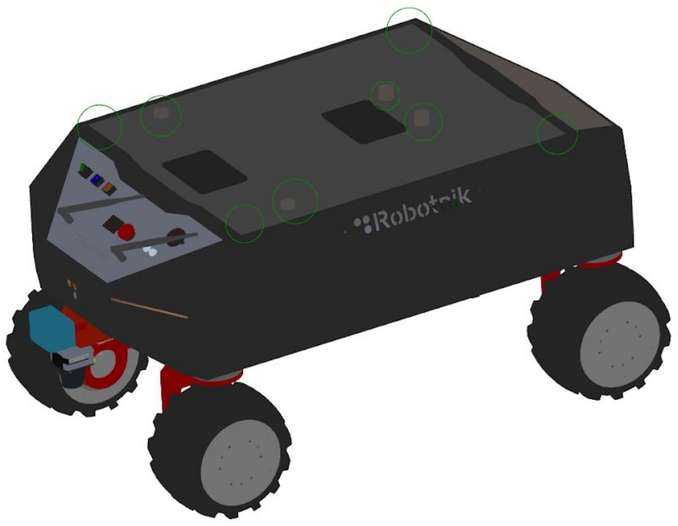


Figure 31. Screws for electronic board access

Check point "3.2.1. Electronic board" to locate the electronic board components.

9.3. Battery

The battery can be separated from the robot opening the top gate robot's and unplugging the power supply connector (red and blue wires) and the charge connector (yellow wire). To open the rear robot's gate, you have to unlock the two locks that keep it closed.

The batteries must be kept clean and dry in order to avoid escape currents. Check the wear out of the battery wires to prevent short circuits.

Recharge the batteries ASAP if fully discharged. Keeping the voltage low for a long time will greatly reduce the life cycles.

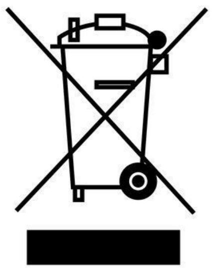
10. Recycling of the robot

Robotnik Automation SL is committed to providing quality products in an environmentally sound manner. For this reason, Robotnik Automation SL continuously improves the design processes of its products to minimize the negative impact in their useful life.

Design for recycling has been incorporated into this product:

- The number of materials has been kept to a minimum while ensuring proper functionality and reliability.
- Dissimilar materials have been designed to separate easily.
- Fasteners and other connections are easy to locate, access, and remove using common tools.
- High-priority parts have been designed so that they can be accessed quickly for efficient disassembly and repair.

The packaging materials for this device have been selected to provide maximum protection for the least cost possible, while attempting to minimize environmental impact and facilitate recycling.

	**European Union**: under the directive 2002/96 / EC of the European Union regarding waste and / or electronic equipment, with a rigorous date since August 13, 2005, products classified as "electrical and electronic equipment" cannot be deposited in usual containers of your municipality, equipment manufacturers electronic, are required to take care of these products at end of his life period. Use the public collection system to return, recycle or treat them in accordance with local regulations.

11. Customer service and technical support

If you have any doubt or problem with your robot, please contact with us by sending an email to support@robotnik.es, indicating the serial number of your robot.
We are constantly improving and upgrading our products and services. Any feedback from users is welcome.

NOTE: Our language for support communication is either in English or Spanish.