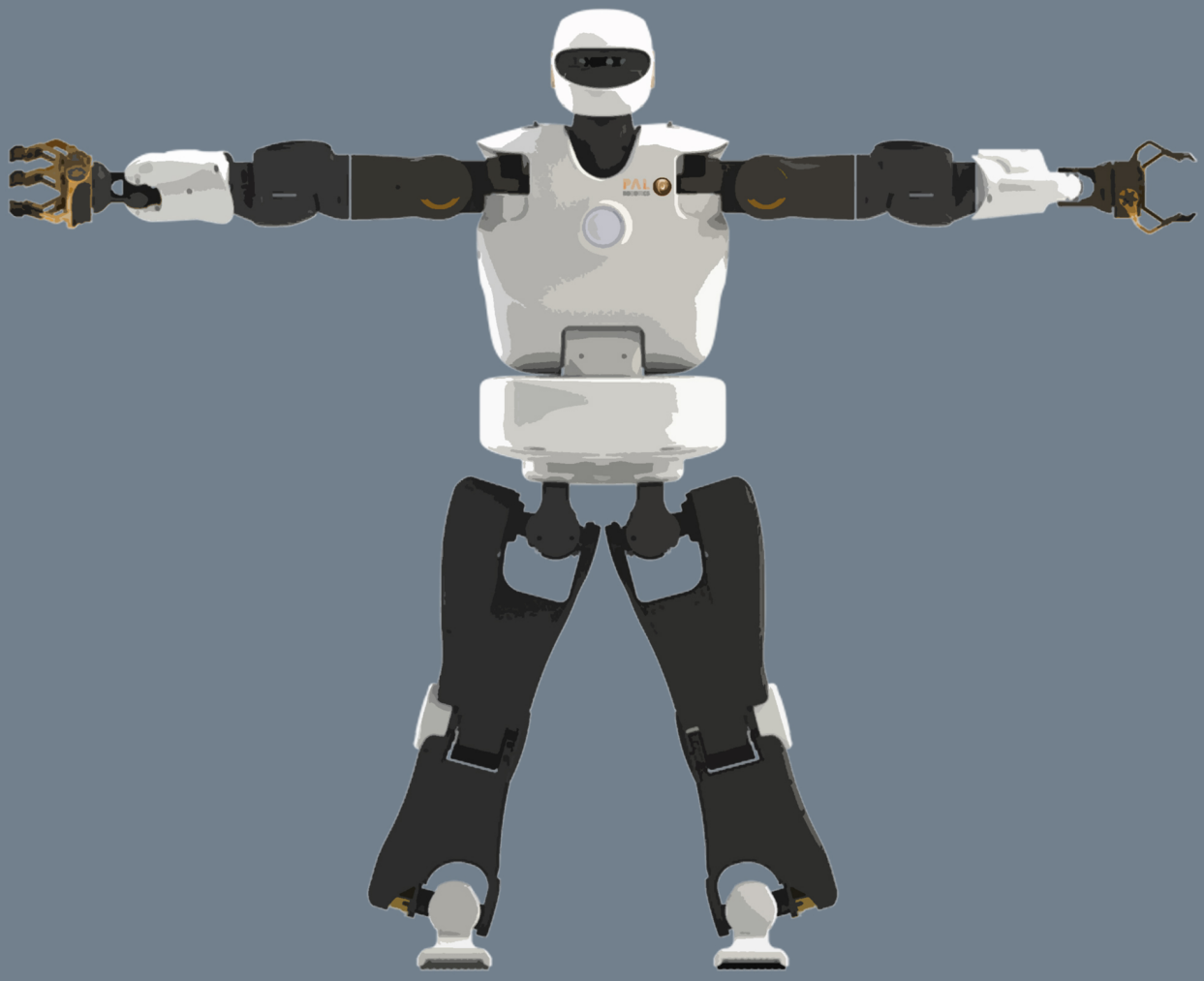
TALOS

Handbook



PAL ROBOTICS

Handbook

Barcelona
2020

Contents

Welcome

Welcome

Thank you for choosing PAL Robotics. This Handbook contains information related to the TALOS robot developed by PAL Robotics. Every effort has been made to ensure the accuracy of this document. All the instructions must be strictly followed for proper product usage. The software and hardware described in this document may be used or replicated only in accordance with the terms of the license pertaining to the software or hardware.

Disclaimers

General Disclaimer

TALOS and its components and accessories are provided "as is" without any representations or warranties, express or implied. PAL Robotics makes no representations or warranties in relation to its products or the information and materials related to them, other than the ones expressly written in this Handbook.

In no event shall PAL Robotics be liable for any direct, indirect, punitive, incidental, special or consequential damages to property or life, whatsoever, arising out of or connected with the use or misuse of TALOS or the rest of our products.

Handbook Disclaimer

Please note that each product application may be subject to standard of identity or other regulations that may vary from country to country. We do not guarantee that the use of TALOS in these applications will comply with such regulations in any country. It is the user's responsibility to ensure that the incorporation and labeling of TALOS complies with the regulatory requirements of their markets.

No warranties This Handbook is provided "as is" without any representations or warranties, express or implied. PAL Robotics makes no representations or warranties in relation to this Handbook or the information and materials provided herein. Although we make a reasonable effort to include accurate and up to date information, without prejudice to the generality of this paragraph, PAL Robotics does not warrant that the information in this Handbook is complete, true, accurate or non-misleading. The TALOS Handbook is provided solely for informational purposes. You should not act upon information without consulting PAL Robotics, a distributor, subsidiary or appropriate professional.

Limitations of liability PAL Robotics will not be liable (whether under the law of contract, the law of torts or otherwise) in relation to the contents of, or use of, or otherwise in connection with, this Handbook:

- to the extent that this Handbook is provided free-of-charge, for any direct loss;
- for any indirect, special or consequential loss; or
- for any business losses, loss of revenue, income, profits or anticipated savings, loss of contracts or business relationships, or loss of reputation or goodwill.

These limitations of liability apply even if PAL Robotics has been expressly advised of the potential loss.

Exceptions Nothing in this Handbook Disclaimer will exclude or limit any warranty implied by law that it would be unlawful to exclude or limit; and nothing in this Handbook Disclaimer will exclude or limit PAL Robotics's liability in respect of any:

- personal injury caused by PAL Robotics's negligence;
- fraud or fraudulent misrepresentation on the part of PAL Robotics; or
- matter which it would be illegal or unlawful for PAL Robotics to exclude or limit, or to attempt or purport to exclude or limit, its liability.

Reasonableness By using this Handbook, you agree that the exclusions and limitations of liability set out in this Handbook Disclaimer are reasonable. If you do not think they are reasonable, you must not use this Handbook.

Other parties You accept that, PAL Robotics has an interest in limiting the personal liability of its officers and employees. You agree that you will not bring any claim personally against PAL Robotics's officers or employees in respect of any losses you suffer in connection with the Handbook.

Without prejudice to the foregoing paragraph, you agree that the limitations of warranties and liability set out in this Handbook Disclaimer will protect PAL Robotics's officers, employees, agents, subsidiaries, successors, assigns and sub-contractors, as well as PAL Robotics.

Unenforceable provisions If any provision of this Handbook Disclaimer is, or is found to be, unenforceable under applicable law, that will not affect the enforceability of the other provisions of this Handbook Disclaimer.

1 Package contents

1.1 Overview

This section includes a list of items and accessories that come with TALOS. Make sure they're all present:

(a) Transport box

(b) Charger

Figure 1: Transportation box and battery charger

(a) Wireless access point

(b) Basestation

Figure 2: Connectivity

(a) Joystick

(b) Installation Flash Disks

Figure 3: Peripherals

Specifications

2 Specifications

2.1 Overall dimensions

- Height: 1750 mm
- Width: ∼775 mm
- Depth: ∼330 mm

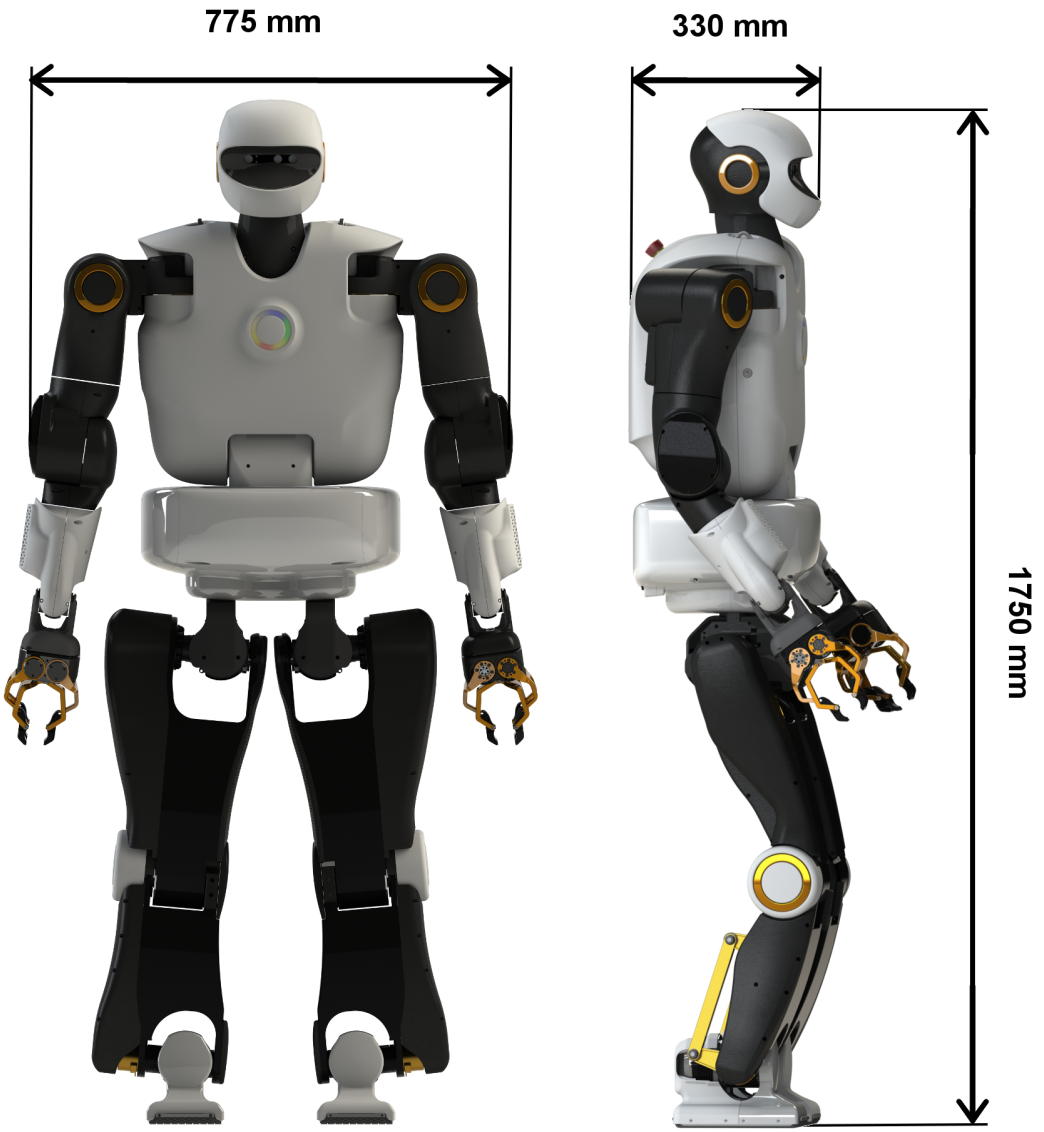


Figure 4: Dimensions

2.2 Weight

The total weight of TALOS is about 100 kg, including batteries.

2.3 Degrees of freedom

TALOS is provided with a total of 32 degrees of freedom, following the distribution, shown in Table 1.

	Qty. of DoF
Head	2
Left Arm	7
Right Arm	7
Left Gripper	1
Right Gripper	1
Waist	2
Left Leg	6
Right Leg	6

Table 1: Degrees of freedom

2.4 Payload

TALOS is able to grasp objects and sustain them. The payload of each arm is up to 6 kg, with the arm completely stretched, and the robot is also able to have a payload of up to 20 kg sustained between the torso and the arms.

2.5 Kinematics

The kinematic structure of TALOS is depicted in Figure 5. Each depicted frame is located at the origin of a joint. All joints are revolute. The base link of the robot is located close to its center of mass, in the waist.

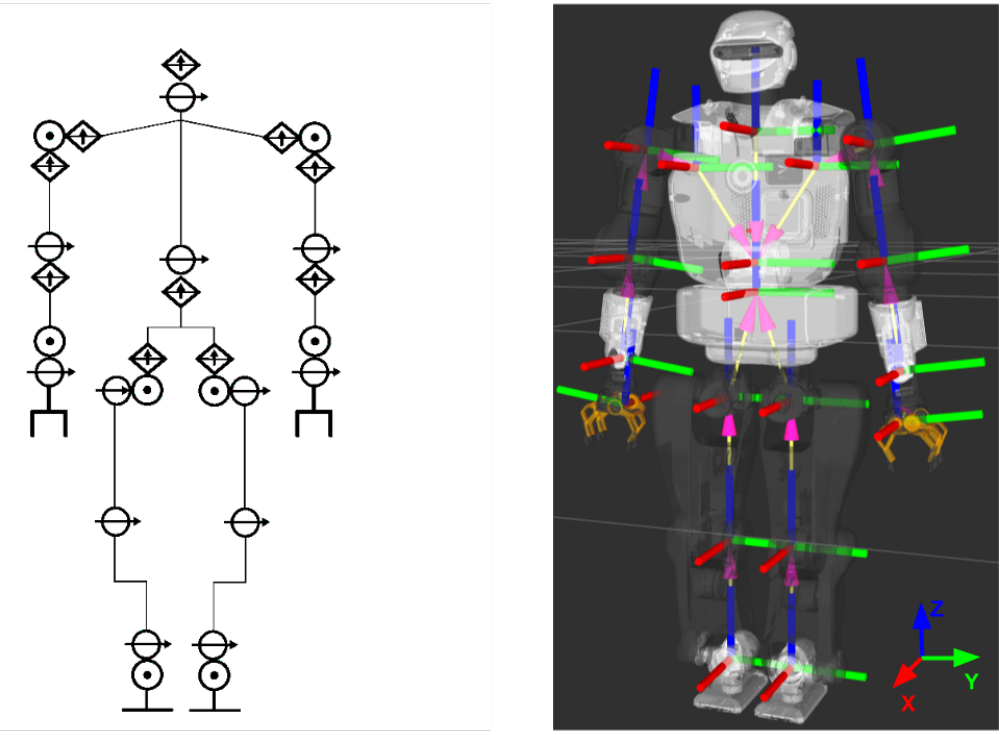


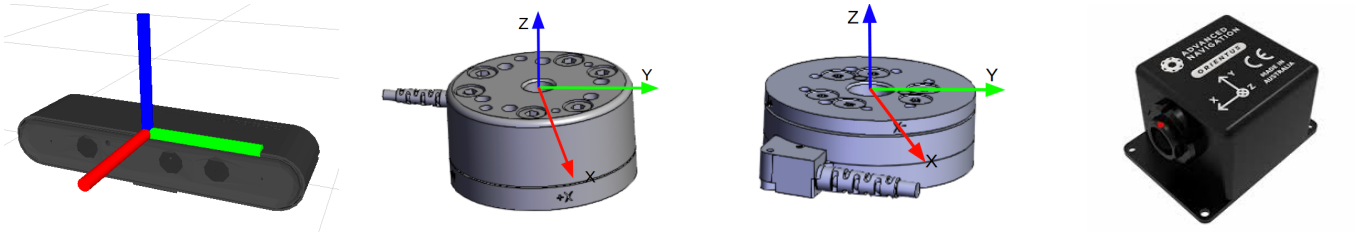
Figure 5: TALOS upper body joints

Table 2 lists the sensors installed in TALOS with their reference frame name.

Sensor	Link	Parent Link
RGB-D	rgbd_link	head_2_link
Right Ankle FT	leg_right_6_link	leg_right_5_link
Left Ankle FT	leg_left_6_link	leg_left_5_link
Right Wrist FT	wrist_right_ft_link	leg_left_5_link
Left Wrist FT	wrist_left_ft_link	leg_left_5_link
IMU	imu_link	torso_2_link

Table 2: Sensors location for TALOS

The reference frames for the sensors are depicted in Figure 6.



RGB-D Foot FT sensor Wrist FT sensor IMU

Figure 6: Refence frames placement on sensors

2.6 Dynamics

The dynamics information, such as body links CoM (Center of Mass) location and intertia matrices, can be found in the URDF (Unified Robot Description File) provided with the robot or available at https://github.com/pal-robotics/talos_robot.

Table 3 shows the details for the output torque, speed and range of motion for each DoF.

		Joint range				
Name joint	Parent joint	Min	Max	Nominal Torque[Nm]	Max Torque [Nm]	Maximum angular velocity [RPM]
Legs						
leg_left_1_joint	base	-20	90	49	82	38
leg_left_2_joint	leg_left_1_joint	-30	30	90	157	56
leg_left_3_joint	leg_left_2_joint	-120	45	87.3	157	56
leg_left_4_joint	leg_left_3_joint	0	150	143.8	308	56
leg_left_5_joint	leg_left_4_joint	-72,5	38	90	157	56
leg_left_6_joint	leg_left_5_joint	-30	30	49	82	56
Torso and head						
torso_1_joint	base	-75	75	160	200	56
torso_2_joint	torso_1_joint	-15	45	160	200	56
head_1_joint	torso_2_joint	-15	45	6	9	30
head_2_joint	head_1_joint	-75	75	6	9	15
Arms						
arm_left_1_joint	torso_2_joint	-30	90	90	157	56
arm_left_2_joint	arm_left_1_joint	0	165	90	157	56
arm_left_3_joint	arm_left_2_joint	-140	140	49	82	56
arm_left_4_joint	arm_left_3_joint	0	135	49	82	56
arm_left_5_joint	arm_left_4_joint	-145	145	15	25	30
arm_left_6_joint	arm_left_5_joint	-80	80	6	9	30
arm_left_7_joint	arm_left_6_joint	-40	40	6	9	30

Table 3: Joint properties. Joint torque/speed represents the expected performance when battery is fully charged.

3 Sensors

3.1 Cameras

3.1.1 Frontal Cameras

There are frontal cameras placed in TALOS 's head, specifications for which are described in Table 4.

Manufacturer	Orbec
Model	Orbec Astra Pro
Interface	USB 2.0
Resolution	1280x720
Range	0.4-8 m
Depth Image Size	640*480
Field of View	60° horizontal, 49.5° vertical

Table 4: Frontal camera

3.2 Six axes force/torque sensors

TALOS has six axes force/torque sensors in each ankle and wrist. Their specifications are shown in Tables 5, 6 and 7.

Manufacturer	ATI
Model	MINI58
Fx,Fy range	2800 N
Fz range	6800 N
Tx,Ty range	120 Nm
Tz range	120 Nm
Fx,Fy resolution	3/4 N
Fz resolution	1 N
Tx,Ty resolution	1/50 Nm
Tz resolution	1/80 Nm

Table 5: Force Torque sensors mounted in the ankles

Manufacturer	ATI
Model	MINI45
Fx,Fy range	580 N
Fz range	1160 N
Tx,Ty range	20 Nm
Tz range	20 Nm
Fx,Fy resolution	1/4 N
Fz resolution	1/4 N
Tx,Ty resolution	1/188 Nm
Tz resolution	1/376 Nm

Table 6: Force Torque sensors mounted in the wrists

3.3 Joint torque sensors

TALOS is equipped with a single axis joint torque sensor in every joint, except the wrists, grippers and head. There are three families of such sensors, depending on the nominal range of measurement: 100Nm, 160Nm, 300Nm. Distribution of these sensors among the DoF is shown in Table 7.

Joint	T range [Nm]
leg_1_joint	100
leg_2_joint	160
leg_3_joint	160
leg_4_joint	300
leg_5_joint	160
leg_6_joint	100
arm_1_joint	160
arm_2_joint	160
arm_3_joint	100
arm_4_joint	100
torso_1_joint	160
torso_2_joint	160

Table 7: Torque sensors in joints

3.4 Inertial measurement unit

There is one inertial measurement unit in TALOS 's torso. Sensor specifications are listed in Table 8.

Roll/Pitch Accuracy (Static)	0.2°
Heading Accuracy (Static)	0.5°
Roll/Pitch Accuracy (Dynamic)	0.6°
Heading Accuracy (Dynamic)	1.0°
Internal filter rate	1000 Hz
Output data rate	1000 Hz
Accelerometers range	2,4,16 g
Gyroscopes range	250,500, 2000 °/s
Magnetometers	2,4,8 G
Accelerometers bias stability	20 µg
Gyroscope bias stability	3 °/hr
Non-linearity	< 0.05 %

Table 8: Inertial Measurement Unit

3.5 Motors

Every degree of freedom of TALOS is powered by an electrical motor. Table 9 shows the input specifications for every joint motor. These values also correspond to the respective joints of both left and right side.

Joint	Nominal voltage (V)	Power (W)	Torque constant (mN.m/A)	Cont. current (A) (RMS)	Reduction Ratio
leg_1_joint	48	180	138.6	3.53	150
leg_2_joint	48	275	145.8	8.57	101
leg_3_joint	48	275	145.8	8.57	100
leg_4_joint	48	430	91.9	15.5	144
leg_5_joint	48	275	145.8	8.57	100
leg_6_joint	48	270	149.9	4.95	101
arm_1_joint	48	275	145.8	8.57	100
arm_2_joint	48	275	145.8	8.57	100
arm_3_joint	48	270	149.9	4.95	100
arm_4_joint	48	270	149.9	4.95	100
arm_5_joint	48	220	91	2.31	100
arm_6_joint	48	160	48	1.75	100
arm_7_joint	48	160	48	1.75	100
arm_8_joint	48	160	48	1.75	100
torso_1_joint	48	275	145.8	8.57	100
torso_2_joint	48	275	145.8	8.57	100
head_1_joint	48	160	48	1.75	100
head_2_joint	48	160	48	1.75	200

Table 9: Motors

3.6 Connectivity

TALOS is equipped with a 802.11a/n/ac 5GHz and a 10/100/1000 Mbps Ethernet port.

It is important to note that there are two ways to communicate with the robot: through VPN access (wired or wireless) or directly (wired or wireless).

- Wireless VPN connection

TALOS wireless interface must be configured in order to communicate with the Access Points. If there is more than one Access Point in the facility, they must share the same SSID. Note that when the robot is connected to the network it requires an IP address. This address can be obtained from a DHCP or it can be statically assigned.

At that point, TALOS initiates a connection to the Basestation whose IP address is known. Once this connection is established, all the VPN traffic is routed through this pathway.

- Direct VPN connection

Removing the user panel cover on the back of the robot, an Ethernet connector can be found. This connector can be configured in order to work with the building network. The VPN connection is then routed using this wired connection. If the Ethernet is disconnected, the robot switches to the Wireless VPN connection automatically.

- Direct wired connection

This Ethernet connector is, by default, a direct connection to the TALOS internal network. The interface provides the IP configuration to the connected device via a DHCP server. This system will work even if the VPN system doesn't. This is typically used for the initial network configuration of TALOS, or for debugging other networking problems.

- Direct wireless connection

It is also possible to connect a computer directly to TALOS by using the wireless connection in Access Point mode. This is essentially identical to plugging into the wired service port.

3.7 Electrical parts and components

Neither TALOS nor any of its electrical components and mechanical parts are connected to an external ground. The chassis of the robot is connected to the negaitve pole of the robot's battery. To avoid any damage in the electromechanical part of TALOS , don't touch any metallic parts directly as this may result in ESD discharges.

Electrical power supply and connectors

The power source supplied with TALOS is compliant with the directive on the restriction of the use of certain hazardous substances in electrical and electronic equipment 2002/95/EC (RoHS) and compliant with the requirements of the applicable EC directives, according to the manufacturer. The power source is connected to the environment ground, whenever the supplied wire is used (Phase-Neutral-Earth).

Battery

The specifications of the battery supplied with TALOS are shown in Table 10.

Type	Li-Ion
V_nominal	72.0V
V_max	84.0V
V_cutoff	60.0V
Nominal capacity	14.75 Ah
Nominal energy	1062 Wh
Continuous discharge current	50A
Pulse discharge current	100A
Max. charging current	10A
Charging method	CC/CV
Weight	8.5 kg

Table 10: Battery specifications

Computers

There are two computers (PCs) in TALOS : one is for real-time control applications and the other for multimedia applications. On the Service Panel, the Control PC is referred to as PC1; Multimedia PC is referred to as PC2. Both PCs are controlled with the same power button, have two USB ports and one HDMI port, accessible through the service panel.

3.8 Service Panel

Service panels and emergency button are located on the back of the robot (as shown in Figure 7).

All the ports and buttons are detailed in Table 11.

- First level: main power switch, computers on/off button and battery indicator (Figure 8).
- Second level: it is accessible after removing the squared cover and it exposes ports of the Control and Multimedia computers plus the EtherCAT network port (Figure 9).

Figure 7: TALOS torso back with Emergency button

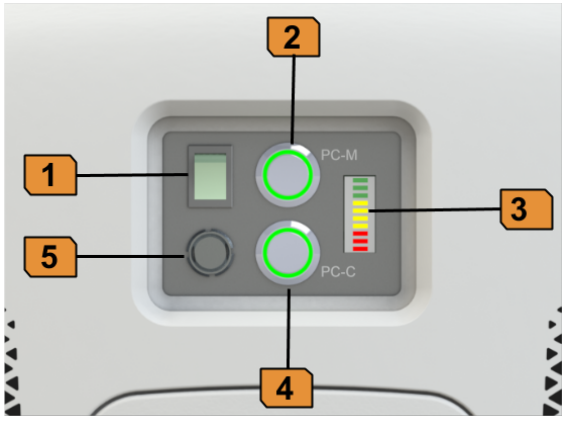


Figure 8: First level user panel

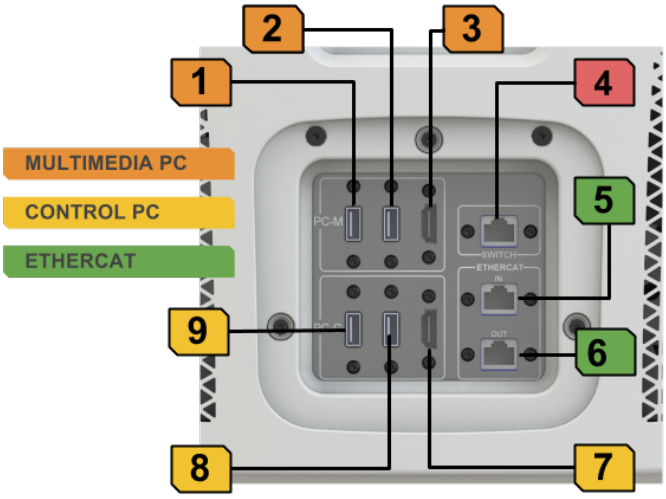


Figure 9: Second level user panel

First Level	
Number	Name / Short description
1	Main Switch
2	PC - ON/OFF
3	Battery Level Indicator
4	User Button
5	Service Connector

Second Level	
Number	Name / Short description
1	USB 1 (PC Multimedia)
2	USB 2 (PC Multimedia)
3	HDMI 1 (PC Multimedia)
4	Ethernet Switch
5	EtherCAT In (Robot)
6	EtherCAT Out (PC Control)
7	HDMI 2 (PC Control)
8	USB 1 (PC Control)
9	USB 2 (PC Control)

Table 11: Service Panel description

Handling

4 Storage

4.1 Overview

Information relating to transport, handling and storage of TALOS will be explained in this section.

4.2 Transport

Based on the the *United Nations´ "Recommendations on the Transport of Dangerous Goods - Model Regulations*" the batteries **are restricted to transport** and **assigned to Class 9**.

The packaging of the batteries has to be properly labeled, tested and certified to satisfy the above requirements.

The batteries cannot be transported within the same crate as TALOS.

4.3 Storage cautions

- Always store TALOS where it will not be exposed to weather.
- The storage temperature range for TALOS is between 0°C $\sim$ +50°C.
- The storage temperature range for the batteries is between +10°C $\sim$ +35°C.
- It is recommended to remove or disconnect the battery from TALOS when storage period exceeds two weeks.
- It is recommended to charge the battery to 50% when storing it for more than two weeks.
- The storage temperature range for the Basestation is between 0°C $\sim$ +60°C.
- If TALOS has to be stored for a long time, it is highly recommended to keep it in its crate. Instructions explaining how to place the robot into the crate can be found in Section 6.4.
- TALOS should be always hung using the hooks that stand out from the shoulders (Figure 10). To reduce the stress on the mechanical structure of the robot, make sure that the feet are clearly in contact with the floor, completely flat and the robot remains straight and vertical. Always use the carabiners and ropes supplied with TALOS.

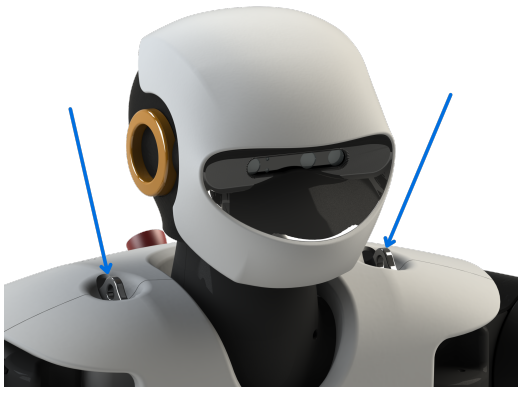


Figure 10: Hooks location

- Never hold TALOS by any other part but the hooks over the shoulders (Figure 10).
- TALOS should never stay out of its crate without being sustained by the hooks over the shoulders (Figure 10).
- Avoid the use or presence of water near TALOS.
- Avoid any generation of dust close to TALOS.
- Avoid the use or presence of magnetic devices or electromagnetic fields near TALOS.

5 Indications for handling

In order to handle TALOS safely, take the following into account:

- TALOS may only be used in perfect technical conditions, in accordance with its designated use, and only by safety-conscious people who are fully aware of the risks involved during the operation.
- Never use TALOS without being sustained by the hooks over the shoulders (Figure 10). Take the weight of the robot into account when dimensioning the holding device (see chapter “Specifications”, section “Weight”).
- While the robot is on, keep away all objects, obstacles and people, respecting the safety distances shown in Figure 20.
- Never drag TALOS.
- Never pull or push TALOS.
- Never lift TALOS by hand.
- Never hit or drop TALOS.
- Never use sharp objects against TALOS.
- While handling TALOS, do not wear watches or any kind of jewelery on hands or wrists. It is recommended to use gloves.
- Do not touch the ultrasound sensors directly, to avoid damaging them.
- Do not touch the camera’s lenses to avoid damaging or dirtying them.
- Avoid the use or presence of liquids near the robot.
- TALOS contains numerous delicate electronic circuits and components which can become damaged as a result of electrostatic discharge (ESD). Prior to handling, carefully read and follow these procedures:
  - Avoid touching any metallic parts of TALOS.
  - It is advisable to wear an electrostatic discharge (ESD) wrist strap when handling external metallic parts or make sure to discharge yourself of ESD.

Assembly

6 Set-up and mounting conditions

6.1 Overview

Information related to the installation of the robot will be explained in this section.

6.2 Robot crate

Approximate external dimensions of the crate:

- Weight: 140 kg (empty), 240 kg (with TALOS inside)
- Length: 1950 mm
- Width: 1050 mm
- Height: 735 mm (including wheels)

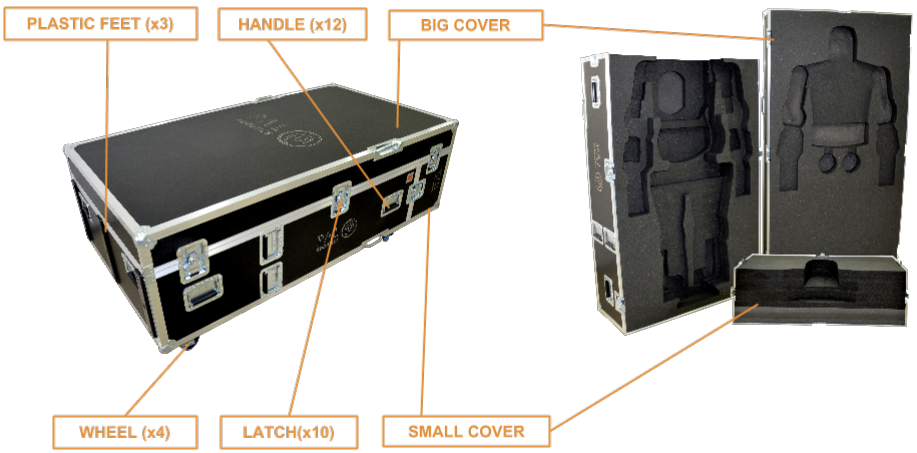


Figure 11: Crate parts

6.3 How to get the robot out of the crate

To get the robot out of its crate safely, the following is required:

- At least two people.
- A crane (i.e. Figure 12), with a maximum cradle height over 1900 mm, and a horizontal leg height of less than 150 mm.
- Dyneema slings and the carabiners supplied with the robot (Figure 13) hanging on the cradle.

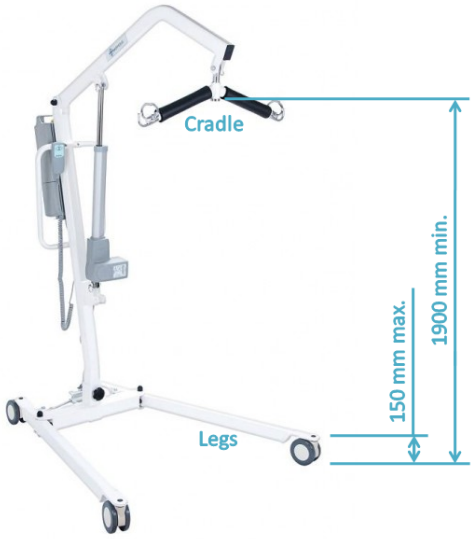


Figure 12: Crane

Figure 13: Carabiners and Slings

Instructions to get TALOS out of the crate (please refer to FIgure 11):

1. With the crate in a horizontal position, remove the big cover.
2. Remove the small cover.
3. Remove the foam protector and make sure the belts that keep the robot secured are tight.
4. Using the corresponding handles at both sides of the box, lift the crate so the box remains on top of the two plastic feet. For optimum safety, release the crate wheel brakes.
5. Move the crane in front of the robot so that the carabiners can be inserted in both black hooks over the shoulders of the robot. The horizontal legs of the crane will be located below or laterally respect the crate. Take care not to hit the robot´s plastic covers with the carabiners to avoid damage.
6. Adjust both slings to the same length so the crane is able to lift the robot up completely. Shorten the slings as much as possible, while taking care to avoid collisions between TALOS´s head and the crane horizontal bar.
7. Lift the robot using the crane until there is tension in the slings.
8. Loosen all the belts that fix the robot to the crate (head, torso, hip and knees).
9. To separate the robot from the back foam protector, lift the robot slowly while pulling it out with the crane.
10. Now the robot can be completely (and carefully) moved away from the crate.

6.4 How to put the robot into its crate

To get the robot into its crate, either for storage or transportation, the following is required:

- At least two people.
- A crane (i.e. Figure 12), with a maximum cradle height of over 1900 mm, and a horizontal leg height of less than 150 mm.
- Two adjustable straps and the carbines supplied with the robot hanging on the cradle.

Once the robot is hanging on the crane, the subsequent instructions must be followed:

1. Using the corresponding handles at both sides of the box, lift the crate so that the box remains on top of the two wooden skids. Later, these two wooden parts will allow tyou to place the legs of the crane below the crate.
2. Release the crate wheel brakes.
3. Remove the big cover.
4. Remove the small cover.
5. Remove the foam protector. A rear foam protector will remain inside the crate.
6. Carefully lift the robot with the crane (in order to let its feet pass clearly over the wall of the crate) and with the front of the robot facing the operator.
7. Move the robot towards the crate. The horizontal legs of the crane will be located bellow the crate, between the wooden skids.
8. When the back of the robot is close to the foam, lower it carefully at the same time than pushing and fitting it into the back foam protector of the crate. The legs have to be separated by a triangular piece of foam. Make sure that the feet, knees, thumbs and eyes of the robot are pointing forward.
9. Once the robot is completely fitted into the back foam protector, adjust the belts to fix the knees, pelvis, torso and head, before releasing the robot from the crane.
10. Release the carabiners from the hooks of the robot.
11. Assemble the front foam protector. Make sure that the thumbs are pointing forward.
12. Assemble the small cover.
13. Assemble the big cover.
14. Using the corresponding handles at both sides of the box, put the crate on its wheels.

7 Power supply

7.1 Battery assembly

To install the battery in the robot, the below procedure must be followed:

1. Make sure that the robot is turned off and the Emergency stop is pressed.
2. Remove the cover in the right side of the robot´s pelvis by loosening the screws (Figure 14, Figure 15).
3. The battery will be inserted from robot´s right side, as shown in Figure 16.
4. Handle the battery carefully. The weight of the battery is about 8.5 kg.
5. Slide the battery inside the compartment (Figure 17).
6. Tighten the four M4 screws supplied with the battery (Figure 17).
7. Remove the protector from the battery connector. Make sure the pins won't touch any conductive surface.
8. Plug in the connector with black cables, facing upwards as shown in Figure 18.
9. Mount the cover (Figure 19) and insert its M3 screws.

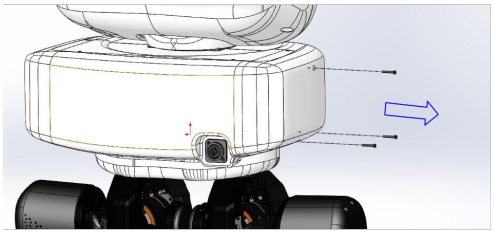


Figure 14: Release screws

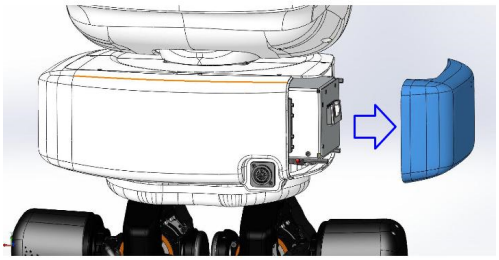


Figure 15: Remove side covers

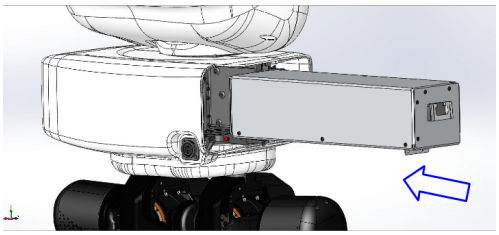


Figure 16: Battery assembly

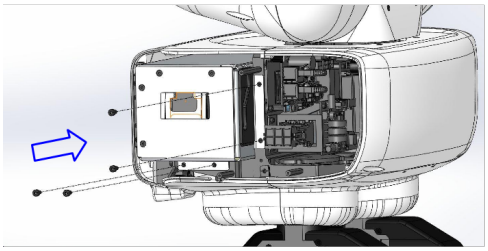


Figure 17: Insert screws

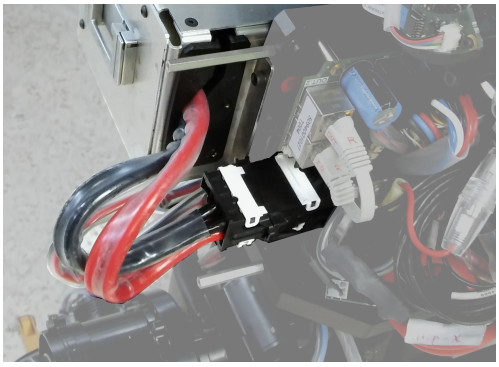


Figure 18: Plugging

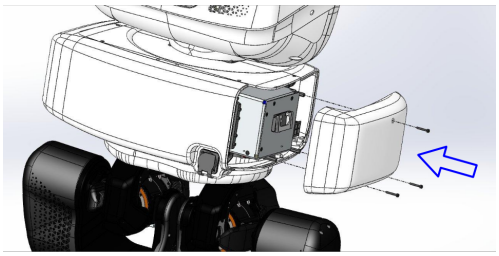


Figure 19: Attach side covers

Safety

8 Introduction to safety

8.1 Overview

This chapter provides an important overview of safety issues, general usage guidelines and some safety-related design features. *Before operating* TALOS*, all users must read and understand this chapter*

8.2 Intended applications

The intended usage of the robot must be clearly set out before any kind of operation.

TALOS is a robotics research and development platform meant to be operated in a controlled environment under supervision by trained staff at all time.

- The hardware and software of TALOS allows users to research and develop activities in the following areas:
  - Biped walking
  - Navigation and SLAM
  - Manipulation, grasping, whole body control
  - Speech recognition
  - Computer vision
  - Human-robot interaction
- Strictly prohibited uses include:
  - Applications that require the robot to be used without the crane.
  - Manipulating objects exceeding the payload specifications of the arms described in Section 2.4.
  - Modifications of the robot´s hardware in any way without prior and appropriate instruction by PAL Robotics.
  - Applications where the robot could cause harm either to people or to itself.
  - Operation by untrained staff.

8.3 Working environment and usage guidelines

The working temperatures are:

- Robot: +10°C $\sim$ +35°C
- Basestation: -5°C $\sim$ +55°C (ambient with air flow)

The space where TALOS operates should have a flat floor and be free of hazards. Stairways and other drop-off points, in particular, can pose an extreme danger. Avoid hazardous or sharp objects (such as knives), sources of fire, hazardous chemicals or furniture that could be knocked over.

Maintain a safe environment:

- The terrain for TALOS's usage must be capable of supporting the weight of the robot (see Section Specifications). It must be horizontal and flat. Do not use carpets, to avoid the robot tripping over.
- Make sure the robot has adequate space for any expected or unexpected operation.

- Make sure the environment is free of objects that could pose a risk if knocked, hit, or otherwise affected by TALOS.
- Make sure there are no cables or ropes that could be caught in the covers, legs or arms, as these could pull other objects over.
- Keep animals away from the robot.
- Be aware of emergency exit locations and ensure they cannot be blocked by the robot.
- Do not operate the robot outdoors.
- Keep TALOS away from flames and other sources of heat.
- Do not allow the robot to come into contact with liquids.
- Keep the room clear of dust.
- Avoid the use or presence of magnetic devices near the robot.
- Be cautious whenever the robots' arms are away from the body, as body parts or other objects could be damaged if caught between the body and the arms.
- Apply extreme caution with children.

8.4 Risks

1. The robot may fall down at any time. There is no mechanism that can enforce balance constraints for all operating conditions, particularly when:
   (a) the emergency stop is activated;
   (b) the robot is externally disturbed (e.g: it is pushed or hits an obstacle);
   (c) the operator sends control commands that make the robot lose balance;
   (d) the robot steps on an uneven surface while walking.
2. The robot can cause significant damage if it runs over someone or if someone runs over it. The robot can wield dangerous implements and can knock heavy objects over. People must always be cautious and attentive when around a TALOS .
3. The behavior and capabilities of TALOS can be changed by user interaction or reprogramming.

8.5 Safety distance

In order to keep both the robot and operators safe while TALOS is working, everyone except authorized personnel should maintain the distance shown in Figure 20.

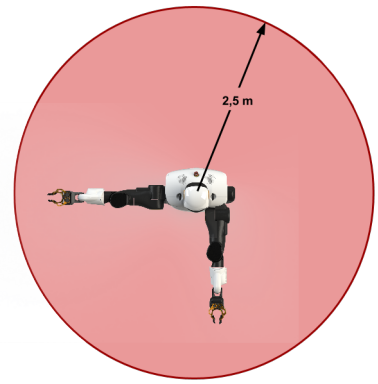


Figure 20: Safety distance

8.6 Battery manipulation

General guidelines when manipulating the battery:

- Do not expose to fire.
- Do not let the battery get wet.
- Do not open or modify the battery case.
- Do not expose the battery to ambient temperatures above 49°C for over 24 hours.
- Do not store the battery in temperatures below -5°C over seven days.
- For long term storage (more than one month), charge the battery to at least 50%.
- Do not use TALOS's battery for other purposes.
- Do not use any devices except the supplied charger to recharge the battery.
- The weight of the battery is about 8.5 kg; be careful not to drop it.
- If any damage or leakage is observed, stop using the battery.

8.7 Transmission bar warning

Be careful with the ankle´s transmission bar (Figure 21). Maintain a safe distance from the transmission bar when the TALOS is moving, and never use it to manipulate the TALOS.

Figure 21: TALOS ankle transmission bar

9 Safety measures in practice

9.1 Control system functions

TALOS has safety functions in both the low- and high-level controllers.

In the low-level controllers, the robot has *joint limits avoidance*. Both *joint position* and *joint velocity* limits are enforced by the controllers. This means that all joint commands exceeding the limits are automatically (and appropriately) scaled to stay within them. Users and developers are discouraged from changing these configurations.

With the high-level controllers, the robot is able to avoid detected obstacles in the environment thanks to its sensor suite.

TALOS is also equipped with an emergency stop button, which is described in more detail in Section 9.2. Do not hesitate to use the emergency stop if the robot produces unexpected behavior.

9.2 Emergency stop

The emergency stop button can be found on the back of the robot between the power button and the battery level display. As the name implies, this button should be used only in exceptional cases when the robot is required to stop immediately.

To activate the emergency stop, push the button. To deactivate the emergency stop, rotate the button clockwise, according to the indications on the button, until it pops out.

- Pushing the emergency button turns off the robot´s power. All electronic components, hardware controllers and computers will be powered down. Be careful when using the emergency stop because the motor controllers will be switched off, and the robot´s arms and legs will fall down without the ability to brake or control.
  - After releasing the emergency stop button, re-start the robot by using the power button.

Figure 22: Emergency button in normal state (left), emergency button in pressed state (right)

ATTENTION! Nothing prevents the robot from falling or colliding with objects in the environment while the emergency stop button is pressed. Collisions could result in irrecoverable damages to the robot, its mechanical parts or electrical components.

9.3 Firefighting equipment

If using TALOS in a laboratory or location with safety conditions, it is recommended that operators keep a C Class or ABC Class extinguisher on site, as these are suitable for putting out an electrical fire.

If there is a fire, please follow these instructions:

1. Always put your own and other people's safety first.

2. Call the firefighters.
3. If you can push the emergency stop button without any risk, please do so.
4. Only trained personnel should tackle a fire.
5. Leave the area and wait for the fire fighters.

9.4 Leakage

The battery is the only component of the robot that is able to leak. To avoid leakage and ensure the battery is maintained correctly, follow the instructions defined in sections 4 and 7.1.

Using TALOS

10 General information for usage

This section contains the information and procedures operators need to use the robot.

10.1 User access

The robot´s computers, basestation and development computer have three users:

- root: the administrator user;
- *pal*: the default user for executing applications;
- aptuser: this user allows connections between elements without a password. It is used for sending and receiving files.

The element´s default passwords can be found in Section 16.

10.2 Power supply

The robot is supplied with a 74V Lithium-Ion battery. This is the only power source authorized for the robot and cannot be replaced by any other battery. When the battery is discharged, only use the battery charger supplied with the robot or by PAL Robotics.

Figure 23: Battery charger

The charger has an output of 74VDC nominal (84VDC maximum) and is suitable to give 4A. The device has a male security connector to connect to a female charger plug.

The battery has to be recharged inside the robot. It is possible to connect the charger using the female connector located on the back of the waist, as shown in Figure 24.

The battery charger has three LED indicators and one power switch. The meanings of the LED indicators are described in the table below.

Charger Led	Color	Description
LED1	Red Light	Battery Charger Powered
LED2	Red Light	Battery is Charging
LED2	Green Light	Battery Full Charged

To charge the battery:

Figure 24: The charger's connector

1. Plug the battery charger to the electrical network, making sure that the switch is in the "0" position.
2. Plug the battery charger to the battery, as shown in Figure 25.
3. Turn the charger switch to position "1". The LED indicator will go into charging mode.

When the battery is charged:

1. Use the switch to turn off the battery charger.
2. Unplug the security connector from the robo,t as indicated in Figure 26.
3. Unplug the battery charger from the electrical network.

Figure 25: To plug in charger: plug and rotate right

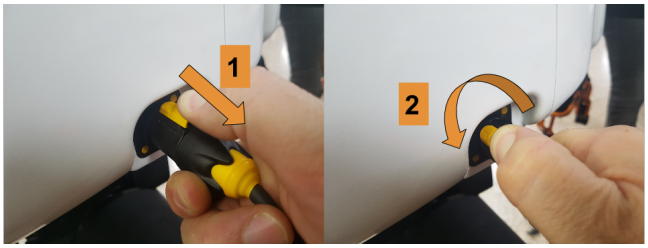


Figure 26: To unplug charger: pull yellow tab outwards and rotate left

10.3 Controls and protective devices

The robot has controllers on several levels.

Software controllers:

Software controllers are running in the robot´s control computer, which is connected to the controller boards. When the emergency button is released, the software controllers of the current mode are active.

On the top level, these controllers are using a hardware abstraction layer, which allows dynamic resource management to ensure that the different controllers are not taking ownership of the same resources: motors, in this case.

Hardware controllers:

Hardware controllers are running on dedicated controller boards and are connected to the motor modules.

Protective devices:

The robot has fuses distributed inside its power management system. These fuses are not accessible by the user.

Never try to replace a damaged fuse. Ask for technical support from PAL Robotics.

10.4 Fault identification and location

To avoid damaging the robot´s electrical, mechanical or structural components, it is important to locate any faults that occur by following the instructions and steps described in Section 13.

In case of any malfunction, first check that the issue is not produced by any mistake in your application. First, restore default parameters and configuration, then restart the operation. If the problem persists, contact PAL Robotics with a complete description, including:

- Description of the context before the fault detection.
- Context of the fault: changes in environment, hardware, software.
- Which parts of the robot are affected by the fault.
- Fault behavior.

In order to identify hardware problems, consult the output of the WebCommander tool, documented in section 20.

11 Network kick-off

11.1 Overview

This chapter explain how to initialize the development computer, the basestation and the network devices in order to access the robot.

11.2 Reserved IP address

The majority of communications between components onboard the TALOS happen via an on-board Ethernet network, referred to as the "Robot Internal Network".

This on-board network can be accessed directly via an Ethernet wired connection (located in the control panel), or by setting the wireless connection as Access Point. Additionally, the robot can be accessed via the basestation through a VPN tunnel.

There are three different network sub-nets used by PAL Robotics infrastructure, indicated below.

10.68.0.0/24: Robot Internal Network. The robot Ethernet Port and the robot´s wireless (when configured as Access Point) are directly connected to this network, allowing a user to connect their laptop to the robot´s internal network. The robot has an internal DHCP server that assigns IP addresses.

10.68.1.0/24: Robot VPN Network. The primary role of the basestation is to function as a VPN server for the robot. Each robot has two unique VPN addresses (one for the control computer and one for the multimedia computer).

> *All devices (robot and development computers) MUST be registered in the Basestation VPN server before trying to establish a connection with it. Otherwise, a reboot of the device will be required in order to reconnect to the VPN server.*

10.68.2.0/24: Automatic upgrade access. In order to have direct access to software upgrades, a connection between the Basestation and the PAL Robotics site is created when the Basestation has internet access available. To create a secure connection, an IP of this segment is used.

> *The IP addresses used in the building´s network MUST not use any of these three ranges, because it can interfere with the robot´s services.*

If needed, these IP ranges can be modified, but it would require PAL Robotics' assistance.

11.3 Basestation

Network interfaces The Ethernet cable must be connected to the eth0 port of the Basestation.

Network configuration The default IP address of the Basestation is *192.168.1.6*. This IP address can be changed in the file */etc/network/interfaces*. Changes are applied when the network is restarted or on the next reboot.

Another important network configuration is the NTP server.

The Basestation is used as the time reference for all the robots. It's important that the date of the Basestation is properly configured. The time synchronization is made using NTP protocol.

The default setup of the NTP uses the date of the Basestation as a time source reference, so the date and time of basestation must be correct (the Basestation is in UTC time). When checked, if it is not correct, it can be set using the *date* command. The NTP service then has to be restarted using the command *service NTP restart*.

If the basestation has connectivity with an external NTP server (internet servers or building network servers), it can be configured to use these external servers as time references. In order to use an external NTP server, the file */etc/ntp.conf should be edited.*

The default ntp.conf has the following configuration:

```
1 # You do need to talk to an NTP server or two (or three).
2 #server ntp.ubuntu.com
3 #Use own clock as server
4 server 127.127.1.0 burst minpoll 4 maxpoll 6
5 fudge 127.127.1.0 stratum 6
```

Listing 1: Default ntp configuration file

To use an external NTP server, uncomment (remove #) line 2 (ntp.ubuntu.com can be changed by the ntp server wanted), and then comment (add # at the beginning of the line) lines 4 and 5. This is required in order to avoid conflicts between the internal clock and the external NTP server.

Register robots and devices For all communication between the robots and devices (tablets and terminals), an encrypted VPN is created. The Basestation acts as a server for this VPN, meaning that all the robots and devices have to be registered in the Basestation in order to join it.

There is a set of tools in the Basestation used to register and unregister VPN users. These tools can only be executed as root.

For robots, there are two commands: *addRobotVpn* and *delRobotVpn.*

Below is an example of how to register a TALOS with serial number 1.

```
addRobotVpn -s 1 -r talos -i 6
```

For the other devices, the following commands must be used: *addVpnUser* and *delVpnUser*.

Below is an example of how to register a development computer desktop1.

```
addVpnUser -u desktop1 -i 23 -a peterDesktop
```

Executing the commands with the -h argument will show the parameters that can be used with these commands.

This command will be explained in more detail in section 18.

In this document it will be used as example a TALOS *robot with serial number 1.*

11.4 Robot

The Network configuration can be changed using a web browser connected to the robot´s control computer.

Plug any computer in to the Ethernet connector of the robot. This computer has to be configured with DHCP so the robot can assign it an IP address.

Enter the following address into a web browser to show the robot´s WebCommander.

```
http://control:8080
```

Configure the Network tab explained in the WebCommander section 20.

11.5 Development computer

Register computer
In order to enter the VPN, use the hostname of the computer. By default the hostname is *development*. It can be used as the user in the VPN but there can not be more computers with this hostname.

To modify the hostname, edit the file */etc/hostname* and the file */etc/hosts* using user root, then reboot.

Once the hostname is adjusted, it can be registered in the basestation (as indicated in the kick-off of the basestation).

Network Configuration
Network configuration is set using the standard NetworkManager in Ubuntu.

Remember that the terminal should be registered in the Basestation before trying to connect with it. A restart of the VPN will be needed if the connection is made before the registration.

Once the terminal is registered, indicate in the terminal the IP address of this basestation.

The IP address is configured in the file */etc/hosts*. This is the default value:

```
1  127.0.0.1       localhost
2  127.0.1.1       development
3
4  192.168.1.6 basestation_public
5
6  # The following lines are desirable for IPv6 capable hosts
7  ::1     localhost ip6-localhost ip6-loopback
8  fe00::0 ip6-localnet
9  ff00::0 ip6-mcastprefix
10 ff02::1 ip6-allnodes
11 ff02::2 ip6-allrouters
```

Listing 2: Default /etc/hosts configuration file

Change the IP address of basestation_public (line 4) for the current IP of the basestation, then restart the VPN.

```
service openvpn restart
```

NTP Configuration The default *ntp.conf* has the following configuration:

```
1  # Use servers from the NTP Pool Project. Approved by Ubuntu Technical Board
2  # on 2011-02-08 (LP: #104525). See http://www.pool.ntp.org/join.html for
3  # more information.
4  server 0.ubuntu.pool.ntp.org
5  server 1.ubuntu.pool.ntp.org
6  server 2.ubuntu.pool.ntp.org
7  server 3.ubuntu.pool.ntp.org
```

Listing 3: Default ntp configuration file

It can be adjusted in order to use the same NTP server indicated to the basestation, or to use the basestation as a server in this way:

```
1  server basestation
```

Listing 4: Default ntp configuration file

DNS Configuration The development computer has to be able to use the DNS domain name *reem*.

The best solution is configuring the DNS server of the building network to have the *reem* domain as a slave.

Another possibility is to use the Basestation for the reem domain and the building´s DNS server for everything else. See figure 27 for an example of this situation.

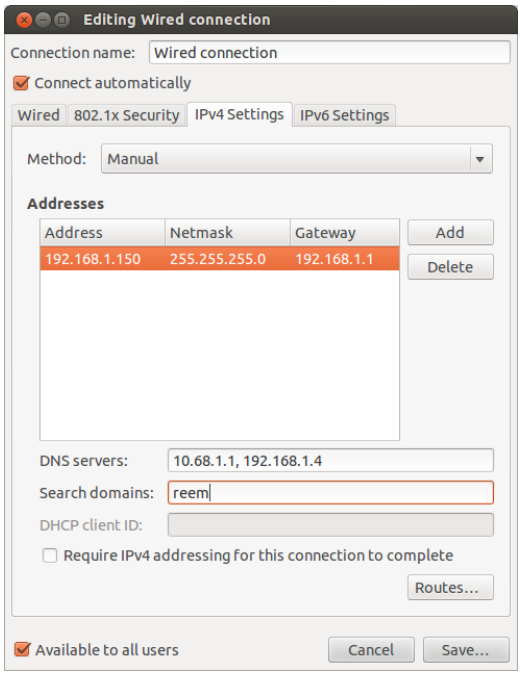


Figure 27: DNS configuration

12 Getting started

Before starting the robot, make sure that the user is familiar with the instructions regarding safety in Section 8, and the general usage information in Section 10.

12.1 Switch on the robot

1. Make sure that the emergency stop button is released, the robot is hanging on the crane and its feet do not touch the ground.

2. Turn on the robot by flipping the power switch. Check the battery level on the indicator and connect the charger if the battery level is red (30% or less)

3. Start the computers by pressing the PC ON button for two seconds.

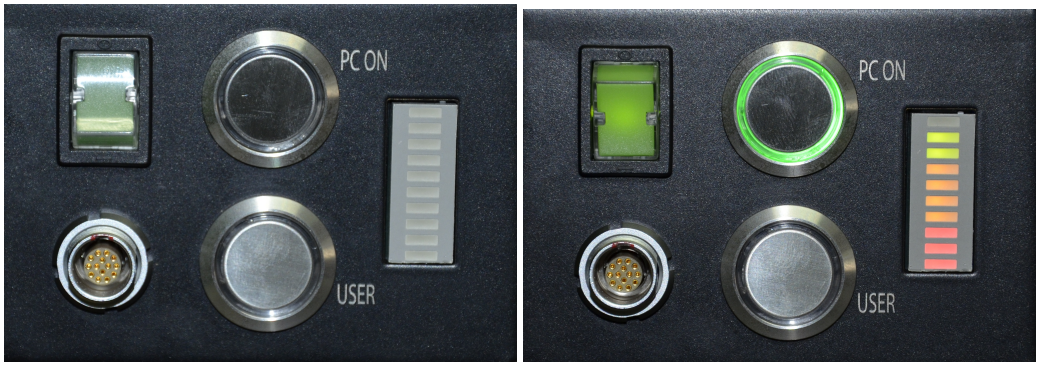


Figure 28: Power button states: off (left) and on (right)

4. Make sure that the emergency stop button is released. This applies power to the motors.

Figure 29: Emergency button states: pressed (left) and released (right)

5. Fans should start spinning and the power button´s light should change to green. Wait for the computers to boot up.

6. Verify that the motors and sensors are running using the WebCommander on page 81. The figure 30 shows the motors' status. If any of the motors do not display a green light, repeat the process by rebooting the robot. If the problem persists, contact PAL Robotics.

7. Now the robot can be operated.

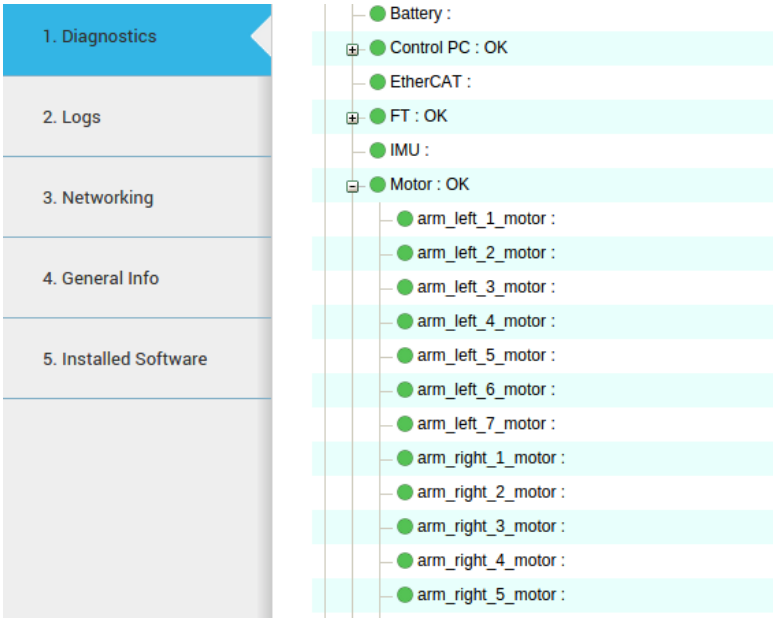


Figure 30: WebCommander motors status

12.2 Shutting down the robot

- Make sure the robot is secured on the crane.
- Hold the PC ON button until the light disappears. Now the computers are off.
- Flip the switch power switch to 0. When the green light turns off, the robot is off.
- Extend the legs and leave the robot secured on the crane and make sure that its feet touch the ground.

Maintenance

13 Nature and frequency of inspections

Performing daily inspection, periodic inspection and general maintenance can keep the robot´s performance stable for a long period.

13.1 Cleaning

- Clean the robot periodically with a damp cloth (not wet).
- Be extremely careful when cleaning the sensors.
- Do not spray the robot directly.

13.2 Daily inspection

Clean and maintain each component of the robot during everyday operations, while checking to see whether:

- there are any cracks or breaks in the components;
- there is any abnormal vibration, noise or heat generation in the motors

Check that each degree of freedom is running smoothly.

13.3 Once every three months

Inspect the following items every three months. Increase the frequency of inspections if the conditions under which the robot is being used have changed.

- Check that the cover retaining bolts or external bolts is not loose.
- Check the functionality of every joint by trying to move each one independently while paying attention to the response with a position reference (speed, etc...)

13.4 Once every year (by technical service)

The technical service will perform the following interventions at regular intervals of one year:

- Check that the motor connectors and other connectors are not loose.
- Check that the timing belts are not loose.
- Check the behaviour of all the joints. If the technical service detects any abnormal performance, or lack of smoothness, it will proceed with grease replacement.
- Clean and review the fans.

13.5 Once every three years (by technical service)

The technical service will perform the following interventions at regular intervals of three years:

- Check the grease conditions of the joints' gearboxes.
- Replace the computers' hard disks.
- Replace the computers' +3.3 Volts batteries.

14 De-commission, dismantling and disposal

At the end-of-life of the robot, to execute dismantling and disposal, contact Pal Robotics who will handle the electronic waste in an environmentally-friendly manner.

Disposal and recycling of the Lithium-Ion battery:

- Lithium-Ion batteries are subject to disposal and recycling regulations that vary by country and region.
- Always check and follow the applicable regulations before disposing of the batteries.
- The disposal of electronic equipment in standard waste receptacles is usually forbidden.
- Contact your local battery recycling organization.

Marking

15 Robot identification

The robot is identified by physical labels containing:

- Business name and full address
- Designation of the machine
- Serial number
- The year of construction

The following label (Figure 31) can be found on the robot and uniquely identifies the robot.

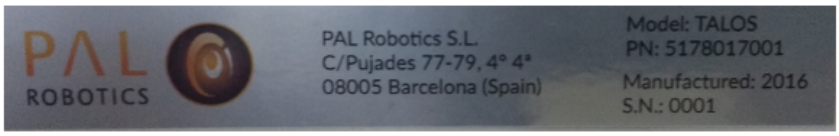


Figure 31: Identification label

Software Recovery

16 Software recovery

16.1 Overview

This section explains the Software installation procedure for TALOS .

16.2 Installation of the Basestation

To begin the installation process, connect a monitor, keyboard and mouse to the computer. This installation does not require internet connection.

BIOS configuration

Software installation

- Insert the Software USB drive
- Turn on the robot and press F2 repeatedly. Wait until BIOS menu appears.
- Enter the *BIOS Features menu* and select the Software USB drive as the *Boot Option #1* in *Hard Drive BBS Priorities.*
- Select the Software USB drive Boot Option #1.
- Press F10 and select Yes.
- The *Language* menu will pop up. Select *English*.

A menu similar to what is shown Figure 32 will appear.

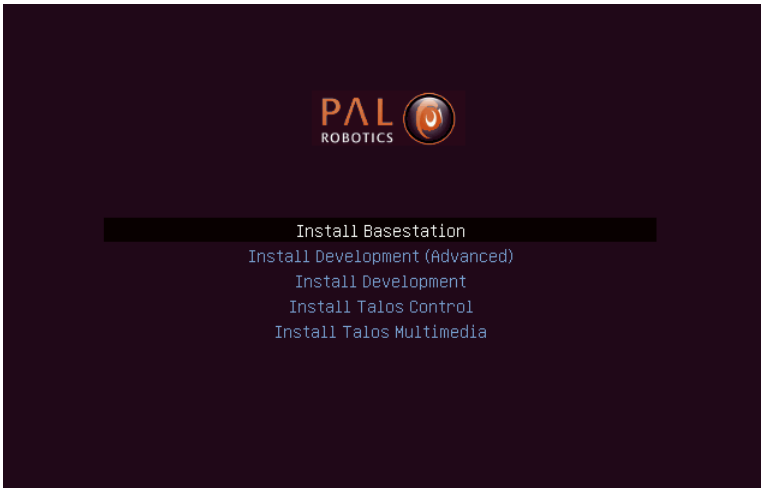


Figure 32: Software installation menu

- Select *Install Basestation*.
- Select the keyboard layout by following the instructions.

Default users The installation creates the following users:

- root: Default password is *palrootbasestation*.
- *pal*: Default password is *pal*.
- aptuser: Default password is *palaptuser*.

16.3 Installation of the robot's computers

TALOS has two computers called Control and Multimedia.

To begin the installation process, connect a monitor in the HDMI port and a USB keyboard to the corresponding computer.

Multimedia BIOS configuration Some options in the Multimedia BIOS computer must be configured as follows:

- Turn on the computer and press F2. Wait until the BIOS menu appears.
- Select the Software USB drive as the boot device.
- Save and exit the BIOS.

Control BIOS configuration Some options in the Control BIOS computer must be configured as follows:

- DO NOT select *Restore Defaults* in the *Save & Exit* menu as it would revert to the VGA graphics mode.
- Turn on the computer and press F2. Wait until the BIOS menu appears.
- In the *Advanced* menu, set Active Processor Cores to All.
- In the *Advanced* menu, set Hyper-Threading to Disabled.
- In the *Advanced* menu and Power Management submenu, set Intel SpeedStep to Disabled.
- In the *Advanced* menu and Power Management submenu, set C3, C6 and C7 Report to Disabled
- In the *ACPI* menu, set Hybernation Support as Disabled.
- In the *ACPI* menu, set ACPI Sleep State to Suspend Disabled.
- In Advanced menu, Graphics submenu, select HDMI/DVI in Digital Display Interface 1 (DDI1), Digital Display Interface 2 (DDI2) and Digital Display Interface 3 (DDI3)

Software installation

- Insert the Software USB drive.
- Select the Software USB drive as the boot device.
- Save and exit the BIOS.
- The *Language* menu will pop up. Select *English*.

A menu similar to what is shown in Figure 33 will appear.

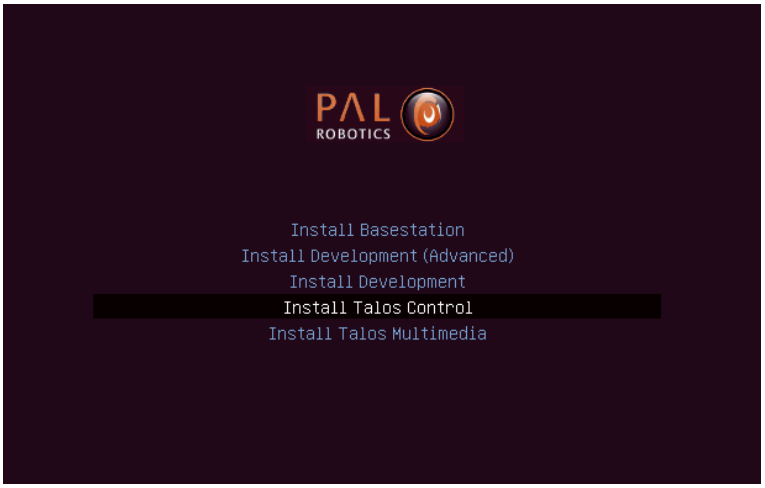


Figure 33: Software installation menu

- Select *Install* TALOS Control or Multimedia.

- Select the keyboard layout by following the instructions.

Control computer EtherCAT setup After the installation of the control computer, some more steps have to be carried out to enable the communication with the motor control boards.

- Log into the control computer and execute the following command to enable the EtherCAT drivers:

```
#rw
#chroot /ro
#depmod -a
#exit
#ro
```

- Customize the /etc/sysconfig/ethercat configuration file by adding the MAC address of the eth1 to the "MASTER0_DEVICE" variable and "e1000e" as the "DEVICE_MODULES" variable as shown in Figure 34.

```
#
# Master devices.
#
# The MASTER<X>_DEVICE variable specifies the Ethernet device for a master
# with index 'X'.
#
# Specify the MAC address (hexadecimal with colons) of the Ethernet device to
# use. Example: "00:00:08:44:ab:66"
#
# The broadcast address "ff:ff:ff:ff:ff:ff" has a special meaning: It tells
# the master to accept the first device offered by any Ethernet driver.
#
# The MASTER<X>_DEVICE variables also determine, how many masters will be
# created: A non-empty variable MASTER0_DEVICE will create one master, adding a
# non-empty variable MASTER1_DEVICE will create a second master, and so on.
#
MASTER0_DEVICE="" <------ (1)
#MASTER1_DEVICE=""

#
# Ethernet driver modules to use for EtherCAT operation.
#
# Specify a non-empty list of Ethernet drivers, that shall be used for EtherCAT
# operation.
#
# Except for the generic Ethernet driver module, the init script will try to
# unload the usual Ethernet driver modules in the list and replace them with
# the EtherCAT-capable ones. If a certain (EtherCAT-capable) driver is not
# found, a warning will appear.
#
# Possible values: 8139too, e100, e1000, e1000e, r8169, generic. Separate
# multiple drivers with spaces.
#
# Note: The e100, e1000, e1000e and r8169 drivers are not built by default.
# Enable them with the --enable-<driver> configure switches.
#
DEVICE_MODULES="" <------ (2)

#
# Flags for loading kernel modules.
#
# This can usually be left empty. Adjust this variable, if you have problems
# with module loading.
#
#MODPROBE_FLAGS="-b"
```

Figure 34: /etc/etherlab/sysconfig

Default users The installation creates the following users:

- root: Default password is *palroot*.
- *pal*: Default password is *pal*.
- aptuser: Default password is *palaptuser*.

16.4 Development computer installation

For the installation process connect a monitor, a keyboard and a mouse to the computer. This installation does not require internet connection.

Software installation

- Insert the Software USB drive.
- Turn on the computer and access the BIOS configuration menu.
- Configure BIOS to boot Software USB drive.
- Save changes and reboot computer.
- The *Language* menu will pop up. Select *English*.

A menu similar to what is shown in Figure 35 appears next.

Figure 35: Software installation menu

- Select the *Install Development* or *Install Development (Advanced)* option. The first is a fully automated installation what will format and partition the first drive in the computer while the later will let you choose drives, set partitions and configure GRUB. Choose the one that better suits your needs.
- Select the keyboard layout by following the instructions.

Default users The installation creates the following users:

- root: Default password is *palroot*.
- *pal*: Default password is *pal*.
- aptuser: Default password is *palaptuser*.

17 Generation of USB installation

17.1 Overview

This section specifies the USB flash drives generation procedure for installation from the ISO files.
Please contact PAL Robotics in order to download the ISOs from a PAL Robotics' FTP server.

17.2 Requirements

The following material is required to generate the USBs:

- The Software ISO image provided by PAL Robotics
- A bootable USB drive with at least 8GB of capacity
- Ubuntu computer

17.3 PAL-Robotics ISO

This section explains how to generate a bootable USB drive with the PAL-Robotics Software ISO.
The generation is made using an application called *usb-creator-gtk*. As *root* user type in a terminal:

```
usb-creator-gtk
```

The window shown in Figure 36 will appear.
Then perform the following sequence:

- Erase all the content in the USB memory by selecting the USB in the field *Disk to use* and press *Erase Disk*.
- Press the *Other* button and select Software ISO image.
- With the USB ready and the ISO selected, press *Make Startup Disk*.
- A window will indicate the progress. When this action is completed, the USB is ready.

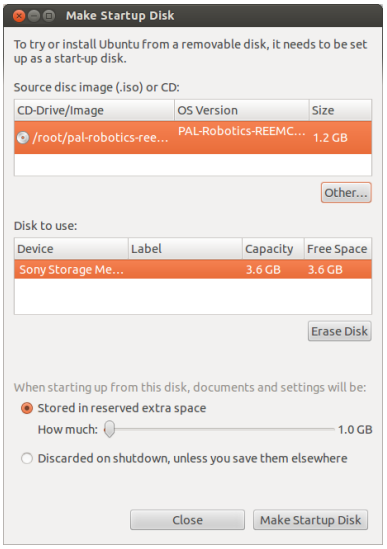


Figure 36: usb-creator-gtk application

Basestation

18 Basestation

18.1 Overview

The Basestation is the central element that coordinates communication between the different elements of the TALOS infrastructure.

It is also the element that stores all the applications and logs, and it receives software updates from PAL Robotics.

18.2 Users

The are three users in the Basestation:

- root: Default password is *palrootbasestation*.
- *pal*: Default password is *pal*. This is the default user for the execution of applications.
- aptuser: Default password is *palaptuser*. This user allows connections between elements without a password. It is used, for example, for collection of logs or debian installation.

18.3 Network configuration

By default, the Basestation is configured with the IP 192.168.1.6. It can be changed by editing the file */etc/network/interfaces*:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 192.168.1.6
    netmask 255.255.255.0
    gateway 192.168.1.1
    dns-search reem
    dns-nameservers 127.0.0.1
```

Listing 5: Default network configuration

The IP address of the Basestation will be configured in all the robots and devices that are connected with it. It is an IP address of the building´s network.

The DNS parameters (*search* and *nameservers*) should not be changed because they are configured to be its own server.

18.4 NTP server

The Basestation is used as the time reference for all the robots. It is important that the date on the Basestation is properly configured. The time synchronization is made using NTP protocol.

The default setup of the NTP uses the date of the Basestation as a time source reference. The time in the Basestation is in UTC). If, after checking, the date is not correct, it can be set using *date* command. The NTP service then has to be restarted using the command *service NTP restart*.

If the Basestation has connectivity with an external NTP server (internet servers or building network servers), it can be configured to use these external servers as time references. The file */etc/ntp.conf* should be edited

The default ntp.conf has the following configuration:

```
1 # You do need to talk to an NTP server or two (or three).
2 #server ntp.ubuntu.com
3 #Use own clock as server
4 server 127.127.1.0 burst minpoll 4 maxpoll 6
5 fudge 127.127.1.0 stratum 6
```

Listing 6: Default ntp configuration file

Uncomment (remove #) line 2 (ntp.ubuntu.com can be changed by the ntp server wanted), and comment (add # at the beginning of the line) lines 4 and 5. This is needed in order to avoid conflicts between the internal clock and the external NTP server.

18.5 OpenVPN

The Basestation creates a Virtual Private Network (VPN) with the robots and all the elements that work with them.

This is the only allowed communication; a firewall is set in order to prevent unprotected connections.

For the robots and development computers, an OpenVPN network is used (http://openvpn.net/index.php/open-source.html), encrypted with AES-128.

By default the VPN uses an IP with the range 10.68.1.X/24[1]. It is possible to use other ranges, but the operator will need to contact PAL Robotics for more information.

18.5.1 Add an element to the VPN

There are two commands that can be used to add elements to the VPN (one that is specific for robots and one for everything else).

- For the robot, use the command *addRobotVpn:*

```
root@basestation:~# addRobotVpn
-h          shows this help
-s SERIAL   serial number of robot
-r ROBOT    robot type
-i NUM      last value of ip address for this user.  will ser NUM and NUM+1
-a ALIAS    alias name to add to DNS additionaly to ROBOT-SERIAL [OPTIONAL]

Examples: addRobotVpn -s 123456789 -r reemc -i 4 [-a palRobot]
              It creates two vpn users: reemc-123456789c and reemc-123456789m
              Ip address will be 4 and 5
              DNS entrance are reemc-123456789c and reemc-123456789m
              [ DNS alias are palRobotc and palRobotm ].
```

The robot type (*reemc*, *reemh3*, *talos*...), the serial number and the final number of the IP address have to be provided. The example shows user reemh3-123456789c and user reemh3-123456789m for the control and multimedia computers of a REEM. The IP addresses assigned are 10.68.1.4 and 10.68.1.5.

The robot can also be configured to have an alias so it's easier to remember. Add a *c* and an *m* at the end of the indicated alias to distinguish control from the multimedia computer.

- For the other elements of the network (including the development computer) use the command *addVpnUser*:

[1]This range is shared with the PPTP and L2TP/IPsec PSK connections. It has to be taken in consideration when assigning IP addresses.

```
root@basestation:~# addVpnUser
-h          shows this help
-u USER     user name for vpn client and DNS
-i NUM      last value of ip address for this user
-d DNSNAME  force a DNS name instead of USER [OPTIONAL]
-a ALIAS    alias name to add to DNS additionaly to USER [OPTIONAL]
-n          not add to the upgrade list (for Android clients) [OPTIONAL]
Examples: addVpnUser -u desktop1 -i 23
          addVpnUser -u desktop1 -i 23 -a peterDesktop.
```

The *hostname* of the computer and the final number of the IP address should be provided. It should also be possible to force the use of another name for the DNS or to add an alias. Finally there is an option to not add the robot to the upgrade system.

18.5.2 Remove an element from the VPN

As before, there are two commands to remove elements from the VPN: one that is specific for robots and one for everything else.

- Robot: use the command *delRobotVpn*:

```
root@basestation:~# delRobotVpn
-h          shows this help
-r robot    robot type
-s SERIAL   serial number of robot to delete

Examples: delRobotVpn -s 123456789 -r reemh3.
```

The robot type (reemc, reemh3...) and the serial number should be provided.

- Development computer: for the other elements, use the command *delVpnUser*:

```
root@basestation:~# delVpnUser
-h          shows this help
-u USER     user name of vpn to delete
-n          not delete of the upgrade list (for Android clients) [OPTIONAL]

Examples: delVpnUser -u desktop1.
```

In this case, only the name of the client is needed.

18.5.3 Copying files between VPN users

They are tools to copy (or move) files from one VPN client to other (including the Basestation) without a password.

Only the user *aptuser* has keys configured to enable *ssh* without a password, so files will be sent and received to this user.

User *pal* and user *aptuser* share the group *users*, allowing the applications (normally using *the user named PAL*) to authorize or deny the network to copy files, or to give group permissions to a file or directory.

There are two commands for making network copies:

- The command copyToVpnUser allows files or directories to be sent to another VPN user. The original file will be removed automatically once it is sent correctly, if indicated as an option. For users *pal* and *aptuser* this command has to be performed with *sudo*, but no password is required.

```
pal@basestation:~$sudo copyToVpnUser
copyToVpnClient performs a rsync -rlzve from the local path
to the destination path of the indicated vpn client.
Optionaly, it can be indicated to remove the source files adding remove-source
Sintaxis: copyToVpnUser localPath vpnClient destinationPath [remove-source]
Example: copyToVpnUser /home/pal/Desktop reemh3-2c /mnt_flash/ remove-source
```

- The command *copyFromVpnUser* enables files to be received from another VPN user. As with the command before, the original file can be removed automatically. For users *pal* and *aptuser* this command has to be performed with *sudo*, but no password is required.

```
pal@basestation:~$sudo copyFromVpnUser
copyFromVpnClient performs a rsync -rlzve from the remote path
of the indicated vpn client to a local path.
Optionaly, it can be indicated to remove the source files adding remove-source
Sintaxis: copyFromVpnUser remotePath vpnClient destinationPath [remove-source]
Example: copyFromVpnUser  /mnt_flash reemh3-2c /home/pal/Desktop remove-source
```

18.6 DNS Server

The Basestation has a DNS server. It is configured for the VPN network with the domain *reem*.

When a VPN user is created, it is automatically added to the DNS, but a new alias can be created or removed for this DNS entrance.

These commands are:

- addDnsAlias: Ceates a new alias for an existing DNS entrance.

```
root@basestation:~# addDnsAlias
-h          shows this help
-d DNSNAME  dns name to add alias
-a ALIAS    new alias for the dns name

Example: addDnsAlias -d basestation -a basesation.archives.
```

- delDnsAlias: Removes an existing alias from the DNS server.

```
root@basestation:~# delDnsAlias
-h          shows this help
-a ALIAS    alias to remove

Example: delDnsAlias -a basesation.archives.
```

Transfer of the domain *reem* to other DNS servers is enabled by default. The DNS servers of the operator´s facilities can have the domain *reem*.

18.7 PPTP Connections

If using a Windows machine, it is possible to access the robot by connecting to the Basestation using a PPTP connection.

18.7.1 Address pool

The configuration of the PPTP daemon can be found in: *etc/pptpd.conf.*

Unlike OpenVPN, there is no static IP assignment. Here it is defined as a pool of IP addresses.

```
1 #
2 # (Recommended)
3 localip 10.68.1.249
4 remoteip 10.68.1.240-248
5 # or
6 #localip 192.168.0.234-238,192.168.0.245
7 #remoteip 192.168.1.234-238,192.168.1.245
```

Listing 7: PPTP address pool

By default, the Basestation will assign for itself the IP *10.68.1.249.* There is a pool of nine IP addresses (from *10.68.1.240* to *10.68.1.248*).

If more addresses are needed, simply change the *remoteip* entrance in the file.

18.7.2 Security

Security configurations can be found in */etc/ppp/pptpd-options*. It is configured to work with Windows.

User and password are defined in /etc/ppp/chap-secrets:

```
1 # Secrets for authentication using CHAP
2 # client server secret IP addresses
3 reem l2tpd pal 10.68.1.1/24
4 l2tpd reem pal 10.68.1.1/24
5 reem pptpd pal *
```

Listing 8: PPTP default user

Line 5 defines the default user: *reem* and the default password: *pal*.

18.7.3 DNS settings

The device that connects to the basestation can have access to the DNS server. These devices have to be configured in order to use the basestation as a server and to use the domain reem to connect to robots.

18.8 Software repositories

The Basestation has repositories with the software provided by PAL Robotics and a repository in order to allow the customer to add their debian packages. These repositories can be found inside directory */srv/upgrade* and are visible for all the elements of the VPN.

There are tools for creation and maintenance of more repositories, but they have to be added to the VPN clients by adding a new file in */etc/apt/sources.list.d/*.

18.8.1 PAL Robotics 's repositories

There are two repositories with software provided and maintained by PAL Robotics.

If the Basestation has internet connection, a VPN tunnel is created with PAL Robotics 's corporate Basestation. This tunnel is used for automatic updates of the repositories.

These repositories are updated every night (this function can be changed by editing */etc/cron.d/apt-mirror*). A synchronization can be forced by executing the command:

```
root@basestation:~# sync_repository
```

aptuser can execute this application without password using *sudo*:

```
aptuser@basestation:~$ sudo sync_repository
```

The repositories are:

- pal-system: contains the basic system and communication software.
- pal-stable: robotic framework and device applications.

18.8.2 Customer software repository

In the same directory as the other repositories, the repository *pal-staging* can be found. This repository is visible by all VPN clients, is not maintained by PAL Robotics and is empty by default.

Customers can add or remove debian packages with their own applications. Once a debian is added or modified the repository has to be refreshed.

This is done using the command *refresh_repository*:

```
root@basestation:~# refresh_repository -h
refresh_repository <dir_of_repository>
```

aptuser can execute this application without a password using *sudo*.

More repositories can be created by *root* and then manually configured in the clients' VPN, using the command *create_repository*:

```
root@basestation:~# create_repository -h
-h          shows this help
-d PATH     directory to create the repository
-l LABEL    description of the repository

Example: create_repository -d directory -l label.
```

If the repository is created in the same directory level as *pal-system*, it will be automatically visible with apache. If not, apache has to be configured.

18.9 System upgrade

In order to update software, there is a tool based in synaptic that allows updates of multiple devices at the same time.

The tool is called *pal-upgrade-synaptic* and can be executed by *root* or *aptuser* using *sudo*. Only one session of *pal-ugrade-synaptic* can be run at the same time.

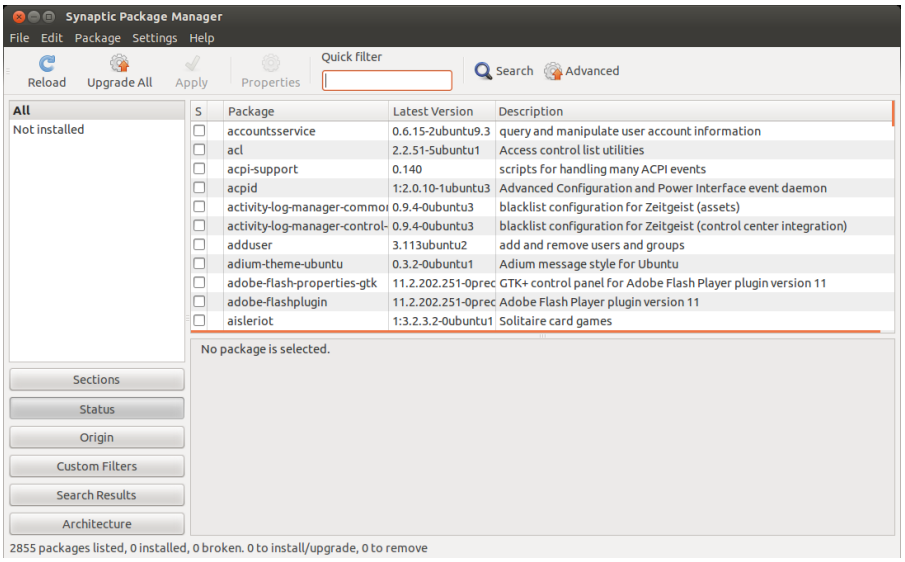


Figure 37: pal-upgrade-synaptic application

As shown in Figure 37, none of the debian packages included in the repositories explained in section 18.8 are installed.

Select the software to be installed (it wil automatically install the latest available), then press *Apply*.

Remember that a particular version can be forced using the *Package* menu.

When the confirmation window appears, press *Apply* again and a new window with the VPN clients currently connected to the network will appear.

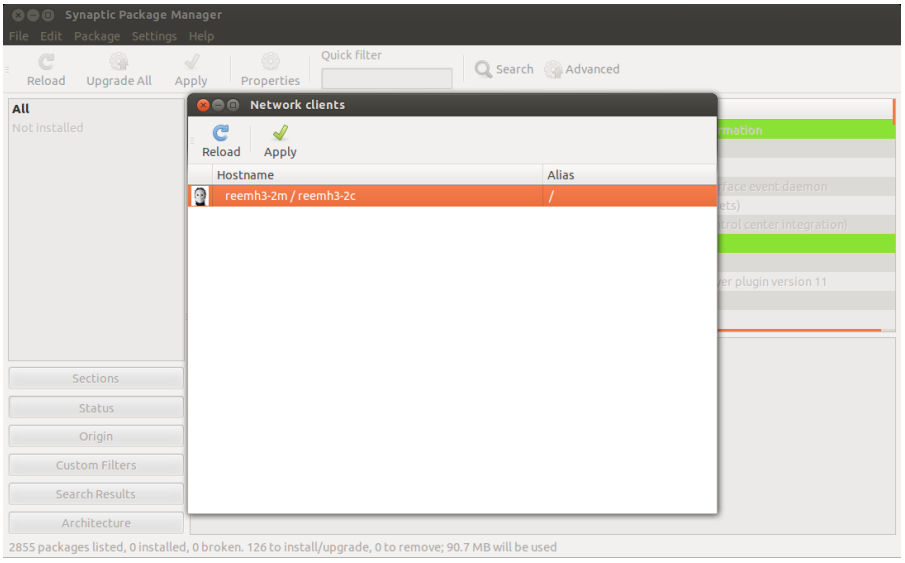


Figure 38: pal-upgrade-synaptic client selections

In this window (shown in Figure 38) select the robots and devices (use the Ctrl button for multiple selection).

The Reload button refreshes the list of connected clients.

Once the clients are selected and the *Apply* button is pressed, the system upgrade will begin.

The system connects to the selected devices and tests that there is a previous version of the software to upgrade. If there was a previous version it will install the new one, but if there wasn't the debian will not be installed.

Only previously existing debian packages can be upgraded directly. This is a security policy in order to avoid the installation of debian packages of one device that can produce a malfunction in other device (for example, debian packages of a control computer in a multimedia computer of a TALOS).

For a general upgrade of all the debians, there is a button labeled *Upgrade All* (in Figure 37). When pressed, the network clients will open and multiple robots and/or devices can be selected. When *Apply* is selected (Figure 38) the same process will begin as before, but with all the debians installed.

For complete control of installation and/or de-installation of debian packages, use the *Advance* button. When this button is pressed a window like the one shown in Figure 38 will appear, but only one device at a time can be selected. When the *Apply* button is pressed, a remote synaptic of the selected device will open. In this synaptic, installations and de-installations can be performed. In robots, the synaptic of the multimedia computer will open first, and when it is closed the synaptic of the control computer will open.

PAL Robotics devices connected to the VPN can access this tool by typing *pal-upgrade-synaptic* in the *Dash*. This command connects to the Basestation and opens the application remotely.

18.10 PAL Robotics 's Corporate Basestation

If the Basestation has internet access, an encrypted tunnel is created between the customer Basestation and the PAL Robotics Corporate Basestation.

This connection is used for the software updates indicated in section 18.8, as it allows remote support from PAL Robotics.

If there is no internet connection available or this service is not wanted, all files which do not use this tunnel will be encrypted.

This allows both sides to authenticate the file.

- Receiving files from PAL Robotics:

  As *root* user in the basestation, the file can be decrypted using this command:

  ```
  root@basestation:~# gpg -decrypt filename.gpg
  ```

- Sending files to PAL Robotics:

  For encryption of a file in the basestation, use this command as *root*:

  ```
  root@basestation:~# gpg -encrypt filename
  ```

18.11 DHCP server

By default the Basestation hasn't installed a DHCP sever in order to avoid problems with the Building's network.

A DHCP server debian is included in the software, so *the root* user can install it by executing:

```
root@basestation:~# apt-get install isc-dhcp-server
```

The configuration file can be found in */etc/dhcp/dhcpd.conf*.

Robot Computers

19 TALOS Robot's Internal Computers

19.1 TALOS LAN

19.2 Users

The users and default passwords in the TALOS computers are:

- root: Default password is *palroot*.
- pal: Default password is *pal*.
- aptuser: Default password is *palaptuser*.

As explained in section 18.5.3 for the Basestation, the computers use the same command to copy files to other VPN users.

The same commands can also be used to copy files between computers inside the internal LAN.

19.3 File system

The TALOS robot's computer has a protection against power failures that could corrupt the filesystem.

These partitions are created:

- / : This is an *union* partition, the disk is mounted in */ro* directory as read-only and all the changes are stored in RAM. So, all the changes are not persistent between reboots.
- /home : This partition is read-write. Changes are persistent between reboots.
- /var/log : This partition is read-write. Changes are persistent between reboots.

In order to work with the filesystem as read-write do the following:

```
root@talos-1c:~# rw
Remounting as rw...
Mounting /ro as read-write
Binding system files...
root@talos-1c:~# chroot /ro
```

rw command remounts all the partitions as read-write. Then with a *chroot* to */ro* we have the same system than the default but all writable. All the changes performed will be persistent.

In order to return to the previous state do the following:

```
root@talos-1c:~# exit
root@talos-1c:~# ro
Remount /ro as read only
Unbinding system files
```

First *exit* command returns from the *chroot*. Then the *ro* script remounts the partitions in the default way.

19.4 Internal DNS

The control computer has a DNS server that is used for the internal LAN of the TALOS with the domain name *reem-lan*. This DNS server is used by all the computers connected to the LAN.

When a computer is added to the internal LAN (using the Ethernet connector, for example) it can be added to the internal DNS with the command *addLocalDns*:

```
root@talos-1c:~# addLocalDns -h
-h shows this help
-u DNSNAME dns name to add
-i IP ip address for this name

Example: addLocalDns -u terminal -i 10.68.0.220.
```

The same command can be used to modify the IP of a name: if the *dnsname* exists in the local DNS, the IP address is updated.

To remove names in the local DNS, exit the command *delLocalDns*:

```
root@talos-1c:~# delLocalDns -h
-h          shows this help
-u DNSNAME  dns name to remove

Example: addLocalDns -u terminal
```

These additions and removals in the local DNS are not persistent between reboots.

19.5 NTP

Since big jumps in the local time can have undesired effects on the robot applications, NTP is setup when the robot starts and before the ROS master is initiated. If no synchronization was possible, for example if the NTP servers are offline, the NTP daemon is stopped after a timeout.

To setup ntp as client edit the `/etc/ntp.conf` file and add your desired ntp servers. You can use your own local time servers or external ones, such as ntp.ubuntu.com. You can also try uncommenting the default servers already present. For example, if the local time server is in 192.168.1.6 add the following to the configuration file.

```
server 192.168.1.6 iburst
```

Restart the ntp daemon to test your servers.

```
systemctl restart ntp.service
```

Run the `ntpq -p` command and check that at least one of the configured servers has a nonzero *reach* value and a nonzero *offset* value. The corrected date can be consulted with the `date` command. Once the desired configuration is working make sure to make the changes in `/etc/ntp.conf` persistant and reboot the robot.

If, on the contrary, you want the robot to act as the NTP server of your network, no changes are needed. The current ntp daemon already acts as server. You will only need to configure NTP for the clients.

To configure NTP on the rest of the clients, like the development PCs, run:

```
systemctl status ntp.service
```

If the service is *active* follow the previous steps to configure the ntp daemon. Once again a private or public NTP server can be used. If, instead the robot is desired as server add this line to `/etc/ntp.conf`.

```
server tiago-Xc iburst
```

If the service is *not found* then that means ntp is not installed. Either install it with `apt-get install ntp` or make use of Ubuntu's default ntp client called `timesyncd`.

To configure `timesyncd` simply edit the `/etc/systemd/timesyncd.conf` file and set the proper NTP server.

Restart the `timesyncd` daemon.

```
systemctl restart systemd-timesyncd.service
```

Check the corrected date with the `date` command. The time update can take a few seconds.

19.6 System upgrade

For performing system upgrades connect to the robot, make sure you have Internet access and run the *pal_upgrade* command as *root* user.

This will install the latest TALOS software available from the PAL repositories.

Reboot after upgrade is complete.

19.7 Meltdown and Spectre vulnerabilities

Meltdown and Spectre exploit critical vulnerabilities in modern processors.

Fortunately the linux Kernel has been patched to mitigate these vulnerabilities, this mitigation comes at a slight performance cost.

PAL Robotics configuration does not interfere with mitigation, whenever the installed kernel provides mitigation, it is not disable by our software configuration.

Below we provide some guidelines to disable the mitigation in order to recover the lost performance, this is **not** recommended by PAL Robotics and it is done on the customer's own risk.

On this website the different tunables for disabling mitigation controls are displayed.

These kernel flags must be applied to the GRUB_CMDLINE_LINUX in `/etc/default/grub`. After changing them, `update-grub` must be executed, and the computer must be rebooted.

These changes need to be made in the persistent partition, as indicated in 19.3

Be extremely careful when performing these changes, since they can prevent the system from booting properly.

Developer's Manual

20 WebCommander

The WebCommander is a web page hosted by TALOS. It can be accessed from any modern web browser that is able to connect to TALOS.

It is an entry point for several monitoring and configuration tasks that require a Graphical User Interface (GUI).

20.1 Accessing the WebCommander website

1. Ensure that the device you want to use to access the website is in the same network and able to connect to TALOS .
2. Open a web browser and type in the address bar the host name or IP address of TALOS's control computer and try to access port 8080:

   ```
   http://talos-1c:8080
   ```

3. If you are connected directly to TALOS, when the robot is acting as an access point, you can also use:

   ```
   http://control:8080
   ```

20.2 Overview

The WebCommander website contains visualizations of the state of TALOS's hardware, applications and installed libraries, as well as tools to configure elements of its behaviour.

20.3 Default tabs

TALOS comes with a set of preprogrammed tabs that are described in this section, these tabs can also be modified and extended, as explained in the 20.4 section. Each tab is an instantiation of a web commander plugin.

For each tab a description and the plugin type used to create it is defined.

20.3.1 Startup tab

Plugin Startup

Description Displays the list of PAL software that is configured to be started in the robot, and whether it has been started or not.

Each application, or group of applications, that provides a functionality, can choose to specify a startup dependency on other applications or groups of applications. There are three possible states:

- Green: All dependencies satisfied, application launched.
- Yellow: One or more dependencies missing or in error state, but within reasonable time. Application not launched.
- Red: One or more dependencies missing or in error state, and maximum wait time elapsed. Application not launched.

Additionally, there are two buttons on the right of each application. If the application is running, a “Stop” button is be displayed, which will stop the application when pressed. If the application is stopped or has crashed, the button “Start” is be displayed, which will start the application when pressed. The “Show Log” button, allows to display the log of the application.

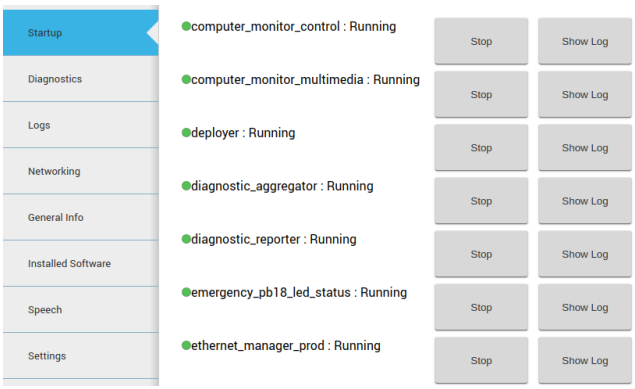


Figure 39: The Startup Tab displays the launched applications and their dependencies

20.3.2 Diagnostics Tab

Plugin Diagnostics

Description Displays the current status of TALOS’s hardware and software.

The data is organized in an hierarchical tree. The first level contains the hardware and functionality categories.

The functionalities are the software elements that run in TALOS, such as vision or text to speech applications.

Hardware diagnostics contain the hardware’s status, readings and possible errors.

Inside the hardware and functionality categories, there’s an entry for each individual functionality or device. Some devices are grouped together (motors, sonars), but each device can still be seen in detail.

The color of the dots indicates the status of the application or component.

- Green: No errors detected.
- Yellow: One or more anomalies detected, but they are not critical.
- Red: One or more errors were detected which can affect the behavior of the robot.
- Black: Stale, no information about the status is being provided.

An example of this display is shown in Figure 40. The status of a particular category can be expanded by clicking on the “+” symbol to the left of the name of the category. This will provide information specific to the device or functionality. If there’s an error, an error code will be shown.

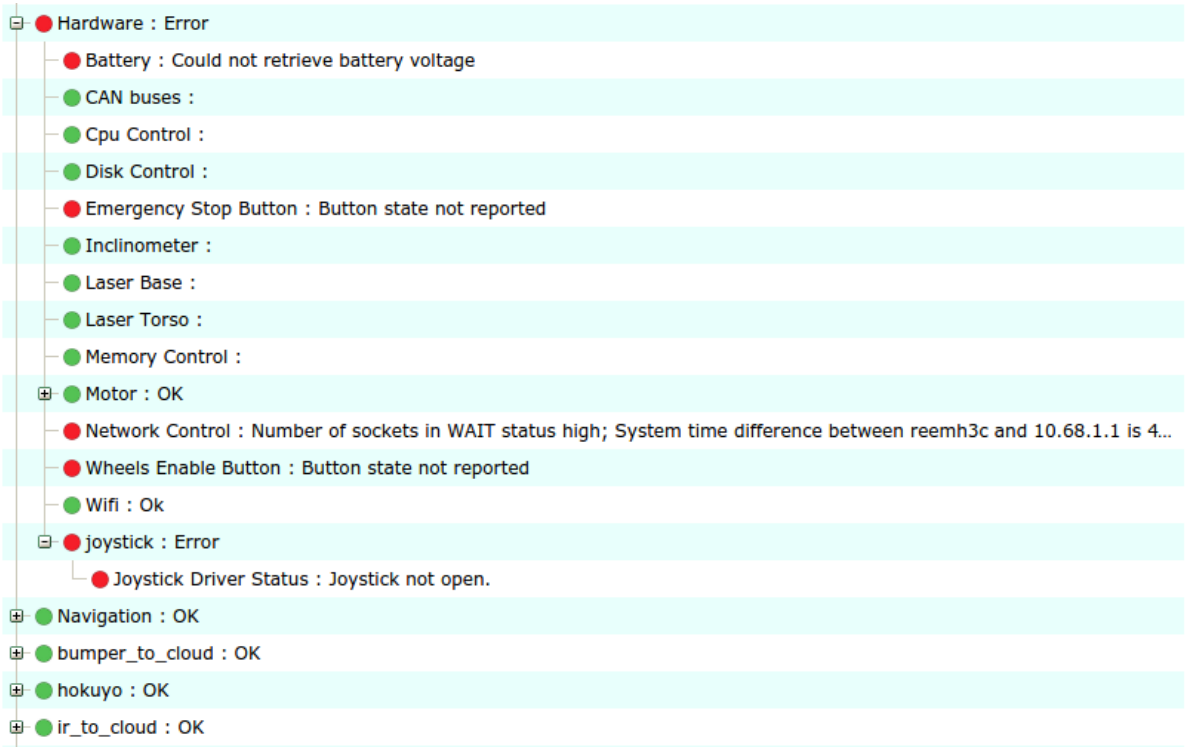


Figure 40: The Diagnostics tab displays the status of the hardware and software components of TALOS

20.3.3 Logs Tab

Plugin Logs

Description Displays the latest messages printed by the applications' logging system.

The logs are grouped by severity levels, from high to low severity: *Fatal*, *Error*, *Warn*, *Info* and *Debug*.

The logs are updated in real time, but messages printed before opening the tab can't be displayed.

The log tab has different check-boxes to filter the severity of the messages that are displayed. Disabling a priority level will also disable all the levels below it, but they can be manually enabled. For instance, unchecking *Error* will also uncheck the *Warn*, *Info* and *Debug* levels, but the user can click on any of them to reenable them.

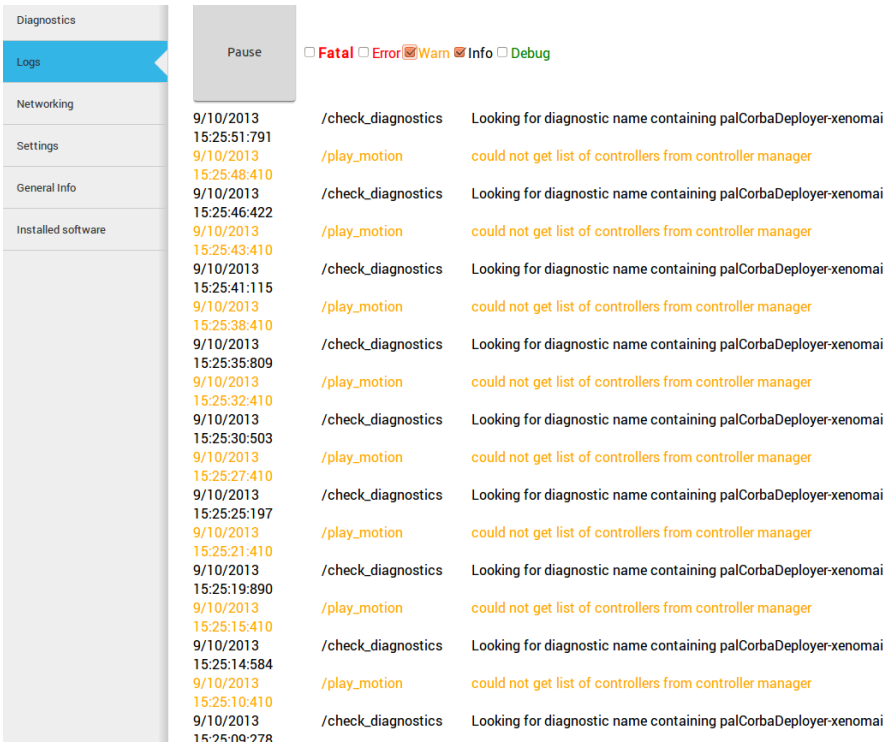


Figure 41: The Log Tab displays the log messages as they are being published in the robot

20.3.4 General Info Tab

Plugin General Info

Description Displays the robot model, part number and serial number.

20.3.5 Installed Software Tab

Plugin Installed Software

Description Displays the list of all the software packages installed in both the robot's computers.

20.3.6 Networking Tab

Plugin Networking

Description Enables the configuration of TALOS's networking .

Figure 42 shows the Networking Tab. By default the controls for changing the configuration are not visible in order to avoid access by multiple users.

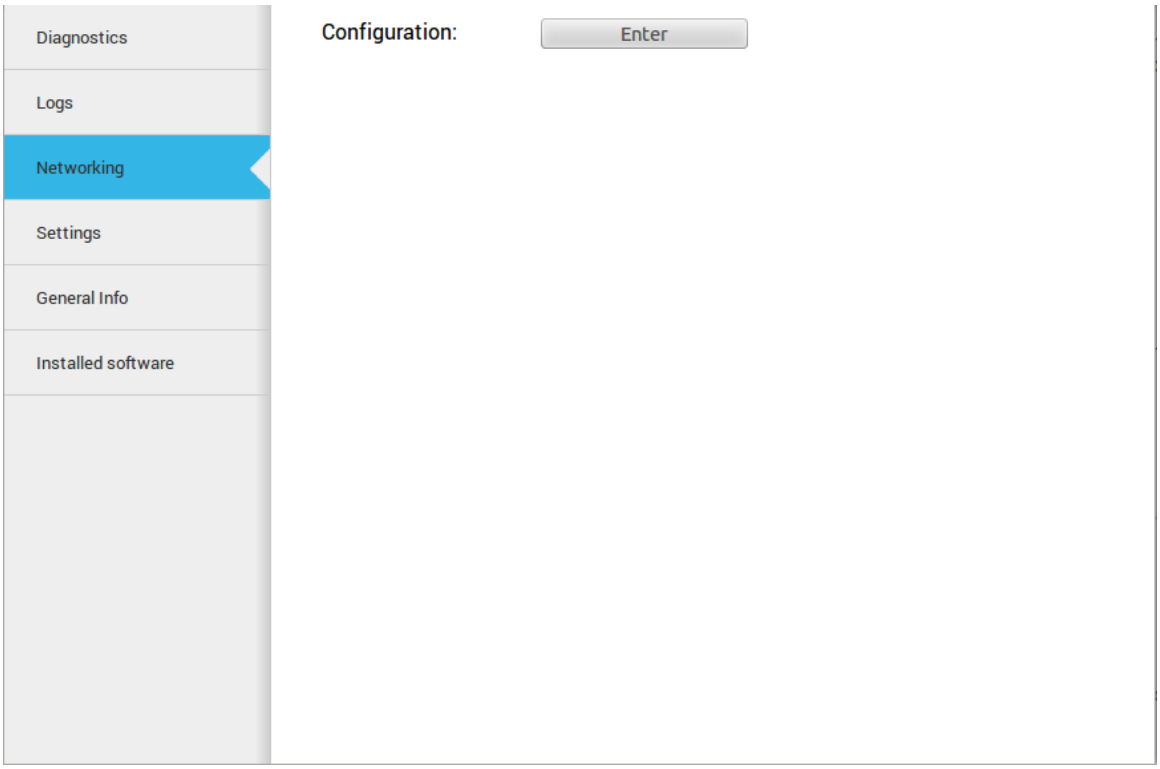


Figure 42: Networking configuration

If *Enter* button is pressed, the Tab connects to the network configuration system and the controls shown in Figure 43 will appear.

When a user connects to the configuration system, all the current clients are disconnected and a message is shown in the status line.

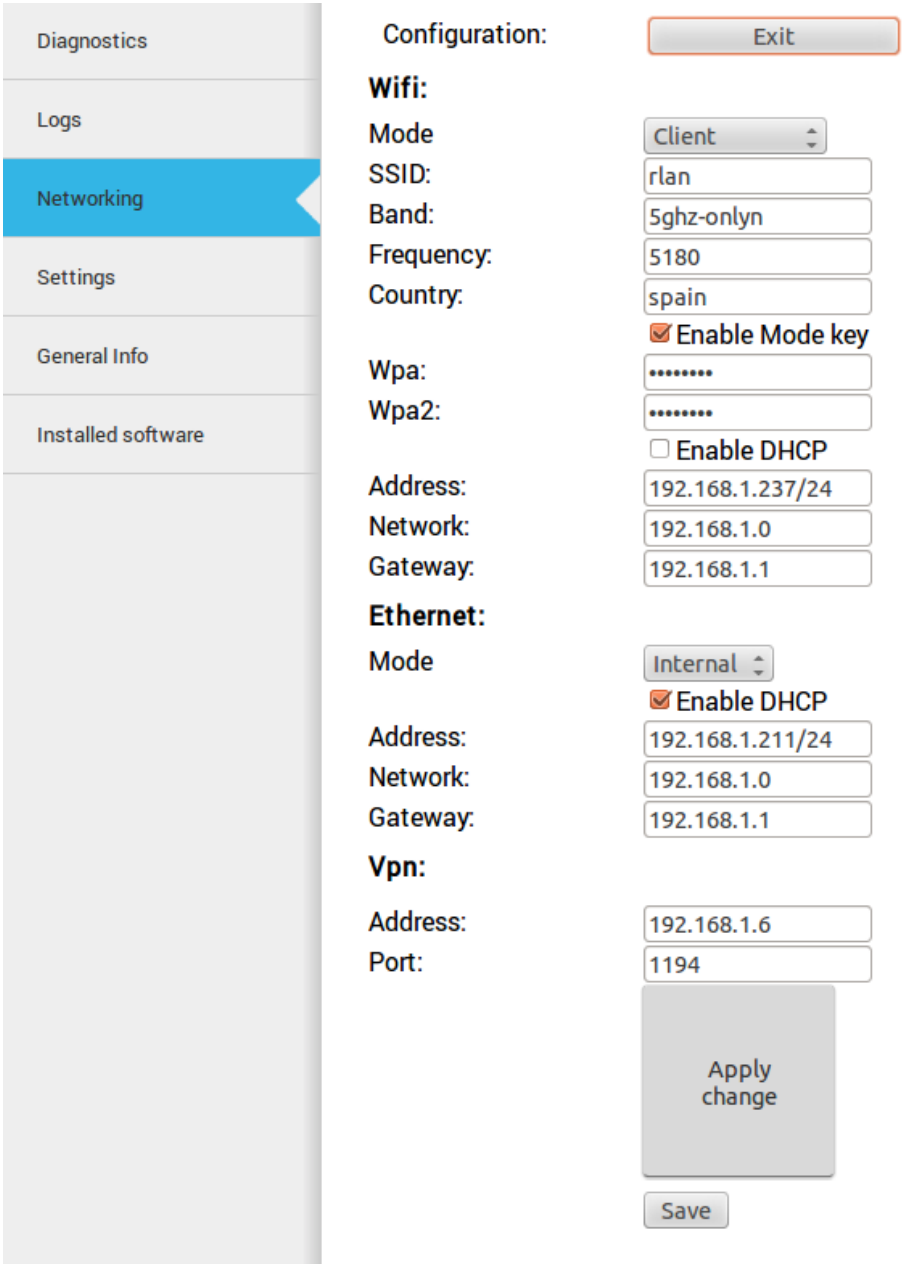


Figure 43: Networking configuration controls

Configurations are separated into three blocks:

- Wifi:
  - Mode: Can be selected whether Wifi connection works as Client or Access Point.
  - SSID: ID of the Wifi to connect to Client mode or to publish in Access Point mode.
  - Band: Available options are: *2ghz-b/g/n, 2ghz-onlyn, 5ghz-a/n, 5ghz-onlyn.*
  - Frequency: To be used only when it is configured as Access Point (not in Client mode).
  - Country: Regulation domain of the Wifi connection.
  - Enable Mode key: Select if the Wifi connection is with or without encryption.
  - Wpa: Password for WPA encryption.

  - Wpa2: Password for WPA2 encryption.
  - Enable DHCP: In Client mode, use DHCP to obtain the IP address of the building network.
  - Address, Network, Gateway: In Client mode, manual values of the building network will be used by the Wifi interface.

- Ethernet:
  - Mode: Can be selected whether the Ethernet connection works as an Internal LAN or VPN connection.
  - Enable DHCP: In VPN mode, use DHCP to obtain the IP address of the building network.
  - Address, Network, Gateway: Manual values of the building network to be used by the Ethernet interface (when in VPN mode).

- VPN
  - Address: Building network IP address of the Basestation.
  - Port: Port of the Basestation where the VPN server is listening.

The changes are not set until the *Apply change* button is pressed.

When the *Save* button is pressed (and confirmed), the current configuration is stored in the hard disk.

> Be sure the networking configuration is correct before saving it. A bad configuration can make it impossible to reconnect with the robot. If this happens, a general reinstallation is required.

Using the Diagnostics Tab, it is possible to see the current state of the Wifi connection.

As shown in Figure 44, click on *Components/hardware/Wifi* in the *Diagnostics* tab, to see the current Wifi state.

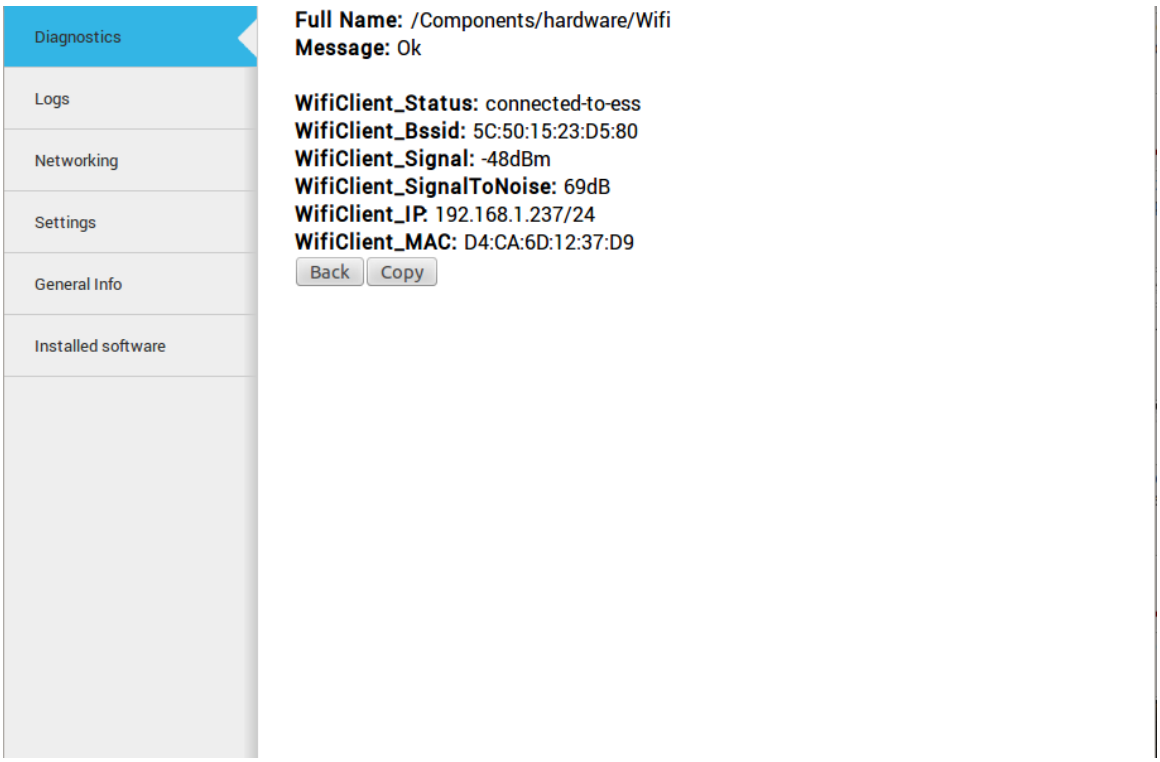


Figure 44: Networking Wifi diagnostics

And to have information about the state of the Ethernet click on *Components/hardware/Ethernet* (Figure 45).

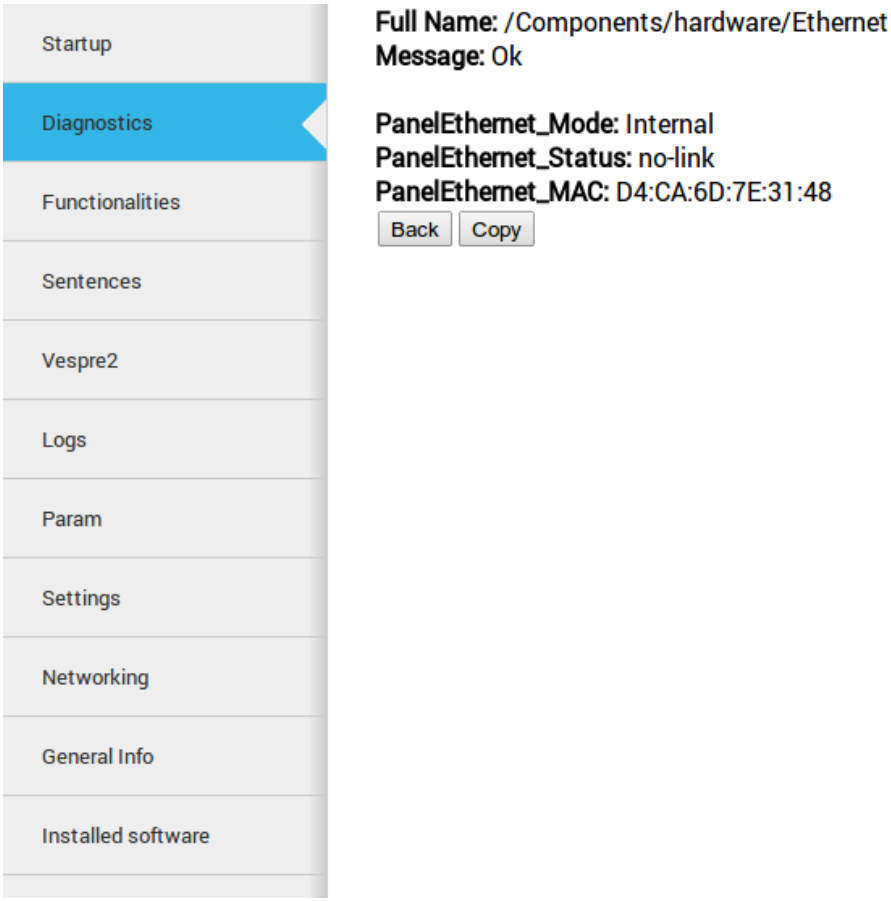


Figure 45: Networking Ethernet diagnostics

20.3.7 Video Tab

Plugin Video

Description Displays the images from a ROS topic in the WebCommander

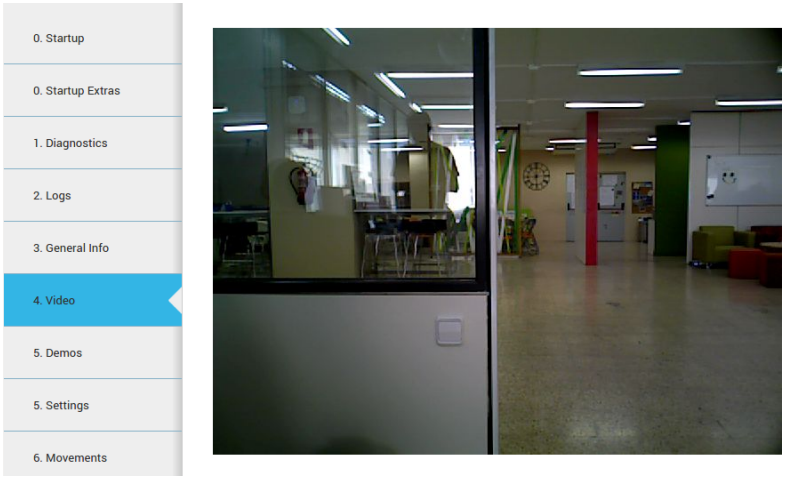


Figure 46: The Video Tab displays live video stream from the robot's camera

20.3.8 Movements Tab

Plugin Movements

Description Enables playing pre-recorded motions on TALOS .

The movement tab that can be seen in figure 47 allows a user to send upper body motion commands to the robot. Clicking on a motion will execute it immediately in the robot. Make sure the arms have enough room to move before sending a movement, to avoid possible collisions.

Movements sent through this interface take into account the surroundings of the robot, if a motion is expected to move the right arm and there is an obstacle detected by the sensors in the right side of the robot, the complete movement will not be executed.

For the same reason, a movement might be aborted if something or someone gets too close to the robot while performing a motion.

To disable these safety features, the "Safety is enabled" button must be clicked, it will then turn red and the next movement command sent will not take into account the sensors information. For security reasons, the disable safety will only affect the next movement command sent. Sending multiple unsafe movements requires pressing the "Safety is enabled" button once before each command.

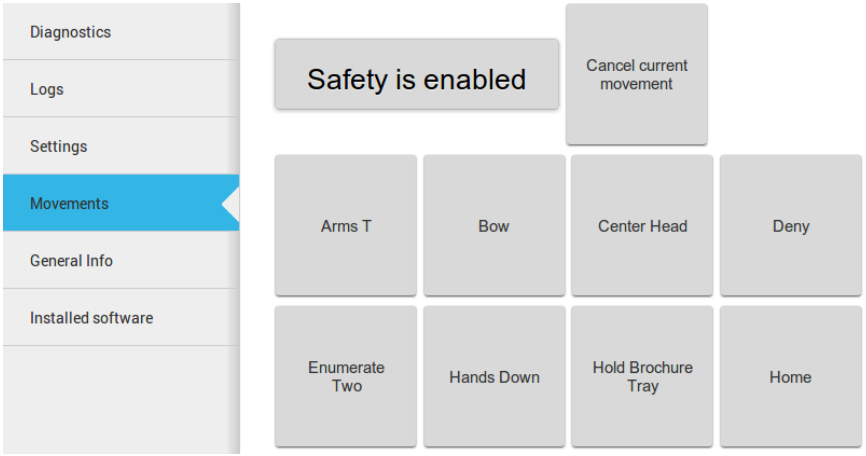


Figure 47: The Movement tab allows to send upper body motions to TALOS

20.4 Tab configuration

The WebCommander is a configurable container for different types of content, and the configuration is done through the `/wt` parameter in the ROS Parameter Server. On the robot's startup, this parameter is loaded by reading all the configuration files in `/home/pal/.pal/wt/`. For a file to be loaded, it needs to have a .yaml extension containing valid YAML syntax describing ROS parameters within the `/wt` namespace.

20.4.1 Parameter format

In the box below, an example of how a WebCommander configuration is displayed. It is a YAML file, where `/wt` is a dictionary and each key in the dictionary creates a tab in the website with the key as the title of the tab.

Each element of the dictionary must contain a `type` key, whose value indicates the type of plugin to load. Additionally, it can have a `parameters` key with the parameters that the selected plugin requires.

```
wt:
    "0. Startup":
        type: "Startup"
    "1. Diagnostics":
        type: "Diagnostics"
    "2. Logs":
        type: "Logs"
    "3. Behaviour":
        type: "Commands"
        parameters:
            buttons:
                - name: "Say some text"
                  say:
                        text: "This is the text that will be said"
                        lang: "en_GB"
                - name: "Unsafe Wave"
                  motion:
                        name: "wave"
                        safe: False
                        plan: True
```

The parameters in the box of this section would create four tabs. Named "0. Startup", "1. Diagnostics", "2. Logs" and "3. Behaviour", of the types *Startup*, *Diagnostics*, *Logs* and *Commands* respectively. The first three plugins do not require parameters, but the *Command* type does, as explained in the Command Plugin section.

20.4.2 Startup Plugin Configuration

Description Displays the list of PAL software that is configured to be started in the robot, and whether it has been started or not.

Parameters

startup_ids A list of strings that contains the startup groups handled the instance of the plugin. See section 32.1.3 on page 130.

20.4.3 Diagnostics Plugin Configuration

Description Displays the current status of TALOS's hardware and software.

Parameters None required

20.4.4 Logs Plugin Configuration

Description Displays the latest messages printed by the applications' logging system.

Parameters None required

20.4.5 General Info Plugin Configuration

Description Displays the robot model, part number and serial number.

Parameters None required

20.4.6 Installed Software Plugin Configuration

Description Displays the list of all the software packages installed in both the robot's computers.

Parameters None required

20.4.7 Networking Plugin Configuration

Description This tab allows to change the network configuration.

Parameters None required

20.4.8 Video Plugin Configuration

Description Displays the images from a ROS topic in the WebCommander

Parameters

topic Name of the topic to read images from, for instance: `/xtion/rgb/image_raw/compressed`

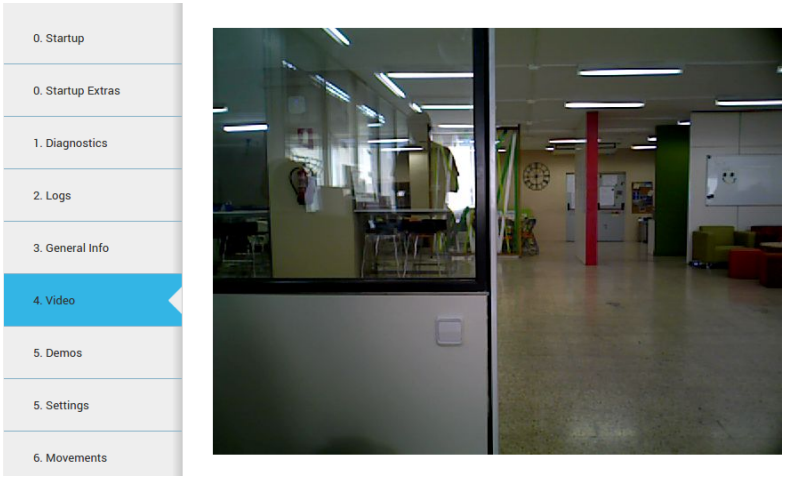


Figure 48: The Video Tab displays live video stream from the robot's camera

20.4.9 Movements Plugin Configuration

Description Enables playing pre-recorded motions on TALOS .

Parameters

goal_type Either "play_motion" or "motion_manager". Determines which action server will be used for sending the motions.

21 Rviz Basics

Description Rviz is a useful tool for visualizing the geometric and kinematic model of TALOS, as well as the robot's view of the world. As shown in Figure 49, rviz consists of a main 3D rendering area which is surrounded by a number of dock widgets. The 3D rendering area allows for the usual scene navigation operations, such as translation, rotation and zoom. We will now focus mainly on the left dock widget containing *Displays*. Refer to the official documentation for an in-depth description of rviz.

Requisites A running TALOS, as described in section 36.

21.1 Setup

Open a terminal connected to TALOS, and execute the following command

```
export ROS_MASTER_URI=http://talos-1c:11311
rosrun rviz rviz
```

Select *Robot Model* and then *base_link* as a *Fixed Frame*.

Rviz can be used with TALOS in a simulated environment as well, starting the simulation in a terminal:

```
roslaunch talos_gazebo talos_gazebo.launch
```

and Rviz from another terminal:

```
rosrun rviz rviz
```

RobotModel Displays a rendering of TALOS links (Figure 49). It allows –among other things– to toggle between visualization and collision geometries, to set the model transparency, to show the axis of the body links' reference frames and the trail followed during motions.

TF Displays the tf transform tree. Each transform in the tree represents a frame that might correspond to robot link origins, sensor locations, or any other feature that can have a coordinate system associated to it. Figure 5 displays part of the current TF tree, in particular the origin of some of the robot's links. Note that we have set the RobotModel display transparency to 0.5 to better visualize the frames contained inside TALOS .

Figure 49: RobotModel display (left), visualization geometry (center) and collision geometry (right).

22 Rqt Basics

Description Rqt is a framework for implementing Graphical User Interfaces (GUIs) in ROS. It allows the user to display multiple visualization plugins in a single window. These graphical tools allow the user to remotely monitor a robot's system, as well as to send control commands.

This test presents an overview of an example *perspective* used to graphically interact with the robot's system. It will be used during the tests in sections 38 and 29.

Requisites A running TALOS, as described in section 36.

22.1 Setup

22.1.1 Default Rviz configuration

This is a step that needs to be performed *only once*, and will persist across subsequent executions of this test. Its purpose is to set a default configuration for the Rviz 3D visualizer.

1. Open a new terminal and set the default Rviz configuration
   ```
   cp `rospack find talos_description`/config/rviz.rviz ~/.rviz/default.rviz
   ```

22.1.2 Launch the GUI

1. Open a terminal connected to TALOS, and start joint trajectory controllers for the full robot.
   ```
   roslaunch talos_controller_configuration position_controllers.launch
   ```
2. Open another terminal connected to TALOS, and launch an instance of Rqt loaded with the example *perspective* to graphically interact with the robot system.
   ```
   roslaunch talos_rqt_tutorials talos_gui.launch
   ```
3. A window similar in appearance to Figure 50 should be displayed.

22.1.3 GUI details

The user interface from Figure 50 has three main components:

3D model A virtual reconstruction of the robot system is shown on the right-most part of Figure 50. This visualization is implemented by an Rviz plugin.

Online data plots The time evolution of selected signals is shown in the center of Figure 50. The top plot displays the actual and desired position of the right arm shoulder and elbow joints, while the bottom plot displays the electrical current being applied to them. This visualization is implemented by an rqt_plot plugin.

Control commands Two kinds of plugins, shown on the left-most part of Figure 50, allow the user to send commands to running controllers:

- **Joint positions:** Allows the user to set the desired position of individual joints. This visualization is implemented by the `rqt_joint_trajectory_controller` plugin. Its operation is detailed in section 38.
- **Actuator current limits:** Allows the user to set the maximum current individual actuators can apply. This visualization is implemented by the `rqt_current_limit_controller` plugin. Its operation is detailed in section 29.

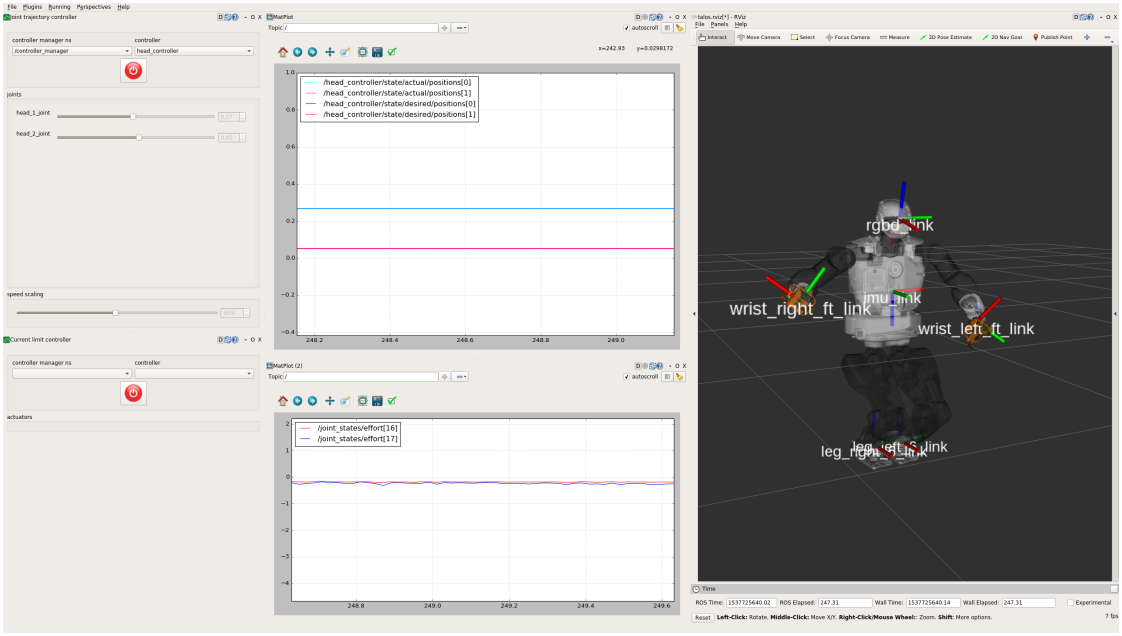


Figure 50: Rqt GUI displaying controls for commanding the robot joints (left) plots displaying the time evolution of selected joint positions and motor currents (center), and a 3D rendering of the robot system (right).

22.2 End test

Press `Ctrl-C` to close all terminals used in this test.

23 Camera

This section explains how the ROS framework manages the depth camera and the RGB camera.

23.1 pal_orbbec_pro

This node is in charge of publishing the images of TALOS's RGB-D camera.

23.1.1 Published topics

`/rgbd/rgb/image_color/compressed`

Color image corresponding to the RGB camera with transport compression.

`/rgbd/rgb/image_color`

Color image corresponding to the RGB camera.

`/rgbd/depth/image_rect_raw`

Rectified depth image corresponding to the RGB-D camera.

`/rgbd/depth/image_raw`

Depth image corresponding to the RGB.

23.1.2 Launching the node

The node in charge of publishing the cameras' topics is launched as follows

```
roslaunch pal_orbbec_pro pal_orbbec_pro.launch
```

This command is automatically executed during the start-up of TALOS so the camera is active and automatically publishing images and camera information on startup.

The node can be stopped by executing:

```
pal-stop pal_orbbec_pro
```

and can be restarted by executing:

```
pal-start pal_orbbec_pro
```

both commands must be executed in the talos-1m computer.

24 Camera Subscription

This section explains the different ways a user can retrieve the images and intrinsic parameters from the ROS topics of TALOSś cameras.

24.1 Command line

24.1.1 Image visualization

A simple way to visualize the images from TALOSś cameras is to use the ROS command made specifically for this purpose. First, open a console to make the environment variable `ROS_MASTER_URI` point to TALOS.

```
export ROS_MASTER_URI=http://talos-1c:11311
```

RGB-D camera

To visualize the depth image, use the rqt_image_view tool

```
rqt_image_view
```

RGB camera To visualize the color image, both rqt_image_view and image_view tools are available

```
rosrun image_view image_view image:=rgbd/rgb/image_color _image_transport:=compressed
```

25 Text-to-Speech synthesis

25.1 Overview of the technology

TALOS incorporates the Acapela Text-to-Speech from Acapela Group.

The technology used in this engine is the one that leads the market of synthetic voices. It is based on unit selection and allows to produce highly natural speeches in formal styles. The system is able to generate speech output, based on a input text utterance [2]. It does the phonetic transcription of the text, predicts the appropriate prosody for the utterance and finally generates the signal waveform.

Every time a text utterance is sent to the text-to-speech (TTS) engine it generates the corresponding waveform and plays it using TALOS speakers. There are several ways to send text to the TTS engine: using the ROS API, by executing ROS commands in the command line or by implementing a client in C++. Each of them is described below.

25.2 Text-to-Speech node

25.2.1 Launching the node

To be able to generate speeches, the `soundServer` should be running correctly.

System diagnostics described in Section 20 allow to check the status of the TTS service runing in the robot. These services are started by default on start-up, so normally there is no need to start them manually. To

[2] In spoken language analysis an utterance is a smallest unit of speech. It is a continuous piece of speech beginning and ending with a clear pause.

start/stop them, the following commands can be executed in a terminal opened in the multimedia computer of the robot:

```
pal-start sound_server
```

```
pal-stop sound_server
```

25.2.2 Action interface

The TTS engine can be accessed via a ROS action server named `/tts`. The full definition and explanation of the action is located in `/opt/pal/dubnium/share/pal_interaction_msgs/action/Tts.action`, below is a summary of the API:

- Goal definition fields:

```
I18nText text
TtsText rawtext
string speakerName
float64 wait_before_speaking
```

- Result definition fields:

```
string text
string msg
```

- Feedback message:

```
uint16 event_type
time timestamp
string text_said
string next_word
string viseme_id
TtsMark marks
```

Text to speech goals need to have either the `rawtext` or the `text` fields defined, as specified in the sections below.

The field `wait_before_speaking` can be used to specify a certain amount of time (in seconds) the system has to wait before speaking aloud the text specified. It may be used to generate delayed synthesis.

Sending a raw text goal

The rawtext field of type *TtsText* has the following format:

```
string text
string lang_id
```

The `rawtext` field needs to be filled with the text utterance TALOS has to pronounce and the text's language should to be specified in the `lang_id` field. The language *Id* must follow the format `language_country` specified in the RFC 3066 document (i.e., en_GB, es_ES, . . .).

Sending a I18nText goal

The text field of type I18nText has the following format:

```
string section
string key
string lang_id
I18nArgument[] arguments
```

I18n stands for Internationalization, this is used to send a pair of section and key that identifies a sentence or a piece of text stored inside the robot.

In this case the `lang_id` and `arguments` fields are optional. This allows the user to send a sentence without the need of specifying which language must be used, the robot will pick the language it's currently speaking and say the sentence in that language.

In the ROS manual you can find examples about how to create an action client that uses these message definitions to generate a speech in TALOS .

25.3 Examples of usage

25.3.1 WebCommander

Sentences can be synthesized using the WebCommander, a text field is provided so that text can be written and then synthesized by pressing the `Say` button.

Additionally buttons can be programmed to say predefined setnences, see the **??** for details.

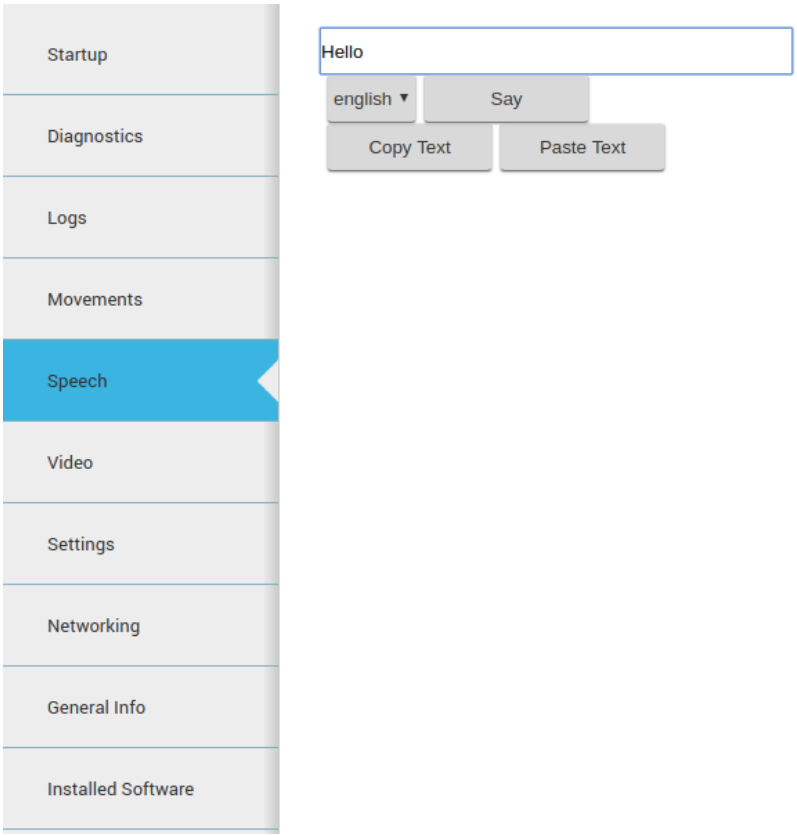


Figure 51: Voice synthesis in a commands tab of the WebCommander

25.3.2 Command line

Goals to the action server can be sent through command line by typing:

```
rostopic pub /tts/goal pal_interaction_msgs/TtsActionGoal
```

Then, by pressing `Tab`, the required message type will be auto-completed. The fields under `rawtext` can be edited to synthesize the desired sentence, as in the following example:

```
rostopic  pub /tts/goal pal_interaction_msgs/TtsActionGoal "header:
  seq: 0
  stamp:
        secs: 0
     nsecs: 0
 frame_id: ''
goal_id:
 stamp:
       secs: 0
       nsecs: 0
 id: ''
goal:
 text:
 rawtext:
        text: 'Hello world'
        lang_id: 'en_GB'
 speakerName: ''
 wait_before_speaking: 0.0"
```

25.3.3 Action client

A GUI included in the `actionlib` package of ROS can be used to send goals to the voice synthesis server.

In order to be able to execute the action successfully, the `ROS_IP` environment variable should be exported with the IP direction of your development computer:

```
export ROS_IP=DEV_PC_IP
```

The GUI shown in Figure 52 can be run as follows:

```
export ROS_MASTER_URI=http://talos-1c:11311
rosrun actionlib axclient.py /tts
```

Editing the fields inside `rawtext` parameter and pressing the `SEND GOAL` button will trigger the action.

26 Sensors

This section contains an overview of the sensors included in the TALOS robot. After a brief description of the sensors themselves, their ROS and C++ API are presented.

26.1 Description of sensors

1. Feet force torque sensors
   TALOS has 6 axis force torque sensor on each ankle, before the foot sole. These sensors measure ground reaction forces and torques.

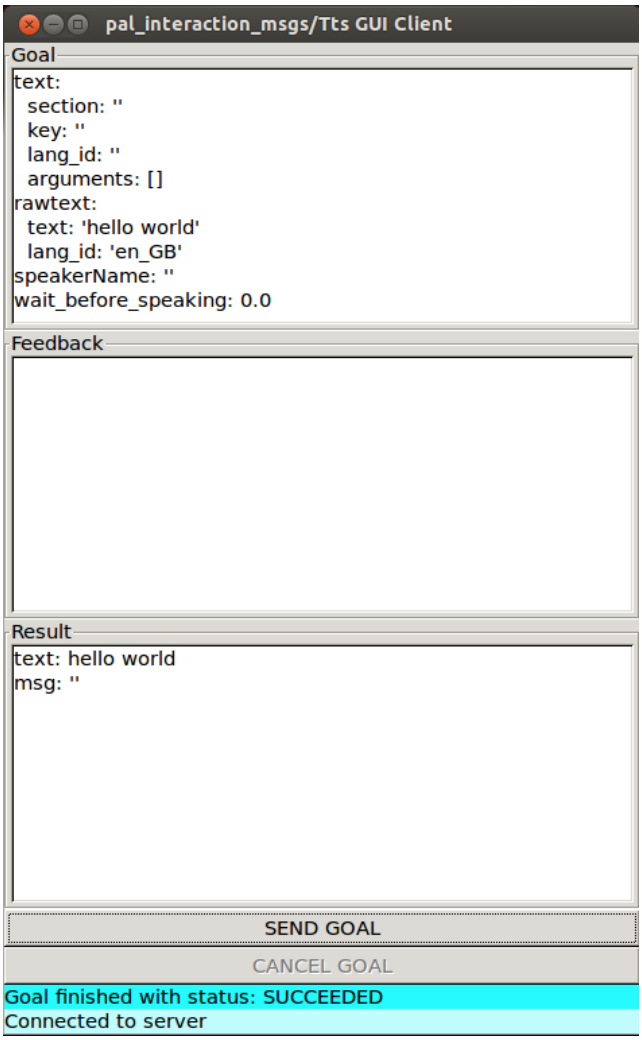


Figure 52: Voice synthesis using the GUI from actionlib

2. Wrists force torque sensors
   The robot features six axis force torque sensors mounted on each wrist, just before the hand. They can measure interaction forces and torques with the environment or with a human interacting with the end effectors.

3. Inertial measurement unit (IMU)
   This sensor unit is mounted in the torso of TALOS and provides acceleration, angular rates, magnetic field and attitude estimations.

4. Cameras
   For documentation about camera specifications go to Section 4; for a description about how to access camera images go to Section 23.

The locations of the sensors are depicted in Figure 53.

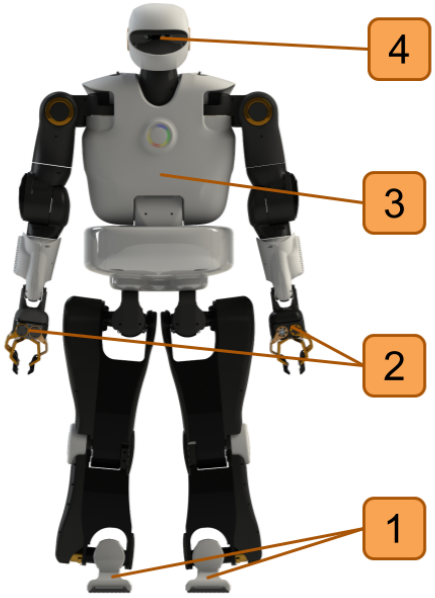


Figure 53: Sensors location

26.2 ROS API

NOTE: Every node that publishes sensor data is launched by default on startup.

26.2.1 Published topics

`/left_ankle_ft` (geometry_msgs/WrenchStamped)

Force torque sensor data from the left sole.

`/right_ankle_ft` (geometry_msgs/WrenchStamped)

Force torque sensor data from the right sole.

`/left_wrist_ft` (geometry_msgs/WrenchStamped)

Force torque sensor data from the left wrist.

`/right_wrist_ft` (geometry_msgs/WrenchStamped)

Force torque sensor data from the right wrist.

`/base_imu` (sensor_msgs/Imu)

Inertial data from inside the waist of TALOS .

`/diagnostics` (diagnostic_msgs/DiagnosticArray)

State of the hardware and software of TALOS. There are different tools for visualization, rqt_robot_monitor, rqt_runtime_monitor or the Diagnostic Tab of the WebCommander (see 20.4.3 on page 91). For details about the diagnostics system see http://wiki.ros.org/diagnostics.

`/joint_states` (sensor_msgs/JointState)

State of all the joints of the robot, *velocity* contains the velocity of the joint in rad/s, *position* contains the position in radians according to the **relative** encoders, *effort* contains the **current consumed** by the joints in amperes.

`/joint_torque_states` (sensor_msgs/JointState)

State of all the joints of the robot, *velocity* contains the velocity of the joint in rad/s, *position* contains the position in radians according to the **absolute** encoders, *effort* contains the **measures of the torque sensors** in Nm.

`/actuator_temperatures` (temperature_sensor_controller/ActuatorTemperatureState)

Temperature of all the actuators of the robot in Celsius degrees.

`/actuator_mode_state` (mode_state_controller/ModeState)

Mode of all the actuators of the robot. The different modes are defined in the msg file:

`/mode_state_controller/ModeState.msg`

```
int32 BEGIN=-1
int32 MODE_POSITION=0
int32 MODE_VELOCITY=1
int32 MODE_EFFORT=2
int32 NOMODE=3
int32 EMERGENCY_STOP=4
int32 SWITCHING=5
int32 ERROR=6
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string[] name
int32[] mode
```

27 Walking

In this section we introduce and describe the parameters, interfaces and conventions used on TALOS by the walking controller.

27.1 Walking_controller

The walking controller is a `ros_control` plugin that allows TALOS to walk following commands from the user or from an application. There are three ways to command TALOS:

- Using a topic;
- A service;
- Or an action client.

27.1.1 Action API

The `walking_controller` provides an implementation of a `SimpleActionServer` (see actionlib documentation) that takes goals containing a desired step list for TALOS to execute, and provides feedback on the the steps executed and the final result. This action should be used carefully, since the user is the one that specifies the footholds in the message. If the list is not generated carefully, it can create self collisions or unfeasible configurations of TALOS. This action is recommended as an input interface for a step planner; if the user only wants to command the direction of TALOS, the topic interface is preferable.

Action subscribed topics

`/walking_controller/footsteps_execution/goal` (humanoid_nav_msgs/ExecFootstepsActionGoal)

A goal with the list of footsteps to be performed.

`/walking_controller/footsteps_execution/cancel` (actionlib_msgs/GoalID)

A request to cancel a specific goal.

Action published topics

`/walking_controller/footsteps_execution/feedback` (humanoid_nav_msgs/ExecFootstepsActionFeedback)

Feedback contains the footsteps performed so far by the walking controller; note that the poses reached might be slightly different from the ones desired, so they can be used to replan if necessary.

`/walking_controller/footsteps_execution/result` (humanoid_nav_msgs/ExecFootstepsActionResult)

The result is the list of footsteps finally performed by TALOS .

`/walking_controller/footsteps_execution/status` (actionlib_msgs/GoalStatusArray)

Report the status of the action server.

27.1.2 Topic API

`/walking_controller/cmd_vel` (geometry_msgs/Twist Message)

A Twist command specifies the linear and angular velocity that TALOS needs to track with its center of mass. The walking controller will update the velocity command every time a step is finalised. If no command is given and the feet are not aligned TALOS will automatically add a final step to align them.

The joystick controller uses the topic interface to command TALOS. Joystick nodes are launched automatically, however it is possible to start them manually using the following command:

```
roslaunch talos_bringup joystick_teleop.launch cmd_vel:=/walking_controller/cmd_vel
```

27.1.3 Published topics

`/walking_controller/walking_status` (walking_msgs/WalkingStatus)

General status of the walking controller

27.1.4 Service API

`/walking_controller/walk_steps` (walking_msgs/WalkSteps)

Call this service to add a list of steps to be executed by TALOS. The list is parametrized by the number of steps, length of steps and step time. The foot positions will be decided automatically by the walking controller.

`/walking_controller/do_step` (humanoid_nav_msgs/StepTargetService)

Call this service to send a step to be executed by TALOS. The step is parametrized by one side (left or right) and the 2D relative position respects the stance foot. Step time is set at one second and can't be modified. Use this interface carefully and take care when switching the swing foot from left to right and providing valid target foot positions.

27.2 Walking parameters

The walking algorithm has parameters that can be modified by the user.

There are three groups of parameters, as shown in Figure 54, that allow to operator to modify the walking behaviour though `rqt_reconfigure`: `biped_controller`, `zmp_solver` and `com_stabilizer` parameters.

In order to run the `rqt_reconfigure` GUI, execute:

```
rosrun rqt_reconfigure rqt_reconfigure
```

Select from the drop-down list the set of parameters to be modified (found under the namespace walking_controller).

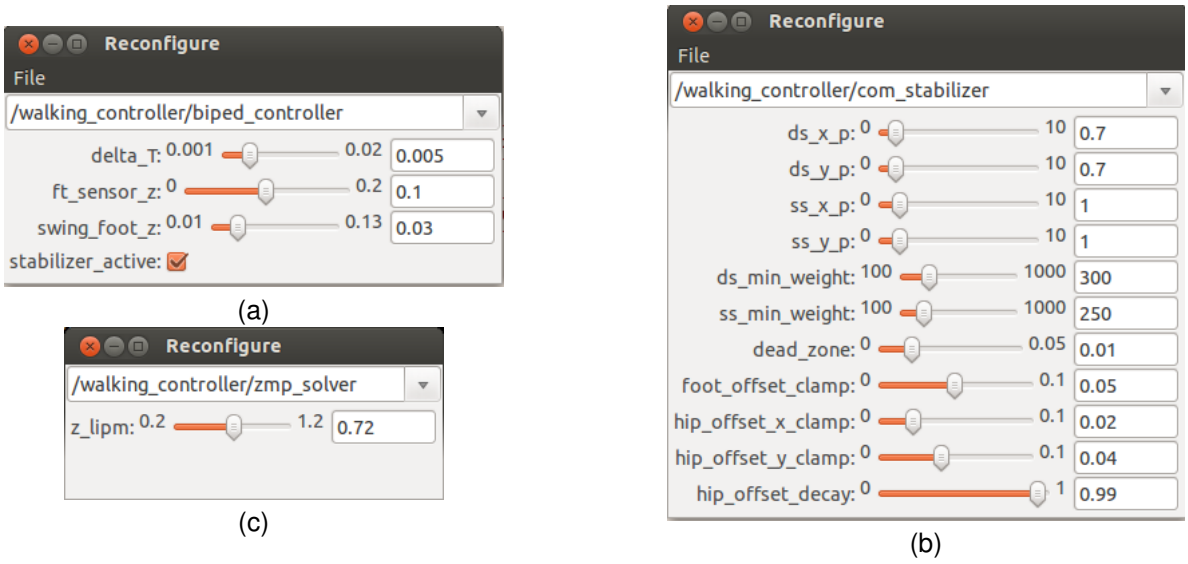


Figure 54: Walking controller parameters: biped_controller (a), com_stabilizer (b), zmp_solver (c).

Biped controller The general parameters for walking specify the height of the swing leg and the status of the stabilizer (enabled or disabled), as shown in Figure 54(a)

`~swing_foot_z` (double, default: 3.5)

Foot height from the ground when foot is in swing (middle of the step).

`~stabilizer_active` (bool, default: true)

Enable/disable stabilizer.

COM Stabilizer This set of parameters allow to tune the walking controller stabilizer in order to add robustness to the walking gait. Figure 54(b)

`~ds_x_p` (double , default: 0.8)

Double support x-axis proportional gain.

`~ds_y_p` (double , default: 0.6)

Double support y-axis proportional gain.

`~ss_x_p` (double , default: 1.0)

Single support x-axis proportional gain.

`~ss_y_p` (double , default: 1.0)

Single support y-axis proportional gain.

`~ds_min_weight` (double , default: 300)

Double support weight threshold for enabling stabilizer.

`~ss_min_weight` (double , default: 250)

Single support weight threshold for enabling stabilizer.

`~dead_zone` (double , default: 0.01)

Dead zone for zmp error tracking.

`~foot_offset_clamp` (double , default: 0.05)

Max offset applied to swing foot by stabilizer.

`~hip_offset_x_clamp` (double , default: 0.02)

Max offset applied to hip x-coord by stabilizer.

`~hip_offset_y_clamp` (double , default: 0.04)

Max offset applied to hip y-coord by stabilizer

`~hip_offset_decay` (double , default: 0.99)

Decay value for hip offset.

Solver The walking controller uses the Linear Inverted Pendulum as a simplified model of the robot's dynamics. The only parameter of the model that can be modified by the user is the height of the pendulum (Figure 54(c)).

`~z_lipm` (double , default: 0.7)

Height of the Linear Inverted Pendulum used in the walking solver.

28 Joint Trajectory Controller

28.1 Overview

Controller for executing joint-space trajectories on a group of joints. Trajectories are specified as a set of waypoints to be reached at specific time instances, which the controller attempts to execute in the best way the mechanism allows. Waypoints consist of positions, velocities and accelerations.

28.1.1 Trajectory representation

The controller is templated to work with multiple trajectory representations. By default, a spline interpolator is provided, but it is possible to support other representations. The spline interpolator uses the following interpolation strategies, depending on the waypoint specification:

Linear Only position is specified. Guarantees continuity at the position level. Discouraged because it yields trajectories with discontinuous velocities at the waypoints.

Cubic Position and velocity are specified. Guarantees continuity at the velocity level.

Quintic Position, velocity and acceleration are specified. Guarantees continuity at the acceleration level.

28.1.2 Hardware interface type

The controller is templated to work with multiple hardware interface types. Currently, joints with **position** and **effort** interfaces are supported: the former simply forwards desired positions to the joints, and the latter maps the (position and velocity) trajectory following error to an effort command through a PID. Example controller configurations for both interfaces can be found in controller's ROS wiki page.

Similarly to the trajectory representation case above, it's possible to support new hardware interfaces, or alternative mappings to an already supported interface (eg. a proxy controller for generating effort commands).

28.1.3 Other features

- **Realtime-safe** implementation.
- Proper handling of **wrapping** (continuous) joints.
- **Robust to system clock changes.** Discontinuous system clock changes do not cause discontinuities in the execution of already-queued trajectory segments.

28.2 Sending trajectories

28.2.1 Available interfaces

There are two mechanisms for sending trajectories to the controller: by means of the **action interface** or the **topic interface**. Both use the trajectory_msgs/JointTrajectory message to specify trajectories, and require specifying values for all the controller joints (as opposed to only a subset).

The primary way to send trajectories is through the **action interface**, which should be favored when execution monitoring is desired. Action goals allow the operator to specify not only the trajectory to execute, but also (optionally) path and goal tolerances. When no tolerances are specified, the defaults given in the parameter server are used (see the ROS API). If tolerances are violated during trajectory execution, the action goal is aborted and the client is notified.

Important note Even when a goal has been aborted, the controller will still attempt to execute the trajectory as best as possible.

The **topic interface** is a fire-and-forget alternative. Use this interface if execution monitoring is not required. The controller's path and goal tolerance specification is *not* used in this case, as there is no mechanism to notify the sender about tolerance violations. Note that although some degree of monitoring is available through the `query_state` service and `state` topic (see the ROS API), it is much more cumbersome to realize than with the action interface.

28.2.2 Preemption policy

Only one action goal can be active at any moment (or none if the topic interface is used). Path and goal tolerances are checked *only* for the trajectory segments of the active goal.

When an active action goal is preempted by another command coming from either the action or the topic interface, the goal is canceled and the client is notified.

Sending an **empty trajectory** message from either the action or the topic interface will stop the execution of all queued trajectories and enter position hold mode. The `stop_trajectory_duration` parameter controls the duration of the stop motion.

28.2.3 Trajectory replacement

Joint trajectory messages allow the operator to specify the time at which a new trajectory should start executing by means of the header timestamp, where zero time (the default) means *"start now"*.

The arrival of a new trajectory command does *not necessarily* mean that the controller will completely discard the current running trajectory and substitute it with the new one. Rather, the controller will take the useful parts of both and combine them appropriately.

The steps followed by the controller for trajectory replacement are as follows:

- Get useful parts of the **new** trajectory. Preserve all waypoints which have times to be reached is in the future, and discard those with times in the past. If there are no useful parts (ie. all waypoints are in the past), the new trajectory is rejected and the current one continues the execution without changes.
- Get useful parts of the **current** trajectory. Preserve the current trajectory up to the start time of the new trajectory; discard the later parts.
- Combine the useful parts of the **current** and **new** trajectories.

The following examples describe this behavior in detail.

28.2.4 Trajectory replacement example: basics

This example, illustrated in Figure 55a shows a joint which is in hold position mode (flat grey line labeled `pos hold`). A new trajectory (shown in red) arrives at the current time (`now`), which contains three waypoints and a start time in the future (`traj start`). The time at which waypoints should be reached (`time_from_start` member of the trajectory_msgs/JointTrajectoryPoint message) is relative to the trajectory start time.

The controller splices the current hold trajectory at time `traj start` and appends the three waypoints. Notice that between `now` and `traj start` the previous position hold is still maintained, as the new trajectory is not supposed to start yet. After the last waypoint is reached, its position is held until new commands arrive.

The controller guarantees that the transition between the current and new trajectories will be smooth. Longer times to reach the first waypoint mean slower transitions.

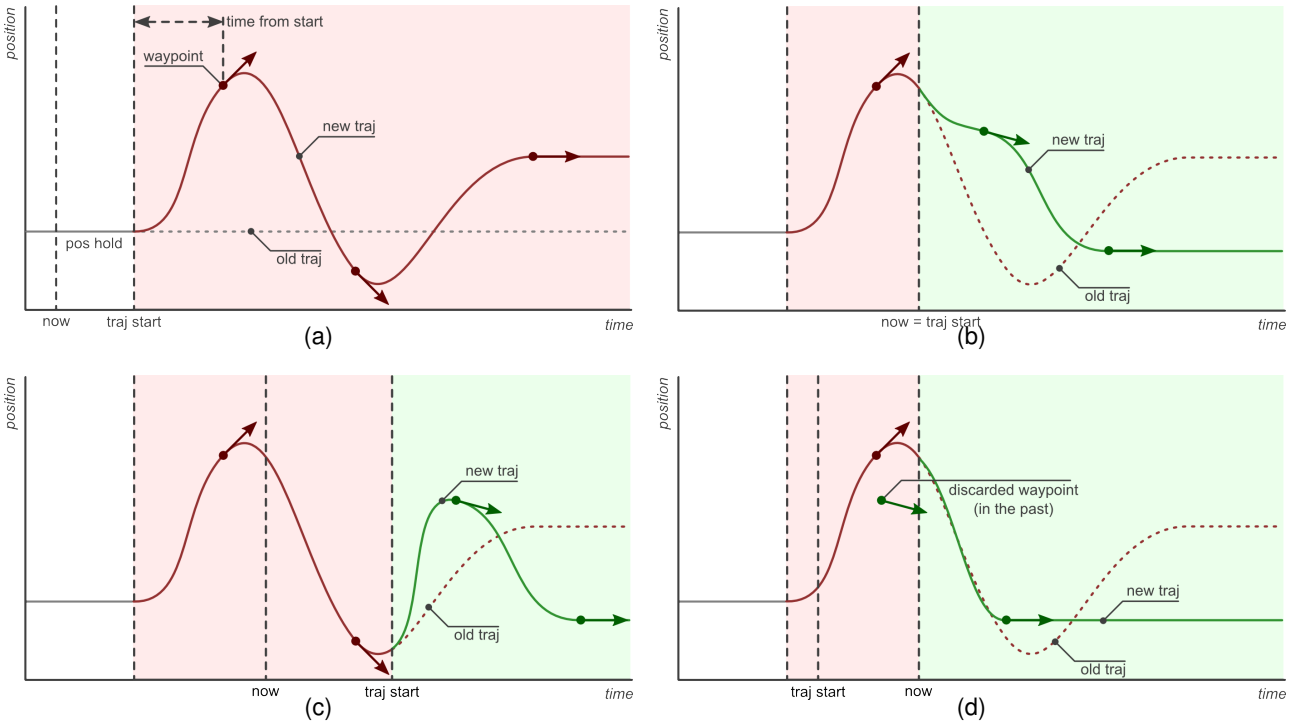


Figure 55: Trajectory replacement. Details of sending a new trajectory (a). Effects of trajectory start time set to: now (b), in the pfuture (c), and in the past (d) .

28.2.5 Trajectory replacement example: effects of trajectory start time

This example describes the effect of sending the same trajectory to the controller with different start times. The scenario is that of a controller executing the trajectory from the previous example (shown in red), and receiving a new command (shown in green) with a trajectory start time set to either zero (start now, Figure 55b), a future time (Figure 55c), or a time in the past (Figure 55d).

Of special interest is Figure 55d, where the new trajectory start time and first waypoint are in the past (before `now`). In this case, the first waypoint is discarded and only the second one is realized.

28.3 ROS API

28.3.1 Action interface

The controller exposes a control_msgs::FollowJointTrajectoryAction interface in the `follow_joint_trajectory` namespace of the controller. See the action definition for more information on what to send.

28.3.2 Subscribed topics

`command` (trajectory_msgs/JointTrajectory)

Trajectory to execute using the **topic interface**.

28.3.3 Published topics

`state` (control_msgs/JointTrajectoryControllerState)

Current controller state.

28.3.4 Services

`query_state` (control_msgs/QueryTrajectoryState)

Query controller state at any future time.

28.3.5 Parameters

`joints` (`string[]`)

The list of joints to control.

`constraints/goal_time` (`double`, default: 0.0)

If the timestamp of the goal trajectory point is t, then following the trajectory succeeds if it reaches the goal within *t* +/- `goal_time`, and aborts otherwise.

`constraints/stopped_velocity_tolerance` (`double`, default: 0.01)

Velocity to be considered approximately equal to zero.

`constraints/<joint>/goal` (`double`, default: 0.0)

Position tolerance for a particular joint to reach the goal. When the joint is within goal_position +/- goal_tolerance, than the trajectory can succeed.

`constraints/<joint>/trajectory` (`double`, default: 0.0)

Position tolerance for a particular joint throughout the trajectory. If the joint position ever falls outside trajectory_position +/- tolerance, then the trajectory aborts.

`gains/<joint>` (`associative array`)

Key value pairs specifying a PID controller. This is only required by the effort interface variant.

`stop_trajectory_duration` (`double`, default: 0.5)

When starting the controller or canceling a trajectory, position hold mode is entered. This parameter specifies the time it takes to bring the current state (position and velocity) to a stop. Its value can be greater or equal than zero.

`state_publish_rate` (`double`, default: 50)

Frequency (in Hz) at which the controller state is published.

`action_monitor_rate` (`double`, default: 20)

Frequency (in Hz) at which the action goal status is monitored. This is an advanced parameter that should not require changing.

29 Setting current limits – graphical user interface

Description This test covers how to set the maximum electrical current individual actuators TALOS can apply, using a graphical user interface. This test is similar to the one presented in section 38, but commands actuator current limits instead of joint positions. This test applies only to real TALOS deployments, not to simulated ones.

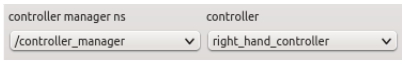
Requisites A running TALOS, as described in section 36.

29.1 Setup

29.1.1 Launch the GUI

1. Launch the Rqt GUI, as described in section 22.1.2 and shown in Figure 50.

   (a) On the bottom-left of the GUI is the panel for controlling actuator current limits, named `Current limit controller`, detailed in Figure 56. Activate the gripper current limit controller by selecting `/controller_manager` and `right_gripper_current_limit_controller`on the panel's combo boxes, as shown below.



   Note: If instead of `right_gripper_current_limit_controller`the user selects any of the other options (arms, legs, torso, head), current limits can be set for the corresponding group.

2. By default, the controller panel is in **monitor mode**, as indicated by the red status button shown below. In monitor mode the actuator sliders display the actual limits but do not accept new commands. By clicking on the status button, the controller panel enters **control mode**, and the status button turns green.

3. In control mode, current limits can be set by either moving the sliders or by providing discrete setpoints on the box to the right of each slider. Values are **normalized** to the [0, 1] interval, where zero means no current applied and one maps to the maximum allowed current.

4. Actuator currents being applied to the actuators are shown in the data plot of Figure 50 (center, bottom). Plot values are not normalized, and represent actual current values.

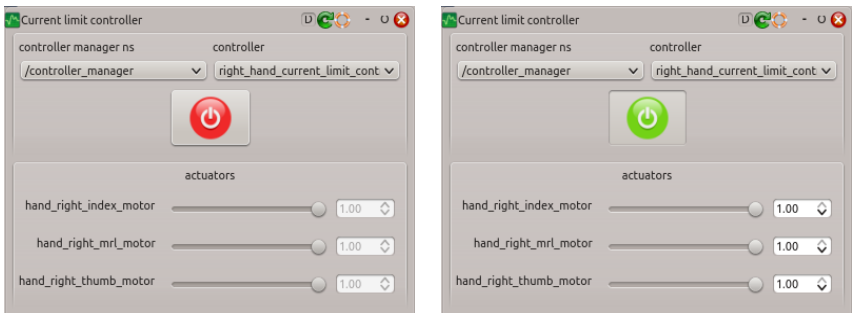


Figure 56: Detail of the joint control panel in monitor (left) and control (right) modes.

29.2 End test

Press `Ctrl-C` to close all terminals used in this test.

s

30 ros_control

This section contains a brief introduction to the `ros_control` architecture and different use cases for creating different types of controllers depending on the users needs.

30.1 Introduction to ros_control

`ros_control` is a consistent interface used to access the actuators and sensors of a robot. It presents the RAW data that comes from hardware components to the various controllers in an organised and intuitive way. It also has the function of being a resource handler for all sensors and actuators, allowing different controllers to access the same device simultaneously using custom policies. Controllers are implemented as plugins that can be loaded and unloaded dynamically at runtime. Since `ros_control` is an abstraction layer for hardware, it allows the operator to write controllers that are robot agnostic, and can be reused in different types of robots with different physical configurations. This is one of the features that makes it possible to write controllers that are 100% compatible between the `gazebo` simulator and the real robot. Figure57 shows the architecture of the system.

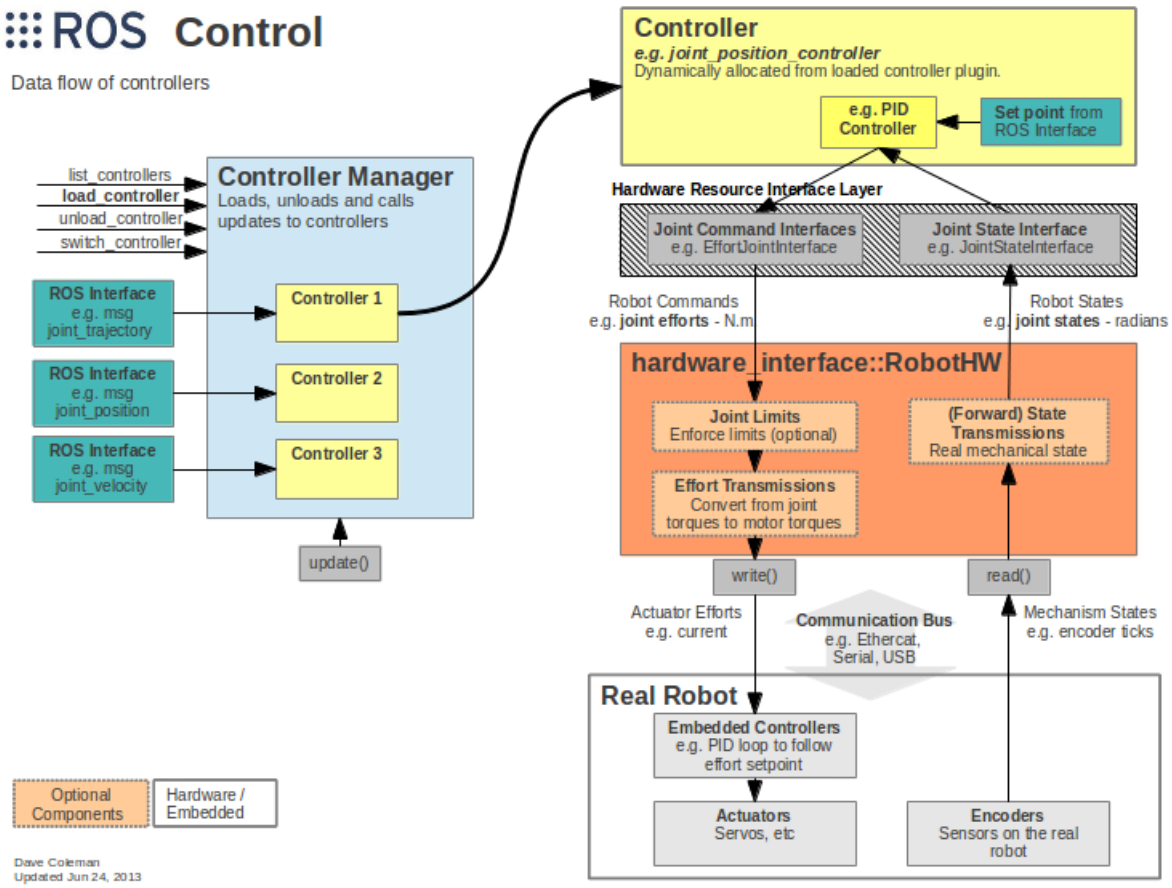


Figure 57: Data flow of controllers

The main hub of `ros_control` is the `controller_manager`. The controller manager exposes its interface through ROS services. It's responsible for managing all the controllers and resources of TALOS, and triggering events. For every robot, a robot hardware interface must be created. This interface abstracts the real robot's hardware to the system by exposing standard interfaces to the custom hardware. We provide two robot hardware interfaces, one for the real TALOS robot and one for the simulated robot.

A controller can have states on two levels based on

1. Presence
   (a) loaded: All controller-specific configurations are loaded in the parameter server and `ros_control` has created an instance of the controller that is in initialised state.
   (b) not loaded: `ros_control` has knowledge of the controller type, but no instance of the controller was created.
2. Activity
   (a) running: The update function of the controller is called at every control cycle.
   (b) stopped: The controller is initialized but is not performing any action.

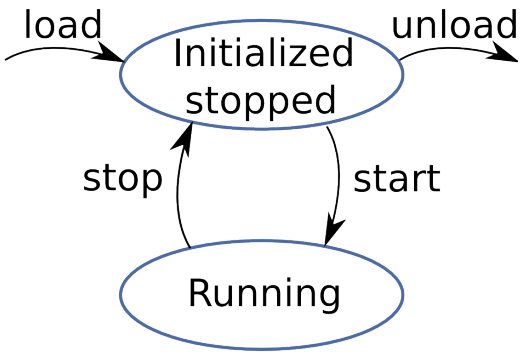


Figure 58: Controller states

30.1.1 How to load and unload controllers

```
rosservice call /controller_manager/load_controller "name: 'hello_controller'"
```

```
rosservice call /controller_manager/unload_controller "name: 'hello_controller'"
```

30.1.2 How to start and stop controllers

```
rosservice call /controller_manager/switch_controller "start_controllers: -
'hello_controller' stop_controllers: - 'imu_controller' strictness: 0"
```

30.2 How to create a custom controller

ATTENTION! Pay attention when using custom controllers! A custom controller might break the real-time features of the original system. When developing a new controller, extreme care should be taken for experiments on the real TALOS .

The following snippets of code are simple examples on how to use `ros_control`, do not run them on the real TALOS since they use the ROS logging system which is NOT real-time safe.

All controllers created for `ros_control` are plugins that allow the operator to directly access TALOS 's hardware - this includes moving the joints and reading the sensors.

This subsection provides example controllers to help the operator get started with custom controllers.

REQUIREMENTS TO CREATE A CONTROLLER

- Create a class that contains your controller. The class must inherit from the base class Controller specified as a template parameter of the type of hardware interface that is going to use.
- A xml file must be created in the package where the plugin is located, that has to contain all the plugins that reside in the package. By doing so, `ros_control` knows where the plugin libraries are stored. The xml file name must contain the name of the package followed by `_plugins.xml`, for example: `ros_controllers_turorials.xml`.
- The xml file has the same following schema:

```
<class name="ros_controllers_tutorials/HelloController"
type=" ros_controllers_tutorials :: HelloController"
      base_class_type="controller_interface::ControllerBase">
  <description>
    The HelloController does nothing useful.
     It 's only a demonstrational controller that prints a hello message.
  </description>
</class>
```

Listing 9: The HelloController is a skeleton controller with a single ROS_INFO in the update function.

- All instances of a controller that you want to load must have its name along with its type in the parameter server. An example yaml file with these parameters would be:

```
hello_controller :
  type: " ros_controllers_tutorials / HelloController"
```

Listing 10: The HelloController is a skeleton controller with a single ROS_INFO in the update function.

In the following subsections, we will explain how to create different types of controllers. All the source code and configuration files can be found in the `ros_controllers_tutorials` package.

30.2.1 Hello controller

This controller shows the most basic components. All controllers must inherit from the base class, specifying in the template constructor what type of hardware_interface they want to use (position, velocity, effort and others. In the case of TALOS , all joints are commanded in position mode). See the end of this section for an explanation of how to use different types of hardware interfaces in the same controller.

Four methods must be implemented that correspond to each of the states of the state machine:

- Init: When the controller is loaded, it calls the init function. All initialisations must be done in this function. Real time safety doesn't to be taken into account, since it is not in the real time loop.
- Starting: Before the controller starts to cycle in the control loop, it calls the start function.
- Stopping: When the controller wants to shutdown, either by an event or external request, it calls this function after its last update loop.
- Update: This method will be called periodically while the controller is in running state.

In this example, we inherit from the JointStateInterface despite the fact we're not going to use the robot's joints. At every control cycle, we are going to print the "Hello World" message.

Source code:

```
#include <controller_interface/controller.h>
#include <hardware_interface/joint_state_interface.h>
#include <pluginlib/class_list_macros.h>

namespace ros_controllers_tutorials{

class HelloController
  : public controller_interface::Controller<hardware_interface::JointStateInterface>
{
 public:
  bool init(hardware_interface::JointStateInterface* hw, ros::NodeHandle &n)
  {
    return true;
  }
  void update(const ros::Time& time, const ros::Duration& period)
  {
    // WARNING! This will not be realtime-safe
    ROS_INFO_NAMED("hello_controller", "Hello ros_control!");
  }
  void starting(const ros::Time& time) { }
  void stopping(const ros::Time& time) { }
private:
};
PLUGINLIB_DECLARE_CLASS(ros_controllers_tutorials,
                        HelloController,
                        ros_controllers_tutorials::HelloController,
                        controller_interface::ControllerBase);
}
```

Listing 11: The HelloController is a skeleton controller with a single ROS_INFO in the update function.

30.2.2 Moving a position controlled joint

Source code:

```
#include <controller_interface/controller.h>
#include <hardware_interface/joint_command_interface.h>
#include <pluginlib/class_list_macros.h>
namespace ros_controllers_tutorials{
  class JointController : public
      controller_interface::Controller<hardware_interface::PositionJointInterface>
  {
  public:
    bool init(hardware_interface::PositionJointInterface* hw, ros::NodeHandle &n)
    {
      cont = 0;
      controlled_joint_name_ = "head_2_joint";
      ROS_INFO_STREAM("LOADING JOINT CONTROLLER ");
      // Get a joint handle
      try
      {
        joint_ = hw->getHandle(controlled_joint_name_);
 // throws on failure
        ROS_INFO_STREAM("Found joint '" << controlled_joint_name_);
      }
      catch (...)           {
        ROS_ERROR_STREAM("Could not find joint '" << controlled_joint_name_);
        return false;
      }
      return true;
    }
    void update(const ros::Time& time, const ros::Duration& period)
    {
      // Move the head using a sine wave
      double joint_comand = 0.2*sin(cont/1000.0);
      joint_.setCommand(joint_comand);
      cont = cont + 1;
    }
```

```
    void starting(const ros::Time& time) { }
    void stopping(const ros::Time& time) { }
  private:
    hardware_interface::JointHandle joint_;
    std::string controlled_joint_name_;
    double cont;
  };
  PLUGINLIB_DECLARE_CLASS(ros_controllers_tutorials,
                JointController,
                ros_controllers_tutorials::JointController,
                controller_interface::ControllerBase);
}
```

Listing 12: The JointController is a controller that moves a single joint in position control

30.2.3 Inertial measurement unit (IMU)

```
#include <controller_interface/controller.h>
#include <hardware_interface/joint_state_interface.h>
#include <pluginlib/class_list_macros.h>
#include <realtime_tools/realtime_buffer.h>
#include <realtime_tools/realtime_publisher.h>
#include <boost/assign.hpp>
#include <hardware_interface/imu_sensor_interface.h>
#include <sensor_msgs/Imu.h>

namespace ros_controllers_tutorials{
  class ImuController :
  public controller_interface::Controller<hardware_interface::ImuSensorInterface> {
  public:
    bool init(hardware_interface::ImuSensorInterface* hw, ros::NodeHandle &n)  {
      // get all imu sensor names
      const std::vector<std::string>& sensor_names = hw->getNames();
      for (unsigned i=0; i<sensor_names.size(); i++)
        ROS_INFO("Got sensor %s", sensor_names[i].c_str());
      for (unsigned i=0; i<sensor_names.size(); i++){
        // sensor handle
        sensor_ = hw->getHandle(sensor_names[i]);
      }
      return true;
    }
    void update(const ros::Time& time, const ros::Duration& period)
    {
      using namespace hardware_interface;
      sensor_msgs::Imu value;
      // Orientation
      if (sensor_.getOrientation())
      {
        value.orientation.x = sensor_.getOrientation()[0];
        value.orientation.y = sensor_.getOrientation()[1];
        value.orientation.z = sensor_.getOrientation()[2];
        value.orientation.w = sensor_.getOrientation()[3];
      }
      ROS_INFO_STREAM("IMU orientation x: "<<value.orientation.x<<" y:
          "<<value.orientation.y<<" z: "<<value.orientation.z<<"
          w: "<<value.orientation.w);
    }
    void starting(const ros::Time& time) { }
    void stopping(const ros::Time& time) { }
  private:
    hardware_interface::ImuSensorHandle sensor_;
  };
  PLUGINLIB_DECLARE_CLASS(ros_controllers_tutorials,
          ImuController,
          ros_controllers_tutorials::ImuController,
          controller_interface::ControllerBase);
}
```

Listing 13: The ImuController is a controller that subscribes and print the value of the IMU.

30.2.4 Force torque sensors

```
#include <controller_interface/controller.h>
#include <hardware_interface/joint_state_interface.h>
#include <pluginlib/class_list_macros.h>
#include <geometry_msgs/Wrench.h>
#include <realtime_tools/realtime_buffer.h>
#include <realtime_tools/realtime_publisher.h>
#include <boost/assign.hpp>
#include <hardware_interface/force_torque_sensor_interface.h>

namespace ros_controllers_tutorials{

  class ForceTorqueController :
  public controller_interface::Controller<hardware_interface::JointStateInterface>
  {
  public:
    bool init(hardware_interface::ForceTorqueSensorInterface* hw, ros::NodeHandle &n)
    {
      // get force torque sensors names package.
      const std::vector<std::string>& sensor_names = hw->getNames();
      for (unsigned i=0; i<sensor_names.size(); i++)
        ROS_INFO("Got sensor %s", sensor_names[i].c_str());
      for (unsigned i=0; i<sensor_names.size(); i++){
        // sensor handle
        sensors_.push_back(hw->getHandle(sensor_names[i]));
      }
      return true;
    }
    void update(const ros::Time& time, const ros::Duration& period)
    {
      geometry_msgs::Wrench ft[2];
      for(unsigned int s = 0; s<2; ++s){
        ft[s].force.x   = sensors_[s].getForce()[0];
        ft[s].force.y   = sensors_[s].getForce()[1];
        ft[s].force.z   = sensors_[s].getForce()[2];
        ft[s].torque.x = sensors_[s].getTorque()[0];
        ft[s].torque.y = sensors_[s].getTorque()[1];
        ft[s].torque.z = sensors_[s].getTorque()[2];
      }
      ROS_INFO_STREAM("Force sensor left: fx = "<<
                      ft[0].force.x<<" fy: "<<
                      ft[0].force.y<<" fz: "<<
                      ft[0].force.z<<" tx: "<<
                      ft[0].torque.x<<" ty: "<<
                      ft[0].torque.y<<" tz: "<<
                      ft[0].torque.z);
      ROS_INFO_STREAM("Force sensor left: fx = "<<
                      ft[1].force.x<<" fy: "<<
                      ft[1].force.y<<" fz: "<<
                      ft[1].force.z<<" tx: "<<
                      ft[1].torque.x<<" ty: "<<
                      ft[1].torque.y<<" tz: "<<
                      ft[1].torque.z);
   }
    void starting(const ros::Time& time) { }
    void stopping(const ros::Time& time) { }
  private:
    std::vector<hardware_interface::ForceTorqueSensorHandle> sensors_;
  };
  PLUGINLIB_DECLARE_CLASS(ros_controllers_tutorials,
                          ForceTorqueController,
                          ros_controllers_tutorials::ForceTorqueController,
                          controller_interface::ControllerBase);
}
```

Listing 14: The ForceTorqueController is a controller that subscribes to the FT sensors and prints its values

30.2.5 Combining different resources (hardware interfaces) in a single controller

This example shows how to write a controller that contains different hardware interfaces, for example a force torque and IMU interface. Due to the size of the example, only the most relevant parts of the code are represented here. The complete file can be found in the `ros_controllers_tutorials` package.

Source code:

```
namespace ros_controllers_tutorials{
  class CombinedResourceController : public controller_interface::ControllerBase
  {  public:
    bool initRequest(hardware_interface::RobotHW* robot_hw,
                     ros::NodeHandle&              root_nh,
                      ros::NodeHandle &             controller_nh,
                      std::set<std::string>&        claimed_resources)
    {
      // Check if construction finished cleanly
      if (state_ != CONSTRUCTED)
      {
        ROS_ERROR("Cannot initialize this controller because it
            failed to be constructed");
        return false;
      }
      // Get a pointer to the joint position control interface
      PositionJointInterface * pos_iface = robot_hw->get<PositionJointInterface>();
      if (!pos_iface)
      {
        ROS_ERROR("This controller requires a hardware interface of type '%s'."
                  " Make sure this is registered in the
                   hardware_interface::RobotHW class.",
                  getHardwareInterfaceType().c_str());
                  return false;
      }
      // Get a pointer to the force-torque sensor interface
      ForceTorqueSensorInterface* ft_iface =
                robot_hw->get<ForceTorqueSensorInterface>();
      if (! ft_iface )
      {
        ROS_ERROR("This controller requires a hardware interface
              of type '%s'." " Make sure this is registered in
              the hardware_interface::RobotHW class.",
              internal::demangledTypeName<ForceTorqueSensorInterface>().c_str());
        return false;
      }
      // Get a pointer to the IMU sensor interface
      ImuSensorInterface* imu_iface = robot_hw->get<ImuSensorInterface>();
      if (!imu_iface)
      {
        ROS_ERROR("This controller requires a hardware interface of type '%s'."
                  " Make sure this is registered in the
                   hardware_interface::RobotHW class.",
                  internal::demangledTypeName<ImuSensorInterface>().c_str());
        return false;
      }
      // Return which resources are claimed by this controller
      pos_iface->clearClaims();
      if (! init(pos_iface,
                 ft_iface,
                 imu_iface,
                 root_nh,
                 controller_nh))
      {
        ROS_ERROR("Failed to initialize the controller");
        std::cerr << "FAILED LOADING WALKING" << std::endl;
        return false;
      }
      claimed_resources = pos_iface->getClaims();
      pos_iface->clearClaims();
      // success
      state_ = INITIALIZED;
```

```
63        return true;
64      }
65      bool init ( PositionJointInterface *       pos_iface,
66                  ForceTorqueSensorInterface* ft_iface,
67                  ImuSensorInterface*         imu_iface,
68                  ros::NodeHandle&            /*root_nh*/,
69                  ros::NodeHandle&            controller_nh)
70      { ... }
71      bool initJoints ( PositionJointInterface * pos_iface,
72                        ros::NodeHandle&       controller_nh)
73      { ... }
74      bool initForceTorqueSensors(ForceTorqueSensorInterface* ft_iface,
75                                  ros::NodeHandle& controller_nh)
76      { ... }
77      bool initImuSensors(ImuSensorInterface* imu_iface,
78                          ros::NodeHandle&  controller_nh)
79      { ... }
80      void update(const ros::Time& time, const ros::Duration& period)
81      {
82        // F-T sensor
83        geometry_msgs::Wrench ft[2];
84        for(unsigned int s = 0; s<2; ++s){
85          ft[s].force.x   = ft_sensors_[s].getForce()[0];
86          ft[s].force.y   = ft_sensors_[s].getForce()[1];
87          ft[s].force.z   = ft_sensors_[s].getForce()[2];
88          ft[s].torque.x = ft_sensors_[s].getTorque()[0];
89          ft[s].torque.y = ft_sensors_[s].getTorque()[1];
90          ft[s].torque.z = ft_sensors_[s].getTorque()[2];
91        }
92        ROS_INFO_STREAM("Force sensor left: fx = "<<
93                        ft[0].force.x<<" fy: "<<
94                        ft[0].force.y<<" fz: "<<
95                        ft[0].force.z<<" tx: "<<
96                        ft[0].torque.x<<" ty: "<<
97                        ft[0].torque.y<<" tz: "<<
98                        ft[0].torque.z);
99        ROS_INFO_STREAM("Force sensor left: fx = "<<
100                        ft[1].force.x<<" fy: "<<
101                        ft[1].force.y<<" fz: "<<
102                        ft[1].force.z<<" tx: "<<
103                        ft[1].torque.x<<" ty: "<<
104                        ft[1].torque.y<<" tz: "<<
105                        ft[1].torque.z);
106
107        // IMU
108        sensor_msgs::Imu value;
109        if (imu_sensor.getOrientation())
110        {
111          value.orientation.x = imu_sensor.getOrientation()[0];
112          value.orientation.y = imu_sensor.getOrientation()[1];
113          value.orientation.z = imu_sensor.getOrientation()[2];
114          value.orientation.w = imu_sensor.getOrientation()[3];
115        }
116        ROS_INFO_STREAM("IMU orientation x: "<<
117                        value.orientation.x<<" y: "<<
118                        value.orientation.y<<" z: "<<
119                        value.orientation.z<<" w: "<<
120                        value.orientation.w);
121      }
122      void stopping(const ros::Time& time)      {}
123      std::string getHardwareInterfaceType()
124            const {return hardware_interface::internal::
125                  demangledTypeName<hardware_interface::PositionJointInterface>();
126  }
127    private:
128      //  Hardware interfaces
129      std::vector<hardware_interface::JointHandle> joints_;
130      hardware_interface::ImuSensorHandle imu_sensor;
131      // We will store two force torque handles,
132      // 0 will be the left ankle sensor and 1 the right ankle sensor
```

```
133     std::vector<hardware_interface::ForceTorqueSensorHandle> ft_sensors_;
134   };
135   PLUGINLIB_DECLARE_CLASS(ros_controllers_tutorials,
136                           CombinedResourceController,
137                            ros_controllers_tutorials::CombinedResourceController,
138                             controller_interface::ControllerBase);
139 }
```

Listing 15: The CombinedResourceController is a controller that subscribes to diferent sensors and prints its values.

30.2.6 Joint torque sensors

All the joints of the robot with the exception of the wrist and head include a torque sensor that measures the exact torque applied by the joint.

```
hardware_interface::JointStateInterface  hw;
hardware_interface::JointStateHandle handle;
handle.getTorqueSensor()
```

The package `joint_torque_sensor_state_controller` contains an example of the usage of this interface.

30.2.7 Absolute encoders

All the joints of the robot include an absolute encoder that is used to initialize every joint of the robot and can be accessed on runtime through the following interface:

```
hardware_interface::JointStateInterface  hw;
hardware_interface::JointStateHandle handle;
handle.getAbsolutePosition();
```

The package `joint_torque_sensor_state_controller` contains an example of the usage of this interface.

30.2.8 Actuator temperature sensors

All the actuators of the robot include a temperature sensor to monitor the motor temperature. The information can be accessed in runtime through the interface

```
hardware_interface::ActuatorTemperatureSensorInterfac hw;
hardware_interface::ActuatorTemperatureSensorHandle handle;
handle.getValue();
```

The package `temperature_sensor_controller` contains an example of the usage of this interface.

30.2.9 Actuator joint mode

All the actuators of the robot support different control modes. The mode of the actuator can be monitored through the following interface:

```
hardware_interface::JointModeHandle hw;
hardware_interface::JointModeHandle handle;
handle.getMode();
```

The actuator control mode is changed automatically to the mode specified by a controller once it is started. As this change is not instantaneous, the controller should wait until the desired control mode is ready for all the claimed resources before sending any command to the actuators.

Keep in mind that this is an actuator's mode not a joint's mode.

The package `mode_state_controller` contains an example of the usage of this interface.

31 Deploying software on the robot

This section contains a brief introduction to the `deploy` script PAL Robotics provides with the development environment.

The `deploy` tool can be used to:

- Install new software onto the robot
- Modify the behaviour of existing software packages by installing a newer version and leaving the original installation untouched.

31.1 Introduction

When TALOS boots up it always adds two sources of packages to its ROS environment. One is the ROS software distribution of PAL Robotics at `/opt/pal/dubnium/`, the other is a fixed location at `/home/pal/deployed_ws`, which is where the `deploy` tool installs to. This location precedes the rest of the software installation, making it possible to overlay previously installed packages.

To maintain consistency with the ROS release pipeline, the `deploy` tool uses the install rules in the CMakeLists.txt of every catkin package. Make sure that everything you need on the robot is declared to be installed.

31.2 Usage

```
usage: deploy.py [-h] [--user USER] [--yes] [--package PKG]
                      [--install_prefix INSTALL_PREFIX]
                      [--cmake_args CMAKE_ARGS]
                      robot

Deploy built packages to a robot. The default behavior is to deploy *all*
packages from any found workspace. Use --package to only deploy a single package.
positional arguments:   robot
                hostname to deploy to (e.g. talos-1c)
optional arguments:
  -h, --help            show this help message and exit
  --user USER, -u USER  username (default: pal)
  --yes, -y             don't ask for confirmation, do it
  --package PKG, -p PKG deploy a single package
  --install_prefix INSTALL_PREFIX, -i INSTALL_PREFIX
                        Directory to deploy files
  --cmake_args CMAKE_ARGS, -c CMAKE_ARGS
                        Extra cmake args like
                        --cmake_args="-DCMAKE_BUILD_TYPE=Release"
e.g.: deploy.py talos-1c -u root -p pal_tts -c="-DCMAKE_BUILD_TYPE=Release"
```

31.3 Notes

- The build type by default is not defined, meaning that the compiler will use the default C++ flags. This is likely to include -O2 optimization and -g debug information, meaning that, in this mode, executables and libraries will go through optimization during compilation and will therefore have no debugging symbols. This behaviour can be changed by manually specifying a different option such as:
  `--cmake_args="-DCMAKE_BUILD_TYPE=Debug"`

- Different flags can also be set by chaining them:
  `--cmake_args="-DCMAKE_BUILD_TYPE=Debug -DPCL_ONNURBS=1"`
- If an existing library is overlayed, executables and other libraries which depend on this library may break. This is caused by ABI / API incompatibility between the original and the overlaying library versions. To avoid this, is it recommended to simultaneously deploy the packages that depend on the changed library.
- There is no tool to remove individual packages from the deployed workspace except to delete the `/home/pal/deployed_ws` folder altogether.

31.4 Deploy tips

- You can use an alias (you may want to add it to your `.bashrc`) to ease the deploy process:
  ```
  alias deploy="rosrun pal_deploy deploy.py"
  ```
- You can omit the `--user pal` as it is the default argument
- You may deploy a single specific package instead of the entire workspace:
  ```
  deploy -p hello_world talos-1c
  ```
- You can deploy multiple specific packages instead of the entire workspace:
  ```
  deploy -p "hello_world other_local_package more_packages" talos-1c
  ```
- Before deploying you may want to do a backup of your previous `~/deployed_ws` in the robot to be able to return to your previous state, if required.

31.5 Use-case example

31.5.1 Adding a new ROS Package

In the development computer, load the ROS environment (you may add the following instruction to the `~.bashrc`)

```
source /opt/pal/dubnium/setup.bash
```

Create a workspace

```
mkdir -p ~/example_ws/src
cd ~/example_ws/src
```

Create a catkin package

```
catkin_create_pkg hello_world roscpp
```

Edit the `CMakeLists.txt` file with the contents in figure 59.

```
cmake_minimum_required(VERSION 2.8.3)
project(hello_world)

find_package(catkin REQUIRED COMPONENTS roscpp)

catkin_package(
)

include_directories(
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ executable
add_executable(hello_world_node src/hello_world_node.cpp)
target_link_libraries(hello_world_node ${catkin_LIBRARIES})

## Mark executables and/or libraries for installation
install(TARGETS hello_world_node
  RUNTIME DESTINATION
${CATKIN_PACKAGE_BIN_DESTINATION})
```

Figure 59: Hello world CMakeLists.txt

Edit the `src/hello_world_node.cpp` file with the contents in figure 60.

```
// ROS headers
#include <ros/ros.h>

// C++ std headers
#include <iostream>

int main(int argc, char** argv)
{
  ros::init(argc, argv, "hello_world");

  ros::NodeHandle nh("~");

  std::cout << "Hello world" << std::endl;

  return 0;
}
```

Figure 60: Hello world C++ source code

Build the workspace

```
cd ~/example_ws
catkin build
```

The expected output is shown in figure 61.

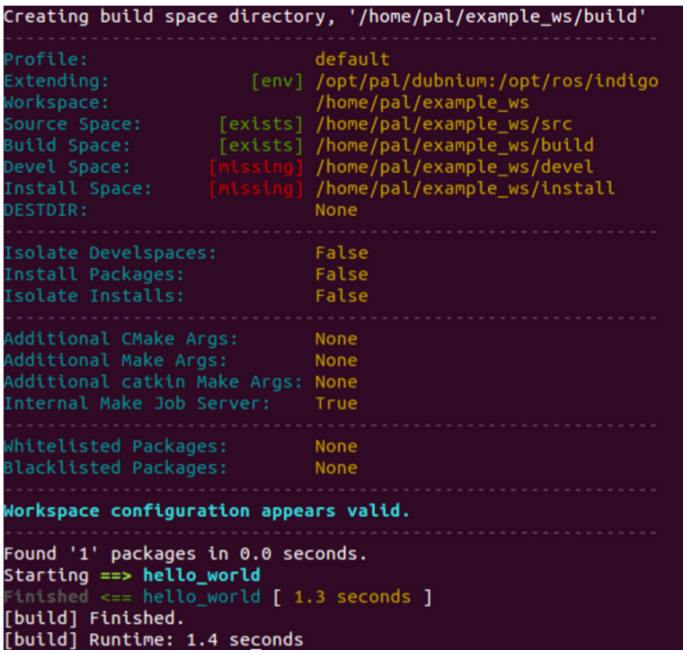


Figure 61: Build output of hello world package

Deploy the package to the robot:

```
cd ~/example_ws
rosrun pal_deploy deploy.py --user pal talos-1c
```

The deploy tool will build the entire workspace in a separate path and, if successful, it will request confirmation in order to install the package on the robot, as shown in figure 62.

```
[clean] No buildspace exists, no CMake caches to clear.
Preparing install space
Creating build space directory, '/home/pal/example_ws/build_pal_deploy'
------------------------------------------------------------------------
Profile:                     pal_deploy
Extending:             [env] /home/pal/example_ws/devel:/opt/pal/dubnium:/opt/ros/indigo
Workspace:                   /home/pal/example_ws
Source Space:       [exists] /home/pal/example_ws/src
Build Space:        [exists] /home/pal/example_ws/build_pal_deploy
Devel Space:       [missing] /home/pal/example_ws/devel_pal_deploy
Install Space:     [missing] /home/pal/example_ws/install_pal_deploy
DESTDIR:                     None
------------------------------------------------------------------------
Isolate Develspaces:         False
Install Packages:            True
Isolate Installs:            False
------------------------------------------------------------------------
Additional CMake Args:       -DCATKIN_BUILD_BINARY_PACKAGE=0 -DCMAKE_CXX_FLAGS_DEBUG=-g -O0 -DCMAKE_C_FLAGS_DEBUG=-g -O0 -DCATKIN_ENABLE_TESTING=OFF
LL_PREFIX=/home/pal/example_ws/install_pal_deploy/home/pal/deployed_ws
Additional Make Args:        None
Additional catkin Make Args: None
Internal Make Job Server:    True
------------------------------------------------------------------------
Whitelisted Packages:        None
Blacklisted Packages:        None
------------------------------------------------------------------------
Workspace configuration appears valid.
------------------------------------------------------------------------

Found '1' packages in 0.0 seconds.
Starting ==> hello_world
Finished <== hello_world [ 2.7 seconds ]
[build] Finished.
[build] Runtime: 2.8 seconds
Using catkin install space: /home/pal/example_ws/install_pal_deploy/home/pal/deployed_ws
=> Deploying package hello_world
I'm about to run the following command in /home/pal/example_ws/install_pal_deploy/home/pal/deployed_ws:
  rsync -avz /home/pal/example_ws/install_pal_deploy/home/pal/deployed_ws/ pal@tiago-0c:/home/pal/deployed_ws
Do it? (Y/n):
```

Figure 62: Deployment of hello world package

Press `Y` so that the package files are installed on the robot computer. Figure 63 shows the files that are copied for the hello world package, according to the installation rules specified by the user in the `CMakeLists.txt`.

```
Syncing binaries with robot...
The authenticity of host 'tiago-0c (192.168.1.33)' can't be established.
RSA key fingerprint is 4c:0e:17:4c:39:48:f7:af:51:87:62:7d:b0:0b:fb:be.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/pal/.ssh/known_hosts).
pal@tiago-0c's password:
sending incremental file list
./
.catkin
_setup_util.py
lib/
lib/hello_world/
lib/hello_world/hello_world_node
lib/pkgconfig/
lib/pkgconfig/hello_world.pc
share/
share/hello_world/
share/hello_world/cmake/

sent 7,576 bytes  received 2,515 bytes  1,552.46 bytes/sec
total size is 291,517  speedup is 28.89

******************
Done. Time to try!
******************
```

Figure 63: Installation of the hello world package to the robot

Then connect to the robot:

```
ssh pal@talos-1c
```

And run the new node as follows:

```
rosrun hello_world hello_world_node
```

If everything goes well you should see 'Hello world' printed on the screen.

31.5.2 Adding a new controller

One use-case for the tool is to add or modifiy controllers. Let's take the `ros_controllers_tutorials` package, as it contains simple controllers, to demonstrate the power of deploying.

First, list the known controller types on the robot. Open a new terminal and execute the following:

```
export ROS_MASTER_URI=http://talos-1c:11311
rosservice call /controller_manager/list_controller_types | grep HelloController
```

As it is a genuine installation, the result should be empty.

Assuming a running robot and a workspace on the development computer called `talos_ws` that contains the sources of `ros_controllers_tutorials`, open a new terminal and execute the following commands:

```
cd talos_ws
catkin_make #-j5 #optional
source devel/setup.bash # to get this workspace into the development environment
rosrun pal_deploy deploy.py --package ros_controllers_tutorials talos-1c
```

The script will wait for confirmation before copying the package to the robot.

Once successfully copied, restart the robot and run the following commands again:

```
export ROS_MASTER_URI=http://talos-1c:11311
rosservice call /controller_manager/list_controller_types | grep HelloController
```

Now, a list of controller types should appear. If terminal highlighting is enabled, “HelloController” will appear in red.

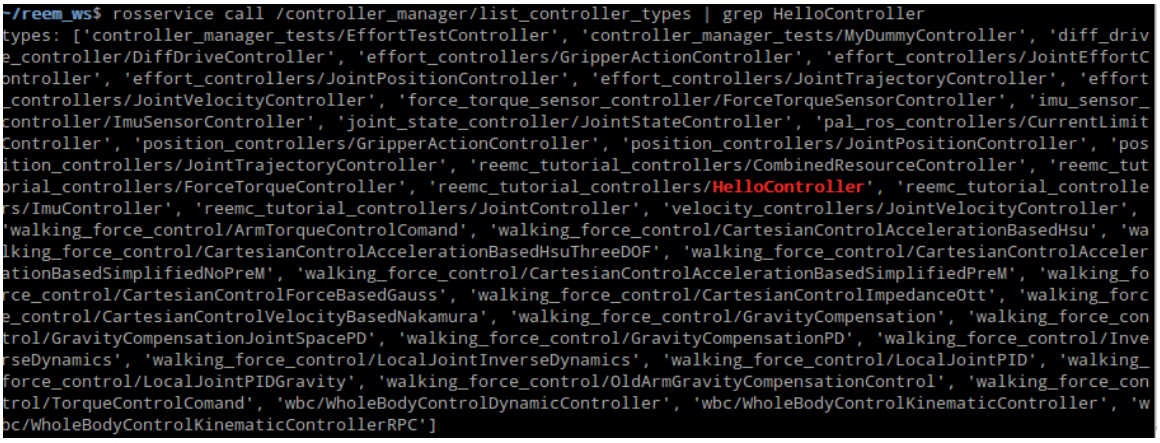


Figure 64: List of contoller types

31.5.3 Modifying an installed package

Now let’s suppose we found a bug on an installed controller inside the robot. In this case, we’ll change the joint_state_controller/JointStateController.

Go to https://github.com/ros-controls/ros_controllers, open a new terminal and execute the following commands:

```
cd talos_ws/src
git clone https://github.com/ros-controls/ros_controllers
# Fix bugs in controller
cd ..
catkin_make #-j5 #optional
source devel/setup.bash # to get this workspace into the development environment
rosrun pal_deploy deploy.py --package joint_state_controller talos-1c
```

After rebooting the robot, the controller with the fixed changes will be loaded instead of the one installed in /opt/

32 Modifying Robot Startup

This section describes how the startup system of the robot is implemented and how to modify it, in order to add new applications, modify how they are launched, or prevent applications from being launched at all.

32.1 Introduction

TALOS startup is configured via YAML files that are loaded as ROS Parameters upon robot startup.

There are two types of files: configuration files that describe how to start an application and files that determine which applications must be started for each computer in a robot.

All these files are in the pal_startup_base package within the config directory.

32.1.1 Application start configuration files

These files are placed inside the apps directory within config.

foo_bar.yaml contains a YAML description on how to start the application foo_bar.

```
roslaunch: "foo_bar_pkg foo_bar_node.launch"
dependencies: ["Functionality: Foo", "Functionality: Bar"]
timeout: 20
```

The required attributes are:

- One of *roslaunch*, *rosrun* or *bash*: used to determine how to start the application. The value of *roslaunch, rosrun* or *bash* is the rest of the commands that you would use in a terminal (you can use bash magic inside such as ‘rospack find my_cfg_dir‘). There are also some keywords that are replaced by the robot’s information in order to make scripts more usable. @robot@ is replaced by the robot name as used in our ROS packages (ie REEMH3 is reem, REEM-C is reemc, ...)
- *dependencies:* a list of dependencies that need to be running without error before starting this application. Dependencies can be seen in the diagnostics tab on page 82. If an application has no dependencies, it should be set to an empty list [].

Optional attributes:

- *timeout*: applications whose dependencies are not satisfied after 10 seconds are reported as an error. This timeout can be changed with the timeout parameter.
- auto_start: Determines whether this application must be launched as soon as its dependencies are satisfied, if not specified defaults to True.

Examples:

localization.yaml

```
roslaunch: "@robot@_2dnav localization_amcl.launch"
dependencies: ["Functionality: Mapper", "Functionality: Odometry"]
```

web_commander.yaml

```
rosrun: "pal_webcommander web_commander.sh"
dependencies: []
```

32.1.2 Computer start lists

The other type of YAML configuration files are the lists that determine what to start for each robot's computer. They are placed within the config directory, inside a directory with the name of the computer that must start them, for instance *control* for the default computer in all of PAL Robotics' robots, or *multimedia* for robots with a dedicated multimedia computer.

Each file contains a single YAML list with the name of the applications, which are the names of the YAML files for the application start configuration files.

Each file has a name that serves as a namespace for the applications contained within it. This allows the user to modify a subset of the applications to be launched.

Examples:

pal_startup_base/config/control/core.yaml

```
# Core
- ros_bringup
- diagnostic_aggregator
- web_commander

# Deployers
- deployer_xenomai

#Navigation
- laser_ros_node
- map_server
- compressed_map_publisher
- map_configuration_server
- vo_server
- localizer
- move_base
- navigation_sm
- poi_navigation
- pal_waypoint

# Utilities
- computer_monitor_control
- remote_shell_control
- rosbridge
- tablet_backend
- ros_topic_monitor
- embedded_networking_supervisor
```

32.1.3 Additional startup groups

Besides the *control* group, and the multimedia group for robots that have more than one computer, additional directories can be created in the config directory at the same level as the *control* directory.

These additional groups are typically used to group different applications in a separate tab in the WebCommander, such as the Startup Extras optional tab.

A startup_manager pal_startup_node.py instance is required to handle each startup group.

For instance if a group called *grasping_demo* is needed to a manage the nodes of a grasping demo started in the control computer, a directory will have to be created called *grasping_demo* containing at least one computer start list yaml file as described in the previous section.

Additionally it is recommended that we add to the control´s computer startup list a new application that will start the startup manager of the *grasping_demo* so it is available from the start.

```
rosrun: "pal_startup_manager pal_startup_node.py grasping_demo"
 dependencies: []
```

32.2 Startup ROS API

Each startup node can be individually controlled using a ROS api that consists of the following services, where {startup_id} must be substituted for the name of the corresponding startup group (ie control, multimedia or grasping_demo).

/**pal_startup_{startup_id}/start** Arguments are **app** (name of the application as written YAML files for the application start configuration files) and **args** (optional command line arguments). Returns a string containing if the app was started successfully.

/**pal_startup_{startup_id}/stop** Arguments are **app** (name of the application as written YAML files for the application start configuration files). Returns a string containing if the app was stopped successfully.

/**pal_startup_{startup_id}/get_log** Arguments are **app** (name of the application as written YAML files for the application start configuration files) and **nlines** (number of lines of the log file to return). Returns up to the last nlines of logs generated by the specified app.

/**pal_startup_{startup_id}/get_log_file** Arguments are **app** (name of the application as written YAML files for the application start configuration files). Returns the path of the log file of the specified app.

32.3 Startup command line tools

pal-start This command will start an application in the background of the computer it is executed on, if it is stopped. Pressing TAB will list the applications that can be started.

pal-stop This command will stop an application launched via pal_startup in the computer it is executed on, if it is started. Pressing TAB will list the applications that can be stopped.

pal-log This command will print the name and path of the log file of the selected application. Pressing TAB will list the applications whose log can be seen.

32.4 ROS Workspaces

The startup system will look for packages in the following directories in order, if a package is found in one of the directories, it will not be looked for any further on directories lower in the list.

- /home/pal/deployed_ws (see 31)
- /opt/pal/dubnium
- /opt/ros/indigo

32.5 Modifying the robot's startup

In order to enable the robot's users to fully customize the startup of the robot, in addition to using the files located in the config directory of the pal_startup_base package, the startup procedure will also load all the parameters within `/home/pal/.pal/pal_startup/` of the robot's *control* computer, if it exists.

To modify the robot's startup, this directory must be created and have the same structure as the config directory within the pal_startup_base package.

32.5.1 Adding a new application for automatic startup

To add a new application, "new_app", to the startup, create a new_app.yaml file within the apps directory. Fill it with the information described in Application start configuration files

The file we created specifies how to start the application, in order to launch the application in the *control* computer, create a *control* directory and place it inside a new yaml file, which must consist of a list containing new_app.

For instance:

/home/pal/.pal/pal_startup/apps/new_app.yaml

```
roslaunch: "new_app_package new_app.launch"
dependencies: []
```

/home/pal/.pal/pal_startup/control/new_app_list.yaml

```
- new_app
```

32.5.2 Modifying how an application is launched

To modify how the application "foo_bar" is launched, copy the contents from the original foo_bar.yaml file in the pal_startup_base package and perform the desired modifications.

32.5.3 Adding a new workspace

In cases where the workspace resolution process needs to be changed, the file at /usr/bin/init_pal_env.sh can be modified to adapt the environment of the startup process.

33 Motion planning with *MoveIt!*

Description This section covers how to perform collision-free motions on TALOS using the graphical interface MoveIt!. Collision-free motion planning is performed by chaining a probabilistic sampling-based motion planner with a trajectory filter for smoothing and path simplification.

For more information, C++ API documentation and tutorials go to the following website:http://moveit.ros.org.

33.1 Getting started with the *MoveIt!* graphical user interface

This subsection provides a brief introduction to some basic use cases of *MoveIt!*. For testing, a TALOS simulation is recommended.

1. Start a simulation with default controllers.
   To launch a simulation in one terminal, execute:
   ```
   roslaunch talos_gazebo talos_gazebo.launch
   ```
   And in another terminal, launch the default controllers.
   ```
   roslaunch talos_controller_configuration_gazebo default_controllers.launch
   ```
2. Start *MoveIt!* with the GUI in another terminal
   ```
   roslaunch talos_moveit_config moveit_rviz.launch
   ```

This command will start the motion planning services along with visualization. Do not close this terminal.

1. The GUI is RViz, with a custom plugin for executing and visualizing motion planning.
2. *MoveIt!* uses planning groups to generate solutions for different kinematic chains. Figure 65 shows that by selecting different planning groups, the GUI shows only the relevant “flying end-effectors”.

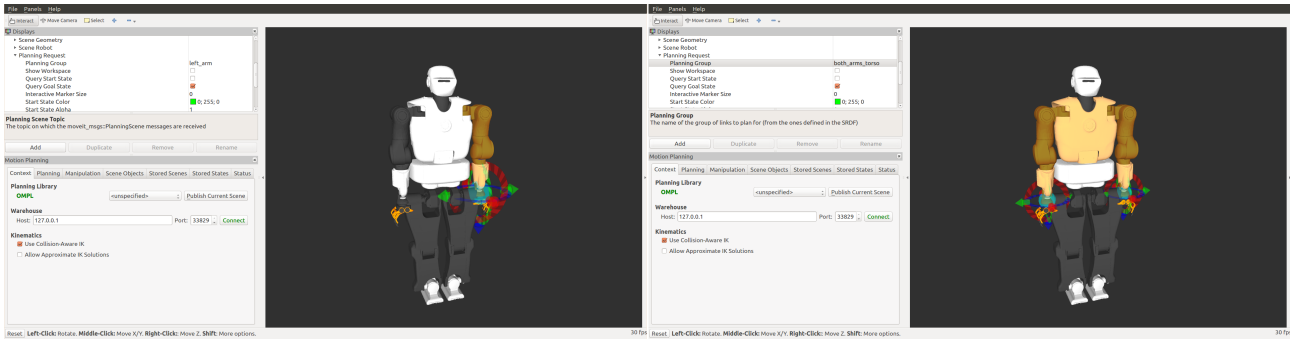


Figure 65: *MoveIt!* graphical interface in RViz. TALOS with the planning group *both_arms* (left) and with the planning group *left_arm* (right).

1. In order to use motion planning, a Start State and Goal State has to be known, to do this with the *MoveIt!* GUI navigate to the tab called “Planning” in the display called MotionPlanning. On this tab, further nested tabs provide the functionality required to update Start State and Goal State depicted in Figure 66.
2. By clicking the “Update” button of the goal state, new poses can be randomized. Figure 67 shows a few randomly generated poses.
3. The sequence of images in Figure 68 shows the result of clicking the *“Update”* goal pose with random in RViz, then clicking *“Plan and Execute”* in the simulator. RViz will visualize the plan before executing.
4. The GUI also allows the operator to define goal poses using the “flying end-effector” method. As shown in Figure 67, the 6DOF pose of the end-effector can be defined using the visual marker attached to it. The red, green and blue arrows define the translation of x, y and z coordinates respectively; the colored rings define the rotation around these axes, following the same logic.

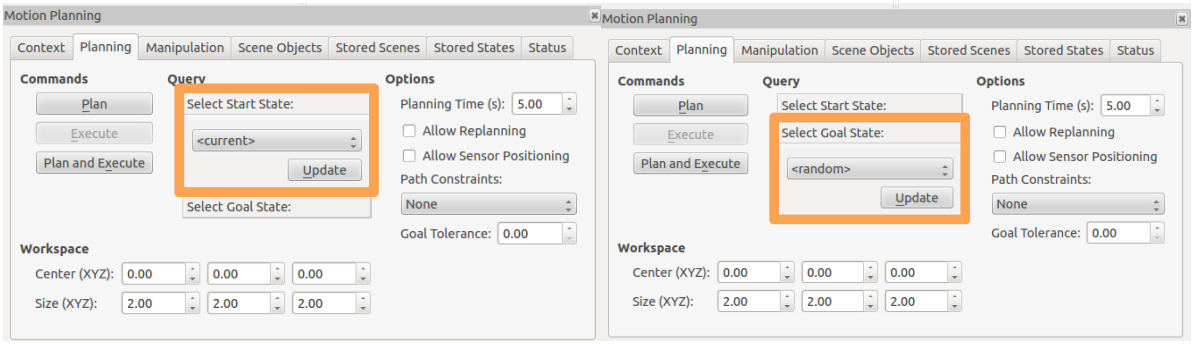


Figure 66: Tab to set start state (left) and tab to set goal state (right).

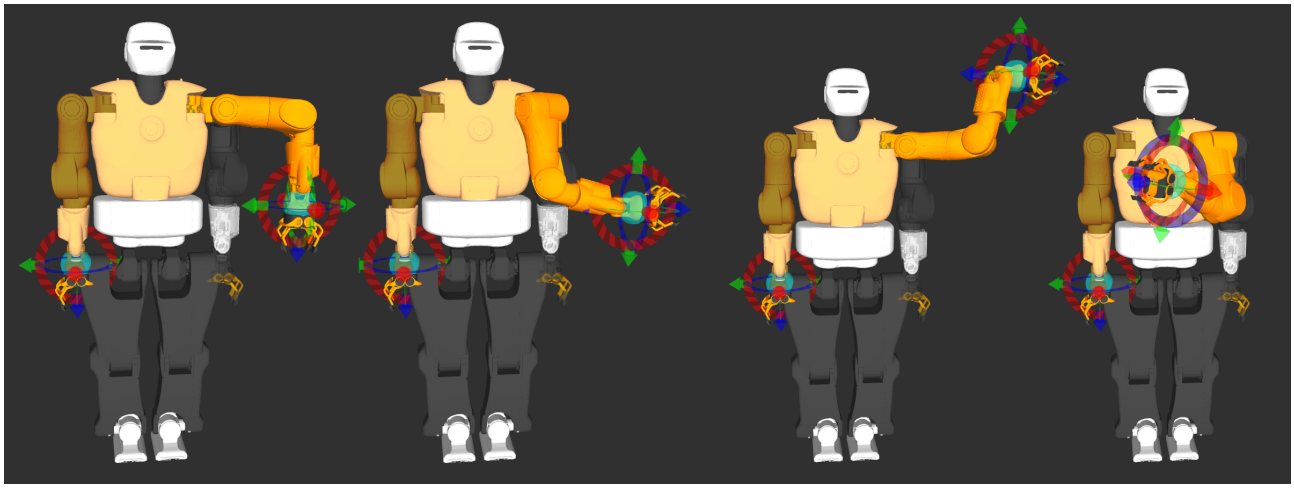


Figure 67: Random poses generated with the GUI

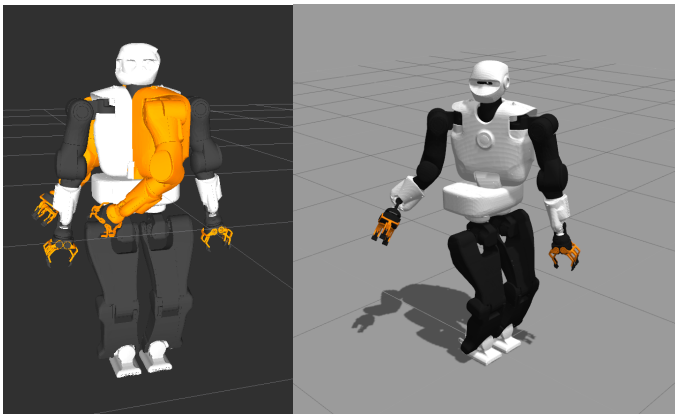


Figure 68: Sequence of plan and execute states

33.2 End test

To close the *MoveIt!* GUI, hit `Ctrl-C` in the terminal used in step 2. The running instance of `Gazebo` can be used for further work, but keep in mind that the robot will remain in the pose it was last sent to.

34 Facial perception

34.1 Overview

This chapter presents the software package for face and emotion recognition included in TALOS [3].

Face and emotion recognition are implemented on top of theVerilook Face SDK provided by Neurotechnolgy.

The ROS package implementing facial perception subscribes to `/xtion/rgb/image_raw` image and processes this topic at 3 Hz in order to provide the following information:

- Multiple face detection
- 3D position estimation
- Gender classification with confidence estimation
- Face recognition with matching confidence
- Facial attributes: eye position and expression
- Emotion confidences for six basic emotions

Figure 69: Face processing example

34.2 Facial perception ROS API

34.2.1 Topic interfaces

`/pal_face/faces` (pal_detection_msgs/FaceDetections)

Array of face data found in the last processed image

`/pal_face/debug` (sensor_msgs/Image)

Last processed image with overlayed face data: face ROIs, gender, facial features, name and emotion.
In order to visualize the debug image, enter the following command from a development computer:

```
export ROS_MASTER_URI=http://talos-1c:11311
rosrun image_view image_view image:=/pal_face/debug
_image_transport:=compressed
```

[3]This software is included in the Facial Perception Premium Software package

34.2.2 Service interface

`/pal_face/set_database` (pal_detection_msgs/SetDatabase)

Service to select the active database in the robot.

For example, in order to select a database named `test_face` and empty its contents, if any:

```
rosservice call /pal_face/set_database test_face True
```

In order to select an existing database named `office_faces` without emptying its contents:

```
rosservice call /pal_face/set_database office_faces False
```

`/pal_face/start_enrollment` (pal_detection_msgs/StartEnrollment

To start learning a new face, select a database and give the name of the person as argument.

As an example, to start learning the face of `Anthony`:

```
rosservice call /pal_face/start_enrollment Anthony
```

During enrollment it is important that the person stands in front of the robot looking at its camera, moving slightly their head to different distances, i.e. between 0.5 and 1.5 m, and changing their facial expression. Face samples of the person will be gathered until `/pal_face/stop_enrollment` is called. An enrollment of 20 - 30 seconds should be enough.

`/pal_face/stop_enrollment` (pal_detection_msgs/StopEnrollment

Service to stop learning a new face:

```
rosservice call /pal_face/stop_enrollment
```

During enrollment, it is important that the person stands in front of the robot looking at its camera, moving their head slightly to different distances, i.e. between 0.5 and 1.5 m, while changing their facial expression.

`/pal_face/recognizer` (pal_detection_msgs/Recognizer

Service to enable or disable the face recognition. By default, the recognizer is disabled.

In order to enable the recognizer, a minimum matching confidence must be specified:

```
rosservice call /pal_face/recognizer True 60
```

Only detected faces matched to a face in the database with confidence greater or equal to the one specified will be reported.

In order to disable the recognizer:

```
rosservice call /pal_face/recognizer False 0
```

34.3 Face perception guidelines

In order to improve the performance of facial perception and to ensure its correct behavior, some basic guidelines have to be taken into account:

- Do not have the robot looking at backlight, i.e. out of a window or towards an indoor lamp. The brightness of the light source will cause high contrast in the images, so that faces may be too dark to be detected.
- Best performance is achieved when the subjects are enrolled and recognized at a distance of between 0.5 and 1.2 m from the camera. The further away the person, the worse recognition confidences will be obtained.
- When enrolling a new person in a database, it is mandatory that no other faces appear in the image. Otherwise, the stored face data will contain features of both people and the recognition will fail.
- In order to reduce the CPU load, the face recognizer should be disabled when possible.

Examples

35 Overview

These sections specify the tests that validate the software support of TALOS. They are useful as learning material with which to familiarize yourself with the robot's different functionalities. In the sections that follow, the scope and goal of each test is described, along with step-by-step instructions for their execution. The expected results of a successful test –or part of it– are explicitly presented in the instructions as command-line or graphical feedback that the user should be able to reproduce.

Requisites These acceptance tests form a self-contained suite. However, since ROS (Robot Operating System) is extensively used, the unfamiliar reader might find it useful to browse through the description and tutorials available online.

All tests have to be executed as user *pal*, which by default has the password *pal*.

35.1 TALOS's ROS environment

The TALOS ROS environment consists of all the ROS packages required to run a TALOS robot. Every command-line terminal of the development computer that will run TALOS software –or build software against it– needs to be aware of its location.

- To setup the **current terminal** to pick up TALOS's ROS environment, run the following command:

  ```
  source /opt/pal/dubnium/setup.bash
  ```

- To avoid typing the above command in every terminal that requires TALOS's ROS environment, it is helpful to automatically execute it every time a new terminal is created. To setup the **current and all future terminals** to pick up TALOS's ROS environment, run the following commands *once:*

  ```
  echo "source /opt/pal/dubnium/setup.bash" >> ~/.bashrc
  source ~/.bashrc
  ```

  The remainder of this document assumes that all terminals have been properly setup to pick up TALOS's ROS environment.

36 Setup

These points describe the setup for running tests on a simulated or a real robot. These instructions are common for all tests. Note that not all tests can be run in both simulated and real robots; when this is the case, the documentation will state it explicitly.

Whenever a test requests to **open a terminal connected to TALOS**, it refers to one of the options below.

36.1 In a simulated environment

36.1.1 Start a simulated TALOS

1. Open a terminal and launch a Gazebo simulation of TALOS in an empty world (Figure 70).

```
roslaunch talos_gazebo talos_empty_world.launch
```

```
roslaunch talos_gazebo talos_gazebo.launch
rosrun rviz rviz
```

Select *Robot Model* and then *base_link* as a *Fixed Frame*.

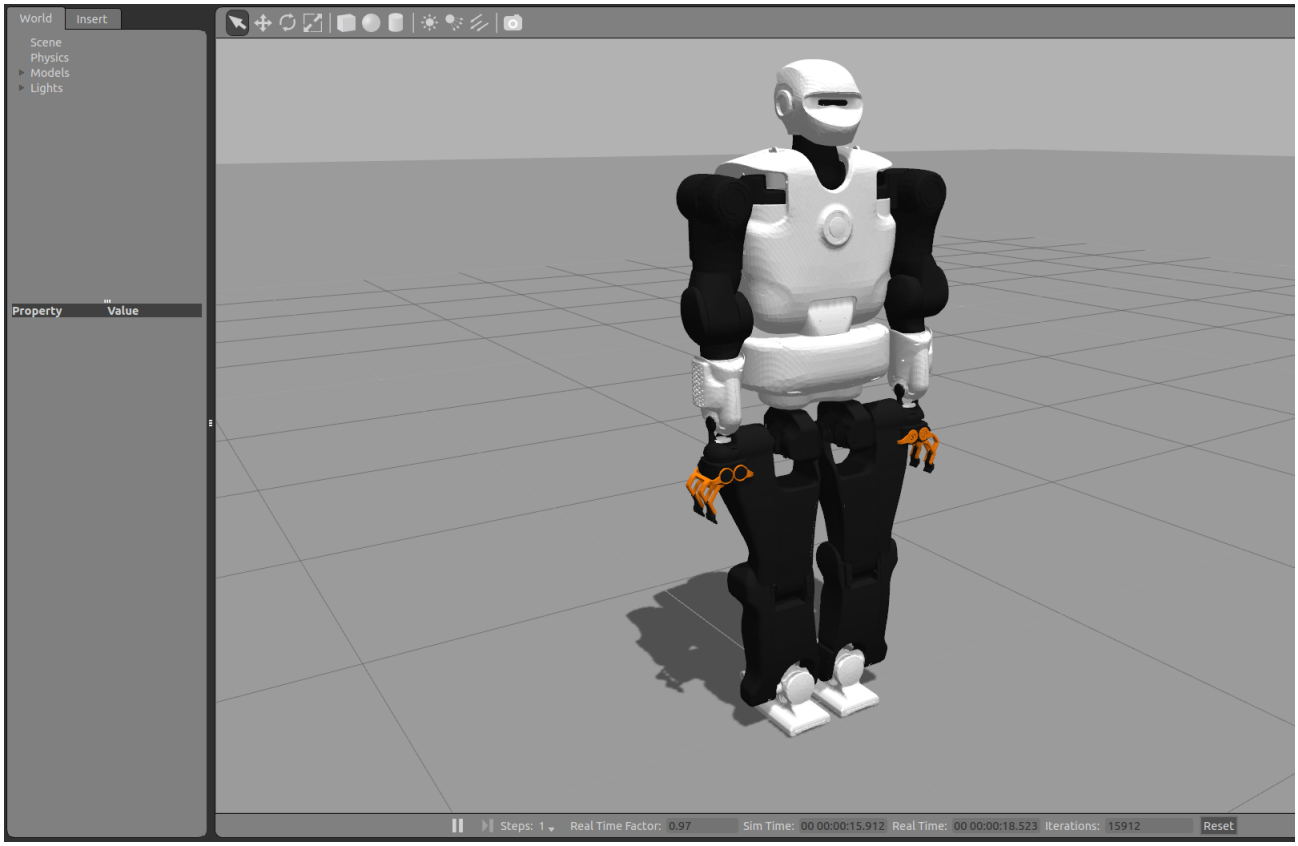


Figure 70: Simulated robot

36.1.2 Connect a terminal to the robot

Since the simulation runs on the local host, a newly opened terminal should be able to connect to it without further settings.

36.2 On the real robot

36.2.1 Start a real TALOS

Please refer to section 12 – Getting started, for details on how to start TALOS.

36.2.2 Connect a terminal to the robot

Open a new terminal and execute the following:

```
export ROS_MASTER_URI=http://talos-1c:11311
```

Until closed, this terminal will be able to connect to the ROS interfaces exposed by the remote computers on-board TALOS.

36.2.3 A note on switching from simulated to real robot deployments (advanced)

If you are reusing an existing `ros_core` and have previously launched the simulation, the `/use_sim_time` ROS parameter will be set to true, and non-simulation tests will not work for lack of a simulated time source (i.e. the `/clock` topic). You can fix the problem by deleting `/use_sim_time` from the parameter server (i.e. `rosparam delete /use_sim_time`), or simply killing the existing `ros_core` instance and starting a new one.

37 Device State Notifications

Description The procedure described in this section is to test the device state notification system embedded in the TALOS robot. This system provides a quick overview of the robot's internal state and helps easily identify any abnormal situation. This test applies only to real TALOS deployments, not to simulated ones.

Requirements The `WebCommanderServer` daemon must be running. To check if it is running, perform the following commands:

```
ssh pal@talos-1c
pgrep -f pal_webcommander_node
exit
```

If the pgrep command returns a number it means the WebCommander server is running and the returned number is its PID (Process Identifier)

37.1 Accessing the device state notification website

1. Open a web browser[4], and connect to the WebCommander website following the instructions in the Software Development Environment section of the manual. In most setups, you must type:

   ```
   http://talos-1c:8080
   ```

[4]We recommend not using Firefox, as it currently fails to load correctly all elements. We have validated WebCommander on Google Chrome.

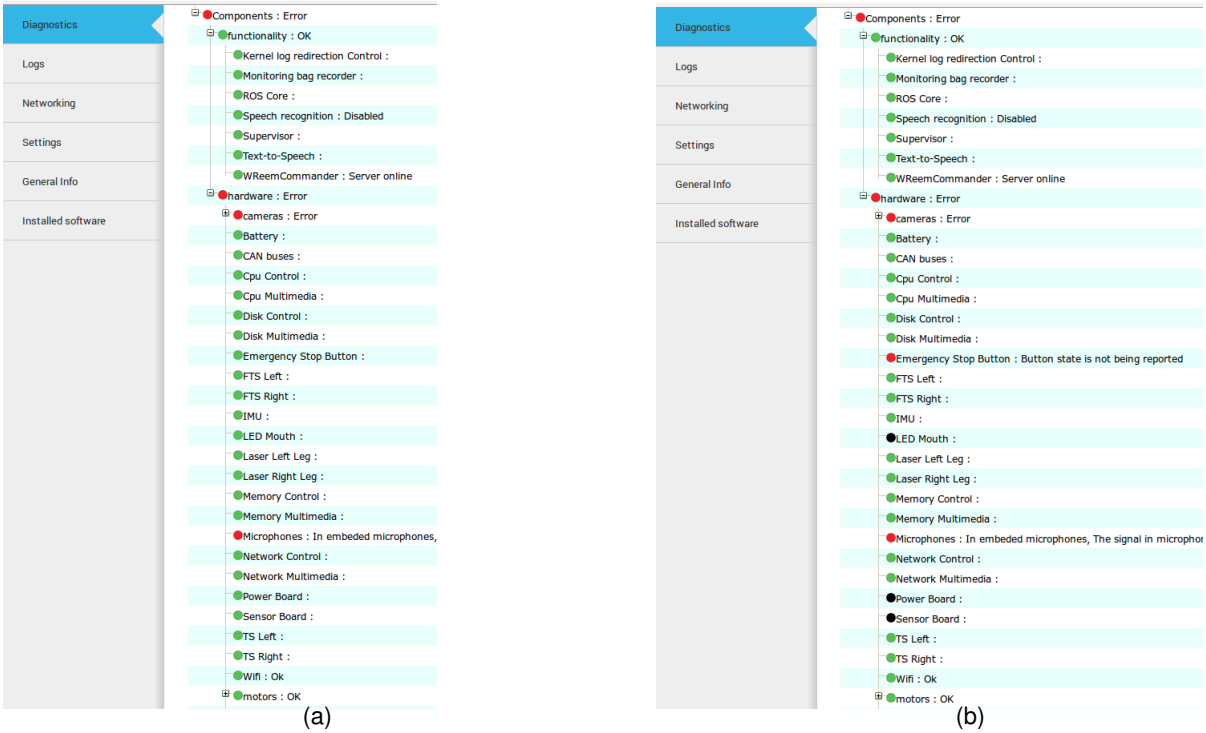


Figure 71: Diagnostics. Before stopping sensor board application (a). After stopping sensor board application (b).

37.2 Diagnostics Tab

To test that the diagnostics are being published correctly, introduce a severe error in the system and check it is being displayed in the diagnostics.

In order to simulate a hardware failure, log into the control computer and stop the application that accesses the Sensor Board 04:

```
ssh pal@talos-1c
```

Once logged in, execute:

```
sb04r03ProdStop.sh
```

After stopping the application, we can see that in the hardware diagnostics, the Power Board and Sensor Board went black, meaning that we stopped receiving diagnostic information from the applications that can be seen in Figure 71(b).

After the errors are displayed, re-start the application to make sure the errors are cleared.

From the same terminal used before:

```
sb04r03ProdStart.sh
```

The applications that had become stale after stopping the application should go back to ok state (green).

Finally, log off TALOS's control computer.

```
exit
```

37.3 Logs Tab

As in the Diagnostics Tab acceptance test, when the error is introduced in the system, one or more log error messages describing the issue will be printed and will be visible in the Logs Tab.

38 Moving individual joints – graphical user interface

Description This test covers how to move TALOS's individual joints to a desired goal configuration using a graphical user interface. This test is similar to the one presented in section 38, the only difference being the mechanism used to send motion commands.

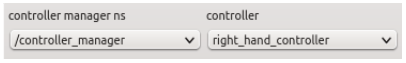
Requisites A running TALOS, as described in section 36.

38.1 Setup

38.1.1 Launch the GUI

1. Launch the Rqt GUI and joint trajectory controllers, as described in section 22.1.2 and shown in Figure 50.

2. On the top-left of the GUI is the panel for controlling joint positions, named `Joint trajectory controller`, detailed in Figure 72.
   Activate the right hand controller by selecting `/controller_manager` and `right_hand_controller` on the panel's combo boxes, as shown below.

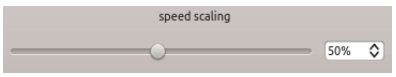


   Note: If instead of `right_hand_controller` the user selects any of the other options (arms, legs, torso, head), joint commands can be sent to the corresponding group.

3. By default, the controller panel is in **monitor mode**, as indicated by the red status button shown below. In monitor mode the joint sliders display the current joint positions but do not accept new commands. By clicking on the status button, the controller panel enters **control mode**, and the status button turns green.

4. In control mode, joint positions can be set by either moving the sliders or by providing discrete setpoints on the box to the right of each slider.

5. As a safety feature, at the bottom of the joint control panel there is an additional slider that allows to scale the maximum velocity of the commands sent to the joints.



6. As the operator sends commands to the joints, their time evolution can be monitored both in the data plots and the 3D rendering of the robot, as depicted in Figure 50.

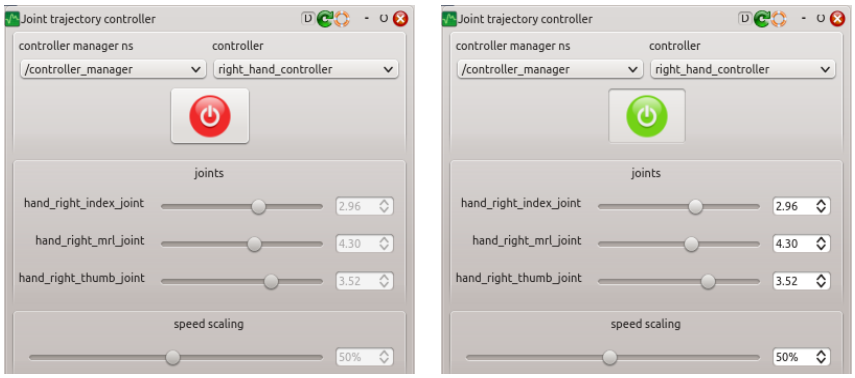


Figure 72: Detail of the joint control panel in monitor (left) and control (right) modes.

Safety note If an emergency stop takes place while the `Joint trajectory controller` panel is in **control mode**, make sure to bring the panel back to **monitor mode**, **before** releasing the emergency stop.

This is important because when the emergency stop is activated the mechanism might change configuration due to the effect of gravity or other external loads. If the emergency stop is released with the panel in **control mode** (undesired), the mechanism will immediately start tracking the specified commands, which could be substantially different from the current configuration, leading to undesired abrupt motions.

38.2 End test

Press `Ctrl-C` to close all terminals used in this test.

39 Walking examples

Description This test shows how to send walking commands to TALOS robot on simulation and on the real hardware using the available interfaces: ROS topics, services and actions. TALOS can walk accepting three types of commands:

1. Specifying the COM (center of mass) velocity using a topic.
2. Specifying a number of steps parametrized by step length and step time.
3. Specifying a list of precomputed steps.

Requisites A running TALOS robot, as described in section 36.

39.1 Setup

39.1.1 Notes when running on the real robot

If running this test on real hardware, TALOS must be turned on hanging on the crane with the feet not touching the ground before it starts walking.

39.1.2 Starting the walking controller

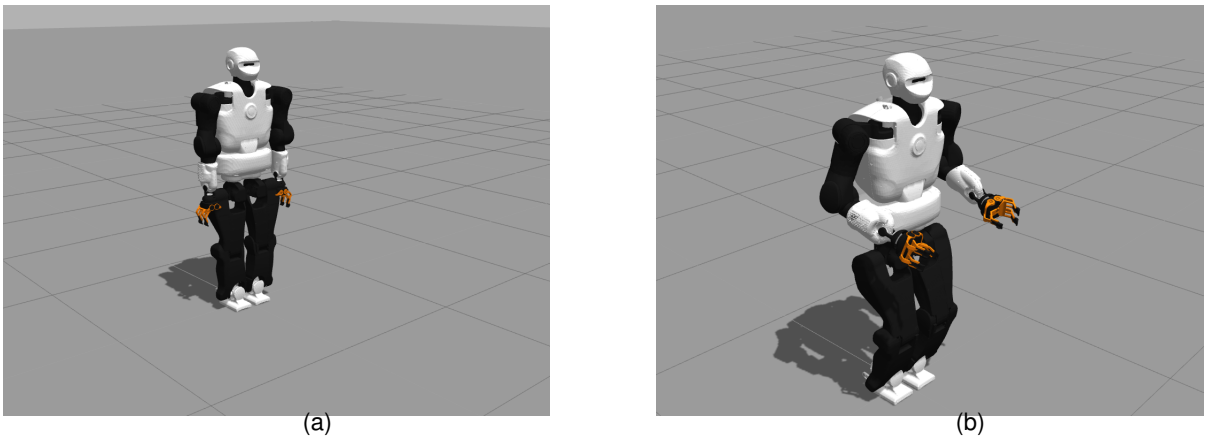
The following commands in the manual are independent of the real robot or simulation.

In order to start the `walking_controller`, the parameters have to be loaded in the parameter server and the controller has to be started. TALOS must be started in a upright position with the knees extended as shown in Figure 73 (a), once the `walking_controller` starts it will place TALOS in the walking configuration, with the knees bent and the arms in walking position depicted in Figure 73 (b). Once the controller has started, TALOS can be lowered from the crane making the feet touch the ground and with enough slackness on the safety ropes so they don't pull the robot while walking.

```
roslaunch talos_controller_configuration default_controllers.launch
```

The `default_controllers` launch file will start `joint_trajectory_controllers` for the upper body, and the `walking_controller` for the lower body. This is the controllers configuration that will be used most frequently in the robot, hence its name. A stabilizer algorithm is included in the walking controller.

Don't execute anything in this terminal and leave it open during all the walking process, open a new terminal for each test and only close this terminal when you want to stop the `walking_controller` and the `joint_trajectory_controllers`.



(a) (b)

Figure 73: Walking configuration. Start position (a). Walking controller started (b).

39.2 Walking using topic interface

Description This acceptance test will show how to send velocity commands to TALOS using velocity references for the center of mass. TALOS will generate the steps automatically in order to try to track this reference velocity. The test will command TALOS to go in the front, back, sideway direction and turn.

Source code The source code for this test can be found in

`pal_walking_tutorials/src/walk_client_example_topic.cpp`.

39.2.1 Setup

1. Open a terminal connected to TALOS, and launch the `walking_client_topic` application as follows

   ```
   rosrun pal_walking_tutorials walk_client_example_topic
   ```

39.3 Walking using a service interface

39.3.1 Setup

Description This acceptance test will show how to send a parametrized list of steps to TALOS . The parameters are the step length, step time and number of steps. TALOS will add the appropriate steps to its step list and execute them. The test will add 5 steps forward and 5 steps backwards.

Source code The source code for this test can be found in `pal_walking_tutorials/src/walk_client_example_service.cpp`.

39.3.2 Setup

1. Open a terminal connected to TALOS, and launch the `walking_client_service` application as follows

```
rosrun pal_walking_tutorials walk_client_example_service
```

39.4 Walking using an action interface

Description This acceptance test will show how to send commands to TALOS specifying directly the foot steps. TALOS will place its feet on the corresponding desired feet positions. This action server should be used with caution since is up to the user to specify a valid sequence of steps. This action service is meant as an interface to a step planner that outputs a valid step sequence given a goal. The action server has feedback on the status of the sent command. This test will generate a step list that resembles the previous test, it commands five steps forward and reports feedback on the steps that have been executed.

Source code The source code for this test can be found in `pal_walking_tutorials/src/walk_client_example_action.cpp`.

39.4.1 Setup

1. Open a terminal connected to TALOS, and launch the `walking_client_action` application as follows

```
rosrun pal_walking_tutorials walk_client_example_action
```

39.5 End test

Press `Ctrl-C` to close all terminals used in this test.

40 Whole Body Control

40.1 Overview

The Whole Body Control (WBC) is PAL's implementation of the Stack of Tasks[5]. It includes a hierarchical quadratic solver, running at 100 Hz, able to accomplish different tasks with different priorities assigned to

[5]N. Mansard, O. Stasse, P. Evrard, A. Kheddar. A versatile generalized inverted kinematics implementation for collaborative working humanoid robot: the stack of tasks. ICAR 2009.

each. In order to accomplish the tasks, the WBC takes control of all TALOS 's upper-body joints, i.e. torso, arm and head.

In TALOS 's WBC package, the following Stack of Tasks have been predefined (from highest to lowest priority):

- Joint limit avoidance: to ensure joint limits are never reached.
- Feet reference: to fix the robot to the floor. This task is necessary for all non fixed base robots.
- Stable COM: to fix the center of mass of the robot between both feet to mantain stability.
- Self-collision avoidance: to prevent the arms from colliding with any other part of the robot while moving.
- Base orientation: to try to maintain the base link straight.
- Torso orientation: to try to maintain the torso straight.
- Torso height: to try to maintain the torso at a certain high.
- Default joint reference: a default reference for the set of controlled joints.

These tasks are automatic managed by the WBC. The first fourth tasks should be always active with the highest priority and not using them may be potentially dangerous for the robot because it could damage himself.

Then the user may push new tasks to send both end-effectors to any spatial configuration and make the robot look to any spatial point. The difference between using WBC and other inverse kinematic solvers is that the WBC finds online solutions, automatically preventing self-collisions ensuring joint limit avoidance and stability.

40.2 WBC documentation

The different tasks could be pushed and removed online, or either new stacks with different configuration could be created.

In order to know more about how to create an stack, the list of diferents tasks or how to start the controller please go to the documentation of the following packages:

- pal_wbc_controller: contains a basic description of WBC
- wbc_tasks: contains the list of the developed tasks, and shows the user how to configure a task once the controller has already started.
- pal_wbc_utils: describes how the tasks can be pushed or removed from the stack. Also describes a set of utilities included in this package that allows to push the most common tasks.
- pal_wbc_tutorials: is a tutorial about how to create new WBC tasks.
- talos_wbc: describes how to load and launch the Kinematic WBC, and how to create a new stack.

Kinematic WBC Uses position control to execute the tasks. The tasks error are defined in terms of position, velocity or acceleration.

Dynamic WBC Uses torque control to execute the tasks. The tasks error are defined in terms of forces and accelerations. Read the torque control WBC and locomotion section (42) to know more about it.

41 Local Sine Sweep Effort Controller

Description This controller allows the debugging of the torque controllers of the robot, and also serves as an example on how to use this interface. The controller allows to send a chirp torque signal to the motor specfying a desired starting frequency, end frequency, amplitude and duration.

41.1 Launching the controller

To launch the controller, configured for example for the joint: leg_right_3_joint, the command would be as follows:

```
roslaunch local_sine_sweep_torque_controller sine_sweep_controller.launch \
local_joint_control:=inertia_shaping_effort_analytic_dob_control \
local_joint_control_pkg:=inertia_shaping_effort_control  \
joint_name:=leg_right_3_joint \
controller_pkg_name:=talos_controller_configuration
```

In a terminal pointing to the robot, you can start the chirp signal though the following command

```
rosrun actionlib axclient.py /sine_sweep_controller/chirp
```

All the joints which have a torque sensor support the inertia_shaping_effort_analytic_dob_control, for the exception of the joints of the robot arms, which support the inertia_shaping_effort_simple_dob_control.

BE VERY CAREFUL NOT TO TURN THE inertia_shaping_effort_analytic_dob_control CONTROLLER TYPE FOR THE JOINTS OF THE ARM.

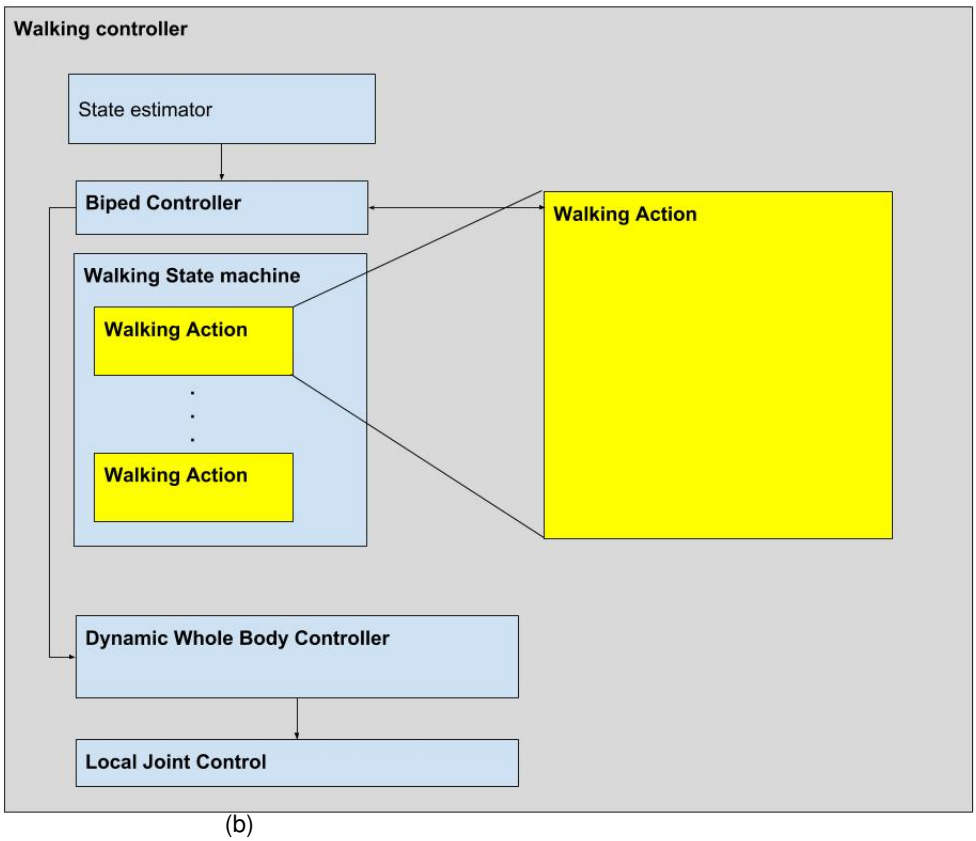
41.2 Differential transmission

There is an especial case for the torso joints, which include a differential transmission, in this case the following command must be used:

```
roslaunch local_sine_sweep_torque_controller  \
sine_differential_sweep_controller.launch \
local_joint_control:=inertia_shaping_differential_effort_analytic_dob_control \
local_joint_control_pkg:=inertia_shaping_effort_control \
joint_name_1:=torso_1_joint joint_name_2:=torso_2_joint \
 controller_pkg_name:=talos_controller_configuration  \
transmission_name:="talos_torso_trans"
```

42 Torque Control WBC and Locomotion

Description: The TALOS locomotion controller is the evolution and unification of previous developed components at PAL. It is meant to be used in a completely torque controlled robot, the following figure shows the architecture.



(b)

42.1 Computation flow

The computation flow is as follows:

1. The robot updates the state of its actuators and floating base state.
2. The state machine computes operational state desired quantities, such as COM acceleration, torso orientation, etc...
3. The operational targets are converted fed to an inverse dynamics wbc controller.
4. The desired torques and command to local joint torque controllers.

42.2 Launching the controller in simulation

How to launch the torque controlled WBC and locomotion controller in simulation:

```
roslaunch talos_pal_locomotion talos_dcm_walking_controller.launch \
estimator:=kinematic_estimator_params
```

42.3 Launching the controller in the real robot

How to launch the torque controlled WBC and locomotion controller in the real robot:

1. Check in the web commander that all the hardware is working properly.

2. Launch the default controllers, which will switch all the robot to position control as configure the starting walking pose: `roslaunch talos_controller_configuration default_controllers.launch`
3. Kill the default controllers `Ctrl-C`
4. Make sure the robot's crane has enough slack.
5. Always having the emergency button ready, launch the following command:

```
roslaunch talos_pal_locomotion talos_dcm_walking_controller.launch \
estimator:=kinematic_estimator_params \
local_joint_control_type:=inertia_shaping_effort_analytic_dob_control \
local_joint_control_pkg:=inertia_shaping_effort_control simulation:=false
```

42.4 Available actions

The current actions are:

42.4.1 Balance DS Action

This is a simple balancing action that tries to regulate the ICP of the robot between to be in the middle point of the two leg ankles.

42.4.2 WBCAction

This action pushes a complete kinematic whole body controller into the state machine. The same functionality of kinematic controller is achieved in the dynamic wbc controller. The only thing that changes is the namespace of the wbc push/pop/status actions.

THIS ACTION IS NOT CURRENTLY SUPPORTED IN THE REAL ROBOT.

```
rosrun pal_locomotion StartWBCAction
```

42.4.3 DCM Walking action

This action implements the locomotion action, it allows to command the robot either using velocity commands or by specifying footsteps.

THIS ACTION IS NOT CURRENTLY SUPPORTED IN THE REAL ROBOT.

```
rosrun pal_locomotion_dcm_planner start_walking_continious_dcm \
-a false -t false -p Cubic -s Zero
```

42.5 Applications

The different apllications working on torque control are:

42.5.1 WBC grasp demo

This demo allows the robot to approach to a table and grasp an object from the table or place it on the table.

The robot detects one aruco above the table, and approaches by performing forward and lateral steps. Then it detects the aruco on the table to pick or place the object.

It uses the DCM Walking action to walk to the table and the WBCAction to grasp and/or place the object.

Moreover this package, the wbc_grasp_demo, is also a tutorial on how to use smach_c, whole body control and generate a list of steps for the robot. Furthermore the whole demo may be easily adapted to another conditions by modifying the set of parameters.

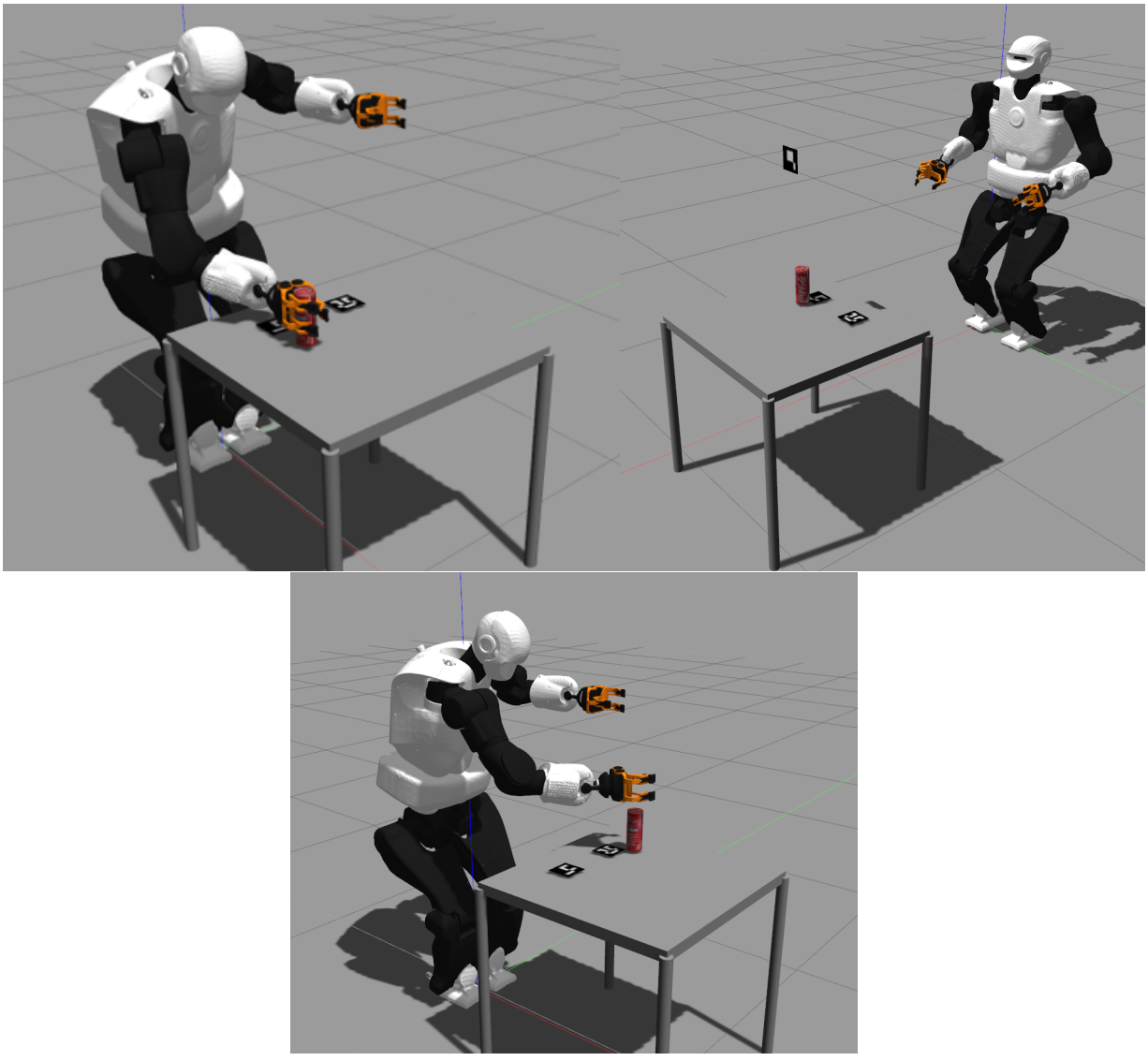


Figure 74: WBC grasp demo

42.5.2 Interactive Markers Control

Once the WBC has started, one of the options is to control the robot by using interactive markers from RVIZ.

Open a terminal in the development computer.

THIS ACTION IS NOT CURRENTLY SUPPORTED IN THE REAL ROBOT.

```
export ROS_MASTER_URI=http://talos-1c:11311
roslaunch talos_wbc interactive_markers.launch \
ns:=/biped_walking_dcm_controller/wbc
rviz
```

In RVIZ will appear three interactive markers that control:

- The gripper left base link
- The gripper right base link
- The gaze position

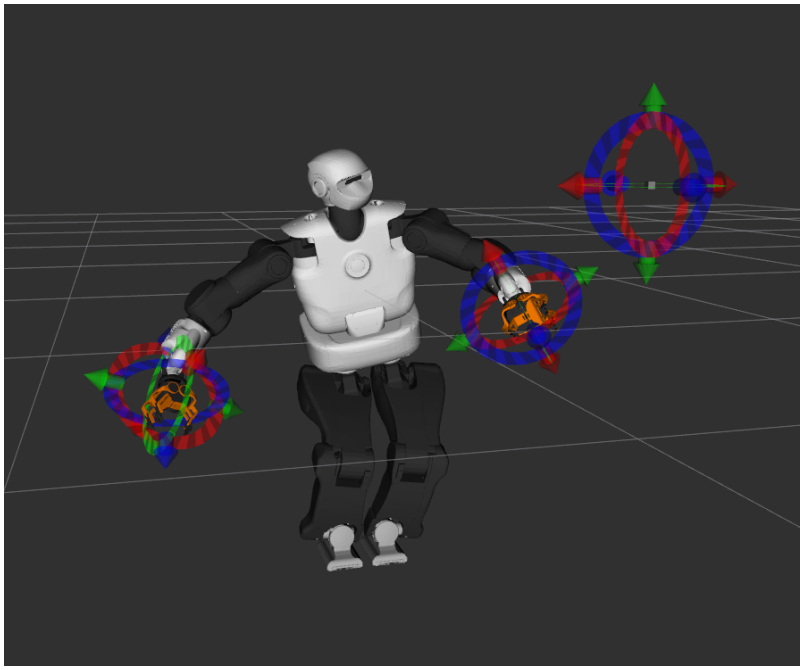


Figure 75: TALOS controlled by interactive markers launch

42.5.3 Talos navigation

The talos navigation is the humanoid navigation framework developed at PAL.

A list of footsteps is generated from the current position of the robot to the specified one. Following the path, the robot is capable to react when it detects an obstacle or when it deviates from his path, by replanning at each step execution.

This framework uses move base costmap to specify the surrounding environment. Thse costmaps are updated online by the depth camera frame mounted in the robot head. It also adds a new plugin in RVIZ which allows to specify goals in the map by using interactive markers.

The DCM Walking action receives the list of footsteps and executes it. The continous replanning plus the torque control allows the robot to reach the goal specified even when applying external disturbances.

THIS APPLICATION IS STILL UNDER DEVELOPMENT

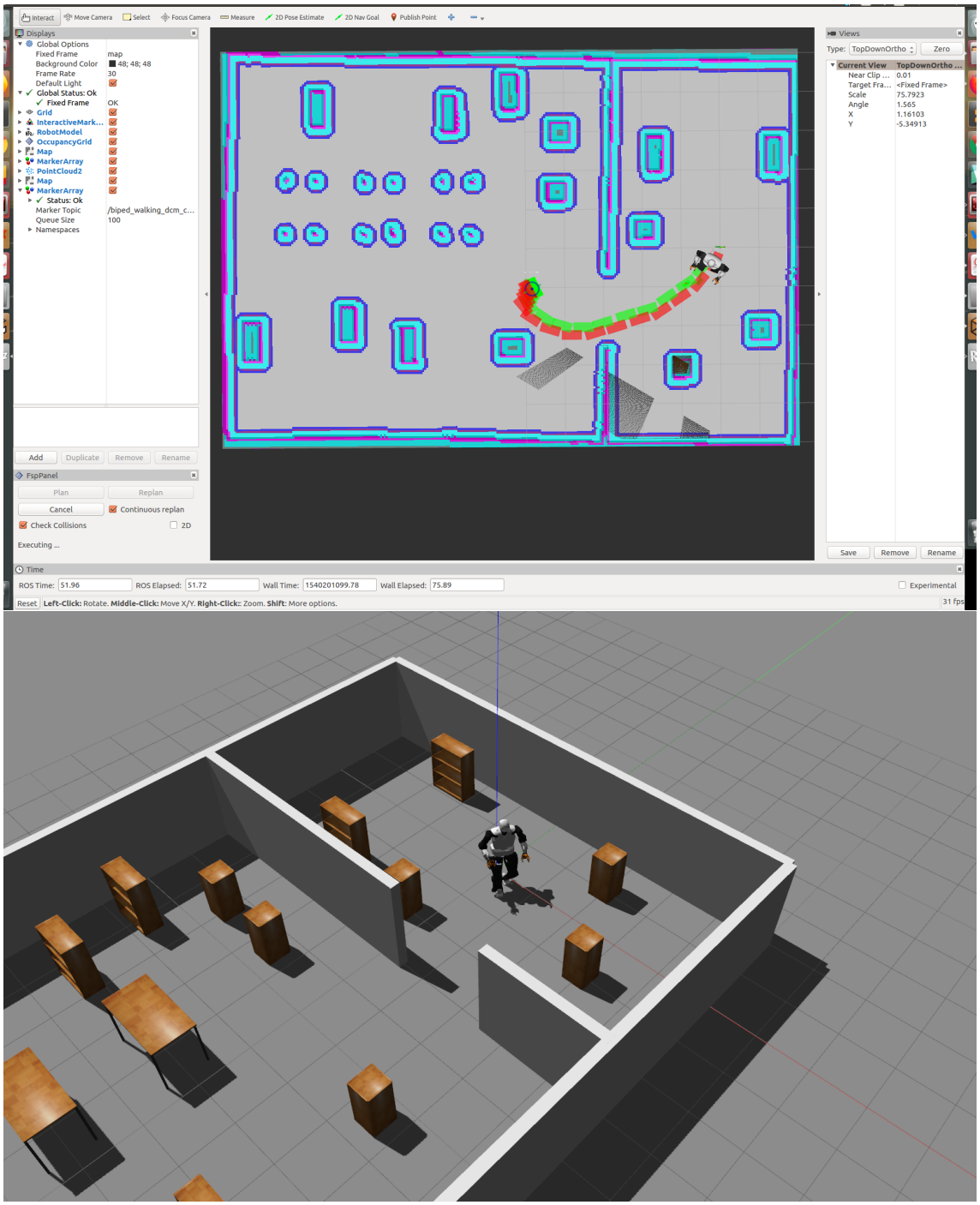


Figure 76: Talos navigation

43 Applications

43.1 Rubber Stamping Demo

The rubber stamping demo is an example of how to perform bi-manual concurrent manipulation using WBC.

The goal of the demo, is to stamp post cards on top of a table. First puting the stamp into an ink container, and then stamping a card or paper.

The documentation of the demo is included with the package rubber_stamping_demo in the repository or on the doc directory of the installed debian package.

Change controllers

44 Change controllers

In order to change robot controllers, as for example to start whole body kinematic controller or gravity compensation controller, and stop position controllers there are two ways to do it.

1- Use the rosservice API with the controller manager services

2- Use the change controllers action

44.1 Controller manager services

There are three main services.

List controllers It lists all the loaded controllers, and shows wich of them are active (running) and which of them are unactive (stopped).

```
ssh pal@talos-1c
rosservice call /controller_manager/list_controllers
```

Also lists the resources used for every controller.

It is important to remark that ROS control doesn't allow two active controllers using a common resorce.

Load controllers It loads a controller. The parameters for the specific controller must be loaded previously on the param server.

```
ssh pal@talos-1c
rosservice call /controller_manager/load_controller "name:
 'controller_name'"
```

It returns true if the controller has been loaded correctly, and false otherwise.

Switch controllers Starts and/or stops a set of controllers. In order to stop a controller this should be active. A controller must be loaded before start it.

```
ssh pal@talos-1c
rosservice call /controller_manager/switch_controller "start_controllers:
- 'whole_body_kinematic_controller'
stop_controllers:
- 'head_controller'
- 'arm_controller'
- 'torso_controller'
strictness: 0"
```

It is recommended to start and stop the desired controllers in one service call.

In the case of the gravity_compensation_controller this is crucial, because once the controller is stopped, the current applied on the arm is zero so it falls down.

The service returns true if the switch has been successfully executed, false otherwise.

Unload controllers It unloads a controller. The controller must be stopped before being unload.

```
ssh pal@talos-1c
rosservice call /controller_manager/unload_controller "name:
 'controller_name'"
```

It returns true if the controller has been unloaded correctly, and false otherwise.

Once a controller has been unloaded, in order to restart it, it will be necessary to load it again.

44.2 Change controllers action

The change controllers action uses the rosservice API of the controller manager, but allows the change of controllers in a simple and intuitive way in a single call. The user doesn't need to know which controllers are active and which resources are being used at any time. The action automatically stops the active controllers that share resources with those who want to be loaded. Moreover it has different flags in the goal msg:

- switch_controllers: If true it stops and starts the controller in a single switch service call. Otherwise, first it calls stop and then it calls start. By default this should be always True. Specially when gravity_compensation_controller needs to be stopped.

- load: Load the controller that is gonna be started

- unload: Unload the request stop controllers. This doesn't affect the controllers that are automatically stopped becasue they share resources with the controllers that are gonna be started.

This action is also optimized to avoid issues with PAL controllers, as for example the whole body kinematic controller needs to be unloaded and loaded every time.

```
export ROS_MASTER_URI=http://talos-1c:11311
rosrun actionlib axclient.py /change_controllers
```

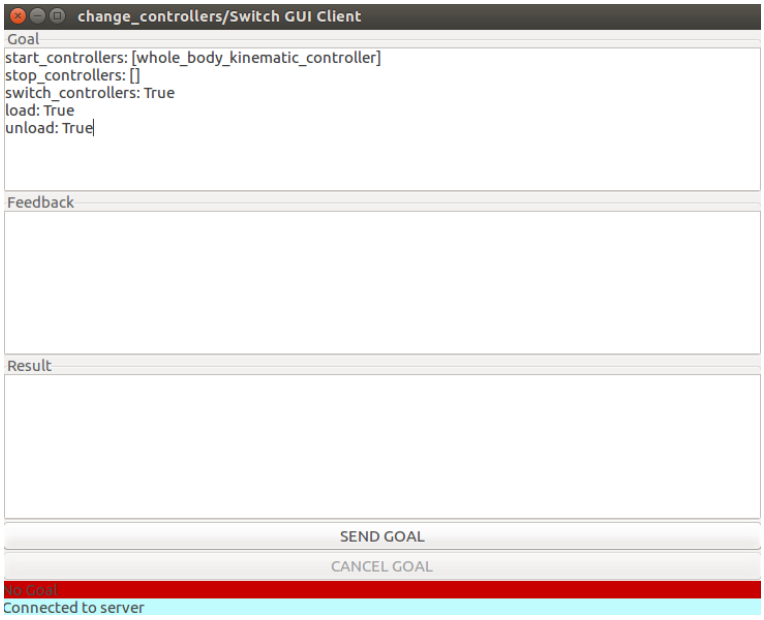


Figure 77: Change controllers action

In the robot the server is automatically launched on the startup. To run it on simulation execute.

```
roslaunch change_controllers change_controllers.launch
```

Introspection controller

45 Introspection controller

The introspection controller is a tool used at PAL to serialize and publish data on the real robot that could be recorded and used later for debugging.

45.1 Start the controller

The introspection controller doesn't use any resource, and it could be activated in parallel with any other controller.

In order to start it run:

```
ssh pal@talos-1c
roslaunch introspection_controller introspection_controller.launch
```

Once the controller is started it will start publishing all the information on the topic: `/introspection_data/full`

45.2 Record and reproduce the data

If you want to record the information from your experiment, it can simply be done using rosbag.

```
ssh pal@talos-1c
rosbag record -O NAME_OF_THE_BAG /introspection_data/full
```

Once you are finished to record your experiment simply close it with `Ctrl-C`

Then copy this file in your development PC

```
ssh pal@talos-1c
scp -C NAME_OF_THE_BAG.bag pal@development:PATH_TO_SAVE_IT
```

Once in your development PC you can reproduce it using PlotJuggler.

```
rosrun plotjuggler PlotJuggler
```

Once PlotJuggler is open load the bag: `File -> Load Data` and select the recorded rosbag.

For more information about PlotJuggler please visit: http://wiki.ros.org/plotjuggler

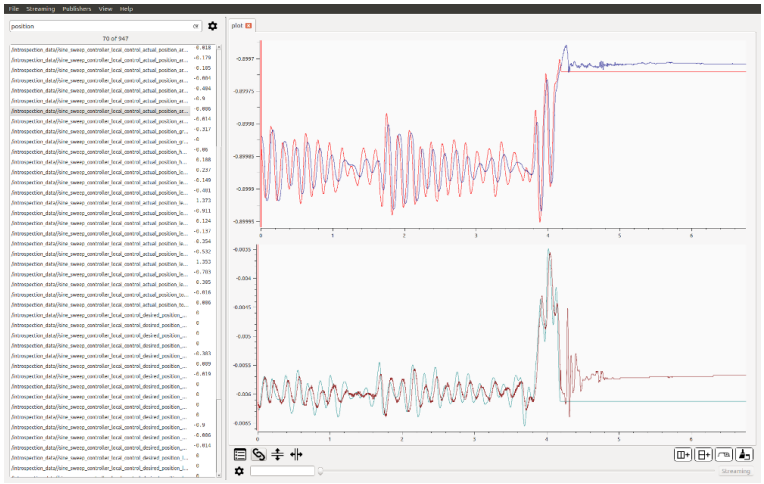


Figure 78: PlotJuggler

45.3 Record new variables

In order to record new variables it will be necessary to register them inside your code as follows.

```
#include < pal_statistics / pal_statistics .h>
#include < pal_statistics /pal_statistics_macros.h>
#include < pal_statistics / registration_utils .h>

 ...

double aux = 0;
 pal_statistics :: RegistrationsRAII registered_variables_;
REGISTER_VARIABLE("/introspection_data", "example_aux", &aux, &registered_variables_);

Eigen::Vector3d vec(0,0,0);
REGISTER_VARIABLE("/introspection_data", "example_vec_x", &vec[0], &registered_variables_);
REGISTER_VARIABLE("/introspection_data", "example_vec_y", &vec[1], &registered_variables_);
REGISTER_VARIABLE("/introspection_data", "example_vec_z", &vec[2], &registered_variables_);
 ...
```

Take in account that the introspection controller only accepts one dimensional variables. For more information please check: https://github.com/pal-robotics/pal_statistics

Customer Service

46 Customer service

46.1 Support portal

All communication between customers and PAL Robotics is made using tickets in a helpdesk software. This web system can be found at http://support.pal-robotics.com. Figure 79 shows the initial page of the site. New accounts will be created on request by PAL Robotics.

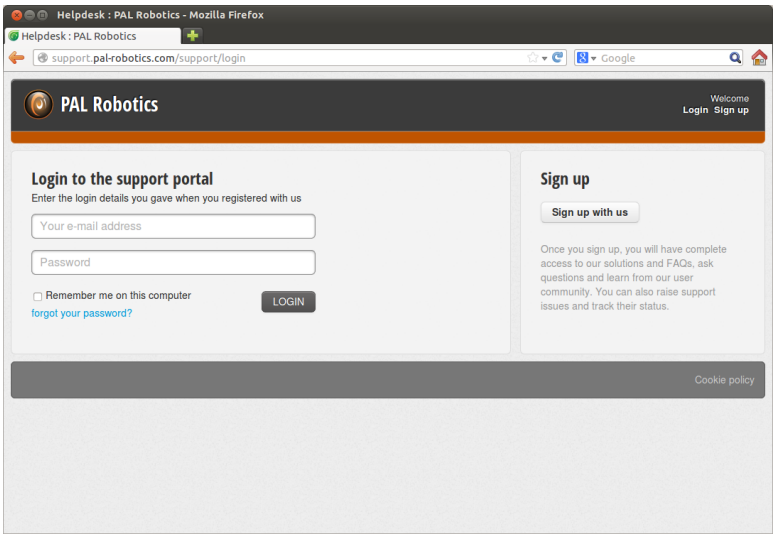


Figure 79: PAL Robotics support website

Once the customer has entered the system (Figure 80), two tabs can be seen: *Solutions* and *Tickets*. The *Solution* section contains *FAQs* and *News* from PAL Robotics . The *Tickets* section contains the history of all tickets the customer has created.

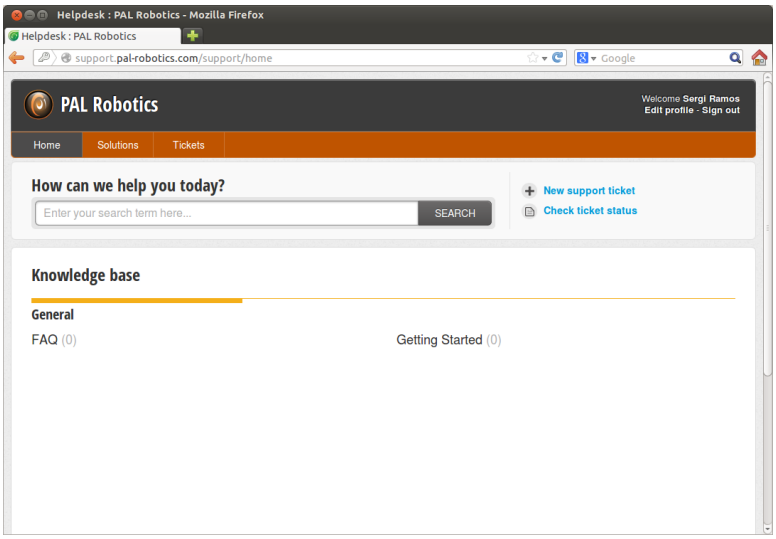


Figure 80: Helpdesk

Figure 81 shows the ticket creation webpage.

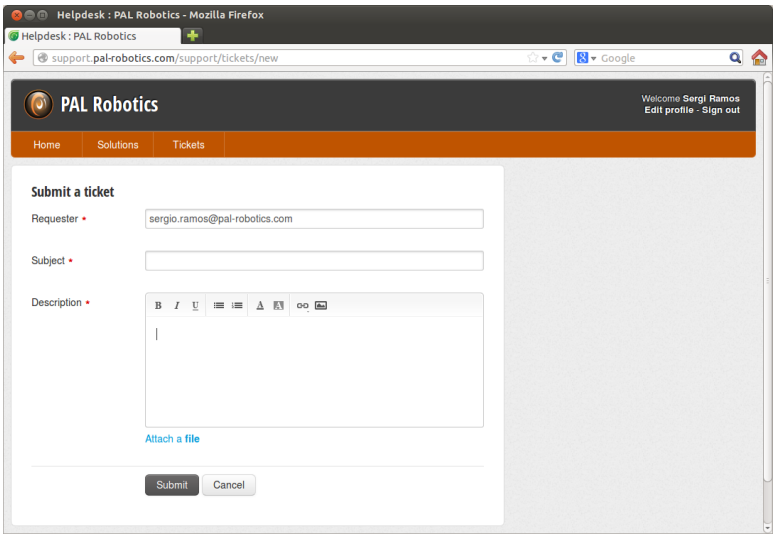


Figure 81: Ticket creation

46.2 Remote support

A technician from PAL Robotics can give remote support. This remote support is disabled by default, so the customer has to activate it manually. If the robot needs to be rebooted, the customer has to activate the remote support after each reboot because it is not persistent.

The robot connects to PAL's remote support system using the command *remoteAssitanceConnect*:

```
root@talos-1c:~# remoteAssistanceConnect ipAddress port
```

Using an issue in the support portal, the PAL technician will provide the IP address and port the customer has to use.

Troubleshooting

47 Troubleshooting

Open the web commander. See section 20. All items in green status mean TALOS is working correctly.

47.1 Common issues

- Low battery: check the *Hardware > Battery* diagnostic. If battery is too low, the diagnostic will be in error state. Shutdown TALOS and leave it charging. Wait until the diagnostic turns green.
- Emergency stop: check the *Hardware > Power* diagnostic. The value of the *Emergency* entry should be **No**. If it is **Yes,** release the emergency stop. The value of the *Emergency* entry should change to **No**. Restart TALOS.
- Motors and FT: check all the *Hardware > Motor* and *Hardware > FT* diagnostics. All must be green. If not, please submit a bug report.
- Controller manager: check the *Functionality > Controller Manager* diagnostic. All controllers must be running. If not, please submit a bug report.

47.2 Bug report

Connect to TALOS and run the following command:

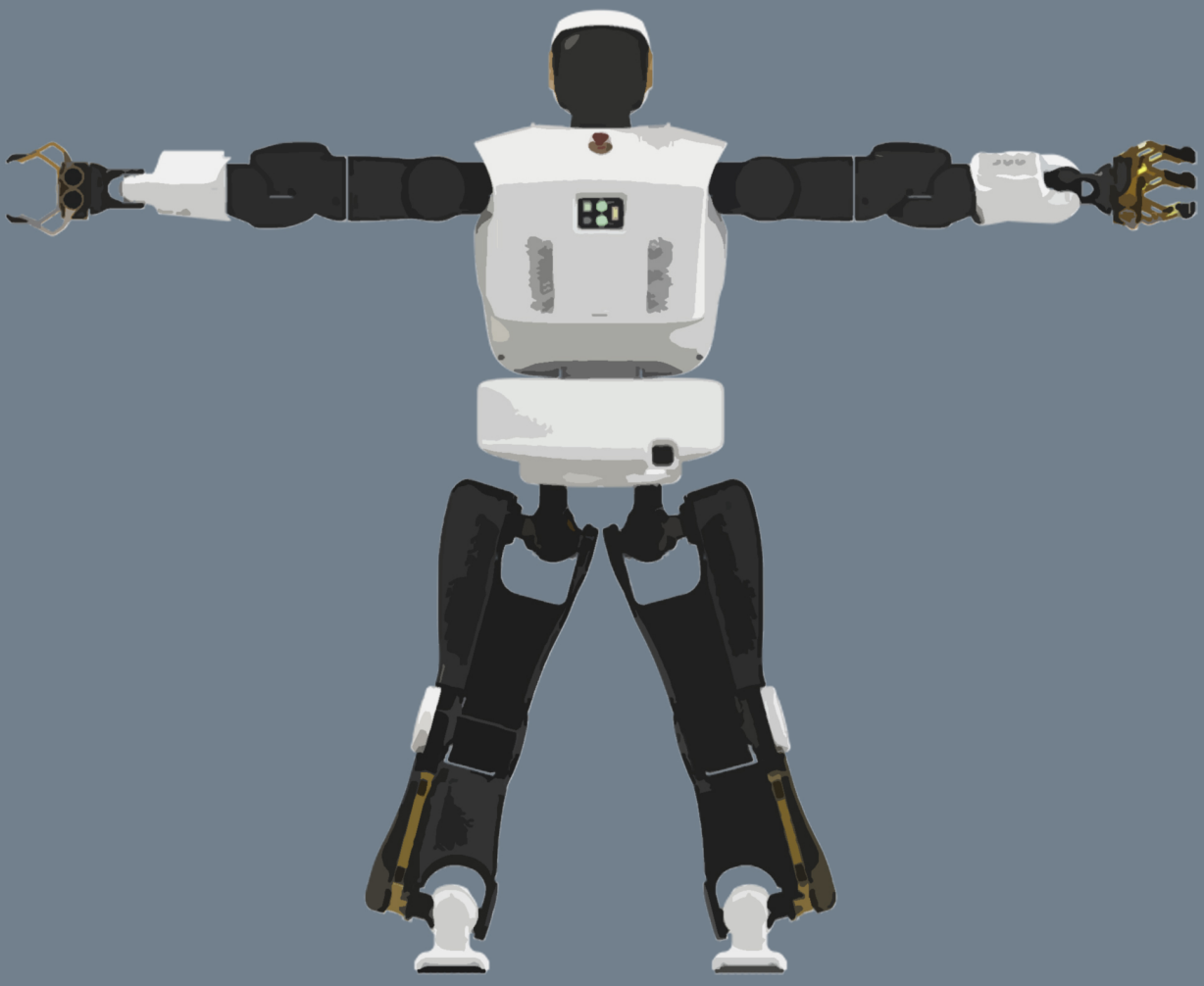
```
rosrun pal_startup_base prepareBugReport.sh
```

A tar file will be generated and placed in */home/pal/bugReports/*. Attach it to the ticket created in the helpdesk. See section 46.

Additionally, execute the following comands and include their outputs in the ticket:

```
ethercat master
ethercat slaves
```

PAL ROBOTICS



PAL ROBOTICS S.L.

Pujades 77-79, 4º 4ª
08005 Barcelona, Spain

Tel.: +34 934 145 347
Fax: +34 932 091 109

info@pal-robotics.com
www.pal-robotics.com