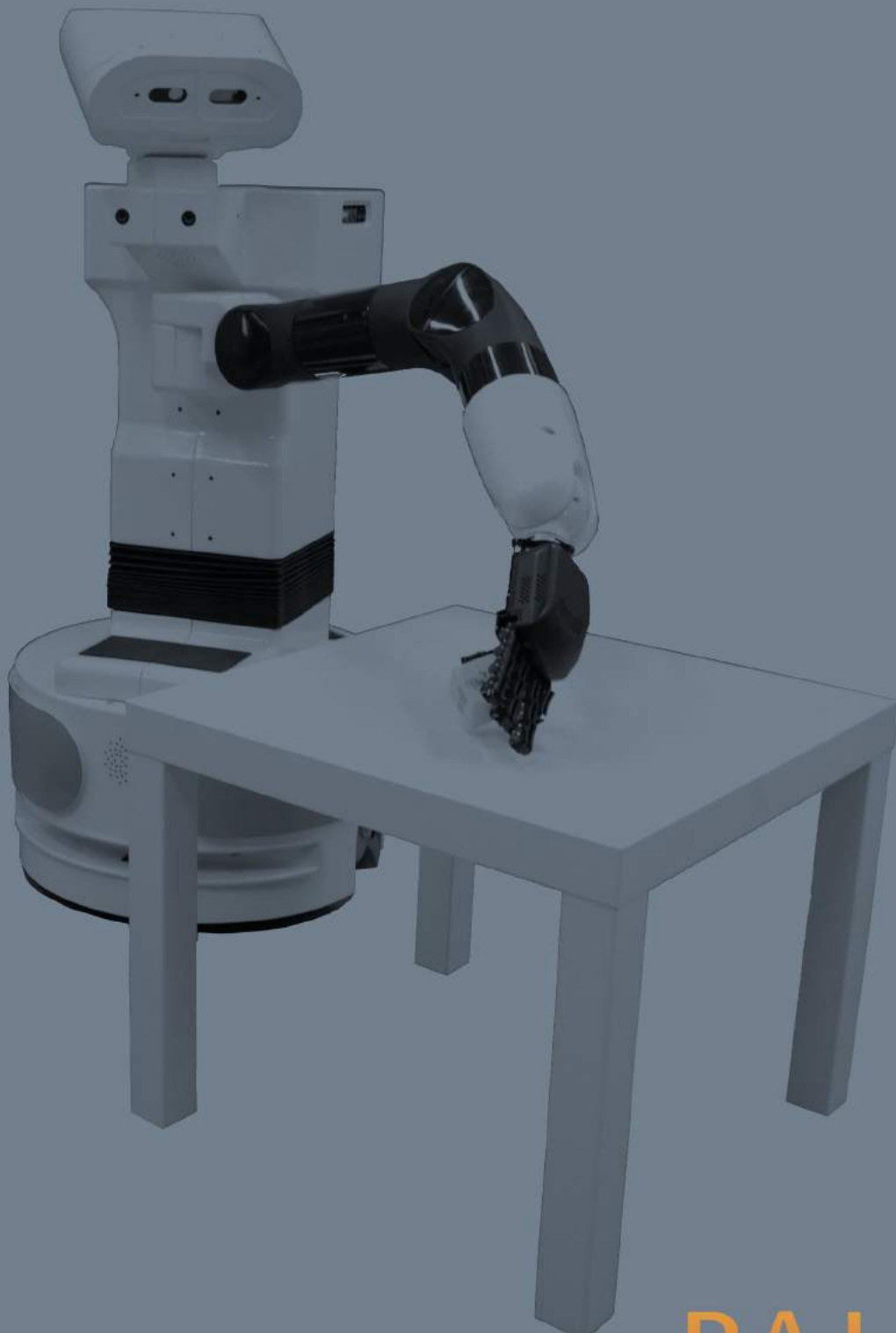


TIA GO

Handbook



PAL
ROBOTICS





Handbook



Barcelona
2016

Contents

Welcome	13
1 Package contents	18
1.1 Overview	18
Specifications	19
2 Specifications	21
2.1 Robot overview	21
2.2 Mobile base	22
2.2.1 Onboard computer	23
2.2.2 Battery	23
2.2.3 Power Connector	23
2.2.4 Laser range-finder	24
2.2.5 Sonars	25
2.2.6 IMU	25
2.2.7 User Panel	25
2.2.8 Service Panel	26
2.2.9 Connectivity	27
2.3 Torso	27
2.3.1 Lifter	27
2.3.2 Expansion Panel	28
2.3.3 Laptop tray	28
2.4 Arm	29
2.5 Force/Torque sensor	30
2.6 End-effector	30
2.6.1 Hey5 hand	30
2.6.2 PAL gripper	31
2.7 Head	32
2.7.1 Pan-tilt mechanism	32
2.7.2 Speaker	32
2.7.3 Stereo microphones	33
2.7.4 RGB-D camera	33
2.8 Robot kinematics	34
2.9 Electrical parts and components	34
Storage	35

3 Storage	37
3.1 Overview	37
3.2 Unpacking	37
3.3 Storage cautions	37
Safety	39
4 Introduction to safety	41
4.1 Overview	41
4.2 Intended applications	41
4.3 Working environment and usage guidelines	41
4.4 Battery manipulation	42
5 Safety measures in practice	43
5.1 Emergency stop	43
5.2 Shutting down the robot properly	43
5.3 How to get the arm out of its initial pose safely	43
5.4 Measures to prevent collisions	44
5.4.1 Measure 1	44
5.4.2 Measure 2	44
5.5 Measures to prevent falls	44
5.5.1 Measure 1	45
5.5.2 Measure 2	45
5.5.3 Measure 3	45
5.5.4 Measure 4	45
5.5.5 Measure 5	46
5.6 How to proceed when an arm collision occurs	46
5.7 Fire fighting equipment	47
5.8 Leakage	47
Marking	49
6 Robot Identification	51
Software Recovery	53
7 Software recovery	55
7.1 Overview	55
7.2 Robot computer installation	55
7.3 Development computer installation	57
Robot Computer	61

8 TIAGo Computer	63
8.1 Computer name	63
8.2 Users	63
8.3 File system	63
8.4 Internal DNS	64
8.5 System upgrade	64
8.6 Firmware update	64
Development Computer	65
9 Development Computer	67
9.1 Overview	67
9.2 ROS communication with the robot	67
9.3 System Upgrade	67
Web Commander	69
10 WebCommander	71
10.1 Accessing the WebCommander website	71
10.2 Overview	71
10.3 The Startup Tab	71
10.4 The Startup Extras tab	71
10.5 The Diagnostics Tab	72
10.6 The Control Joints Tab	73
10.7 The Logs Tab	73
10.8 The Video Tab	73
10.9 The Demos Tab	73
10.10The Movements Tab	73
10.11Networking Tab	74
Software architecture	79
11 Software architecture	81
11.1 Overview	81
11.2 Operating system layer	81
11.3 ROS layer	81
11.4 Software startup process	82
Deployment	83

12 Deploy software on the robot	85
12.1 Introduction	85
12.2 Usage	85
12.3 Deploy notes	85
12.4 Deploy tips	86
12.5 Example of deployment	86
12.5.1 Create the ROS package	86
Sensors	89
13 Sensors	91
13.1 Description of sensors	91
14 Sensors ROS API	91
14.1 Laser range-finder	91
14.1.1 Topics published	91
14.2 Sonars	92
14.2.1 Topics published	92
14.3 Inertial Measurement Unit	92
14.3.1 Topics published	92
14.4 RGB-D camera	92
14.4.1 Topics published	92
14.4.2 Services advertised	92
14.5 Force/Torque sensor	93
14.5.1 Topics published	93
15 Sensors visualization	93
Power status	95
16 Power status	97
16.1 ROS API	97
16.2 Description	97
Voice synthesis	99
17 Voice synthesis	101
17.1 ROS API	101
17.1.1 Action interface	101
17.2 Examples of usage	101
17.2.1 WebCommander	101
17.2.2 Command line	101
17.2.3 Action client	102

Base motions	105
18 Base motions	107
18.1 Overview	107
18.2 Base motion joystick triggers	107
18.2.1 Forward/backward motion	107
18.2.2 Rotational motion	108
18.2.3 Changing the speed of the base	108
18.3 Mobile base control ROS API	108
18.4 Mobile base control diagram	109
Torso motions	111
19 Torso motions	113
19.1 Overview	113
19.2 Torso motion joystick triggers	113
19.3 Torso control ROS API	113
19.3.1 Topic interfaces	113
19.3.2 Action interfaces	113
Head motions	115
20 Head motions	117
20.1 Overview	117
20.2 Head motion joystick triggers	117
20.3 Head motions with rqt GUI	117
20.4 Head control ROS API	118
20.4.1 Topic interfaces	118
20.4.2 Action interfaces	118
Arm motions	119
21 Arm motions	121
21.1 Overview	121
21.2 Arm motions with rqt GUI	121
21.3 Arm control ROS API	121
21.3.1 Topic interfaces	121
21.3.2 Action interfaces	122
Hand motions	123

22 Hand motions	125
22.1 Overview	125
22.2 Hand motion joystick triggers	125
22.3 Hand motions with rqt GUI	125
22.4 Hand control ROS API	125
22.4.1 Topic interfaces	125
22.4.2 Action interfaces	126
Gripper motions	127
23 Gripper motions	129
23.1 Overview	129
23.2 Gripper motion joystick triggers	129
23.3 Gripper motions with rqt GUI	129
23.4 Gripper control ROS API	129
23.4.1 Topic interfaces	129
23.4.2 Action interfaces	130
23.4.3 Service interfaces	130
Upper body motions engine	133
24 Upper body motions engine	135
24.1 Motions library	135
24.2 Motions specification	136
24.3 Predefined motions safety	137
24.4 ROS interface	137
24.4.1 Action interface	137
24.5 Action clients	137
Navigation	139
25 Navigation	141
25.1 Overview	141
25.2 Navigation architecture	141
25.3 Navigation ROS API	141
25.3.1 Topic interfaces	141
25.3.2 Action interfaces	142
25.3.3 Service interface	142
25.4 SLAM and path planning in simulation	143
25.4.1 Mapping	143
25.4.2 Saving the map	144

25.4.3	Localization and Path Planning	145
25.5	SLAM and path planning on the robot	147
25.5.1	Saving the map on the robot	147
25.5.2	Localization and Path Planning	147
25.5.3	Change the active map on the robot	148
25.6	Map Editor	148
Motion planning		151
26 Motion planning with <i>Movel!</i>		153
26.1	Overview	153
26.2	Getting started with the <i>Movel!</i> graphical user interface	153
26.3	The planning environment of <i>Movel!</i>	154
26.4	End test	156
Whole Body Control		157
27 Whole Body Control		159
27.1	Overview	159
27.2	WBC upper-body teleoperation with leap motion	159
27.3	WBC upper-body teleoperation with joystick	160
27.4	WBC with Rviz interactive markers	162
27.5	WBC through ROS topics interface	164
Simulation		167
28 Simulation		169
28.1	Overview	169
28.1.1	Empty world	169
28.1.2	Office world	169
28.1.3	Table with objects world	170
28.2	Grasping demo in simulation	170
LEDs		171
29 LEDs		173
29.1	ROS API	173
29.1.1	Available services	173
Tutorials		175

30 Tutorials	177
30.1 Overview	177
30.2 Download source code	177
30.2.1 Look_hand	177
30.2.2 Look_to_point	177
30.2.3 Play_with_sensors	177
30.2.4 Run_motion	178
30.2.5 Say_something	178
Demos	179
31 Demos	181
31.1 Learning-by-demonstration	181
31.2 Download source code and build demo	181
31.3 Run the demo	181
31.3.1 Select joints	181
31.3.2 Start learning	182
31.3.3 Moving the joints	182
31.3.4 Stop the learning	183
31.3.5 Testing the new motion	183
Customer Service	185
32 Customer service	187
32.1 Support portal	187
32.2 Remote support	188



Welcome



Welcome

Thank you for choosing PAL Robotics. This Handbook contains all the information related to the TIAGo 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. Reproduction, publication, or duplication of this manual, or any part of it, in any manner, physical, electronic, or photographic, is prohibited without the explicit written permission of PAL Robotics.

Disclaimers

General Disclaimer

TIAGo 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 TIAGo 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 TIAGo 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 TIAGo 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 TIAGo 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 to you (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:

- as 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, assignees 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 TIAGo . Make sure you can find all of them:



(a) TIAGo



(b) Battery charger



(c) Electric keys



(d) Installation pendrives



(e) Joystick



(f) Leap motion
(included in the Whole Body Control
premium package)

Figure 1: Components inside transportation box

TIA Go

Specifications



2 Specifications

2.1 Robot overview

TIAGo main parts are depicted in figure 2 and its main specifications are summarized in table 1.

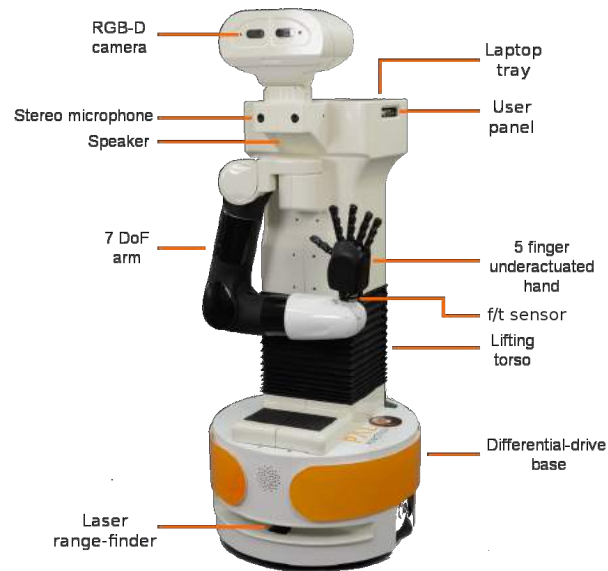


Figure 2: TIAGo main components

Dimensions	Height	110 – 145 cm
	Weight	72 Kg
Degrees of freedom	Base footprint	Ø 54 cm
	Mobile base	2
	Torso lift	1
	Arm	4
	Wrist	3
	Head	2
	Hey5 hand	19 (3 actuated)
	PAL gripper	2
	Drive system	Differential
	Max speed	1 m/s
Mobile base	Lift stroke	35 cm
Torso	Payload	2 Kg
Arm	Reach	87 cm
Electrical features	Battery	36 V, 20 Ah
Sensors	Base	Laser range-finder
		Sonars
		IMU
	Torso	Stereo microphones
	Arm	Motors current feedback
	Wrist	Force/Torque
	Head	RGB-D camera

Table 1: Robot main specifications

2.2 Mobile base

The mobile base of TIAGo is provided of a differential drive mechanism and contains an onboard computer, batteries, power connector, a laser-range finder, three rear sonars, a user panel, a service panel and two Wifi to ensure wireless connectivity.

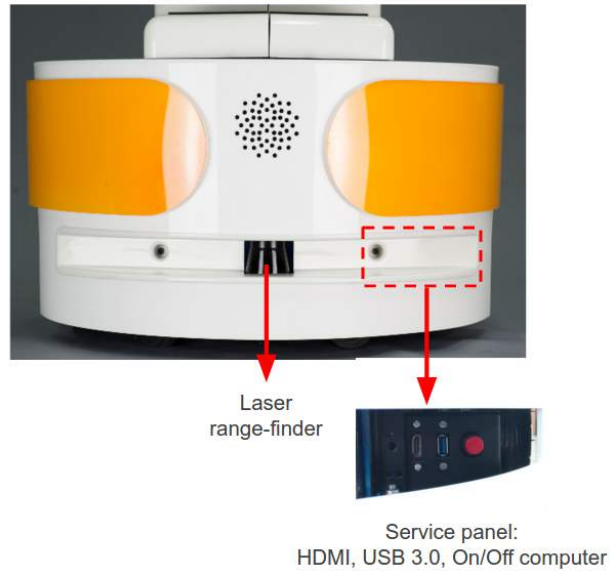


Figure 3: Mobile base frontal view



Figure 4: Mobile base rear view

2.2.1 Onboard computer

The specifications of TIAGo onboard computer depends on the configuration options you may have ordered. The different possibilities are shown in table 2.

Component	Description
CPU	Intel i5 / i7 Haswell
RAM	4 / 8 / 16 GB
Hard disk	60 / 120 / 240 GB SSD
Wi-Fi	802.11 a/b/g/n/ac
Bluetooth	Smart 4.0 Smart Ready

Table 2: Onboard computer main specifications

2.2.2 Battery

The specifications of the battery supplied with TIAGo are shown in table 3.

Type	Li-Ion
V _{nominal}	36.0 V
V _{max}	42.0 V
V _{cutoff}	30.0 V
Nominal capacity	20 Ah
Nominal energy	720 Wh
Max. continuous discharge current	20 A
Pulse discharge current	60 A
Max. charging current	15 A
Charging method	CC/CV
Weight	7.5 kg

Table 3: Battery specifications

TIAGo can be equipped with 2 batteries. In this case the *total Nominal capacity is 1440 Wh*.

2.2.3 Power Connector

TIAGo must be charged only with the supplied charger!! To insert the charger connector, open the lid located on the rear part as shown in figure 5.



Figure 5: Charging connector entry

Connection Insert charging connector with metal lock facing up, push it, as shown in figure 6 until you hear a 'click'.



Figure 6: Charger connector insertion procedure

Disconnection Once charge is completed, the connector can be removed. In order to do so, press the metal lock and pull firmly the connector, see figure 7.



Figure 7: Charger connector removal procedure

2.2.4 Laser range-finder

The specifications of the laser on the frontal part of the mobile base are shown in 4.

Manufacturer	SICK
Model	TIM561-2050101
Range	0.05 - 10 m
Frequency	15 Hz
Field of view	180°
Step angle:	0.33°

Table 4: Laser range-finder specifications

2.2.5 Sonars

The rear part of the mobile base has 3 ultrasound sensors, here referred as sonars. One is centered and the other two are placed at 30° at its left and its right. See table 5 for the specifications of the sonars.

Manufacturer	Devantech
Model	SFR05
Frequency	40 kHz
Measure distance	0.03 - 1 m

Table 5: Sonars specifications

2.2.6 IMU

The Inertial Measurement Unit is mounted at the center of the mobile base and may be used to monitor inertial forces and it also provides the attitude. The specifications are presented in table 6.

Manufacturer	InvenSense
Model	MPU-6050
Gyroscope	3-axis
Accelerometer	3-axis

Table 6: IMU main specifications

2.2.7 User Panel

On top of the rear part of TIAGo mobile base there is the user panel. It provides the buttons to power up and shutdown the robot and visual feedback about the status of the robot. All the specific elements of the user panel are shown in figure 8 and the description of each element is presented in table 7.



Figure 8: User panel

Number	Name / Short description
1	Emergency Stop
2	Information Display
3	On/Off button
4	Electric key

Table 7: User Panel description

Electric key This is the main power control switch. When TIAGo is not going to be used for a long period of time, please, turn off the power by selecting OFF position (figure 9).

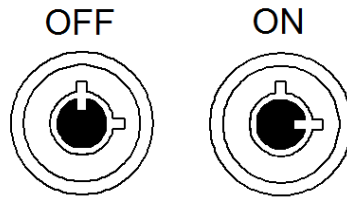


Figure 9: Electric key positions

Information Display 320x240 Color TFT display that shows battery level on the top-right corner. When touched for more than 1 second, internal voltage levels and PC power status will appear on top-left corner for 5 seconds.

On/Off button Standby control button. It is a pushbutton with a green light to indicate the current system status.

Light	State	Name / Short description
Off	Fixed	Standby
On	Fixed	Running
On	Slow-Blink	System in process of shutdown
On	Fast-Blink	Emergency state

Table 8: Green light indicator possible modes

After main power is connected, i.e. electric key is ON, see figure 9, user must press this button during 1 second in order to start the TIAGo.

To set again the system in standby mode when is running, press again the button. The green light will blink slowly during shutdown procedure and light-off when standby mode reached.

Emergency Stop When pushed, motors are stopped and disconnected. Green indicator will blink fast in order to notify the emergency state.

To start the normal behaviour again, a two step validation must be executed: emergency button must be released rotating clockwise, and then On/Off button must be pressed for 1 second. The green light will change to fixed state.

2.2.8 Service Panel

Removing the cover behind the laser it is possible to access the Service Panel, see figure 10.

This service panel gives access to video, usb and on/off button of the computer inside the TIAGo. It can be used for reinstallation or debug propouses.

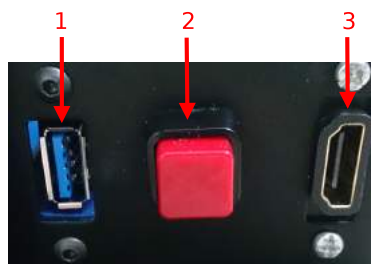


Figure 10: Service panel

Number	Name / Short description
1	USB 3.0
2	On/Off button computer
3	HDMI

Table 9: Service Panel description

2.2.9 Connectivity

TIAGo is equipped with a dual band Wireless 802.11b/g/n/ac interface plus Bluetooth 4.0 and two Wifi antennas. When the Wifi interface is configured as access point it is a 802.11g interface.

There are two Gigabit Ethernet, ports 2 and 3 in figure 13, that can be use to connect to the internal network of the robot. For this network the IP address range *10.68.0.0/24* has been reserved. *The IP addresses used in the building network MUST not use this range because it can interfere with the services of the robot.*

2.3 Torso

The torso of TIAGo is the structure that support the arm and the head of the robot and is equipped with an internal lifter mechanism which allows changing the height of the robot. Furthermore, it provides an expansion panel and a laptop tray.

2.3.1 Lifter

The lifter mechanism is placed underneath the industrial bellows shown in figure 11. The lifter is able to move at 50 mm/s and it has a stroke of 350 mm. The minimum and maximum height of the robot is shwon in figure 12.



Figure 11: Industrial bellows of the lifting torso

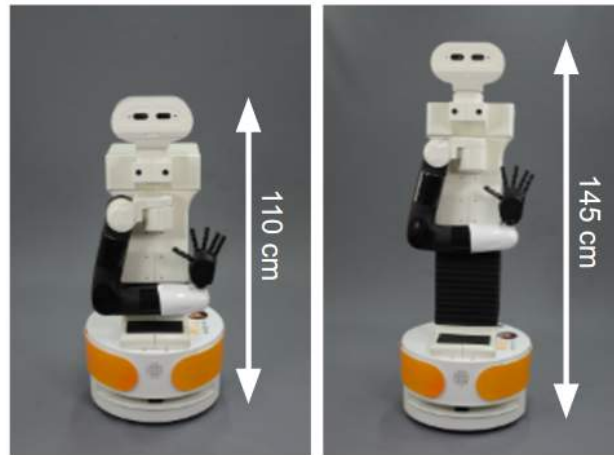


Figure 12: Industrial bellows of the lifting torso

2.3.2 Expansion Panel

The expansion panel is located on the top of the left part of the torso and the connectors exposed are shown in figure 13 and specified in table 10.



Figure 13: Expansion panel

Number	Name / Short description
1	CAN Service connector
2	Mini-Fit Power supply 12 V and 5 A
3	Fuse 5 A
4	GigE port
5	GigE port
6	USB 3.0 port
7	USB 3.0 port

Table 10: Expansion Panel description

The CAN service connector is reserved for maintenance purposes and shall not be used.

2.3.3 Laptop tray

The laptop tray, see figure 14 is the flat surface on top of the torso just behind the head of the robot, see figure 14. It has mounting points to add new equipment, up to 5 Kg can be supported, or it can be used to place a laptop in order to work in place with the robot making use of the Wifi connectivity or using one of the ethernet ports in the expansion panel.

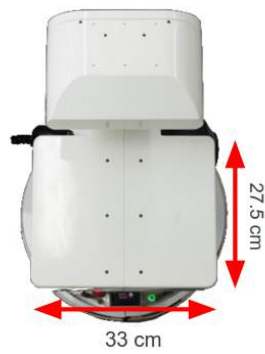


Figure 14: Laptop tray dimensions



Figure 15: Laptop placed on the rear tray of the robot

2.4 Arm

The arm of TIAGo is composed of 4 modules M90 and one 3 DoF wrist M3D as shown in figure 16. The main specifications of the arm and wrist are shown in table 11.

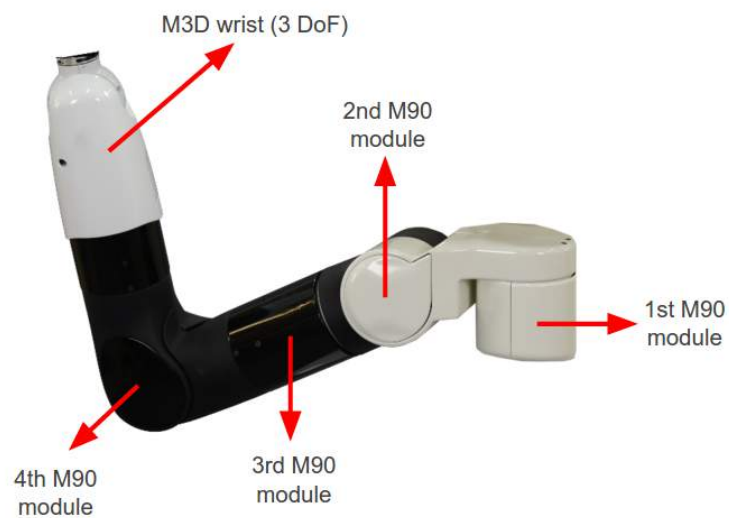


Figure 16: Arm components

Weight	10 Kg					
Payload	2.8 Kg					
Joints	7					
Onboard control modes	Modules	position, velocity and current			Encoders (bits)	
	Wrist	position and velocity				
Actuators	Description	Reduction	Max speed [rpm]	Nominal torque [Nm]	Motor	Absolute
	1st module	100:1	18	39	12	12
	2nd module	100:1	18	39	12	12
	3rd module	100:1	22	22	12	12
	4th module	100:1	22	22	12	12
	Wrist 1st DoF	336:1	17	3	11	12
	Wrist 2nd DoF	336:1	17	5	11	13
	Wrist 3rd DoF	336:1	17	5	11	13

Table 11: Arm and wrist specifications

2.5 Force/Torque sensor

The force/torque sensor integrated on the end-point of the wrist is a ATI mini45, see figure 17. The main specifications of the sensor are summarized in table 12.



Figure 17: Force/torque sensor placement and close view

Physical Specs	Weight	Diameter	Height		
	0.0917 Kg	45 mm	15.7 mm		
	Fx, Fy	Fz	Tx, Ty	Tz	
Sensing ranges	290 N	580 N	10 Nm	10 Nm	
Resolution	1/8 N	1/8 N	1/376 Nm	1/752 Nm	

Table 12: Main specifications of the force/torque sensor

2.6 End-effector

TIAGo can be used with two different inter-changeable end-effectors: the Hey5 hand and the PAL parallel gripper.

2.6.1 Hey5 hand

The Hey5 hand is shown in figure 18. The main specifications of this underactuated self-contained hand are summarized in table 13.

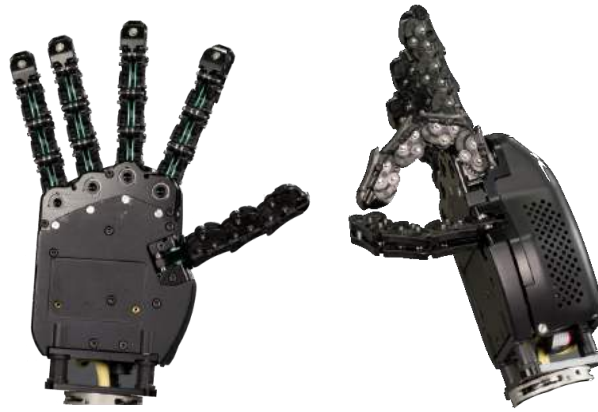


Figure 18: Hey5 hand

Weight	720 g		
Payload	1 Kg		
Joints	19		
Actuators	Description	Max speed [rpm]	Max torque [Nm]
	thumb	32	0.23
	index	32	0.23
	middle+ring+little	34	0.45

Table 13: Hey5 hand main specifications

Credits and attribution

- The **Hey5 hand** has been developed by **PAL Robotics Inc.**, with contributions from **QBrobotics srl**.
- The **Hey5 hand** is a derivative of the **Pisa/IIT SoftHand** open source project by M. G. Catalano, G. Grioli, E. Farnioli, A. Serio, C. Piazza and A. Bicchi.
- The **Pisa/IIT SoftHand** project is distributed under [Creative Commons Attribution 4.0 International License](#) and is available at [NaturalMachineMotionInitiative.com](#).

2.6.2 PAL gripper

The PAL parallel gripper is shown in figure 19. The gripper contains 2 motors, each one controlling one of the fingers. Each finger has a linear range of 4 cm.



Figure 19: PAL gripper

Weight	800 g				
Payload	2 Kg				
interchangeable fingers	Yes				
Actuators	Description	Reduction	Max speed [rpm]	Max torque [Nm]	Absolute encoder
	left finger	193:1	55	2.5	12 bits
	right finger	193:1	55	2.5	12 bits

Table 14: PAL gripper main specifications

2.7 Head

The head of TIAGo is equipped with a pan-tilt mechanism, i.e. 2 DoF, and is equipped with stereo microphones, a speaker and a RGB-D camera. Furthermore, on top of the head there is a flat surface with mounting points to let the user add new sensors or equipment. Note that the head has a payload of 0.5 Kg when adding new equipment on top of it. Figure 20 show the location of each component and the two joints of the pan-tilt mechanism.

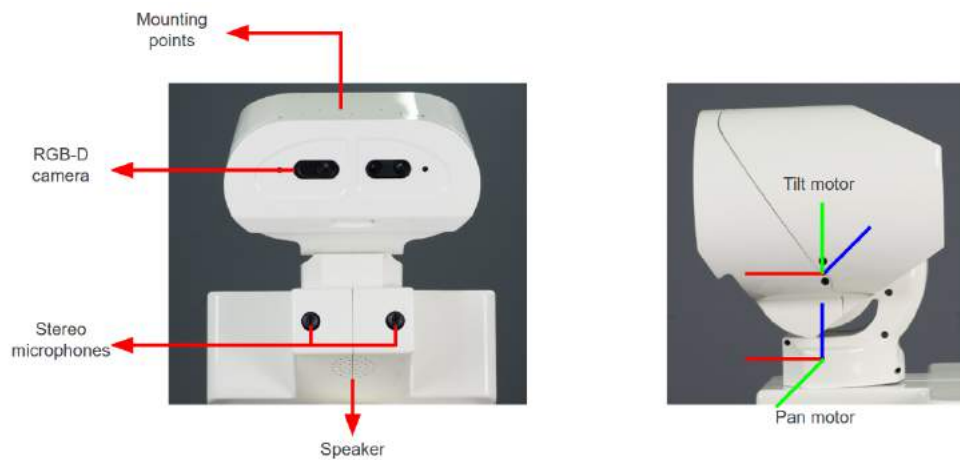


Figure 20: Head parts and components

2.7.1 Pan-tilt mechanism

Actuators	Description	Reduction	Max speed [rpm]	Max torque [Nm]	Absolute encoder
	pan motor	200:1	63	6	12 bits
	tilt motor	200:1	63	6	12 bits

Table 15: Head pan-tilt main specifications

2.7.2 Speaker

The speaker specified in table 16 is provided below the head of the robot.

Manufacturer	VISATON
Model	FRS 5
Rated power	5 W
Max power	8 W
Nominal impedance Z	8 Ohm
Frequency response	150-20000 Hz

Table 16: Speaker main specifications

2.7.3 Stereo microphones

The Andrea SuperBeam Stereo Array Microphone is integrated in TIAGo just below the head of the robot. The specifications of the audio card are presented in table 17 and the specifications of the Stereo Array Microphone are detailed in table 18.

Manufacturer	Andrea Electronics
Model	SuperBeam Stereo Array Microphone
Electrical characteristics	
Mic supply voltage	1.4-5.0 VDC
Supply Bias Resistor	2.2k-39.9k Ohm
Operating Current (each channel)	0.5 mA
Output Impedance at 1 kHz	200 Ohm
Max Input Sould Level at 1 kHz, 3% THD	115 dB
Output Signal Level at THD < 3% @ 1 kHz	24-120 mVrms
Sensitivity at 1 kHz (0 dB = 1 V/Pa Vdc=1.6 V)	-40 to -37 dBV
Frequency Response at 3 dB Variation Noise	20 uVrms
Operating temperature	0-70C° C
Acoustic characteristics	
Recommended Operating Distance	30.5-122 cm
Acoustic Signal Reduction at 1 kHz Outside of 30° Beamform	15-30 dB
Noise reduction	20-25 dB

Table 17: Stereo microphones main specifications

Manufacturer	Andrea Electronics
Model	PureAudio USB-SA
Supply voltage	4.5 - 5.5 VDC
Total power consumption	120 mA
A/D Conversion Resolution	16 bit
THD + N	-84 dB
Supply Bias Resistor	2.2 kOhm @ 3.3 VDC
Frequency response	20-20000 Hz
Input Range	0 - 1.25 Vrms
Dynamic Range	95 dB
Record Gain Range	-6 to 33 dB

Table 18: USB external audio card main specifications

2.7.4 RGB-D camera

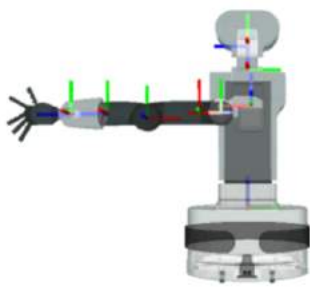
The head of TIAGo includes a RGB-D camera specified in table 19.

Manufacturer	ASUS
Model	Xtion Pro Live
Field of View	58° H, 45° V, 70° D
Interface	USB 2.0
Color stream modes	QVGA 320x240 @ 60 fps, VGA 640x480 @ 30 fps, XGA 1280x720 @ 15 fps
Depth stream modes	QVGA 320x240 @ 60 fps, VGA 640x480 @ 30 fps
Depth sensor range	0.4 - 8 m

Table 19: RGB-D main specifications

2.8 Robot kinematics

Figure 21 presents the kinematics specification of TIAGo upper body including the motion range of each joint.



Joint	Type	Lower limit	Upper limit
torso	prismatic	0 mm	350 mm
head_1	revolute	-75°	75°
head_2	revolute	-60°	45°
arm_1	revolute	0	157.5°
arm_2	revolute	-90°	62.5°
arm_3	revolute	-202.5°	90°
arm_4	revolute	-22.5°	135°
arm_5	revolute	-120°	120°
arm_6	revolute	-90° (*)	90° (*)
arm_7	revolute	-120°	120°

(*) Wrist version with force/torque sensor arm_6 range is [-81°, 81°]

Figure 21: Upper body joints specifications

2.9 Electrical parts and components

Neither TIAGo nor any of its electrical components and mechanical parts are connected to an external ground. The chassis and all electromechanical components of the robot are physically isolated from the environment ground with the isolation rubber under its feet. To avoid any damage in the electromechanical part of TIAGo, don't touch any metal parts directly to avoid discharges.

Electrical power supply and connectors

The power source supplied with TIAGo 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).

TIA Go

Storage



3 Storage

3.1 Overview

Information relating to the storage of TIAGo will be explained in this section.

3.2 Unpacking

3.3 Storage cautions

- Always store TIAGo in a place where it will not be exposed to weather conditions.
- The storage temperature range for TIAGo is between $0^{\circ}\text{C} \sim +60^{\circ}\text{C}$.
- The storage temperature range for the batteries is between $+10^{\circ}\text{C} \sim +35^{\circ}\text{C}$.
- It is recommended to remove the battery from TIAGo when storage period exceeds 2 weeks.
- It is recommended to charge the battery to 50% when storing it for more than 2 weeks.
- Avoid the use or presence of water near TIAGo.
- Avoid any generation of dust close to TIAGo.
- Avoid the use or presence of magnet devices or electromagnetic fields near TIAGo.

TIA Go

Safety



4 Introduction to safety

4.1 Overview

Safety is important when working with TIAGo. This chapter provides an overview of safety issues, general usage guidelines to support safety, and describes some safety-related design features. *Before operating the robot all users must read and understand this chapter!*

4.2 Intended applications

It is important to clarify the intended usage of the robot prior to any kind of operation.

TIAGo 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 TIAGo allow to research and develop activities in the following areas:
 - Navigation and SLAM
 - Manipulation
 - Perception
 - Speech recognition
 - Human-robot interaction

4.3 Working environment and usage guidelines

The working temperatures are:

- Robot: $+10^{\circ}\text{C} \sim +35^{\circ}\text{C}$

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

Maintain a safe environment:

- The terrain for TIAGo usage must be capable of supporting the weight of the robot (see Section Specifications). It must be horizontal and flat. Do not use any carpet, to avoid tripping over.
- Make sure the robot has adequate space for any expected or unexpected operation.
- Make sure the environment is free from objects that could pose a risk if knocked, hit, or otherwise affected by TIAGo .
- Make sure there are no cables or ropes that could be caught in the covers or wheels; these could pull other objects over.
- Make sure no animals are near the robot.
- Be aware of the location of emergency exits and make sure the robot cannot block them.
- Do not operate the robot outdoors.
- Keep TIAGo away from flames and other heat sources.
- Do not allow the robot to come in contact with liquids.
- Avoid dust in the room.
- Avoid the use or presence of magnetic devices near the robot.
- Apply extreme caution with children.

4.4 Battery manipulation

Some guidelines must be respected when handling the robot in order to prevent risks related to the internal batteries of the robot.

- Do not expose the robot to fire.
- Do not expose the robot to water or salt water, or allow the robot to get wet.
- Do not open or modify the robot. Avoid in any case opening the internal battery case.
- Do not expose to ambient temperatures above 49°C for over 24 hours.
- Do not store at temperatures below -5°C for more than seven days.
- For long-term storage (more than 1 month) charge the battery to 50%.
- Do not use the TIAGo's batteries for other purposes.
- Do not use other devices but the supplied charger to recharge the battery.
- Do not drop the batteries.
- If any damage or leakage is observed, stop using the battery.

5 Safety measures in practice

5.1 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 shall be used only in exceptional cases, when the immediate stop of the robot's motors is required.

To activate the emergency stop the user has to push the button. To deactivate the emergency stop, the button has to be rotated clockwise according to the indications on the button until it pops out.

- Be careful using this emergency stop action because the motors will be switched OFF, note that the arm will fall down, while the computer remains on.
 - After releasing the emergency stop button the user has to start the robot by using the power button. After this operation the robot status should be restored to that previous to pressing the emergency button.

5.2 Shutting down the robot properly

Special care needs to be taken when shutting down or powering off the motors by using the emergency button. In order to avoid bumping the arm against the base of the robot the following procedure must be followed as depicted in figure 22:

- a) Execute **Home** predefined motion using, for instance, the Web commander
- b) Hold the wrist of the robot
- c) Press the On/Off button for 1 second
- d) Keep holding the wrist until the robot is totally powered off and you may accompany the wrist to its lying position on top of the mobile base

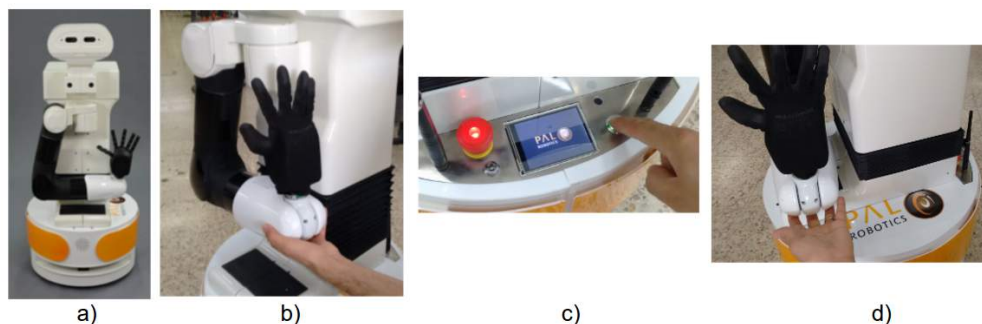


Figure 22: Shutdown procedure

5.3 How to get the arm out of its initial pose safely

When the robot is started the arm will be lying on top of the mobile base. The procedure depicted in figure 23 and hereafter detailed provides a safe way to get the arm of this initial pose preventing self-collisions:

- a) The arm is lying on top of the mobile base. In order to get the arm out of this collision either press the button **Get out of collision** in the **StartStop** tab of the Web commander or run the following command line instruction:

```
export ROS_MASTER_URI=http://tiago-0c:11311
rosservice call /get_out_of_collision
```

- b) Raise the torso to its maximum height with the joystick

c) Execute the **unfold_arm** movement using, for instance, the Web commander

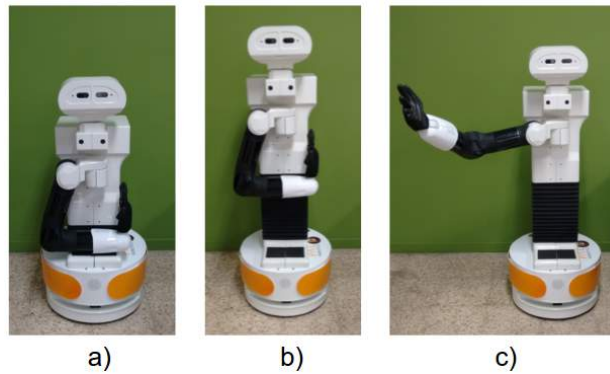


Figure 23: Procedure to start moving the arm safely

5.4 Measures to prevent collisions

Most of collisions might occur when moving the arm TIAGo. It is important to take the following measures into account in order to minimize the risk of collisions.

5.4.1 Measure 1

Make sure that there are no obstacles in the surroundings of the robot when playing back a predefined motion or when moving joints of the arm. Provided that the maximum reach of the arm is 86 cm without end-effector, a safe way to move the arm is to prevent having obstacles closer than this distance as shown in figure 24.

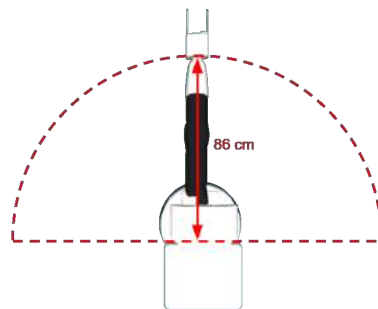


Figure 24: Area of influence of the arm

5.4.2 Measure 2

An active measure to mitigate damage due to collisions is to enable the `soft on effort` node. When enabled, this node monitors the current consumption of the 4 first joints of the arm and if some joint is consuming above a given threshold it switches the control mode of the arm to gravity compensation. After a given time-out, the position control mode is restored. This mechanism enables to prevent the arm to keep pushing an unexpected obstacle which may cause damage to the arm or the environment. Nevertheless, **this node needs to be enabled carefully** because when it is running, the arm will not be able to handle some heavy objects as the extra payload will cause extra current consumption in some joints which will trigger the control mode switch.

5.5 Measures to prevent falls

TIAGo has been designed to be stable even when the arm is holding its maximum payload in its most extreme kinematic configuration. Nevertheless, some measures need to be respected in order to avoid the robot falling.

5.5.1 Measure 1

Do not apply external downward forces to the arm when it is extended and holding a weight, see figure 25

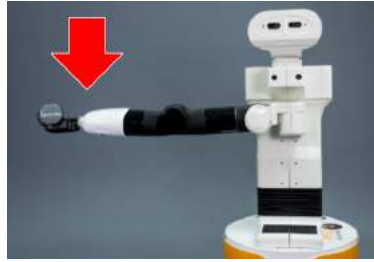


Figure 25: Fall prevention measure 1

5.5.2 Measure 2

Do not navigate at high speeds with the arm extended and holding a weight specially when the torso is also extended, see figure 26



Figure 26: Fall prevention measure 2

5.5.3 Measure 3

Do not navigate on floors with slope higher than 6° , see figure 27



Figure 27: Fall prevention measure 3

5.5.4 Measure 4

Avoid to navigate close to downstairs as the laser range finder of TIAGo will not detect this situation and the robot may fall down the stairs.

5.5.5 Measure 5

In order to ensure a safe navigation it is highly recommended to do it with the arm folded and the torso at a low extension, like in the predefined **Home** configuration, see figure 28. This pose provides the following advantages:

- Reduces the footprint of the robot lowering the probability that the arm collides with the environment
- Ensures the stability of the robot



Figure 28: Fall prevention measure 4

5.6 How to proceed when an arm collision occurs

When the arm collides with the environment, see figure 29 for an example, the motors of the arm will continue exerting force leading to potential damages to the environment or the arm covers. Serious damage to the motors will not occur as they have integrated self-protection mechanisms to detect over-temperature and over-current which switch them off automatically if necessary. Nevertheless, in order to minimize potential harm to the environment and to the robot the following procedure should be undertaken, as depicted in figure 30:

- Press the emergency button to power off the motors of the robot.
- Move the robot out of the collision by pushing the mobile base and pulling the arm to a safe place.
- Release the emergency button by rotating it clockwise according to the indications on the button until it pops out. When the On/Off button flashes press it for 1 second.
- The power and control mode of the motors are restored and the robot is operational and safe again.



Figure 29: Example of arm collision onto a table



Figure 30: How to proceed when a collision occurs

There is another way to prevent damages: enabling the `soft_on_effort` mode through the Startup Extras tab in the WebCommander as explained in section 10.4. When this mode is enabled, the robot is monitoring the current consumption of the arm modules so that unusual high current consumption will automatically switch the arm control mode to **gravity compensation**. In this way, the arm will stop exerting force against the obstacle for several seconds. Note however that the `soft_on_effort` mode can not be used when the end-effector is holding a considerable weight as some arm motions combined with the extra payload may cause high current consumptions that will trigger the control mode switch and may cause the arm to fall down.

5.7 Fire fighting equipment

For a correct use of TIAGo robot in a laboratory or location with safety conditions it is recommended to have in place a C Class fire extinguisher or an ABC Class extinguisher instead (based on halogenated products). These extinguishers are suitable to stifle an electrical fire.

If a fire-fighting action is required, please follow these instructions:

1. Call the fire fighters.
2. If you can push the emergency stop button without any risk, please do it.
3. Only tackle a fire in its very early stages.
4. Always put your own and other people's safety first.
5. On discovering the fire, immediately raise an alarm
6. Make sure you can escape if you need to and never let a fire block your exit.
7. Fire extinguishers are only for fighting a fire in its very early stages. Never tackle a fire if it is starting to spread or has spread to other items in the room or if the room is filling with smoke.
8. If you cannot stop the fire or if the extinguisher becomes empty, get out and get everyone else out of the building immediately, closing all doors behind you as you go. Then ensure the fire brigade has been called.

5.8 Leakage

The battery is the only component of the robot able to produce leakages. To avoid leakage of any substance from the battery, follow the instructions defined in section 3, to make sure that a correct manipulation and use of the battery is achieved.

TIA Go

Marking



6 Robot Identification

The robot is identified by a physical label that can be found close to the power connector.

This label (Figure 31) contains:

- Business name and full address.
- Designation of the machine.
- Part Number (P.N.).
- Year of construction.
- Serial number (S.N.).



Figure 31: Identification Label

TIA Go

Software Recovery

PAL 
ROBOTICS

7 Software recovery

7.1 Overview

The procedure of how to re-install the elements TIAGo integrates will be explained in this section.

7.2 Robot computer installation

Hardware installation TIAGo has one computer called Control. During the installation process it is needed to plug a monitor in the HDMI connector and an USB keyboard to the Service panel, see Section 2.2.8.

Software Installation The installation of the computer is performed in two steps:

1. Operating System installation:
Installs the operating system and system configuration.
2. Installation of applications:
Installation of robotic frameworks and PAL applications.

BIOS configuration The control computer BIOS requires the following options to be configured as follows:

- Insert the 4GB USB memory in a USB slot.
- Turn on the computer pressing F2. Wait until BIOS menu appears.
- In “*System Performance*” selects “*ASUS Optimal*”.
- Enter in *Advance Mode* pressing F7.
- In the ‘*Ai Tweaker\CPU Power Management*’ menu:
 - disable ‘*Enhanced Intel SpeedStep Technology*’.
- In the ‘*Advanced\CPU Configuration*’ menu:
 - disable ‘*Intel Adaptive Thermal Monitor*’.
 - disable ‘*Intel Virtualization Technology*’.
- In the ‘*Advanced\CPU Configuration\CPU Power Management Configuration*’ menu:
 - disable ‘*CPU C states*’.
- In the ‘*Boot*’ menu:
 - set ‘*Wait for F1 if error*’ to disabled.
- Go to ‘*Exit*’ and execute ‘*Save Changes & Reset*’.
- When the computer turns on again, press F2 to enter in BIOS.
- Press F8 and choose the USB device where to boot from.

Operating System installation If the BIOS has been configured as indicated in the previous section, the menu shown in Figure 32 will appear.

After “Install TIAGo” is selected, the layout of the keyboard must be chosen.

The installation will continue and no other question will be made.

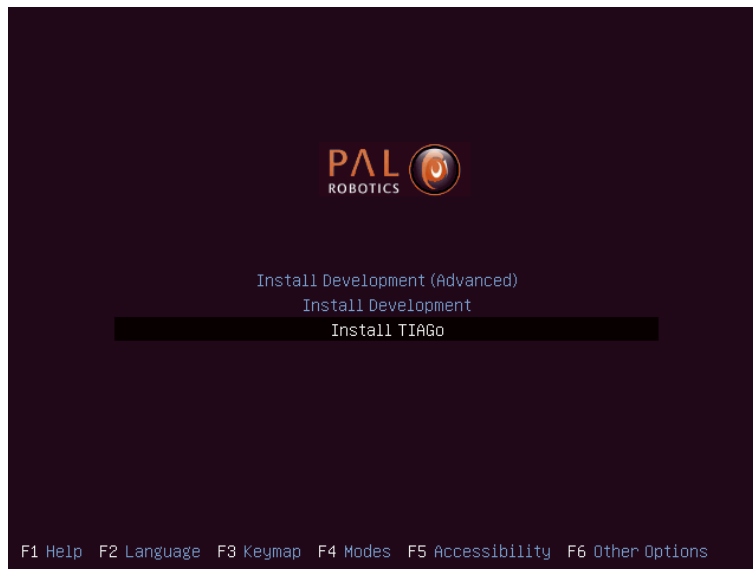


Figure 32: System installation menu

Default users Once the operating system is installed the following users have been created:

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

Applications installation Once the Operating System Installation is finished it is possible to perform the installation of the applications.

- Remove 4GB USB memory if it is connected.
- Power up the robot, open a console using Ctrl+Alt+F2 and log in the computer using *root* as username and *palroot* as password.
- Insert the 16GB USB memory in a USB slot.
- The file system has to be prepared to accept the installation, then the USB PAL-Robotics-Software (16GB) has to be mounted. In the example it can be found in *sdb*, but it can change, so use the *dmesg* command to know where the USB memory is mapped.

```
rw
chroot /ro
mkdir palApp
mount /dev/sdb palApp
cp palApp/tools/install.sh .
umount palApp
rmdir palApp
```

- Remove the USB memory.
- Then execute `install.sh`.

```
./install.sh
```

- The menu, shown in Figure 33 will appear, select embedded.


```
Install:
1. Embedded
2. Development
Q. Quit

Enter choice
1
Indicate Part Number (P.N.):
5178015001
Indicate Serial Number (S.N.):
```

Figure 33: Applications installation menu

- Select Embedded option. Then it will ask you to enter the Part Number that can be found in the ID label located on the back of the robot.
- Then you need to indicate the serial number of the robot. This serial number can be found in the ID label too.
- The application will ask for the PAL-Robotics-System ISO (4GB) and PAL-Robotics-Software ISO (16GB) in different moments of the installation. When a new ISO is asked to be inserted, the USB can be removed and replaced by the new one. There is no need to unmount.
- Once software is successfully installed user may choose to install firmware.

It is very important to let this process to finish completely. If it is aborted by the user, the device may be left in an inconsistent state. This installation process modifies `/etc/apt/source.list` and creates a backup file (`/etc/apt/sources.list.bak`) that can be used to restore it later.

7.3 Development computer installation

Hardware installation Connect the computer to the electric plug, the mouse and the keyboard. Internet access is not required as the installation is self-contained.

Software Installation The development computer installation is performed in two steps:

1. Operating System installation:
Installs the operating system and system configuration.
2. Installation of applications:
Installation of robotics frameworks, PAL applications and Acceptance Tests.

Operating System installation In order to install the Operating System it is necessary to boot from the USB memory following the next steps:

- Insert the 4GB USB memory in a USB slot.
- Press PC Start button.

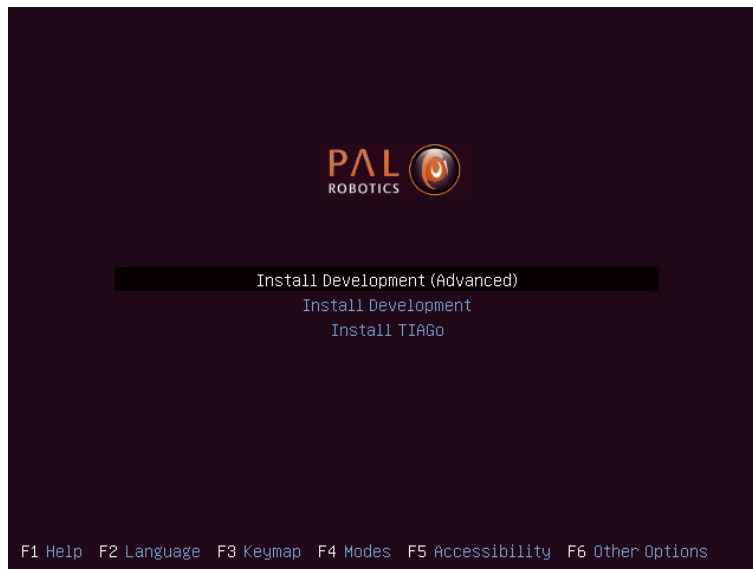


Figure 34: System installation menu

- Choose the *USB memory* as the booting device.
- Once the computer boots from the USB, the menu shown in Figure 34 will appear.

- Select the “Install Development (Advanced)” option which allows you to select a target partition in the development computer. Using “Install Development” option will install erase all partitions in the disk and install the system in a new partition. A menu asks for the layout of the keyboard. Please, choose the appropriate layout.

Default users Once the operating system is installed, the following users have been created:

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

```
Install:
1. Embedded
2. Development
Q. Quit

Enter choice
2
Choose development to install: development-tiago
█
```

Figure 35: Applications installation menu

Applications installation Once the Operating System Installation is finished, it is possible to perform the installation of the applications.

- Remove the 4GB USB memory if it is connected.
- Log in the computer using *root* as username and *palroot* as password.
- Insert the 16GB USB memory in a USB slot.
- Browse inside the USB memory to a directory called *tools*.
- Drag and drop the file *install.sh* to the Desktop of the computer.
- Open a Linux Terminal and go to Desktop (*cd Desktop*). Then execute *install.sh*.

```
./install.sh
```

- The menu, shown in Figure 35 will appear, select Development.
- Indicate the robot for which the development applications need to be installed. Enter *development-tiago*.
- The application will ask for PAL Robotics ISO. Please, insert the 4GB USB memory in a USB slot.

It is very important to let this process finish completely. If it is finished by the user, the device may be left in an inconsistent state. This installation process modifies */etc/apt/source.list* and creates a backup file (*/etc/apt/sources.list.bak*) that can be used to restore it later.

Once the installation process is finished, the *install.sh* file can be removed.

TIA Go

Robot Computer

PAL 
ROBOTICS

8 TIAGo Computer

8.1 Computer name

The name of TIAGo computer is *tiago-0c*, where *0* needs to be replaced by the serial number of your robot. For sake of clarity hereafter we will use *tiago-0cto* refer to the computer name of TIAGo.

In order to connect ot the robot use ssh as follows:

```
ssh pal@tiago-0c
```

8.2 Users

The users and default passwords in the TIAGo computer are:

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

8.3 File system

The TIAGo 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.
- /opt : This partition is mounted as read-only.
- /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@tiago-0c:~# rw
Remounting as rw...
Mounting /ro as read-write
Mounting /opt as read-write
Mounting /var/lib/dpkg as read-write
Binding system files...
root@tiago-0c:~# 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@tiago-0c:~# exit
root@tiago-0c:~# ro
Remount /ro as read only
Remount /var/lib/dpkg as read only
Remount /opt as read only
Unbinding system files
```

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

8.4 Internal DNS

Control computer has a DNS server that is used for the internal LAN of the TIAGo with 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@tiago-0c:~# 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 exists the command *delLocalDns*:

```
root@tiago-0c:~# delLocalDns -h
-h shows this help
-u DNSNAME dns name to remove

Example: addLocalDns -u terminal
```

These adds and removes in the local DNS are not persistent between reboots. A way to do the change in the local DNS persistent to reboots is to add it to the */etc/hosts* file as follows:

```
ssh root@tiago-0c
rw
chroot /ro
vi /etc/hosts
```

Append a line stating the *IP* and the *dnsname*. Exit *vi* by saving changes and then:

```
exit
ro
```

8.5 System upgrade

In order to make updates of software, there is a tool based on [synaptic](#) called *PAL Robot Synaptic*.

This tool can be executed in the development computer using Dash Home.

It asks which robot to connect to and it opens the synaptic of this robot.

Changes are performed in the persistent filesystem, so in order to keep coherence with the volatile filesystem, a reboot is performed in the robot when the application is closed.

8.6 Firmware update

Use the application described in section 8.5. Check for updates for the *pal-dubnium-firmware-** packages and install them.

After that run the *update_firmware.sh* script located in */opt/pal/dubnium/bin/firmware_update_robot* as root. Update will take a few minutes.

```
root@tiago-0c:~# cd /opt/pal/dubnium/bin/firmware_update_robot/
root@tiago-0c:/opt/pal/dubnium/bin/firmware_update_robot/# ./update_firmware.sh
```

Finally, shut down completely and power up the robot using the electric key as described in section 2.2.7.

TIA Go

Development Computer

PAL 
ROBOTICS

9 Development Computer

9.1 Overview

The Operating system used in the SDE Development Computer is based on the Linux Ubuntu 14.04 LTS distribution. Any documentation related to this specific Linux distribution applies to SDE. This document only points out the peculiarities of the PAL SDE from standard the Ubuntu 14.04.

9.2 ROS communication with the robot

When developing applications for robots based on ROS it is usual to have the `rosmaster` running on the robot's computer and the development computer running ROS nodes connected to the `rosmaster` of the robot. This is achieved by setting in each terminal of the development computer running ROS nodes the following environment variable:

```
export ROS_MASTER_URI=http://tiago-0c:11311
```

Note that in order to successfully exchange ROS messages between different computers, each of them needs to be able to resolve the **hostname** of the others. This means that the robot computer needs to be able to resolve the hostname of any development computer and the other way around. Otherwise, ROS messages will not be properly exchanged and unexpected behavior will occur.

Do the following checks before start working with a development computer running ROS nodes pointing to the `rosmaster` of the robot:

```
ping tiago-0c
```

Make sure that the `ping` command reaches the robot's computer.

Then do the same from the robot:

```
ssh pal@tiago-0c
ping devel_computer_hostname
```

If `ping` does not reach the development computer then proceed to add the hostname to the local DNS of the robot as explained in section 8.4.

9.3 System Upgrade

In order to upgrade the software of the robot and the development computers, the TIAGo's public repositories have to be used.

The source list file has to be copied in the proper directory using this command as root:

```
root@development:~# cp /etc/apt/sources.list.d.public/* /etc/apt/sources.list.d
```

PAL Robotics will notify TIAGo's whenever software upgrades are published.



Web Commander



10 WebCommander

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

It is an entry point for some monitoring tasks as well as for configuration tasks that require a Graphical User Interface.

10.1 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 TIAGo .
2. Open a web browser and type in the address bar the host name or IP address of TIAGo 's control computer and try to access port 8080:

```
http://tiago-0c:8080
```

3. If you are connected directly to TIAGo , which means using robot as access point, you can also use:

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

10.2 Overview

The WebCommander website contains visualizations of the state of TIAGo hardware, applications and installed libraries, as well as tools to configure part of it's behaviour.

10.3 The Startup Tab

The startup tab displays the list of PAL software that is configured to be started in the robot, as well as if it has been started, or if it cannot be started for any reason.

Each application, or group of applications that provide a functionality, can choose to specify a startup dependency on other applications or group 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.

10.4 The Startup Extras tab

This tab shows the list of PAL software which is not started by default during the boot up of the robot. These are optional features that need manual execution by the user. For the moment it allows the start/stop of the `soften_on_effort` mode, which switches the control mode of the arm from position to gravity compensation when an unusual current consumption occurs.

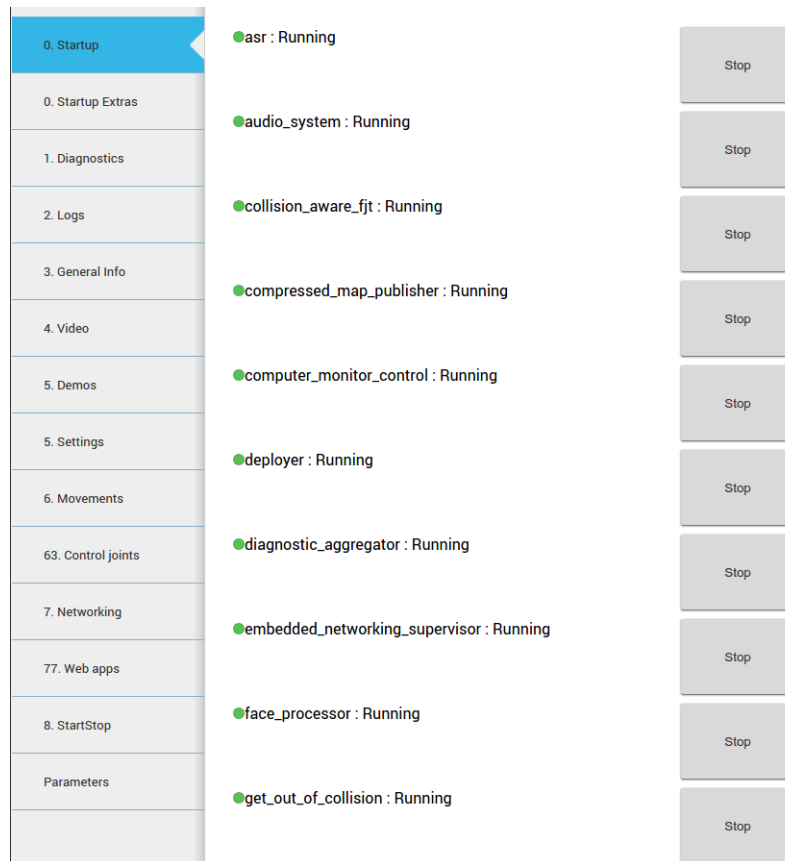


Figure 36: The Startup Tab displays the launched applications and their dependencies

10.5 The Diagnostics Tab

The diagnostic tab displays the current status of TIAGo hardware and software.

The data is organized in a hierarchical tree, in the first level there are the hardware and functionalities categories.

The functionalities are the software elements that run in TIAGo , such as navigation applications.

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

Inside the hardware and functionality categories, there's an entry for each individual functionality or device. Some devices are grouped together if there's a high number of them (motors, sonars), still each of them can be seen in detail.

The colour of the dot indicates the status of the application or component.

- Green: No errors detected
- Yellow: One or more anomalies were 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.

The status of a particular category can be expanded by clicking on the "+" symbol on the left of the name of the category. Doing this will provide information specific to the device or functionality. If there's an error, an error code will be shown.

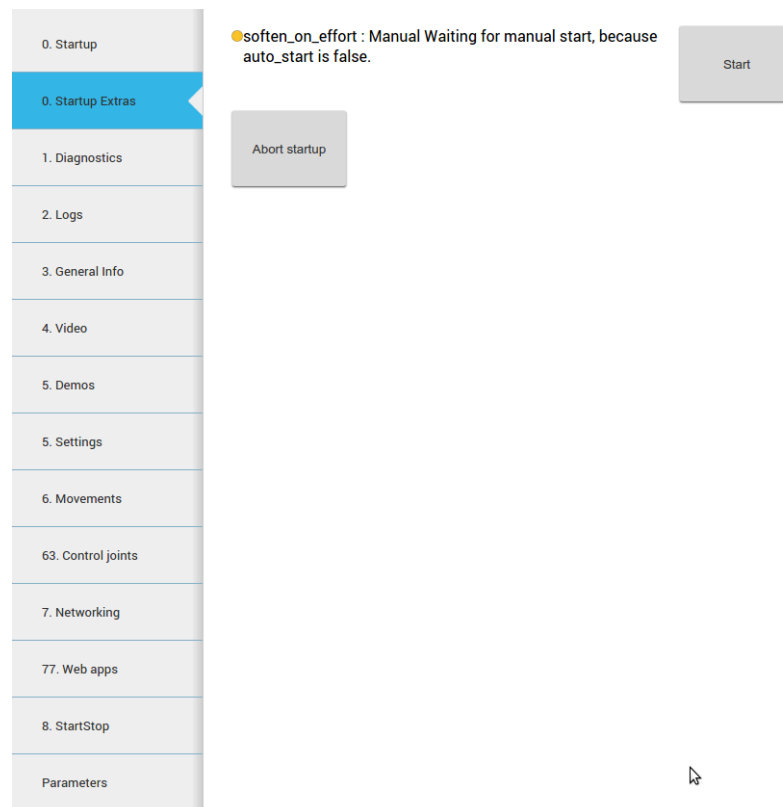


Figure 37: The Startup Tab displays the launched applications and their dependencies

10.6 The Control Joints Tab

This tab provides slides to move each joint of TIAGo upper body.

10.7 The Logs Tab

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

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

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 *Warn*, *Info* and *Debug* levels, but we can click on any of them to reenale them.

10.8 The Video Tab

The Video tab shows live video footage from the RGB-D camera on TIAGo head.

10.9 The Demos Tab

This tab provides buttons to start and stop different demos and synthesize voice messages with TIAGoeither using predefined buttons or by typing new messages on a text box.

10.10 The Movements Tab

This tab provides buttons to trigger upper body motions with TIAGo.

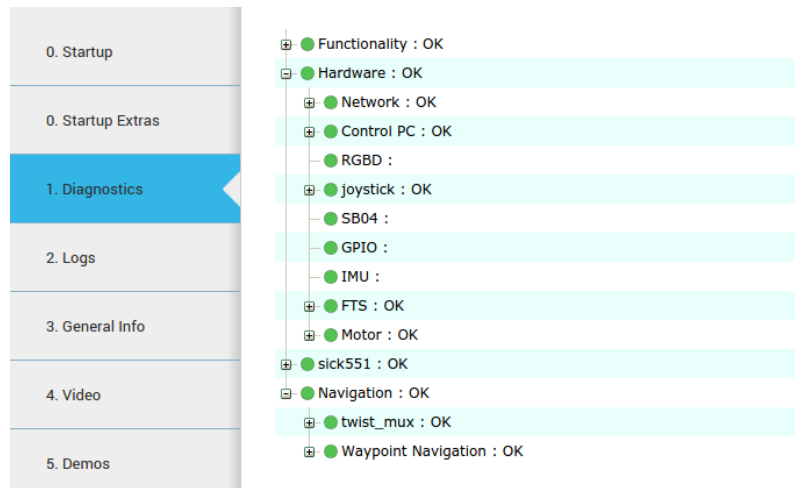


Figure 38: The Diagnostics tab displays the status of the hardware and software components of TIAGo

10.11 Networking Tab

Figure 44 shows the Networking Tab. By default the controls for changing the configuration are not visible in order to avoid access of multiple users.

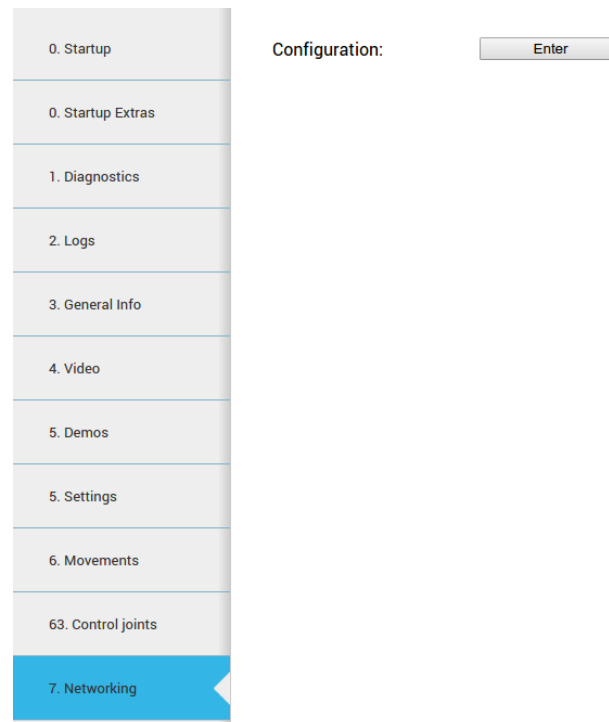


Figure 44: Networking configuration

If *Enter* button is pressed, the Tab connects to the network configuration system and the controls shown in Figure 45 appear.

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

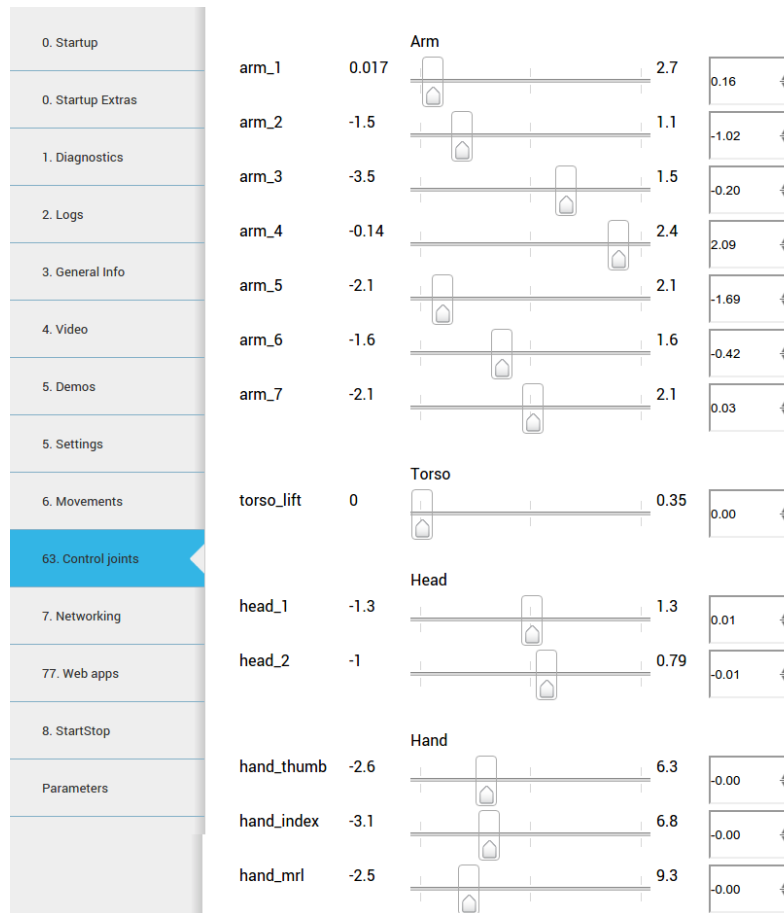


Figure 39: The Control Joint Tab provides slides to move individual joints

The Networking configuration controls are organized into sections:

- Configuration:** Exit button
- Wifi:**
 - Mode: Client
 - SSID: rian
 - Channel: 1
 - Mode Key: WPA-PSK
 - Password: *****
- Ethernet:**
 - Mode: Internal
- IPv4:**
 - ☒ Enable DHCP Wifi
 - ☐ Enable DHCP Ethernet
 - Address: 192.168.1.212
 - Network: 255.255.255.0
 - Gateway: 192.168.1.1
- DNS:**
 - Server: 192.168.1.1
 - Domain: reem
 - Search: reem
- NTP:**
 - Server: 192.168.1.1
- VPN:**
 - ☐ Enable VPN
 - ☐ Enable Firewall
 - Address: 192.168.1.6
 - Port: 1194

Buttons: Apply change, Save

Figure 45: Networking configuration controls

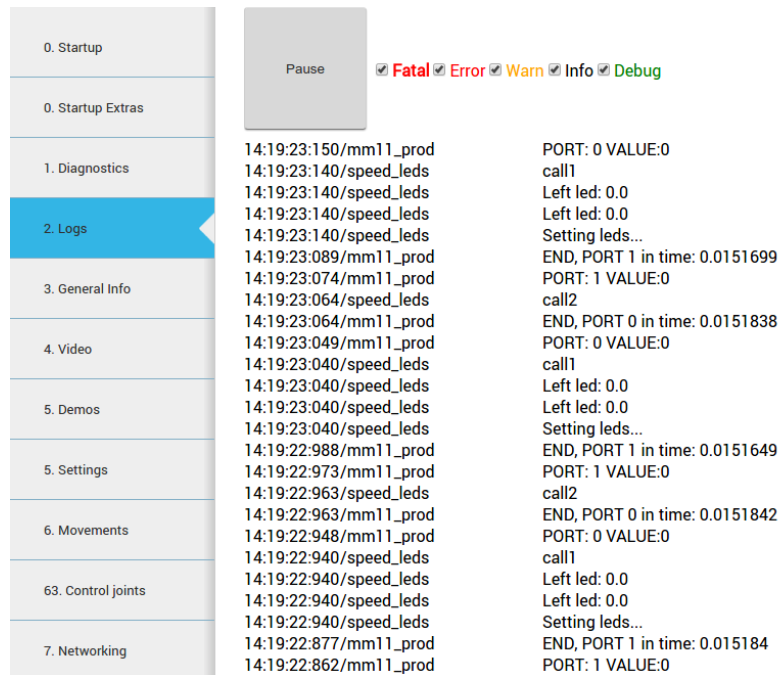


Figure 40: The Log Tab displays the log messages as they are being published in the robot

Configurations are separated in different blocks:

- Wifi:
 - Mode: Can be selected whether Wi-Fi connection works as Client or Access Point.
 - SSID: ID of the Wi-Fi to connect to Client mode or to publish in Access Point mode.
 - Channel: When robot is in Access Point mode this is the channel to use.
 - Mode Key: Encryption of the connection.
 - Password: Password for the Wi-Fi connection
- Ethernet:
 - Mode: Can be selected whether Ethernet connection works as Internal LAN or External connection (see Section 2.3.2).

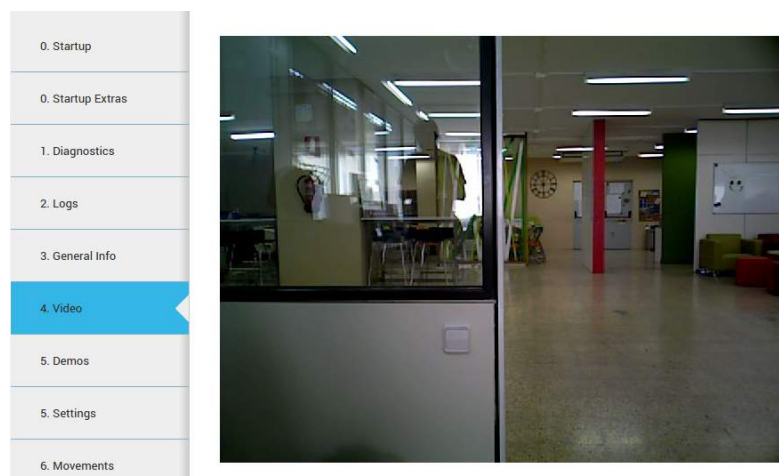


Figure 41: The Video Tab displays live video stream from the robot's camera



Figure 42: The Demos Tab provides buttons to start/stop demos and synthesize voice messages

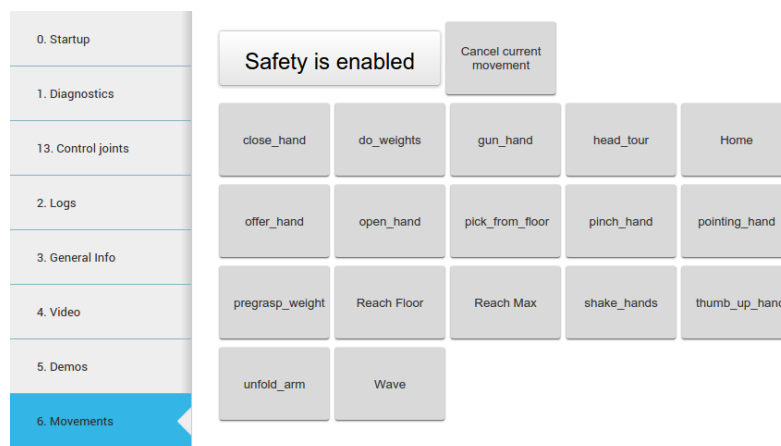


Figure 43: The Movements Tab with the predefined motions available

- IPv4

- Enable DHCP Wifi: Enables DHCP client in Wi-Fi interface.
- Enable DHCP Ethernet: Enables DHCP client in the external ethernet port.
- Address, Network, Gateway: In Client mode, manual values of the building network to use by the Wi-Fi interface. Same for external ethernet port.

- DNS

- Server: DNS server
- Domain: Domain to use in the robot.
- Search: Domains to use in the search.

- VPN

- Enable VPN: If the customer has a PAL Basestation, this robot can be connected to the VPN of the customer.
- Enable Firewall: When activating the VPN it can be selected if a firewall to avoid incoming connection from outside the VPN wants to be allowed.
- Address: Building network IP address of the Basestation.

- Port: Port of the Basestation where the VPN server is listening.

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

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

Be sure to have a correct networking configuration before saving it. A bad configuration can make impossible to connect to the robot. If this happens a general reinstallation is needed.

Changes of the Wi-Fi between Client and Access Point could require a reboot of the computer to be correctly applied.

Using Diagnostic Tab is possible to see the current state of the Wi-Fi connection.



Software architecture



11 Software architecture

11.1 Overview

The software provided in TIAGo is summarized in figure 46.



Figure 46: Tiago software summary

11.2 Operating system layer

As can be seen there are two main software blocks: the operating system, which is Ubuntu with the real-time kernel patch Xenomai; and the robotics middleware, which is based on Orocos for real-time safe communication between processes.

11.3 ROS layer

ROS is the standard robotics middleware used in TIAGo. The comprehensive list of ROS packages used in the robot are classified in three categories:

- Packages belonging to the official ROS distribution indigo.
- Packages specifically developed by PAL Robotics, which are included in its own distribution called dubnium.
- Packages developed by the customer.

The three categories of packages are installed in different locations of the SSD as shown in figure 47. The ROS indigo packages and PAL dubnium packages are installed in a read-only partition as explained in section 8. Note that even if these software packages can be modified or removed, at the customer's own risk, a better strategy is to overlay them using the deployment tool presented in section 12. The same deployment tool can be used in order to install ROS packages in the user space.

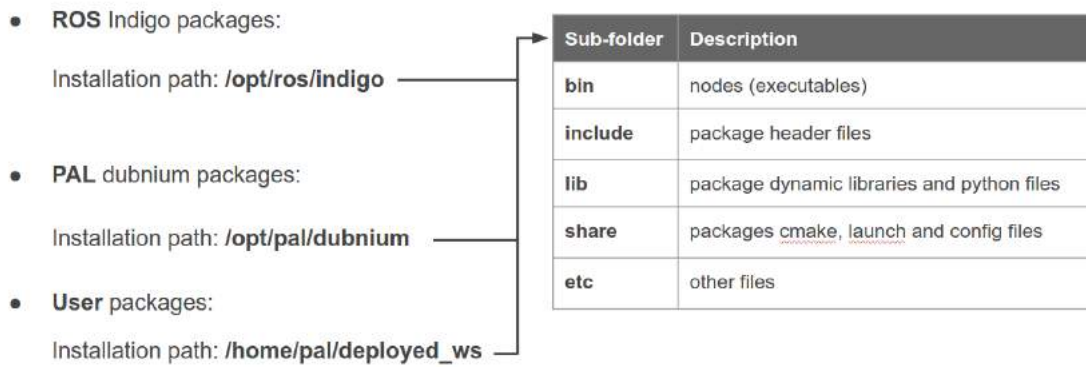


Figure 47: PAL Software overlay structure

11.4 Software startup process

When the robot boots up the software required for its operation is automatically started. The startup process can be monitored in the Web commander as shown in figure 48.

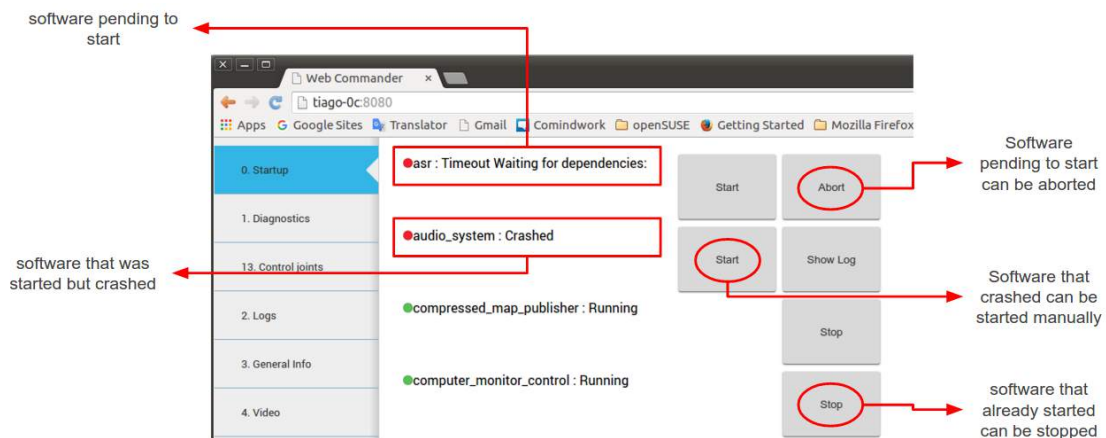


Figure 48: The Startup Tab displays the launched applications and their dependencies

TIA Go

Deployment



12 Deploy 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 deploy new software onto the robot or overlay existing ones from a development computer directly from the catkin workspace while leaving the original installation untouched.

12.1 Introduction

As explained in section 11, when TIAGo boots up it always adds two sources of packages to its ROS indigo environment. One of them is the ROS software distribution of PAL Robotics named dubnium, the other one 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 already installed packages.

To keep 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.

12.2 Usage

```
usage: rosrn pal_deploy deploy.py [-h] [--user USER] [--dirty] [--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. `tiago-5c`)

optional arguments:

```
-h, --help            show this help message and exit
--user USER, -u USER username (default: pal)
--dirty              do it the quick and dirty way, from devel space
--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.: rosrn pal_deploy deploy.py tiago-5c -u root -p my_package -c="-DCMAKE_BUILD_TYPE=Release"
```

12.3 Deploy notes

- The build type by default is `RELEASE`, meaning that in this mode executables and libraries will go through optimization during compilation and as a result will 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 recommended to deploy the packages depending on the changed library as well.
- The use of `--dirty` is not recommended as it installs using only the `devel` folder which is often dirty, as the option states. Install rules give control over what gets copied to the robot.
- There is no tool to remove individual packages from the deployed workspace. The suggested method is to delete the `/home/pal/deployed_ws` folder altogether.

12.4 Deploy tips

- You can use an alias (you may want to add it to your `.bashrc`) to ease the deploy process:

```
alias deploy="roslaunch 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 all the workspace:

```
deploy -p hello_world tiago-0c
```
- You can deploy multiple specific packages instead of all the workspace:

```
deploy -p "hello_world other_local_package more_packages" tiago-0c
```
- Before deploying you may want to do a backup of your previous `~/deployed_ws` in the robot to come back to your previous state in case of need.

12.5 Example of deployment

This section provides a simple example on how to deploy a new ROS package to the robot.

12.5.1 Create the ROS package

In the development computer load the ROS environment (you may 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 49.

```
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 49: Hello world CMakeLists.txt

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

```
// 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 50: Hello world C++ source code

Build the workspace

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

The expected output is shown in figure 51.

```
Creating build space directory, '/home/pal/example_ws/build'
-----
Profile:                        default
Extending:                      [env] /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
Devel Space:                    [missing] /home/pal/example_ws/devel
Install Space:                  [missing] /home/pal/example_ws/install
DESTDIR:                        None
-----
Isolate Develspaces:           False
Install Packages:              False
Isolate Installs:              False
-----
Additional CMake Args:         None
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 [ 1.3 seconds ]
[build] Finished.
[build] Runtime: 1.4 seconds
```

Figure 51: Build output of hello world package

Deploy the package to the robot

```
cd ~/example_ws
roslaunch pal_deploy deploy.py --user pal tiago-0c
```

Deploy tool will build the entire workspace in a separate path and if successful it will request confirmation in order to install the package to the robot as shown in figure 52.

```
[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/dubntum:/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
CMAKE_PREFIX_PATH: /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 52: Deployment of hello world package

Press Y so that the package files are installed to the robot computer. Figure 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:ff: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 53: Installation of the hello world package to the robot

Then you may connect to the robot

```
ssh pal@tiago-0c
```

And then run the new node as follows

```
roslaunch hello_world hello_world_node
```

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

TIA Go

Sensors



13 Sensors

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

13.1 Description of sensors

- Mobile base:

Laser range-finder Located at the front of the base. This sensor measures distances in a horizontal plane. It is valuable asset for navigation and mapping. Bad measurements can be caused by reflective or transparent surfaces.

Sonars These sensors are capable of measuring from low to mid-range distances. In robotics, ultrasound sensors are commonly used for local collision avoidance. Ultrasound sensors work by emitting a sound signal and measuring the reflection of this signal that returns to the sensor. Bad measurements can be caused by either blocking the sensor (will report max range in this case) or by environments where the sound signals intersect with each other.

Inertial Measurement Unit (IMU) This sensor unit is mounted at the center of TIAGo and can be used to monitor inertial forces and it also provides the attitude.

- Head:

RGB-D camera This camera is mounted inside the head of TIAGo and provides RGB images along with a depth image obtained by using an IR projector and an IR camera. The depth image is used to obtain a point cloud of the scene.

Stereo microphones An array composed of two microphones that can be used to record audio and process it in order to perform tasks like speech recognition.

- Wrist:

Force/Torque sensor (optional) This 6-axis force/torque sensor is used to obtain feedback about forces exerted on the end-effector of TIAGo.

Detailed specifications of the above sensors are provided in section 2.

14 Sensors ROS API

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

14.1 Laser range-finder

14.1.1 Topics published

/scan ([sensor_msgs/LaserScan](#))

Laser scan data of the laser scanner.

14.2 Sonars

14.2.1 Topics published

/sonar_base ([sensor_msgs/Range](#))

All measurements from sonars sensors are posted here as individual messages.

14.3 Inertial Measurement Unit

14.3.1 Topics published

/base_imu ([sensor_msgs/Imu](#))

Inertial data from the IMU .

14.4 RGB-D camera

14.4.1 Topics published

/xtion/depth_registered/camera_info ([sensor_msgs/CameraInfo](#))

Intrinsic parameters of the depth image.

/xtion/depth_registered/image_raw ([sensor_msgs/Image](#))

32-bit depth image. Every pixel contains the depth of the corresponding point.

/xtion/depth_registered/points ([sensor_msgs/PointCloud2](#))

Point cloud computed from the depth image.

/xtion/rgb/camera_info ([sensor_msgs/CameraInfo](#))

Intrinsic and distortion parameters of the RGB camera.

/xtion/rgb/image_raw ([sensor_msgs/Image](#))

RGB image.

/xtion/rgb/image_rect_color ([sensor_msgs/Image](#))

RGB rectified image.

14.4.2 Services advertised

/xtion/get_serial ([openni2_camera/GetSerial](#))

Service to retrieve the serial number of the camera.

/xtion/rgb/set_camera_info ([sensor_msgs/SetCameraInfo](#))

Changes the intrinsic and distortion parameters of the color camera.

14.5 Force/Torque sensor

14.5.1 Topics published

/wrist_ft ([geometry_msgs/WrenchStamped](#))

Force and torque vectors currently detected by the f/t sensor.

15 Sensors visualization

Most of the sensors readings of TIAgo can be visualized in rviz. In order to start the rviz GUI with a pre-defined configuration do as follows from the development computer:

```
export ROS_MASTER_URI=http://tiago-0c:11311
roslaunch tiago_bringup config/tiago.rviz
```

An example of how the laser range-finder is visualized in rviz is shown in figure 54.

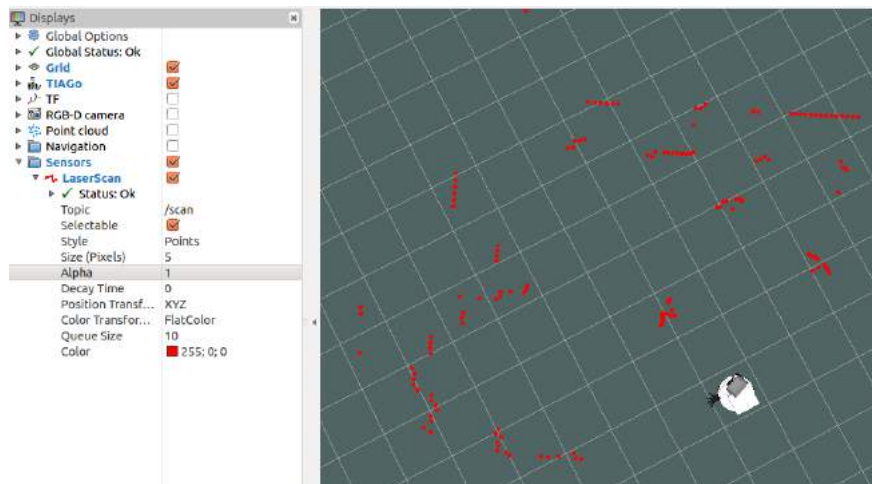


Figure 54: Visualization of the laser range-finder

Figure 55 shows an example of visualization of the RGB image, depth image and point cloud provided by the RGB-D camera.

Finally, figure 56 presents an example of force vector detected by the force/torque sensor.

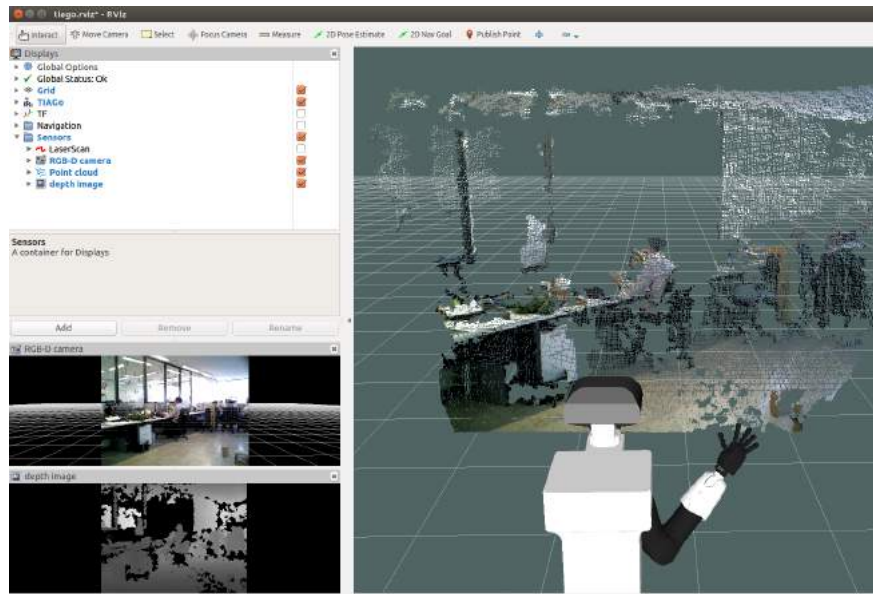


Figure 55: Visualization of the RGB-D camera

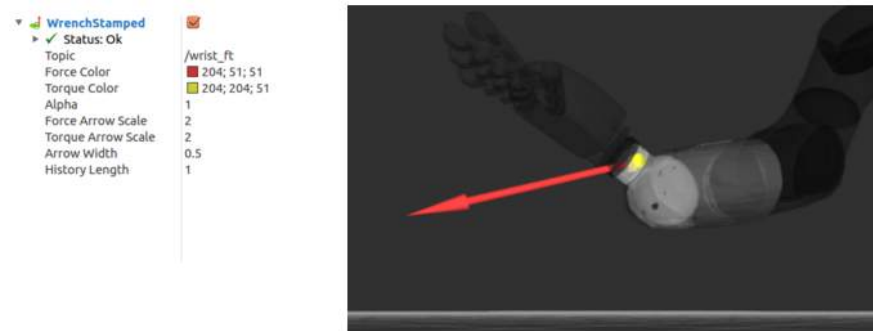


Figure 56: Visualization of force/torque sensor



Power status



16 Power status

This section contains an overview of the power related status data reported by the TIAGo robot. The ROS API and a brief description of the information available is presented.

16.1 ROS API

Robot power status is reported in the `/power_status` ROS topic.

NOTE: This node is launched by default on startup.

16.2 Description

The following data is reported.

- `input`: the voltage coming from the batteries.
- `charger`: the voltage coming from the charger.
- `dock`: the voltage coming from the dock station (not available with TIAGo).
- `pc`: the voltage coming from the PC.
- `charge`: the percentage battery charge.
- `is_connected`: whether TIAGo is currently connected to the charger.
- `is_emergency`: whether the emergency stop button is currently enabled.

TIA Go

Voice synthesis

PAL
ROBOTICS 

17 Voice synthesis

TIAGo is provided by Acapela's voice synthesis software which can be used through a ROS interface.

17.1 ROS API

17.1.1 Action interface

`/tts` ([pal_interaction_msgs/Tts Action](#))

This action allows to send a text to the voice synthesizer. `tts` stands for "text-to-speech".

17.2 Examples of usage

17.2.1 WebCommander

Sentences can be synthesized using the WebCommander. Several buttons corresponding to different predefined sentences are provided in the lower part of the *Demos* tab as shown in figure 57. Furthermore, a text field is provided in the top part of the tab so that any other sentence can be written and then synthesized by pressing *Say* button.



Figure 57: Voice synthesis in the Demo tab of the WebCommander

17.2.2 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 and 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"
```

17.2.3 Action client

Another way to send goals to the voice synthesis server is to use the GUI included in the `actionlib` package of ROS. This can be run as follows:

```
export ROS_MASTER_URI=http://tiago-0c:11311
roslaunch actionlib axclient.py /tts
```

The GUI shown in figure appears and the fields of the goal message under `rawtext` can be filled and then pressing the **SEND GOAL** button will trigger the action.

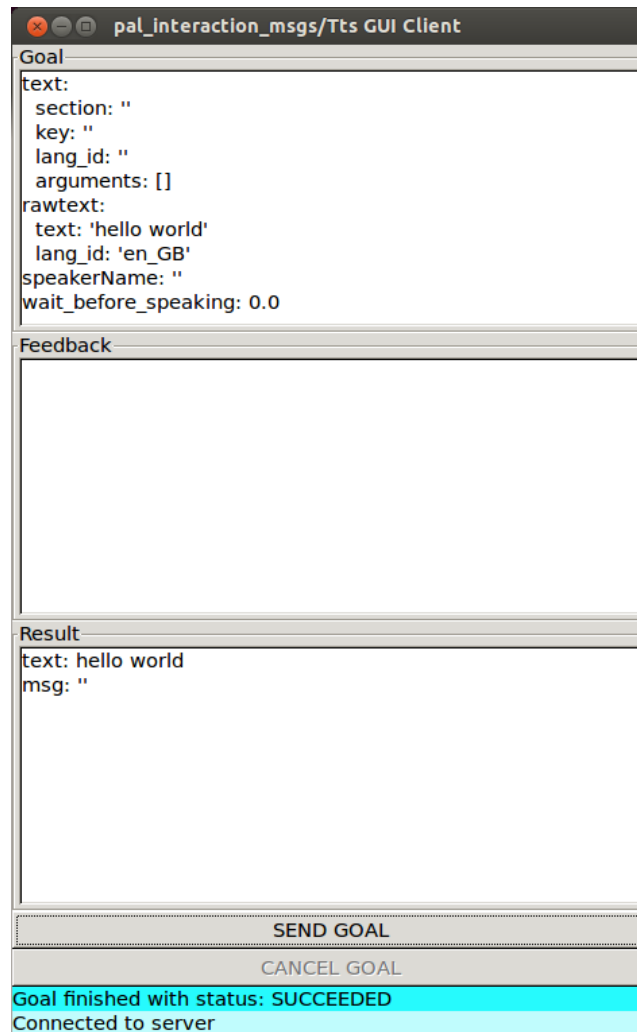


Figure 58: Voice synthesis using the GUI from actionlib

TIA Go

Base motions



18 Base motions

18.1 Overview

This section explains the different ways to move the base of TIAGo. The mobile base is based on a differential drive, which means that a linear and an angular velocity can be set as shown in figure 67. TheFirst of all the motion triggers implemented in the joystick will be exposed. Then, the underlying ROS API to access the base controller will be presented.

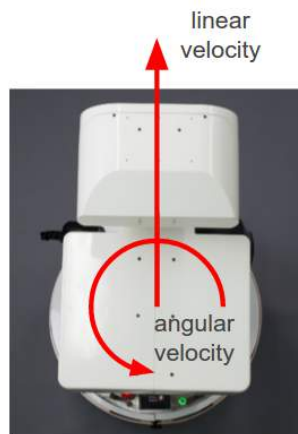


Figure 59: Mobile base velocities that can be commanded

18.2 Base motion joystick triggers

In order to start moving the base with the joystick the priority has to be given to this peripheral. In order to gain priority with the joystick just press the button shown in figure 60. The joystick can release the priority by pressing again the same button.



Figure 60: Taking the priority with the joystick

18.2.1 Forward/backward motion

To move the base forward use the left analog stick as shown in figure 68.



Figure 61: Base linear motion with the joystick

18.2.2 Rotational motion

In order to make the base rotate about its Z axis the right analog stick has to be operated as shown in figure 62



Figure 62: Base rotational motion with the joystick

18.2.3 Changing the speed of the base

The default linear and rotation speed of the base can be changed with button combinations of the joystick:

- a) Increase linear speed, see figure 63a
- b) Decrease linear speed, see figure 63b
- c) Increase angular speed, see figure 63c
- d) Decrease angular speed, see figure 63d



Figure 63: Joystick buttons combinations to change speed

18.3 Mobile base control ROS API

At user level linear and rotational speeds can be sent to the mobile base controller using the following topic:

`/mobile_base_controller/cmd_vel` ([geometry_msgs/Twist](#))

The given linear and angular velocities are internally translated to the required angular velocities of each one of the two drive wheels.

18.4 Mobile base control diagram

Different ROS nodes publish velocity commands to the mobile base controller through the `/mobile_base_controller/cmd_vel` topic. In figure 64 the default nodes trying to gain control of the mobile base are shown. From one side, there are velocities triggered from the joystick and from the other side, commands from the `move_base` node, which is used by the navigation pipeline that is presented in section 25.

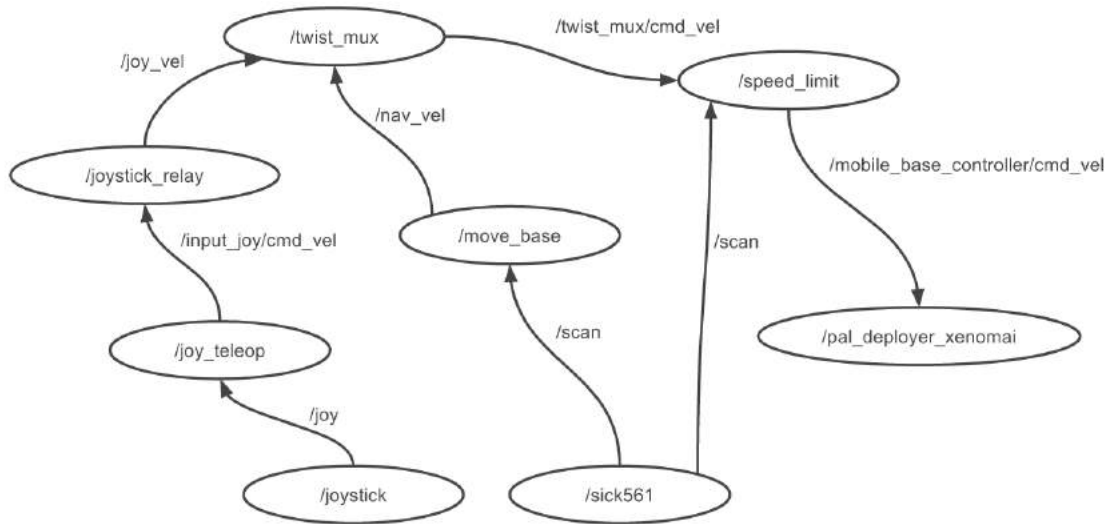


Figure 64: Mobile base control diagram



Torso motions



19 Torso motions

19.1 Overview

This section explains how to move the prismatic joint of the lifting torso of TIAGObot using the joystick or using the available ROS APIs.

19.2 Torso motion joystick triggers

In order to start moving the torso with the joystick the priority has to be given to this peripheral as explained in section 18.2. Then, the torso is moved by using the LB and LT buttons of the joystick see figure 65. Press LB to raise the torso and LT to move it downwards, see figure 66.



Figure 65: Buttons to move the torso



Figure 66: Torso vertical motions triggered with the joystick

19.3 Torso control ROS API

19.3.1 Topic interfaces

`/torso_controller/command` ([trajectory_msgs/JointTrajectory](#))

sequence of positions that the torso joint needs to reach in given time intervals.

`/torso_controller/safe_command` ([trajectory_msgs/JointTrajectory](#))

Idem as before but the motion is only executed if it does not lead to a self-collision.

19.3.2 Action interfaces

`/torso_controller/follow_joint_trajectory` ([control_msgs/FollowJointTrajectory Action](#))

This action encapsulates the `trajectory_msgs/JointTrajectory` message.

`/safe_torso_controller/follow_joint_trajectory` ([control_msgs/FollowJointTrajectory Action](#))

Idem as before but the goal is discarded if a self-collision will occur.



Head motions



20 Head motions

20.1 Overview

This section explains how to move the two rotational joints of the head of the robot with the joystick and then the underlying ROS API is presented.

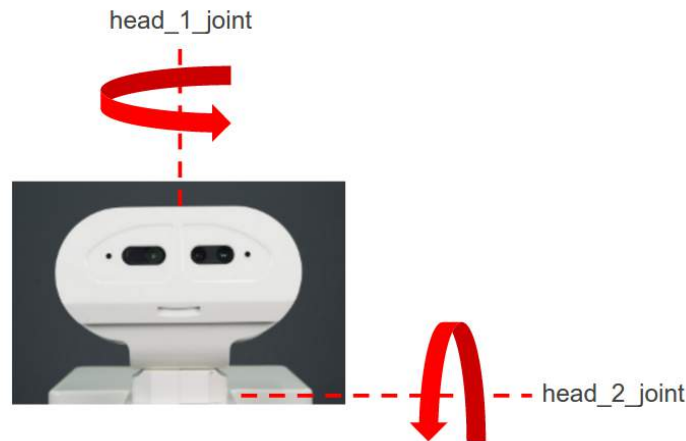


Figure 67: Rotational joints of the head

20.2 Head motion joystick triggers

The head is moved by using the X, Y, A and B buttons on the right part of the joystick, see figure 68.



Figure 68: Buttons to move the head

20.3 Head motions with rqt GUI

The joints of the head can be moved individually using a GUI implemented on the rqt framework. The GUI is launched as follows:

```
roslaunch rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```

The GUI is shown in figure 69. Note that other groups of joints, i.e. head, torso, hand, gripper, can be also moved using this GUI. Furthermore, this is equivalent to use the Control Joints tab in the Web commander as explained in section 10.6. In order to move the head joints select `/controller_manager` in the combo box at the left and the `head_controller` at the right. Sliders for the two actuated joints of the head will show up.

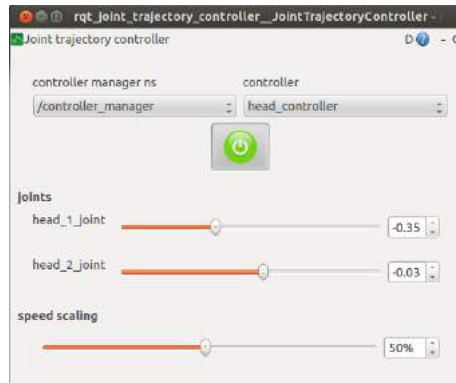


Figure 69: rqt GUI to move individual joints of the head

20.4 Head control ROS API

20.4.1 Topic interfaces

`/head_controller/command` ([trajectory_msgs/JointTrajectory](#))

sequence of joint positions that needs to be achieved in given time intervals.

20.4.2 Action interfaces

`/head_controller/follow_joint_trajectory` ([control_msgs/FollowJointTrajectory Action](#))

This action encapsulates the `trajectory_msgs/JointTrajectory` message.

`/head_controller/point_head_action` ([control_msgs/PointHeadAction Action](#))

This action is used to make the robot look to a given cartesian space.

TIA Go

Arm motions



21 Arm motions

21.1 Overview

This section explains how to move the 7 DoF of TIAGo arm using a GUI or using the available ROS APIs.

21.2 Arm motions with rqt GUI

The joints of the arm can be moved individually using a GUI implemented on the rqt framework. The GUI is launched as follows:

```
roslaunch rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```

The GUI is shown in figure 70. Note that other groups of joints, i.e. head, torso, hand, gripper, can be also moved using this GUI. Furthermore, this is equivalent to use the Control Joints tab in the Web commander as explained in section 10.6. In order to move the arm joints select `/controller_manager` in the combo box at the left and the `arm_controller` at the right. Sliders for the seven joints of the arm will show up.



Figure 70: rqt GUI to move individual joints

21.3 Arm control ROS API

21.3.1 Topic interfaces

`/arm_controller/command` ([trajectory_msgs/JointTrajectory](#))

sequence of positions that the arm joints have to reach in given time intervals.

`/arm_controller/safe_command` ([trajectory_msgs/JointTrajectory](#))

Idem as before but the motion is only executed if it does not lead to a self-collision.

21.3.2 Action interfaces

/arm_controller/follow_joint_trajectory ([control_msgs/FollowJointTrajectory](#) Action)

This action encapsulates the trajectory_msgs/JointTrajectory message.

/safe_arm_controller/follow_joint_trajectory ([control_msgs/FollowJointTrajectory](#) Action)

Idem as before but the goal is discarded if a self-collision will occur.

TIA Go

Hand motions



22 Hand motions

22.1 Overview

This section explains how to move the 3 motors of the Hey5 hand using the joystick or the rqt GUI. Then, the ROS API of the hand is presented.

22.2 Hand motion joystick triggers

There are joystick triggers defined in order to close and open the hand using buttons RT and BT, respectively, as shown in figure 71.



Figure 71: Joystick buttons to trigger hand close/open motions

22.3 Hand motions with rqt GUI

The joints of the hand can be moved individually using a GUI implemented on the rqt framework. The GUI is launched as follows:

```
roslaunch rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```

The GUI is shown in figure 72. Note that other groups of joints, i.e. head, torso, hand, gripper, can be also moved using this GUI. Furthermore, this is equivalent to use the Control Joints tab in the Web commander as explained in section 10.6. In order to move the hand joints select `/controller_manager` in the combo box at the left and the `hand_controller` at the right. Sliders for the three actuated joints of the hand will show up.



Figure 72: rqt GUI to move individual joints of the hand

22.4 Hand control ROS API

22.4.1 Topic interfaces

`/hand_controller/command` ([trajectory_msgs/JointTrajectory](#))

sequence of positions that the hand joints have to reach in given time intervals.

`/hand_current_limit_controller/command` ([pal_control_msgs/ActuatorCurrentLimit](#))

Set maximum allowed current for each hand actuator specified as a factor in $[0, 1]$ of the actuator's maximum current.

22.4.2 Action interfaces

`/hand_controller/follow_joint_trajectory` ([control_msgs/FollowJointTrajectory](#) Action)

This action encapsulates the `trajectory_msgs/JointTrajectory` message.



Gripper motions



23 Gripper motions

23.1 Overview

This section explains how to command the gripper of TIAGo using a GUI or using the available ROS APIs.

23.2 Gripper motion joystick triggers

There are joystick triggers defined in order to close and open the gripper incrementally using buttons RT and BT, respectively, as shown in figure 73.



Figure 73: Joystick buttons to close/open the gripper

23.3 Gripper motions with rqt GUI

Launch the rqt gui to command groups of joints as follows:

```
roslaunch rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```

Select the controller manager namespace available and the gripper_controller and two sliders, one for each gripper joint, will appear as shown in figure 74. Each slider controls the position of each finger.

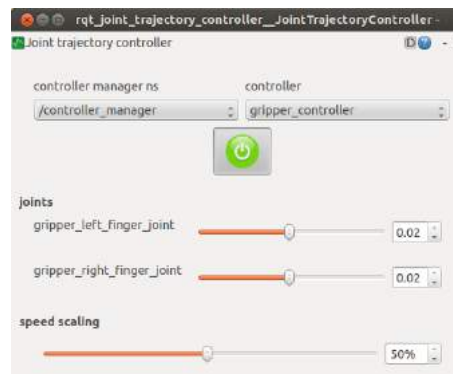


Figure 74: rqt GUI to move the gripper

23.4 Gripper control ROS API

23.4.1 Topic interfaces

/parallel_gripper_controller/command ([trajectory_msgs/JointTrajectory](#))

This topic is used to specify the desired distance between fingers. A single target position or a sequence of positions can be specified. For instance, the following command moves the gripper motors so that the distance between fingers becomes 4 cm:

```
rostopic pub /parallel_gripper_controller/command trajectory_msgs/JointTrajectory "
header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: ''
joint_names:
- 'parallel_gripper_joint'
points:
- positions: [0.04]
  velocities: []
  accelerations: []
  effort: []
  time_from_start:
    secs: 1
    nsecs: 0" --once
```

/gripper_controller/command ([trajectory_msgs/JointTrajectory](#))

sequence of positions to send to each motor of the gripper. The joints are `gripper_left_finger_joint` and `gripper_right_finger_joint`. Position 0 correspond to closed gripper and 0.04 corresponds to open gripper. An example to set the fingers to different positions from command line is shown below:

```
rostopic pub /gripper_controller/command trajectory_msgs/JointTrajectory "
header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: ''
joint_names: ['gripper_left_finger_joint', 'gripper_right_finger_joint']
points:
- positions: [0.04, 0.01]
  velocities: []
  accelerations: []
  effort: []
  time_from_start:
    secs: 1
    nsecs: 0" --once
```

23.4.2 Action interfaces

/parallel_gripper_controller/follow_joint_trajectory ([control_msgs/FollowJointTrajectory Action](#))

This action encapsulates the `trajectory_msgs/JointTrajectory` message in order to perform gripper motions with information about when it ends or whether it has executed successfully.

23.4.3 Service interfaces

/gripper_controller/grasp ([std_msgs/Empty](#))

This service makes the gripper close the fingers until a grasp is detected. When that happens the controller keeps the fingers in the position reached at that moment in order to hold the object and not overheating the motors.

An example on how to call the service from command line is:

```
rosservice call /gripper_controller/grasp
```


TIA Go

Upper body motions engine



24 Upper body motions engine

TIAGo is provided with a motions engine to play back predefined motions involving joints of the upper body. A default library with several motions is provided and the user can add new motions that can be played at any time. The motions engine provided with TIAGo is based on the `play_motion` ROS package which is available in http://wiki.ros.org/play_motion.

This package contains a ROS Action Server that acts as a demultiplexer to send goals to different action servers each one in charge of commanding different groups of joints. Figure 75 shows the role of `play_motion` in order to play back predefined upper body motions.

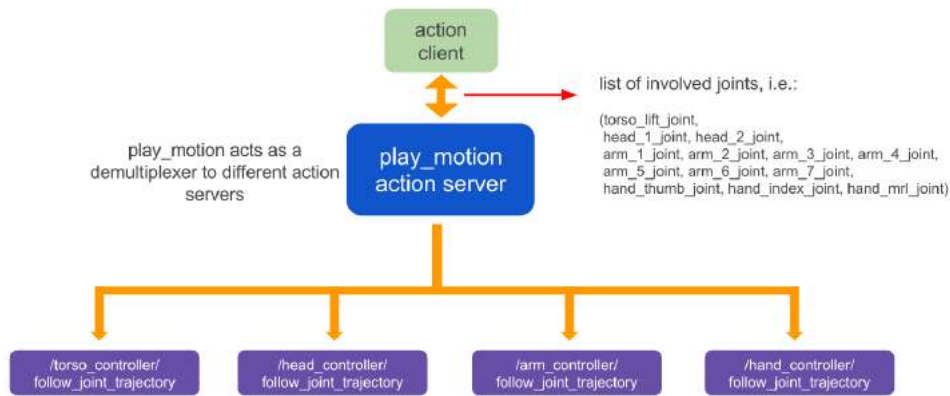


Figure 75: `play_motion` action server role

The different groups of actuated joints defined in TIAGo are the ones shown in table 20.

Group of joints	Joints included
torso	torso_lift
head	head_1, head_2
arm	arm_1, arm_2, arm_3, arm_4, arm_5, arm_6, arm_7
hand	hand_thumb, hand_index, hand_mrl
gripper	gripper_left_finger, gripper_right_finger

Table 20: Groups of joints

The motions that `play_motion` is able to play back are based on a sequence of joints positions that need to be reached within given time intervals as it will be explained in more detail in the following sections.

24.1 Motions library

The motion library is stored in the yaml file `tiago_bringup/config/tiago_motions.yaml`. The contents of this file are uploaded to the ROS param server during the boot up of the robot in `/play_motion/motions`. When the `play_motion` action server is launched, it looks for the motions defined in the param server. The yaml sotring the predefined motions file can be edited as follows:

```

roscd tiago_bringup
cd config
vi tiago_motions.yaml
  
```

The motions already defined in the library are:

- home

- `unfold_arm`
- `reach_floor`
- `reach_max`
- `head_tour`
- `wave`
- `pregrasp_weight`
- `do_weights`
- `pick_from_floor`
- `shake_hands`
- `open_hand`
- `close_hand`
- `pointing_hand`
- `gun_hand`
- `thumb_up_hand`
- `pinch_hand`

New motions can be added to the library by editing the yaml file in `tiago_bringup` package.

24.2 Motions specification

Every motion is specified with the following data structure:

- **joints**: list of joints used by the motion. Note that by convention, when defining a motion involving a given joint, the rest of joints in the subgroup must be also included in the motion specification. For example, if the predefined motion needs to move `head_1` joint, then it needs also to include `head_2` joint as they belong both to the same group, see table 20.
- **points**: list of the following tuple:
 - *positions*: list of positions that need to be reached by every joint
 - *time_from_start*: time given to reach the positions specified above
- **meta**: meta information that can be used by other applications

As example, the specification of the `wave` motion is shown in figure 76.

As can be seen, the joints included in the motion are the ones in the `arm` group.


```

joints: [arm_1_joint, arm_2_joint, arm_3_joint, arm_4_joint, arm_5_joint, arm_6_joint, arm_7_joint]
meta:
  description: wave
  name: Wave
  usage: demo
points:
- positions: [0.06337464909724033, -0.679638896132783, -3.1087325315620733, 2.0882339360702575,
-1.1201172410014792, -0.031008601325809293, -2.0744261217334135]
  time_from_start: 0.0
- positions: [0.06335930908588873, -0.7354151774072313, -2.939624246421942, 1.8341256735249563,
-1.1201355028397157, -0.031008601325809293, -2.0744261217334135]
  time_from_start: 1.0
- positions: [0.06335930908588873, -0.7231278283145929, -2.9385504456273295, 2.2121050027803877,
-1.1201355028397157, -0.031008601325809293, -2.0744261217334135]
  time_from_start: 2.0
- positions: [0.06335930908588873, -0.7354151774072313, -2.939624246421942, 1.8341256735249563,
-1.1201355028397157, -0.031008601325809293, -2.0744261217334135]
  time_from_start: 3.0

```

Figure 76: play_motion action server role

24.3 Predefined motions safety

Special care needs to be taken when defining predefined motions as if they are not well defined self-collisions or collisions with the environment may occur.

A safety feature is included in `play_motion` in order to minize the risk of self-collisions in the begining of the motion. As the upper body joints can be in any arbitrary position before starting the motion, the joint movements in order to reach the first predefined positions in the `points` tuple are the most dangerous ones. For this reason `play_motion` can use motion planning included in MoveIt! in order to find joint trajectories that will prevent self-collisions when reaching the first desired position of the motion.

The planning stage takes only a few amount of time, i.e. less than 2 seconds. The user can disable this feature as it will be shown in the next section. Nevertheless, it is **strongly recommended** not to do so unless the position of the joints before executing the motion is well known and a straight trajectory of the joints towards the first intermediate position is safe.

24.4 ROS interface

The motion engine exposes an Action Server interface in order to play back predefined upper body motions.

24.4.1 Action interface

`/play_motion` ([play_motion_msgs/PlayMotion Action](#))

This action plays back a given predefined upper body motion given its name.

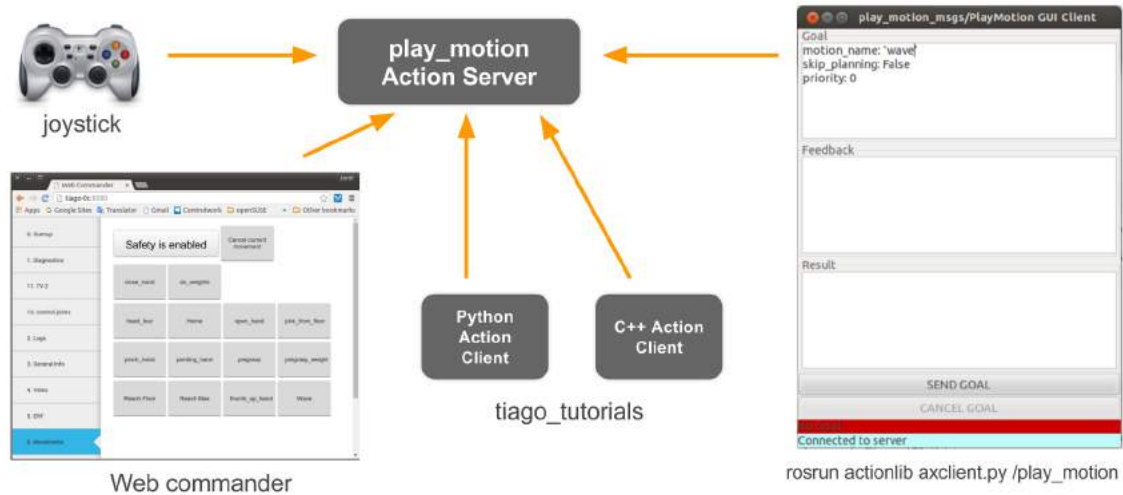
The goal message includes the following data:

- **motion_name**: name of the motion as specified in the motions yaml file.
- **skip_planning**: when true, motion planning is not used to move the joints from the initial position to the first position specified in the list of positions of the motion. This parameter should be set to `False` by default to minimize the risk of self-collisions as explained in section 24.3.
- **priority**: unimplemented feature. For future use.

24.5 Action clients

There are multiple ways to send goals to `play_motion` Action Server. TIAGo provides several action clients in order to request the execution of predefined upper body motions. These action clients are summarized in figure 77.

Note that predefined motions can be executed from:

Figure 77: `play_motion` action clients provided

- **Joystick**: several combinations of buttons are already defined to trigger torso and hand/gripper motions.
- **Web commander**: all the predefined motions including the meta data in `tiago_motions.yaml` will appear in the `Movements` tab of the web interface.
- **Axclient GUI**: using this graphical interface any predefined upper body motion can be executed.
- **Examples in the tutorials**: examples in C++ and in Python are provided in `tiago_tutorials/run_motion` package on how to send goals to the `/play_motion` action server.



Navigation



25 Navigation

25.1 Overview

This section presents the autonomous navigation framework of TIAGo. The [ROS 2D navigation stack](#) is the basis of TIAGo navigation. The navigation software is composed of all the ROS nodes running in the robot that are able to perform SLAM, i.e. mapping and localization using the laser on the mobile base, and path planning to bring the robot to any map location avoiding obstacles and preventing collisions using laser and sonars readings.

25.2 Navigation architecture

The navigation software provided with TIAGo can be seen as a black box with the inputs and outputs shown in figure 78.

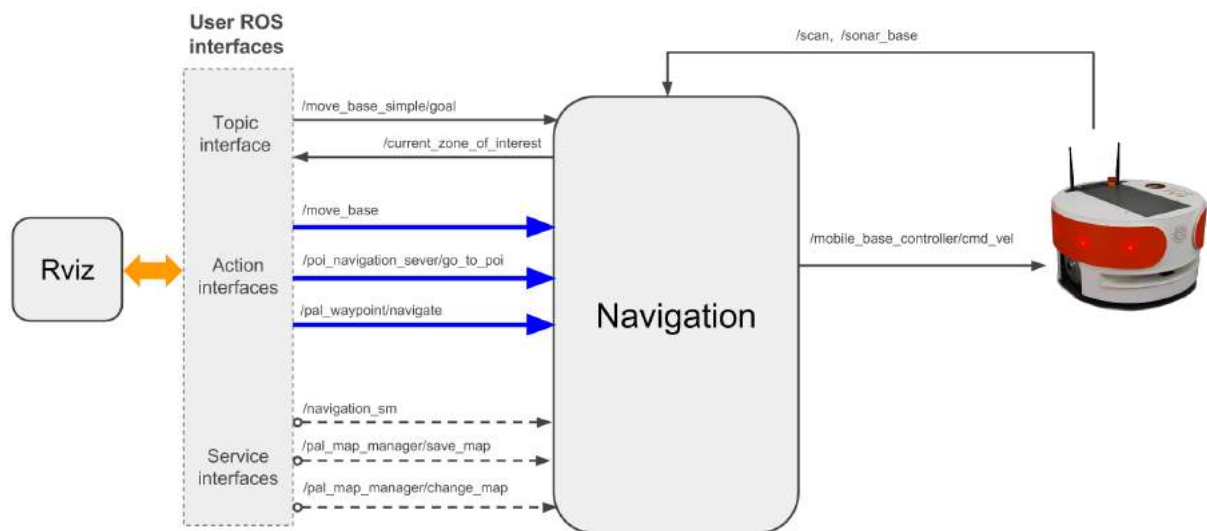


Figure 78: TIAGo navigation black box

As can be seen, the user can communicate with the navigation software through a ROS topic, three different ROS actions and one ROS service. Note that Rviz also can use these interfaces to help the user perform navigation tasks.

The ROS nodes composing the navigation architecture are shown in figure 79. Note that nodes communicate using topics, actions, services but also using parameters in the ROS param server (note depicted in the figure).

25.3 Navigation ROS API

25.3.1 Topic interfaces

`/move_base_simple/goal` ([geometry_msgs/PoseStamped](#))

Topic interface to send the robot to a pose specified in `/map` metric coordinates. Use this interface if no monitoring of the navigation status is required.

`/current_zone_of_interest`¹ ([pal_zoi_detector/CurrentZoi](#))

This topic prints the name of the region of interest where the robot is at present, if any.

¹The publisher of this topic is included in the Advanced Navigation package.

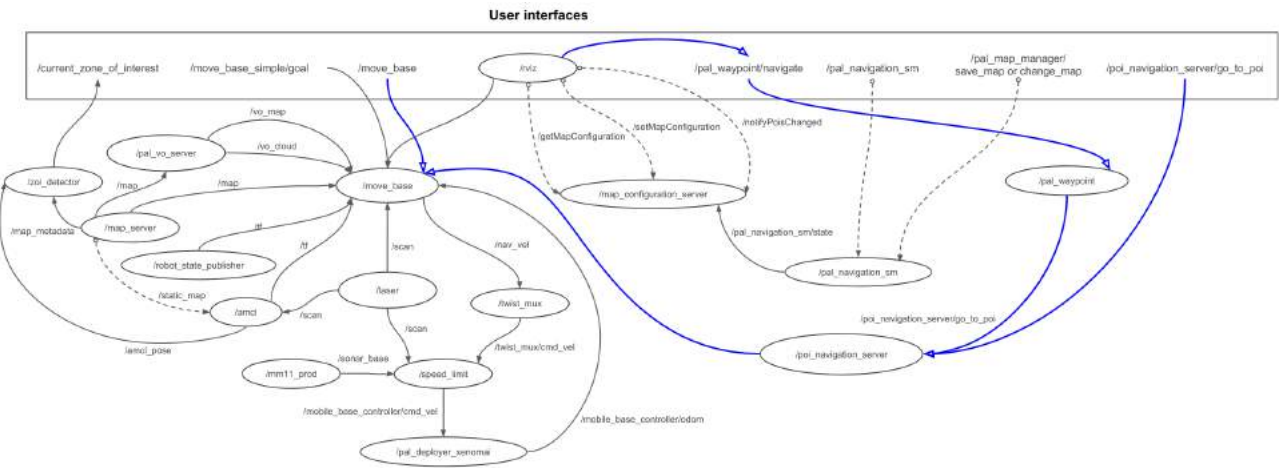


Figure 79: TIAGo navigation nodes overview

25.3.2 Action interfaces

```
/move_base (move_base_msgs/MoveBaseAction)
```

Action to send the robot to a pose specified in /map metric coordinates. Use of this interface is recommended when the user needs to get noticed when the goal has been reached or whether something fails in the process.

```
/poi_navigation_server/go_to_poi2 (pal_navigation_msgs/GoToPOIAction)
```

Action to send the robot to an existing Point Of Interest, hereafter POI, by providing its identifier. POIs can be set using the Map Editor, see section 25.6.

```
/pal_waypoint/navigate2 (pal_waypoint_msgs/DoWaypointNavigationAction)
```

Action to make the robot visit all the POIs of a given group or just a subset of them. POIs and groups of them can be defined using the Map Editor, see section 25.6.

25.3.3 Service interface

```
/pal_navigation_sm (pal_navigation_msgs/Acknowledgment)
```

Service to set the navigation mode to mapping or to localization mode.

In order to set the mapping mode do as follows:

```
rosservice call /pal_navigation_sm "input: 'MAP'"
```

In order to set the robot in localization mode:

```
rosservice call /pal_navigation_sm "input: 'LOC'"
```

In the localization mode the robot is able to plan paths to any valid point of the map.

```
/pal_map_manager/save_map (pal navigation msgs/SaveMap
```

²This software is included in the Advanced Navigation package

Service to save the map with a given name. Usage example:

```
rosservice call /pal_map_manager/save_map "directory: 'my_office_map'"
```

The directory argument is the name of the map. If empty a timestamp will be used. The maps are stored in `$HOME/.pal/productname_maps/configurations`.

`/pal_map_manager/change_map` ([pal_navigation_msgs/Acknowledgment](#))

Service to choose the active map. Usage example:

```
rosservice call /pal_map_manager/change_map "input: 'my_office_map'"
```

25.4 SLAM and path planning in simulation

25.4.1 Mapping

The mapping system uses the readings provided by the 2D laser scanner while the robot is moved around with keyboard/joystick. The map obtained is an Occupancy Grid Map (OGM) that later can be used to make the robot localize and navigate autonomously in the environment.

We can run the simulator and the mapping functionality with the following launch file :

```
export PAL_HOST=tiago
roslaunch tiago_2dnav_gazebo tiago_mapping.launch
```

The Gazebo window shown in figure 80 will open and the robot in an office like environment should be visible. Furthermore a Rviz window will open where the robot model, sensor and map being built will be visualized.

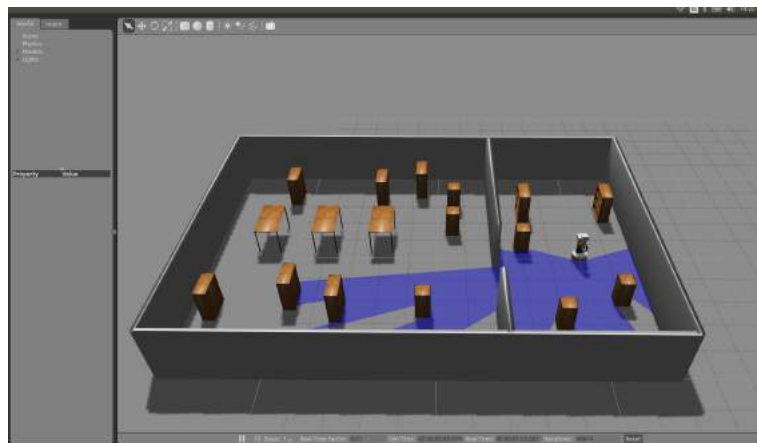


Figure 80: Small office world simulated in Gazebo

If you have a USB joystick plugged in your computer, it will be used to control the robot. Otherwise the following command could be run in a terminal to control the robot with the keyboard arrow keys:

```
roslaunch key_teleop key_teleop.py
```

While moving the robot around the environment, the map will start appear in Rviz, as the sequence of snapshots of Figure 81.

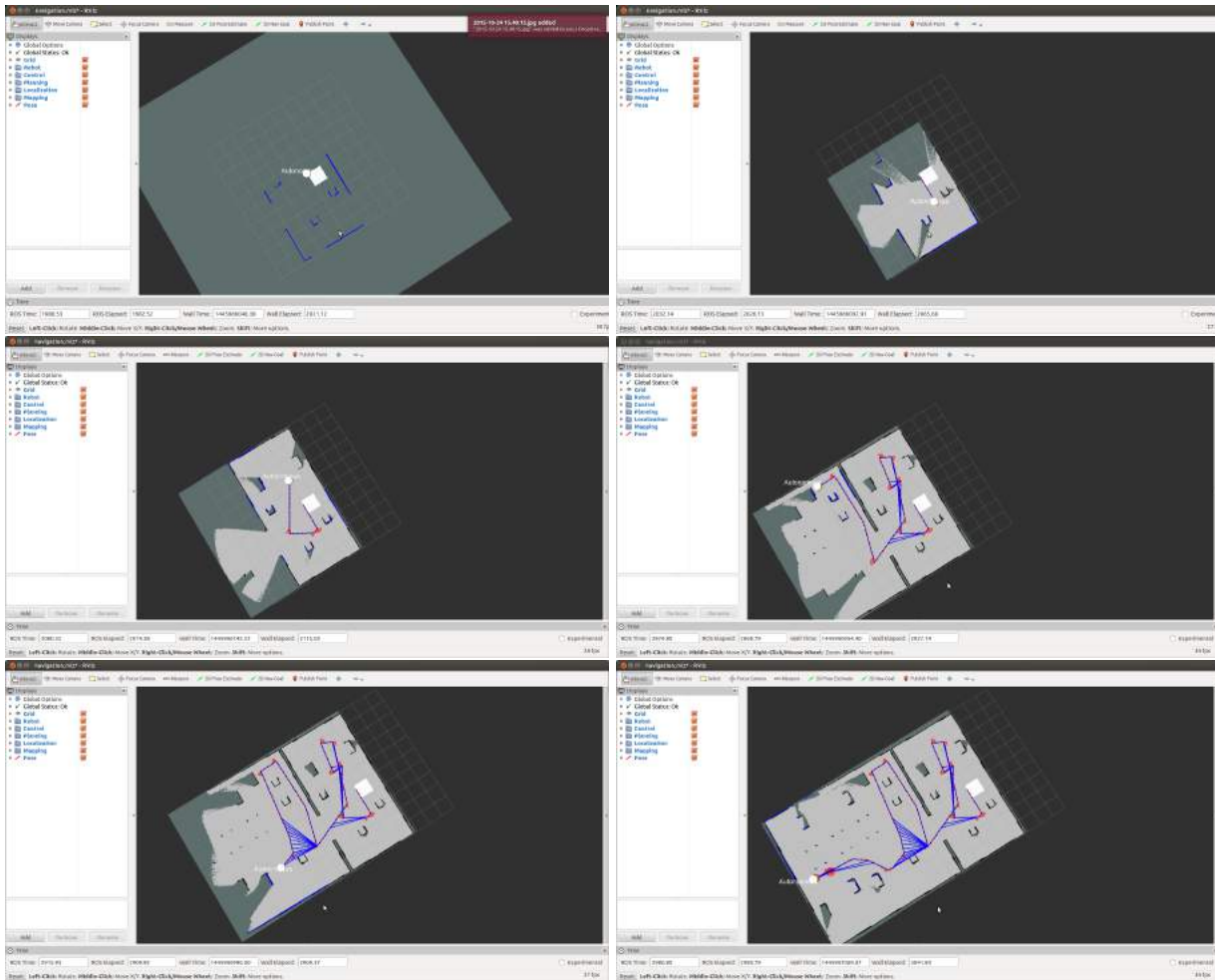


Figure 81: Partial maps built by the SLAM algorithm, starting (top left) and moving until a successful map is built (bottom right). The SLAM graph is represented by the nodes (red dots) and edges (blue segments).

25.4.2 Saving the map

The map can be saved manually as many times as wanted in the current directory by executing:

```
roslaunch map_server map_saver
```

This command saves two files (as shown in Figure 82): a *map.pgm* image file and a *map.yaml* configuration file. They both will be saved in the folder where the terminal is opened.

```

$ roslaunch map_server map_saver
[ INFO ] [1355921852.080625725]: Waiting for the map
[ INFO ] [1355921852.368149481]: Received a 467 X 529 map @ 0.050 m/pix
[ INFO ] [1355921852.368191868]: Writing map occupancy data to map.pgm
[ INFO ] [1355921852.378558406]: Writing map occupancy data to map.yaml
[ INFO ] [1355921852.378696099]: Done

```

Figure 82: Output of `map_saver` call

The *map.pgm* file will contain a graphic representation of the map built.

To save the map just built in the right path automatically, please use the `save_map` service :

```
rosservice call /pal_map_manager/save_map "directory: ''"
```



```
~$ rosservice call /pal_map_manager/save_map "directory: '"
success: True
name: 2015-10-27_185249
full_path: /home/luca/.pal/pmb2_maps/configurations/2015-10-27_185249
message: Map saved: 2015-10-27_185249
```

Figure 83: Output of map_saver call

The map will be named after the date and time at which it has been saved as in Figure 83

The save_map service will store all the map files in the path `$HOME/.pal/productname_maps/configurations`. The current map in use is the one pointed by the symbolic link `$HOME/.pal/productname_maps/config`.

Once mapping is finished, in order to save the map and start the localization mode, the following command should be used:

```
rosservice call /pal_navigation_sm "input: 'LOC'"
```

25.4.3 Localization and Path Planning

To run a simulation with the robot in localization and path planning mode this command should be run:

```
export PAL_HOST=tiago
roslaunch tiago_2dnav_gazebo tiago_navigation.launch
```

A Gazebo window will open, with the robot in the same office like environment, but this time the Rviz window shown in figure 84 will show the map, the localization particles, the robot model localized, the laser sensor readings and some virtual obstacles.

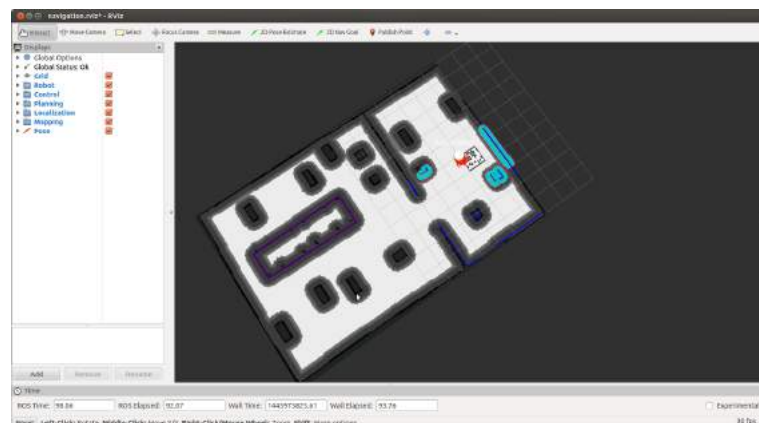


Figure 84: Small office world simulated navigation as visualized in Rviz just after running `roslaunch tiago_2dnav_gazebo tiago_navigation.launch`

To choose a goal, the user should select the “2D Nav Goal” tool in Rviz (clicking on it with left mouse button) as shown in figure 85 then click in the map to select the target location the robot has to reach.

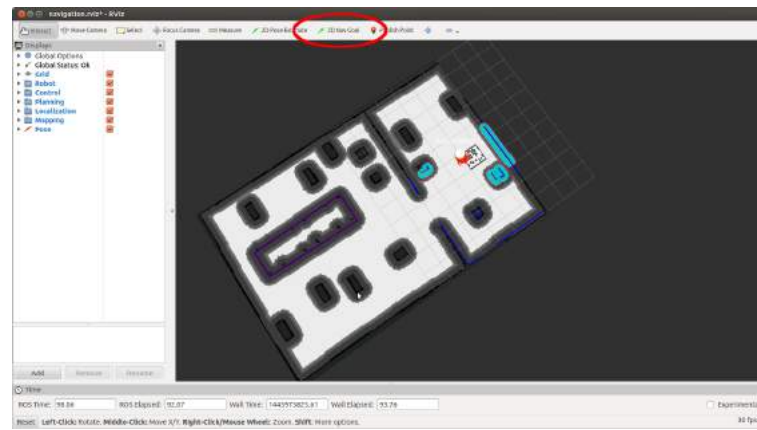


Figure 85: “2D Nav Goal” tool in Rviz should be clocked before selecting in the map the target location the robot has to reach.

The plan will be generated and the robot will start moving along the trajectory towards the goal as illustrated in the sequence of snapshots of figure 86.

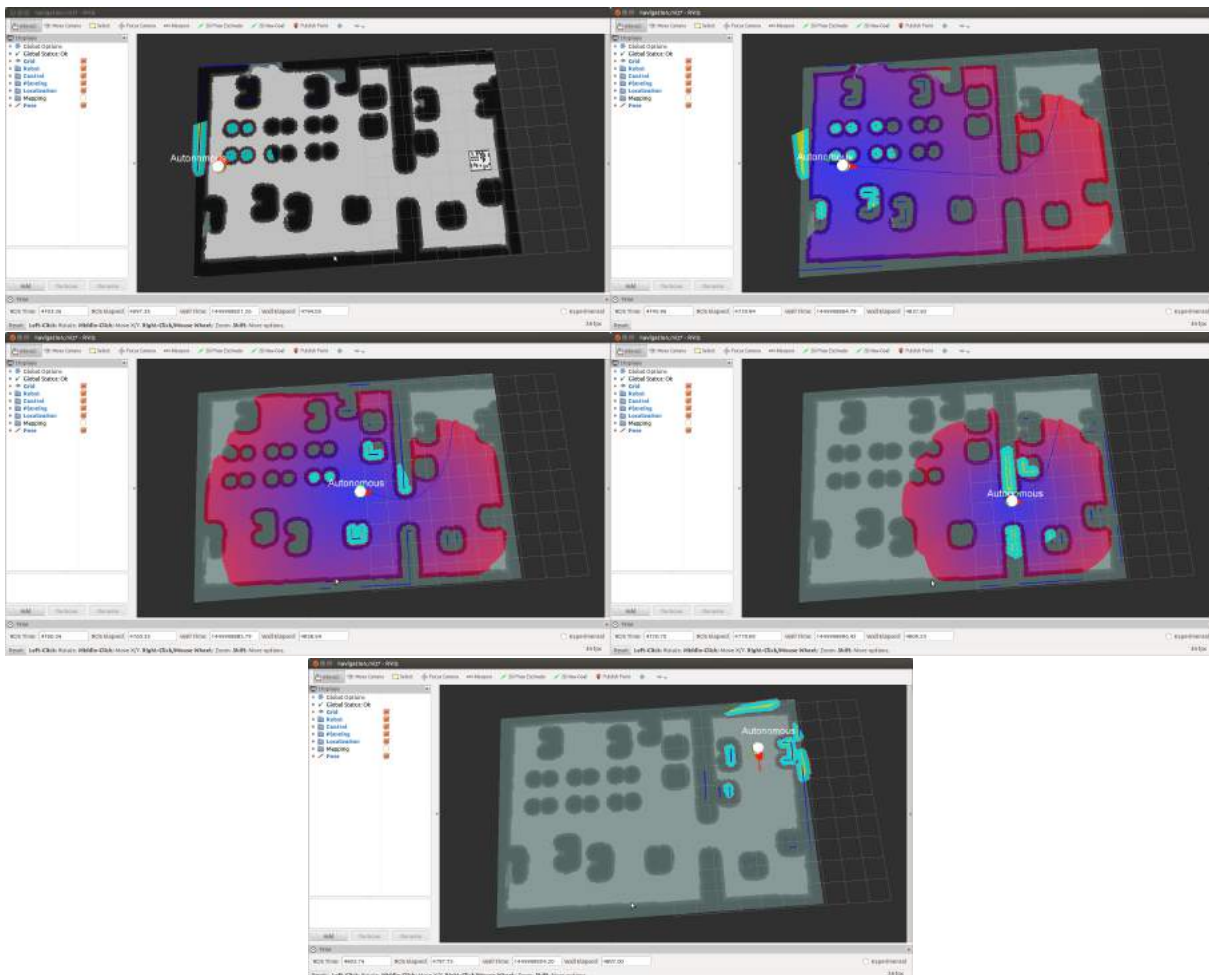


Figure 86: Autonomous navigation: the robot continuously localizes itself in the map, plans the path for reaching the target location and execute velocity commands for following the trajectory and avoiding obstacles. Costmaps and inflated obstacles are visible.

25.5 SLAM and path planning on the robot

When the robot boots all the mapping and localization nodes conforming the whole navigation pipeline are automatically started. Furthermore, the localization mode based on the last map is active.

The procedure for the real robot is almost the same, we just do not need the simulator, and the rviz visualization must be run separately in the development computer outside the robot.

On the development computer:

```
export ROS_MASTER_URI=http://tiago-0c:11311
rosservice call /pal_navigation_sm "input: 'MAP'"
```

For the rest, just follow the same steps described above for the simulation case.

In order to visualize the map in rviz from your development computer, the following command should be run:

```
export ROS_MASTER_URI=http://tiago-0c:11311
roslaunch rviz rviz -d `rospack find tiago_2dnav`/config/rviz/navigation.rviz
```

In order that rviz works properly make sure that the robot computer is able to resolve the development computer hostname as explained in section 9.2.

25.5.1 Saving the map on the robot

The map can be saved on the development computer in the current directory by executing from a terminal:

```
export ROS_MASTER_URI=http://tiago-0c:11311
roslaunch map_server map_saver
```

To save the map built in the robot, please use the save_map service :

```
export ROS_MASTER_URI=http://tiago-0c:11311
rosservice call /pal_map_manager/save_map "directory: ''"
```

The save_map service will store all the map files in the path `$HOME/.pal/tiago_maps/configurations` in the robot file system. The current map in use is the one pointed by the symbolic link `$HOME/.pal/tiago_maps/config`.

In order to automatically save the map, finish mapping and start using the map for localization and path planning, the following command should be used:

```
export ROS_MASTER_URI=http://tiago-0c:11311
rosservice call /pal_navigation_sm "input: 'LOC'"
```

25.5.2 Localization and Path Planning

Localization and path planning is automatically started in the robot during the boot up. A specific rviz configuration file for navigation is provided in `tiago_2dnav`. In order to launch rviz using this configuration file do as follows:

```
export ROS_MASTER_URI=http://tiago-0c:11311
roslaunch rviz rviz -d `rospack find tiago_2dnav`/config/rviz/navigation.rviz
```

To choose a goal, the user should select the "2D Nav Goal" tool in Rviz (clicking on it with left mouse button) as shown in figure 85 then click in the map to select the target location the robot has to reach.

The plan will be generated and the robot will start moving along the trajectory towards the goal as illustrated in the sequence of snapshots of figure 86.

If the robot does not move make sure that the robot computer is able to resolve the development computer hostname as explained in section 9.2.

Navigation goals can be send programatically to the Action Server `/move_base_simple`.

25.5.3 Change the active map on the robot

When the navigation is in localization mode, any of the maps stored in `$HOME/.pal/tiago_maps/configurations` can be selected. In order to select a map the following command can be used:

```
ssh pal@tiago-0c
rosservice call /pal_map_manager/change_map "input: 'MAP_NAME'"
```

where `MAP_NAME` is the name of the map that we want to select.

25.6 Map Editor

As seen in the previous sections with `rviz` we can easily visualize the mapping process. Then, in localization map we can send the robot to any point of the map and provide a new localization guess when necessary.

The PAL Map Editor³ is an `rviz` plugin providing advanced navigation functionalities to `rviz`:

- Downloading and uploading maps
- Renaming maps
- Changing the active map
- Definition of regions of interest
- Definition of POIs and groups of POIs
- Definition of virtual obstacles

Furthermore, a graphical joystick is provided in order to remotely teleoperate the mobile base of TIAGo.

In order to launch `rviz` with the Map Editor plugins do as follows:

```
roslaunch rviz rviz -d `rospack find tiago_2dnav`/config/advanced_navigation.rviz
```

The different plugins added to `rviz` are shown in figure 87.

³Software provided in the Advanced Navigation package

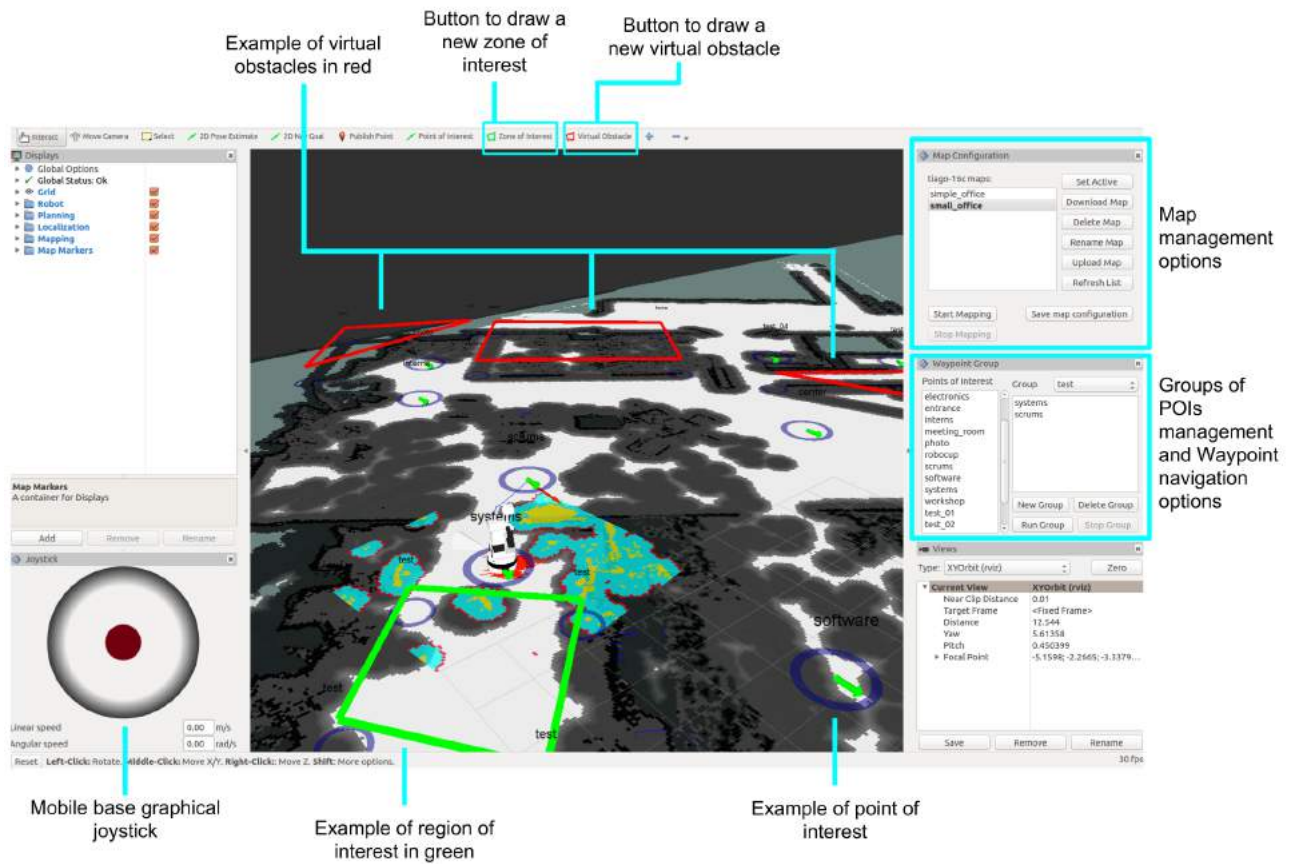


Figure 87: Rviz with the Map Editor plugins



Motion planning



26 Motion planning with *MoveIt!*

26.1 Overview

This section covers how to perform collision-free motions on TIAGo using the graphical interface of *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>.

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

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

1. Start a simulation:

```
roslaunch tiago_gazebo tiago_gazebo.launch robot:=steel world:=empty
```

2. Start *MoveIt!* with the GUI in another terminal:

```
roslaunch tiago_moveit_config moveit_rviz.launch config:=true
```

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 88 shows that by selecting different planning groups, the GUI shows only the relevant “flying end-effectors”.

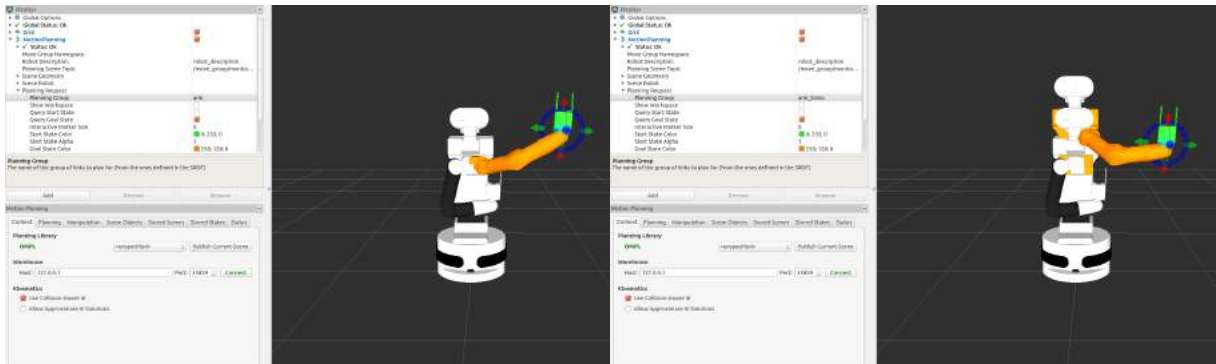


Figure 88: *MoveIt!* graphical interface in RViz. TIAGo with the planning group *arm* (left) and with the planning group *arm_torso* (right).

3. 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 to update Start State and Goal State depicted in figure 89.
4. By clicking the “Update” button of the goal state new poses can be randomized. Figure 90 shows a few random generated poses.
5. The sequence of images in figure 91 shows the result of a clicking “Update” goal pose with random in RViz, then clicking “Plan and Execute” in the simulator. RViz will visualize the plan before executing.
6. The GUI also allows to define goal poses using the “flying end-effector” method. As shown in figure 92, the 6DOF pose of the end-effector can be defined by using the visual marker attached to it. The red, green and blue arrows define the translation of x,y and z coordinates respectively as well as the colored rings define the rotation around these axes following the same logic.

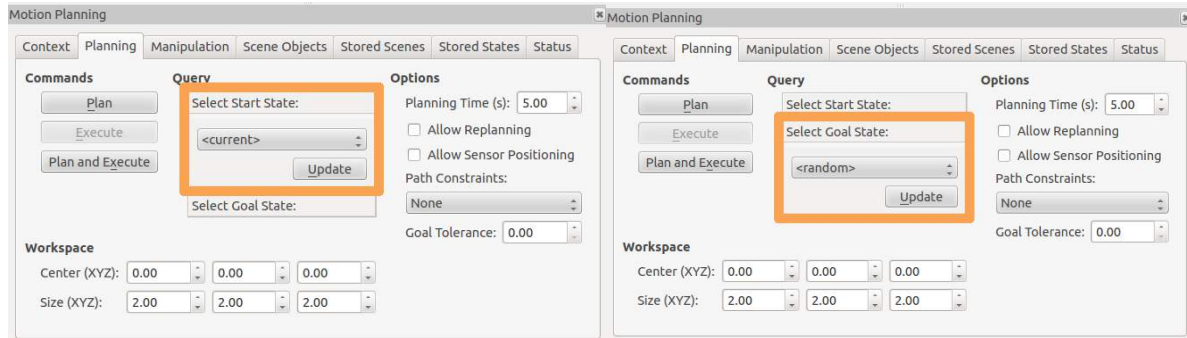


Figure 89: Tab to set start state (left) and tab to set goal state (right).

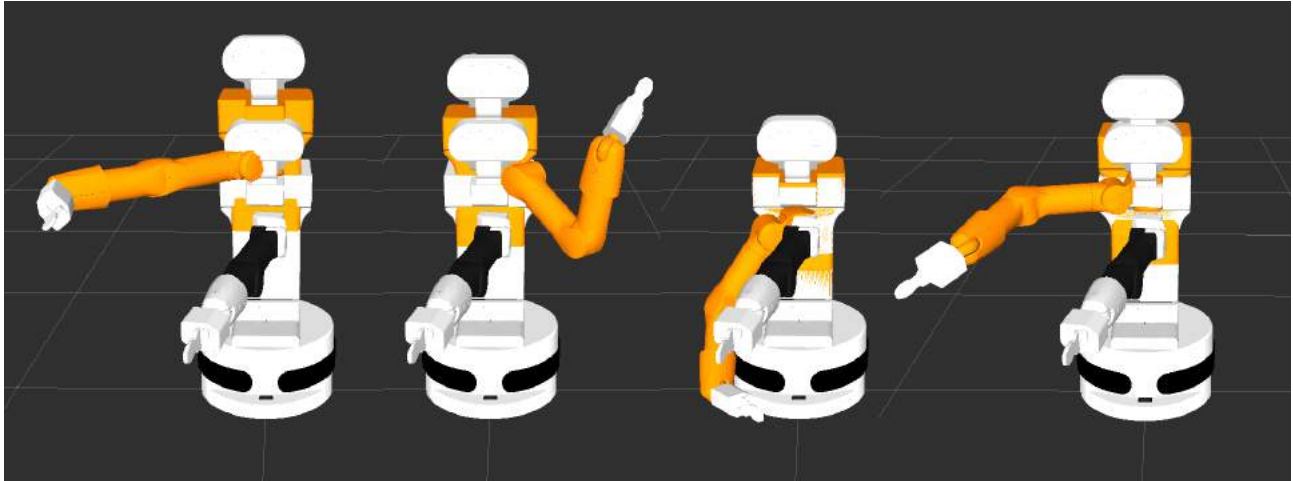


Figure 90: Random poses generated with the GUI

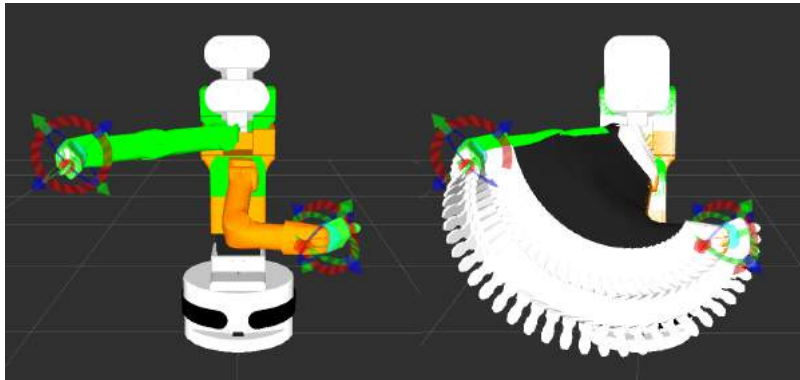


Figure 91: Sequence of plan

26.3 The planning environment of *MoveIt!*

For a complete description, see http://moveit.ros.org/wiki/Environment_Representation/Overview.

Adding a collision object to the motion planning environment will result in plans avoiding collisions with these objects. Such an operation can be done using the C++ or Python API, or with the *GUI presented in the previous subsection*. For this demonstration we are going to use the GUI.

1. To add such an object to the planning environment, navigate to the “*Scene objects*” tab in the planning environment window, then click “*Import file*” and navigate to the `tiago_description` package, where the mesh of the head can be found in the `mesheshead` folder as shown in figure 93.

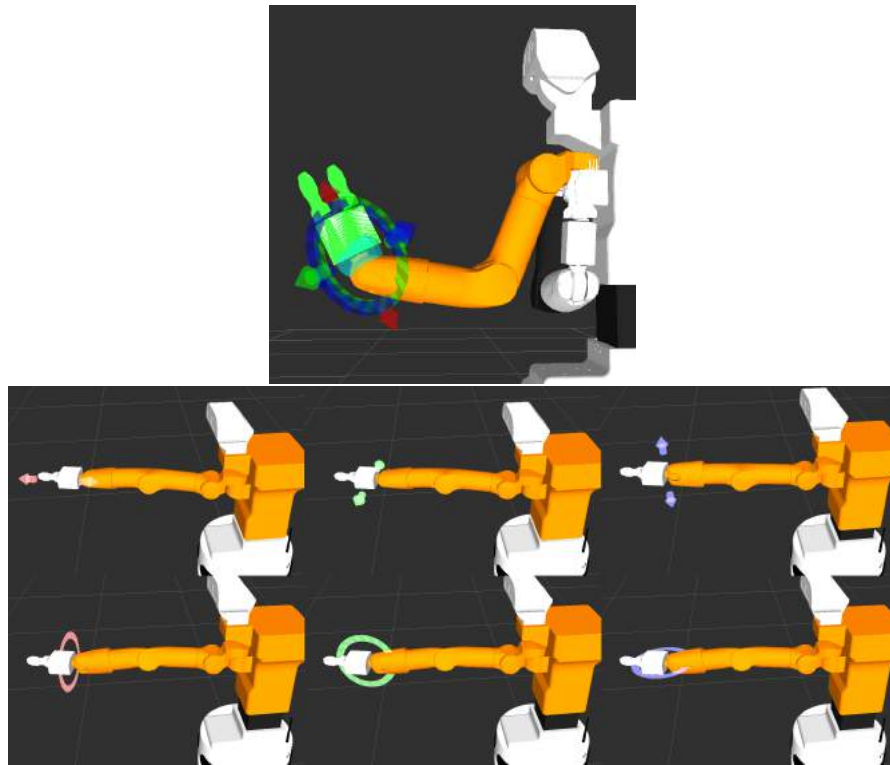


Figure 92: End effector tool (left) and the usage of each DOF (right)

2. Place the mesh close in front of the arm, then select the planning group “arm_torso.”
3. Move the end-effector using the “*flying end-effector*” to a goal position like the one shown at the right part of figure 94. Make sure that no body part is in red, which would mean that *Movel!* detected a collision in the goal position. Goals which result in a collision state will not be planned at all by *Movel!*.
4. (Optional) To execute such a motion, go the “*Context*” tab and click “*Publish current scene*”. This is step necessary so that the planning environment in the GUI will be moved to the one actually used by *Movel!* for planning. Go to the “*Planning*” tab again, and now move the arm to a position. Since this setup can be a very complex one for the planner, it might fail at the first attempt, keep planning until there is a successful plan. A successful plan for the current example is shown in figure 95.

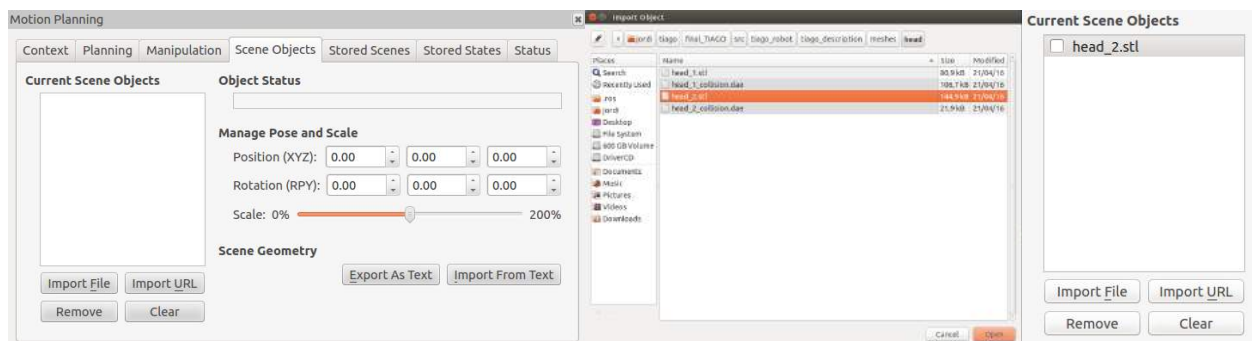


Figure 93: Scene objects tab (left), select the mesh (center), and the object listed in scene objects (right)

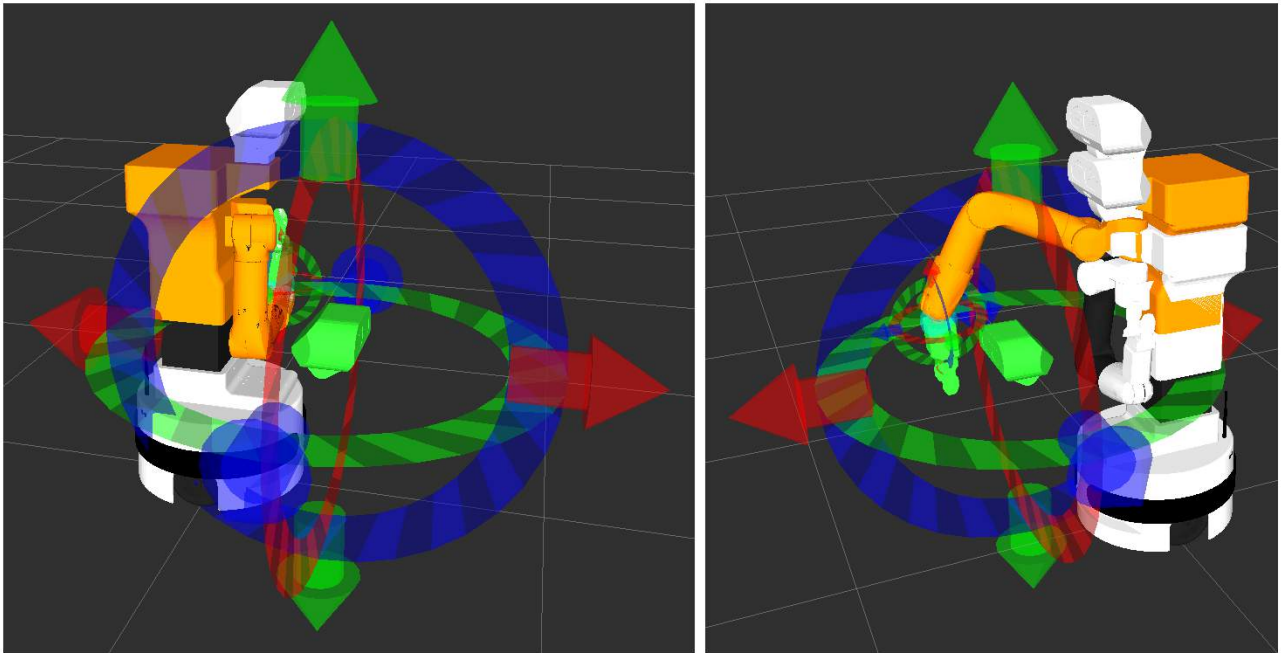


Figure 94: Example with an object in the scene. Initial pose (left) and goal pose (right).

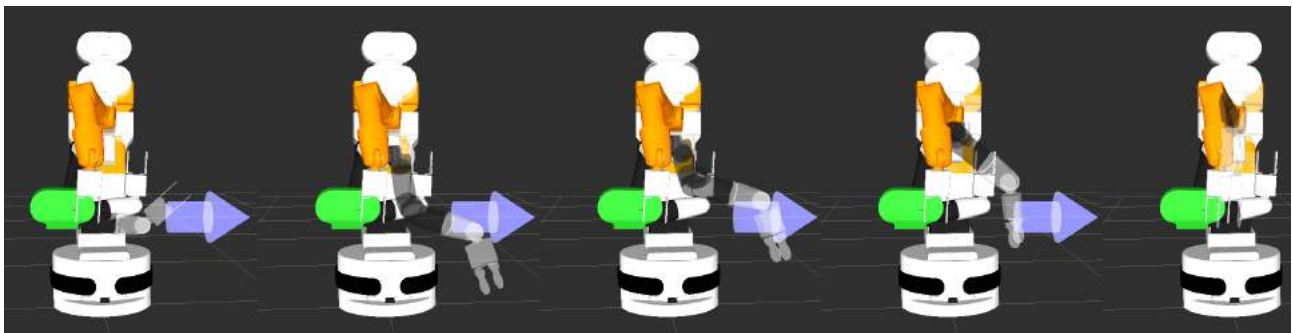


Figure 95: Example of plan found with the object in the scene.

26.4 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.

TIA Go

Whole Body Control

PAL
ROBOTICS 

27 Whole Body Control

27.1 Overview

The Whole Body Control (WBC)⁴ is a PAL's implementation of the Stack of Tasks⁵. It includes a hierarchical quadric solver running at 100 Hz able to accomplish different tasks with different priorities assigned to each of them. In order to accomplish the tasks the WBC takes control all the joints of TIAGo upper-body, i.e. torso, arm and head.

In the WBC package of TIAGo the following stack of tasks have been predefined (from highest priority to lowest priority):

- Self-collision avoidance: to prevent that the arm does not collide with any other part of the robot while moving.
- Joint limit avoidance: this task ensures that joint limits are never reached.
- Command the pose of `/arm_7_link` in cartesian space: this task allows sending the end-effector to any spatial pose.
- Command the pose `/xtion_optical_frame` link in cartesian space: this task allows making the robot look towards any direction.

The first two tasks are automatically managed by the WBC while the goals of the last two tasks are defined by the user. The user may request moving the end-effector to any spatial configuration and make the robot look to any spatial point and these goals can be changed dynamically.

The difference of using this configuration of WBC rather than other inverse kinematic solvers is that WBC finds online solutions preventing self-collisions and ensuring joint limit avoidance.

27.2 WBC upper-body teleoperation with leap motion

When the robot boots up it is ready to be teleoperated with a leap motion camera. In order to execute the demo it is necessary to use a development computer to connect the leap motion camera and run the software that will send teleoperation commands to the robot. First of all, in order that the ROS communication works properly between the robot and the development computer, it is necessary to set up the development computer hostname and IP address in the robot local DNS as explained in Section 9.2.

Then, in the development computer open a terminal and run the following software:

```
export ROS_MASTER_URI=http://tiago-0c:11311
source /opt/LeapSDK/leap_setup.bash
roslaunch wbc_state_machines leap_motion_sm_separated_tf.launch
```

Wait until the robot moves to the initial pose shown in figure 96. Then, wait until the terminal in the development computer shows the following messages:

```
No /right_hand frame...
No /right_hand frame...
No /right_hand frame...
No /right_hand frame...
No /right_hand frame...
```

At this point the user can place his/her opened hand above the leap motion camera as show in figure 96b. The terminal will start printing messages like the following ones:

⁴Included in the corresponding Premium software package.

⁵N. Mansard, O. Stasse, P. Evrard, A. Kheddar. A versatile generalized inverted kinematics implementation for collaborative working humanoid robot: the stack of tasks. ICAR 2009.

```
[INFO] [WallTime: 1472541721.499519] Sending gripper goal: header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: ''
joint_names: ['gripper_left_finger_joint', 'gripper_right_finger_joint']
points:
-
  positions: [0.0017199941490628687, 0.0017199941490628687]
  velocities: []
  accelerations: []
  effort: []
  time_from_start:
    secs: 0
    nsecs: 200000000
```

These messages appear when the user hand is properly tracked and teleoperation commands are being sent to the robot WBC. You should see the robot arm, torso and end-effector moving according to the user hand movements. The leap motion camera is able to track the human hand at several heights, closing and opening it. Even if rotations of the hand are pretty well tracked the leap motion tracking works better by keeping the hand as flat as possible.

In order to stop the demo press CTRL+C in the terminal. The robot will move back to the initial pose.

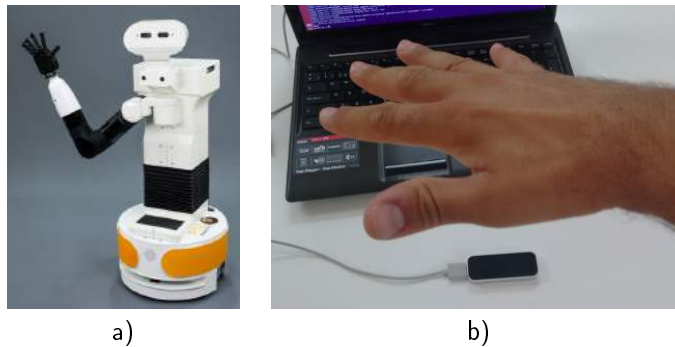


Figure 96: a) Robot initial pose of wbc teleoperation demo. b) Operator hand pose

Finally, in order to set the control mode back to position control you may press the Switch to POSITION in Demos of the WebCommander, see figure 97, or run in the robot the following call:

```
rosservice call /controller_manager/switch_controller "start_controllers:
- 'head_controller'
- 'arm_controller'
- 'torso_controller'
stop_controllers:
- 'whole_body_kinematic_controller'
strictness: 0"
```

27.3 WBC upper-body teleoperation with joystick

WBC provides a simple way to command the end-effector pose using the joystick. The torso lift and arm joints will be controlled in order to move and rotate the end-effector in cartesian space. In order to start using this modality first connect to the robot through a terminal:

```
ssh pal@tiago-0c
```

Then start the required node as follows:

```
pal-start joystick_arm_pose
```

The arm of the robot will extend and once this motion is over the joystick will be able to command the pose of the arm. In order to the teleoperation the user has to gain the joystick priority by pressing the button START

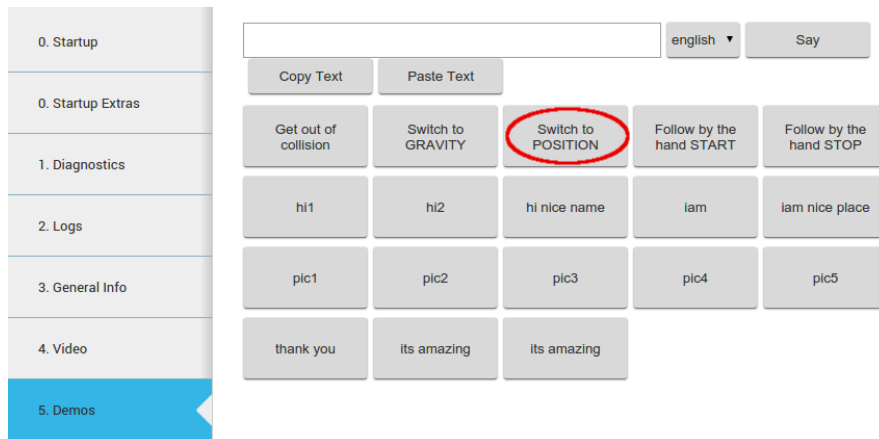


Figure 97: Button to restore position control



Figure 98: Button to gain control of the arm

as shown in figure 98. Note that pressing this button we gain control of the mobile base and the upper body alternatively. Press once or twice until the mobile is no longer controlled with the joystick.

The user can control the position of the end-effector with respect to the `base_footprint` frame, which lies at the center of the mobile base of the robot as shown in figure 99.



Figure 99: Reference frame of the joystick teleoperation demo. X axis in red, Y axis in green and Z axis in blue.

The following motions can be controlled with the joystick:

- Move the end-effector along the X axis of the reference frame: press the left analog stick as shown in figure 100a. Pressing upwards will move the end-effector forward; pressing downwards will move the end-effector backward.
- Move the end-effector along the Y axis of the reference frame: press the left analog stick as shown in

figure 100b. By pressing this joypad leftwards or rightwards the end-effector will move laterally in the same sense.

- Move the end-effector along the Z axis of the reference frame: press the right analog stick as shown in figure 100c. Pressing upwards will move the end-effector upwards; pressing downwards will move the end-effector downwards.
- Rotate the end-effector along the Y axis of the reference frame: press the left pad as shown in figure 101a.
- Rotate the end-effector along the Z axis of the reference frame: press the left pad as shown in figure 101b.

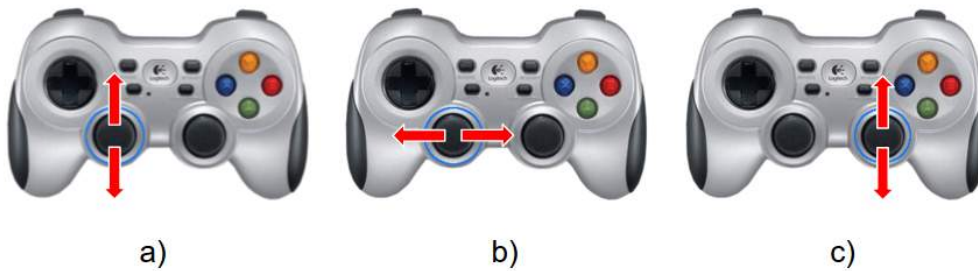


Figure 100: Buttons to move the end-effector in cartesian space



Figure 101: Buttons to rotate the end-effector in cartesian space

In order to stop the demo run the following command in the robot:

```
pal-stop joystick_arm_pose
```

Then, to restore the arm, head and torso position controllers and stop the WBC press the Switch to POSITION in Demos of the WebCommander, as shown in figure 97 or run the following service call in the robot:

```
rosservice call /controller_manager/switch_controller "start_controllers:
- 'head_controller'
- 'arm_controller'
- 'torso_controller'
stop_controllers:
- 'whole_body_kinematic_controller'
strictness: 0"
```

27.4 WBC with Rviz interactive markers

A way to test the WBC capabilities is using rviz and move the head and arm of the robot by moving interactive markers around.

We need to open a terminal from the development computer and connect to the robot:

```
ssh pal@tiago-0c
```

Once connected to the robot the first thing is to stop several controllers:

```
rosservice call /controller_manager/switch_controller "start_controllers:
- ''
stop_controllers:
- 'head_controller'
- 'arm_controller'
- 'torso_controller'
- 'whole_body_kinematic_controler'
strictness: 0"
```

and then unload the WBC loaded by default when powering up the robot as follows:

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

Now make sure that there are no obstacles in front of the robot as the arm will be first extended when the WBC is launched as follows:

```
roslaunch tiago_wbc tiago_wbc.launch source_data:=interactive_marker
```

Then, open a new terminal in the development computer and run rviz as follows:

```
source /opt/pal/dubnium/setup.bash
export ROS_MASTER_URI=http://tiago-0c:11311
roslaunch rviz rviz -d `rospack find tiago_wbc`/config/rviz/tiago_wbc.rviz
```

Rviz will show up as in figure 102 and the user will be able to command the end-effector and the head of TIAGo by dragging and rotating the two interactive markers that will appear.

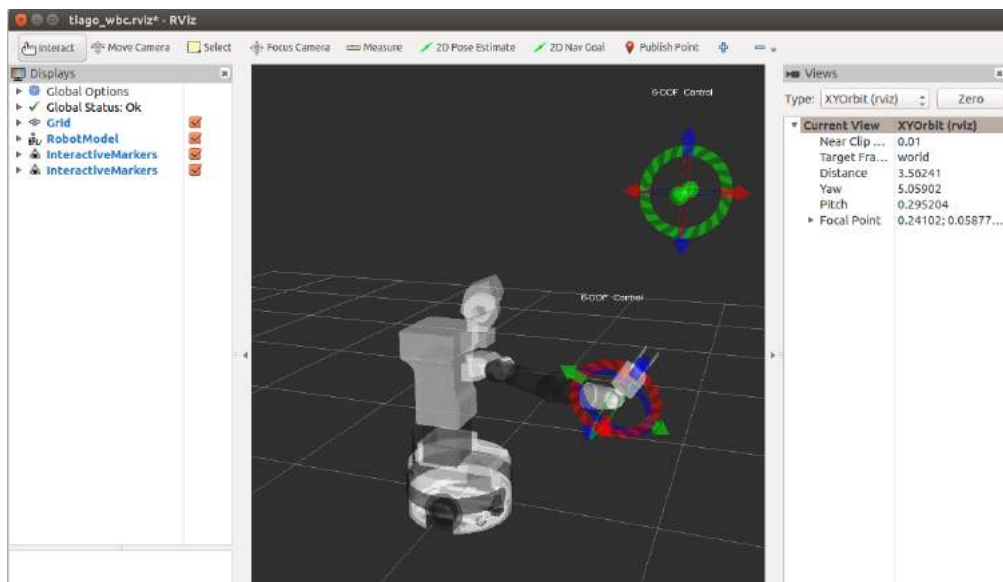


Figure 102: WBC demo with rviz

In order to stop the demo rviz needs to be closed and back to the terminal where the WBC was launched we need to kill it by pressing CTRL+C. Then the following commands need to be executed in the same terminal in order to restore the state of the robot:

```
rosservice call /controller_manager/switch_controller "start_controllers:
- 'head_controller'
- 'arm_controller'
- 'torso_controller'
stop_controllers:
- 'whole_body_kinematic_controller'
strictness: 0"
```

The current WBC needs to be unloaded:

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

And the default WBC is loaded as follows:

```
roslaunch tiago_controller_configuration whole_body_controller.launch
```

27.5 WBC through ROS topics interface

When creating new functionalities in TIAGo involving WBC the user may require sending goals for the arm and head control programmatically. In order to so first the WBC needs to be started as explained in the following steps.

Connect to the robot through a terminal:

```
ssh pal@tiago-0c
```

Stop several controllers:

```
rosservice call /controller_manager/switch_controller "start_controllers:
- ''
stop_controllers:
- 'head_controller'
- 'arm_controller'
- 'torso_controller'
- 'whole_body_kinematic_controller'
strictness: 0"
```

Unload the current WBC:

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

Now make sure that there are no obstacles in front of the robot as the arm will be first extended when the WBC is launched as follows:

```
roslaunch tiago_wbc tiago_wbc.launch source_data:=topic
```

There are now the following ROS topics:

/whole_body_kinematic_controller/arm_7_link_goal ([geometry_msgs/PoseStamped](#))

Topic to specify the goal pose of the arm end link.

/whole_body_kinematic_controller/gaze_objective_xtion_optical_frame ([geometry_msgs/PoseStamped](#))

Topic to specify the spatial point that the camera of the head has to look at.

Example to send a goal for the arm A goal can be sent through command line, for instance, as follows:

```
rostopic pub /whole_body_kinematic_controler/arm_7_link_goal geometry_msgs/PoseStamped "
header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: '/base_footprint'
pose:
  position:
    x: 1.0
    y: 0.0
    z: 0.5
  orientation:
    x: 0.0
    y: 1.0
    z: 0.0
    w: 0.0"
```

When running this example you will notice how the WBC moves the torso and arm in order to set `/arm_7_link` to the desired pose, which is bringing the origin of this frame to the point (1,0,0.5) expressed in `/base_footprint`, which is a point that lies 1 m in front of the robot and 0.5 m above the floor. Notice that the desired orientation of `/arm_7_link` is also specified using a quaternion.

Example to send a goal for the head A goal can be sent through command line, for instance, as follows:

```
rostopic pub /whole_body_kinematic_controler/gaze_objective_xtion_optical_frame_goal \
geometry_msgs/PoseStamped "
header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: '/base_footprint'
pose:
  position:
    x: 1.0
    y: 0.0
    z: 0.5
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 1.0"
```

With this goal the head of the robot will move so that the camera looks towards the point at which the arm had been sent with the previous example.

Stopping the WBC In order to stop the WBC and enable back the torso, arm and head controllers do as follows:

```
rosservice call /controller_manager/switch_controller "start_controllers:
- 'head_controller'
- 'arm_controller'
- 'torso_controller'
stop_controllers:
- 'whole_body_kinematic_controler'
strictness: 0"
```

The current WBC needs to be unloaded:

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

And the default WBC is loaded as follows:

```
roslaunch tiago_controller_configuration whole_body_controller.launch
```

TIA Go

Simulation



28 Simulation

28.1 Overview

When installing a development computer as explained in section 7.3 the user can run Gazebo simulations of the specific TIAGo robot acquired.

Three different simulation worlds are provided with TIAGo, which are described in the following subsections.

28.1.1 Empty world

This is the default world loaded when launching the simulation. The robot is spawned in an empty world with no more objects as shown in figure 103. In order to launch the simulation the following instruction needs to be executed in a terminal:

```
source /opt/pal/dubnium/setup.bash
roslaunch tiago_gazebo tiago_gazebo.launch
```

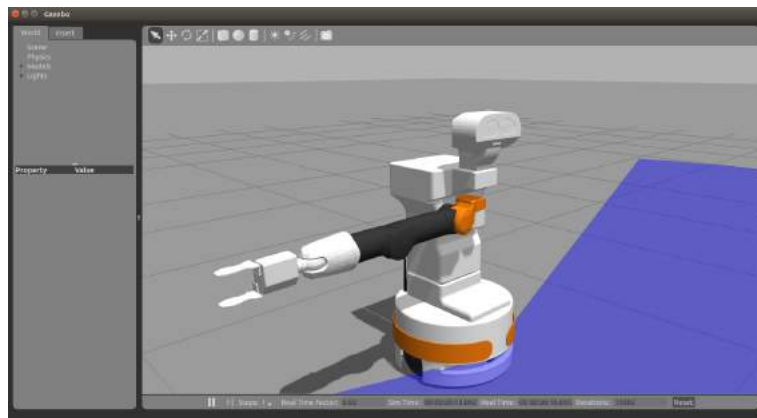


Figure 103: Empty world simulated in Gazebo

28.1.2 Office world

The simple office shown in figure 104 can be simulated with the following instruction:

```
source /opt/pal/dubnium/setup.bash
roslaunch tiago_gazebo tiago_gazebo.launch world:=small_office
```

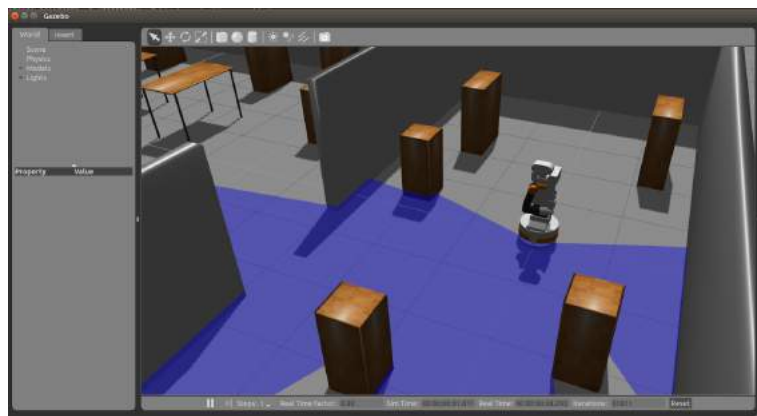


Figure 104: Small office world simulated in Gazebo

28.1.3 Table with objects world

In this simulation TIAGo is spawned in front of a table with several objects on its top, see figure 105. The instruction needed is:

```
source /opt/pal/dubnium/setup.bash
roslaunch tiago_gazebo tiago_gazebo.launch world:=objects_on_table
```

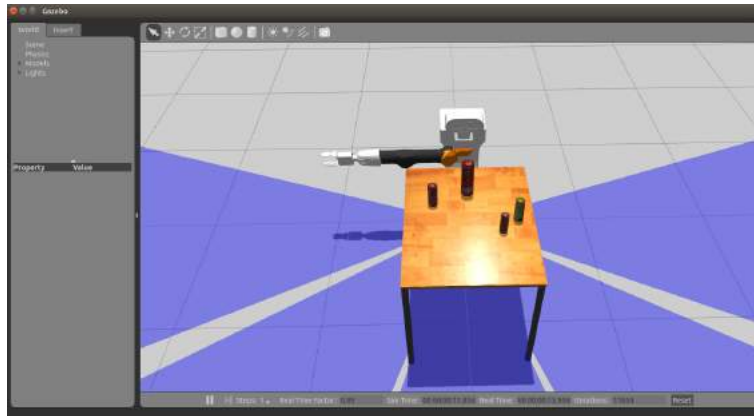


Figure 105: Tabletop scenario simulation

28.2 Grasping demo in simulation

This demo shows TIAGo grasping one of the cans on top of the table by using a sequence of pre-defined upper body movements. In order to run the demo do as follows:

```
source /opt/pal/dubnium/setup.bash
roslaunch tiago_gazebo tiago_gazebo_grasp_demo.launch
```

The sequence of motions is shown in figure 106.

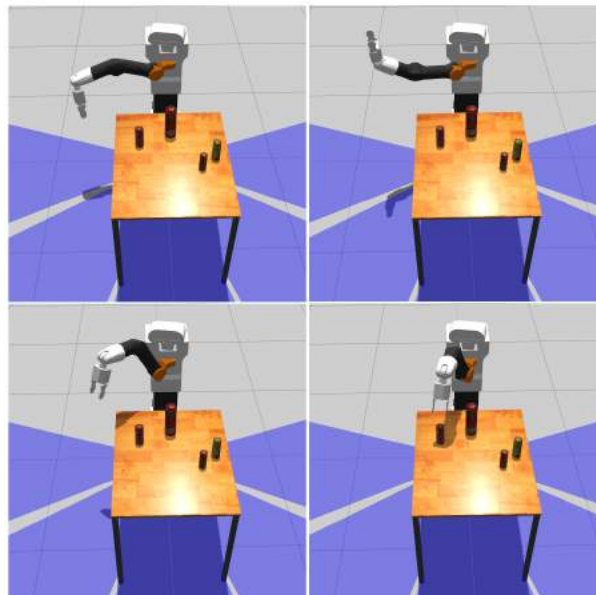


Figure 106: Grasping demo in simulation

TIA Go

LEDS

PAL
ROBOTICS



29 LEDs

This section contains an overview of the LEDs included in the TIAGo robot.

29.1 ROS API

NOTE: The services provided for controlling the LEDs are launched by default on startup.

Keep in mind there is an application running in the TIAGo robot that changes the LEDs strips according to the robot's speed. This application must be stopped if any operation with the LED strips is to be performed. It can be done by calling the `pal-stop leds_app` script.

```
pal@tiago-0c:~$ pal-stop leds_app
result: leds_app stopped successfully
```

29.1.1 Available services

The following services are used for controlling the LED strips present in TIAGo robot. All of them require a *port* argument that specifies which strip the command will affect. A *port* of value 0 refers to the left strip and 1 to the right strip. LED strips are composed of pixel LEDs.

NOTE: Resulting color may be different or not visible at all due to the plastic that covers the LED strips.

- `/mm11/led/set_strip_color port r g b`

Sets all the pixels in the strip to a color. *r*, *g* and *b* refers to the color to set in RGB scale.

```
pal@tiago-0c:~$ rosservice call /mm11/led/set_strip_color 0 255 255 255
```

- `/mm11/led/set_strip_pixel_color port pixel r g b`

Sets one pixel in the strip to a color. *pixel* is the position of the LED in the strip. *r*, *g* and *b* refers to the color to set in RGB scale.

```
pal@tiago-0c:~$ rosservice call /mm11/led/set_strip_pixel_color 0 5 255 0 0
```

- `/mm11/led/set_strip_flash port time period r_1 g_1 b_1 r_2 g_2 b_2`

Sets a flashing effect to the LED strip. *time* is the duration of the flash in milliseconds. *period* is the time between flashes in milliseconds. *r_1*, *g_1* and *b_1* refers to the color of the flash in RGB scale. *r_2*, *g_2* and *b_2* refers to the background color of the flash in RGB scale.

```
pal@tiago-0c:~$ rosservice call /mm11/led/set_strip_flash 0 100 1000 255 0 0 0 0 255
```

- `/mm11/led/set_strip_animation port animation_id param_1 param_2 r_1 g_1 b_1 r_2 g_2 b_2`

Sets an animation effect to the LED strip. *animation_id* sets the type of animation: 1 for pixels running left, 2 for pixels running right, 3 for pixels running forth and back starting from the left and 4 for pixels running forth and back starting from the right. *param_1* is the time between effects in milliseconds. *param_2* is the distance between animated pixels. *r_1*, *g_1* and *b_1* refers to the color of the animation in RGB scale. *r_2*, *g_2* and *b_2* refers to the background color of the animation in RGB scale.

```
pal@tiago-0c:~$ rosservice call /mm11/led/set_strip_animation 0 1 100 5 250 0 0 0 0 255
```


TIA Go

Tutorials



30 Tutorials

30.1 Overview

This section explains how to retrieve and build the public tutorials for TIAGo from PAL Robotics's GitHub repositories.

30.2 Download source code

First of all create an empty workspace:

```
mkdir ~/tutorials_ws
cd ~/tutorials_ws
mkdir src
cd src
```

Then clone the following repositories:

```
git clone https://github.com/pal-robotics/pal_msgs.git
git clone https://github.com/pal-robotics/tiago_tutorials.git
```

Then the workspace is ready to build:

```
cd ~/tutorials_ws
source /opt/pal/dubnium/setup.bash
catkin build
```

Then, the following tutorials are ready to be executed.

30.2.1 Look_hand

This package includes a Python script that shows how to retrieve a frame transformation from TF and then send a goal to the `point_head_action` server to move the head so that the camera looks to the last arm frame.

30.2.2 Look_to_point

This package provides a C++ node that shows live image from TIAGo camera and when the user clicks with the mouse in some pixel, the robot looks towards that direction. The examples shows how to use useful ROS packages like `image_transport`, `cv_bridge` and `actionlib`.

30.2.3 Play_with_sensors

Python examples are included in this package to get data from the different sensors in TIAGo. Concretely, the following nodes are included:

- `current_play.py`: prints the total current consumed by the motors.
- `force_torque_play.py`: example on how to retrieve data from the force/torque sensor on the wrist.
- `imu_play.py`: retrieves data from the IMU in the mobile base.
- `laser_play.py`: example on how to subscribe to the laser scanner of the mobile base.
- `rgbd_play.py`: node that subscribes to the data from the RGBD camera of the head.

30.2.4 Run_motion

C++ and Python examples on how to trigger a pre-defined upper body motion using `play_motion` action server.

30.2.5 Say_something

Python action client that shows how to synthesize a sentence through the `tts` action server.

- `play_with_sensors`
- `run_motion`
- `say_something`

TIA Go

Demos



31 Demos

31.1 Learning-by-demonstration

This demo shows how to make the robot learn arm movements by demonstration.

31.2 Download source code and build demo

First of all create an empty workspace in a development computer:

```
mkdir ~/learning_ws
cd ~/learning_ws
mkdir src
cd src
```

Then clone the following repositories:

```
git clone https://github.com/pal-robotics/simple_learning.git
git clone https://github.com/pal-robotics/learning_gui.git
```

First deploy the `simple_learning` package to the robot as follows:

```
cd ~/learning_ws
roslaunch pal_deploy deploy.py --user pal --package simple_learning tiago-0c
```

When the package has been successfully deployed to the robot the local workspace can be built:

```
catkin build
```

and finally the packages in the workspace need to be include in `ROS_PACKAGE_PATH` by doing:

```
source ../devel/setup.bash
```

31.3 Run the demo

In order to run the demo the node in the package `simple_learning` deployed to the robot needs to be started:

```
ssh pal@tiago-0c
roslaunch simple_learning simple_learning_server.launch
```

Then, in the development computer the following GUI can be started:

```
export ROS_MASTER_URI=http://tiago-0c:11311
roslaunch learning_gui learning_gui.py
```

The GUI in figure 107 will show up.

The steps required to learn a motion are detailed below.

31.3.1 Select joints

Click on the checkboxes at the right of the GUI to select the joints that will be involved in the motion. Note that there are several groups of joints which are coupled, i.e. when selecting one the rest of the joints of the group are automatically selected. The groups are:

- Head: 2 joints
- Arm: 7 joints
- Torso: 1 joint
- Hand / Gripper: 3 / 2 joints respectively

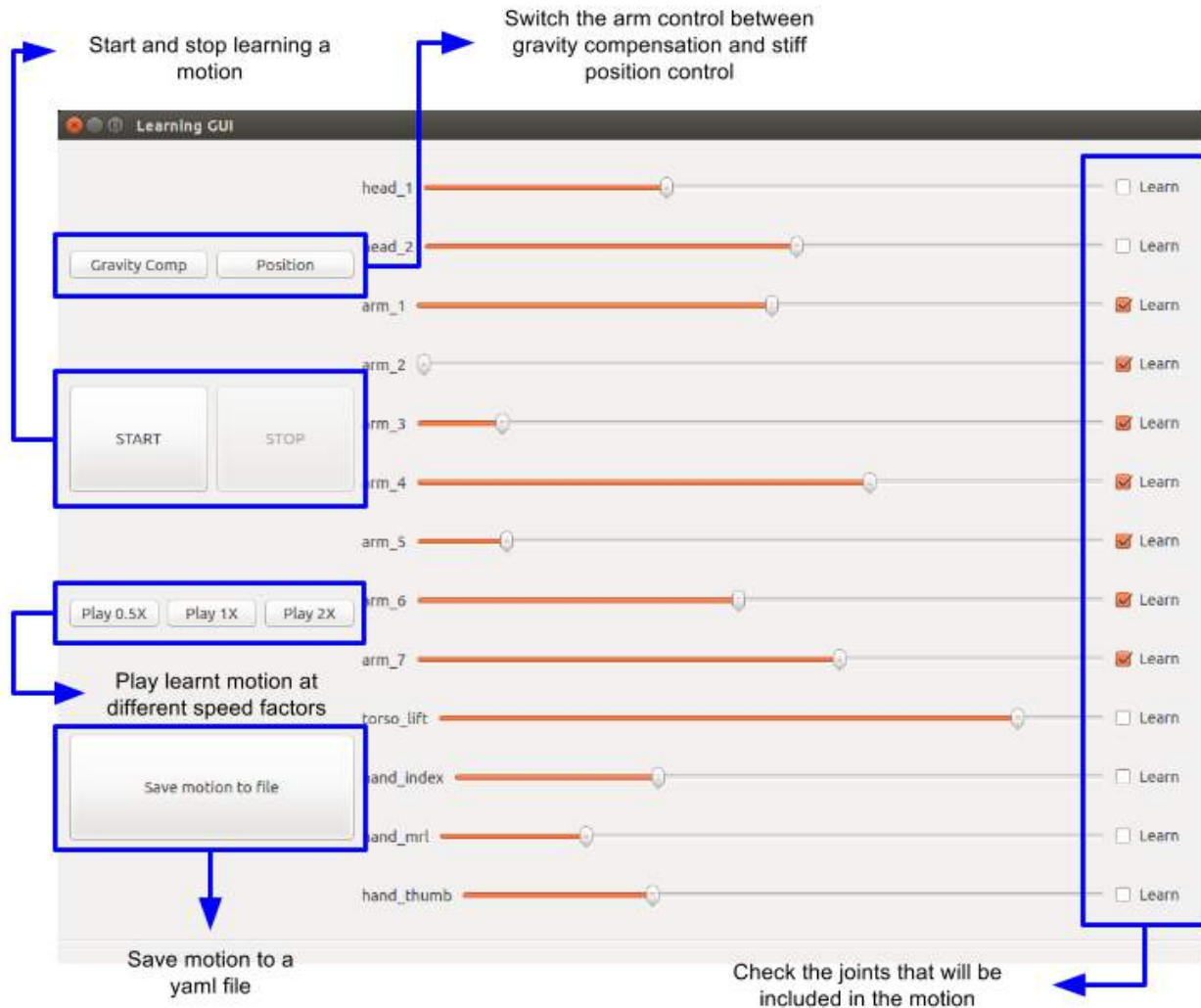


Figure 107: GUI to assist the user in the learning-by-demonstration demo

31.3.2 Start learning

Click on the **START** button to start learning a new motion.

31.3.3 Moving the joints

There are two ways to move the joints:

- Moving by a demonstrator:** in this case the user can physically push and pull the arm of the robot by first pressing the button **Gravity Comp** at the upper-left part of the GUI. TIAGO has current control in the first 4 joints of its arm so that the user can apply forces to the arm up to the 4th DoF. The best way to move the arm in this mode is to push/pull the different links of the arm up to the wrist like in figure 108. It is important not to apply forces to the 3 DoF of the wrist as they are always controlled in stiff position mode. Forcing the joints of the wrist in order to make it turn may cause damage to its mechanisms.



Figure 108: Arm movements taught by a demonstrator

- **Moving with sliders:** all the joints can be moved using the sliders in the GUI. In order to move the arm joints with the sliders first press the `Position` button at the upper-left part of the GUI. The rest of joints are always controlled in stiff position mode. Note that if the head joints need to be included in the learning it is convenient to disable first the `head_manager` node through the WebCommander to disable autonomous motions.

31.3.4 Stop the learning

Press the `STOP` button in order that the taught motion is properly learnt by the learning server.

31.3.5 Testing the new motion

If the learning process succeeds three motions are automatically uploaded in the ROS param server under the namespace `/play_motion/motions`:

- `LBD_HALFX`: learnt motion played at half the speed
- `LBD_1X`: original learnt motion
- `LBD_2X`: learnt motion played at double speed

In order to test the motions the three buttons in the GUI can be pressed. It is wise to test first the `LBD_HALFX` motion for safety.

Note that these motions are stored in the ROS param server and will be lost after a reboot of the robot computer. In order to store the learnt motion press the `Save motion to file` button. In the pop up window choose the destination folder and a namefile with `.yaml` extension. The stored file looks like in figure 109.

Note that editing the file the name of the motion can be changed by modifying the `LBD_1X` word highlighted in yellow.

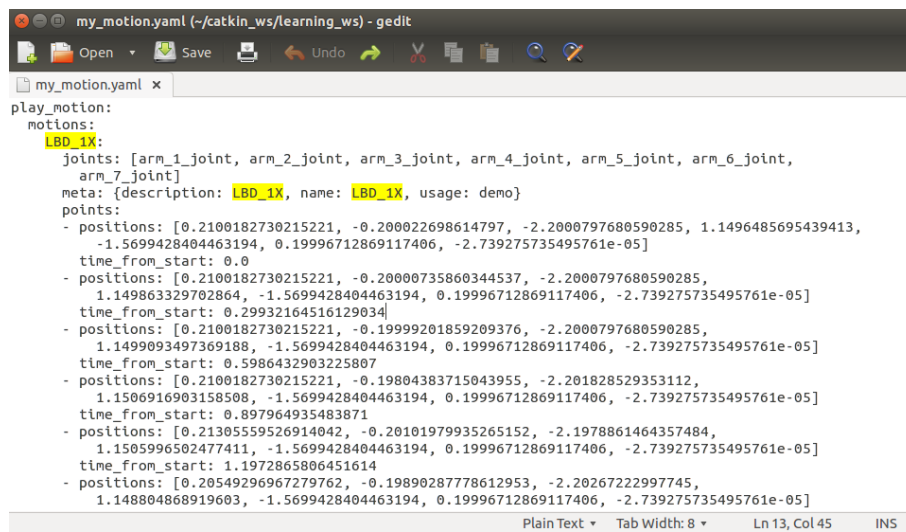
As example, let us assume that we store the motion as `MyMotion` in a file named `my_motion.yaml`. Then, in order to upload the new motion to the ROS param server of the robot so that it is available in the upper body motions library, see 24, the user can do:

```
rosparam load my_motion.yaml
```

The motion file can be copied to the robot or be kept in the development computer. In the latter case, before executing the above command remember to point to the robot's master:

```
export ROS_MASTER_URI=http://tiago-0c:11311
```

Note that the new motion will appear in the WebCommander `Movements` tab after refreshing the browser as shown in figure 110.



```

my_motion.yaml (~/.catkin_ws/learning_ws) - gedit
play_motion:
  motions:
    LBD_1X:
      joints: [arm_1_joint, arm_2_joint, arm_3_joint, arm_4_joint, arm_5_joint, arm_6_joint,
               arm_7_joint]
      meta: {description: LBD_1X, name: LBD_1X, usage: demo}
      points:
        - positions: [0.2100182730215221, -0.200022698614797, -2.2000797680590285, 1.1496485695439413,
                      -1.5699428404463194, 0.19996712869117406, -2.739275735495761e-05]
                      time_from_start: 0.0
        - positions: [0.2100182730215221, -0.2000735860344537, -2.2000797680590285,
                      1.149863329702864, -1.5699428404463194, 0.19996712869117406, -2.739275735495761e-05]
                      time_from_start: 0.29932164516129034
        - positions: [0.2100182730215221, -0.19999201859209376, -2.2000797680590285,
                      1.1499093497369188, -1.5699428404463194, 0.19996712869117406, -2.739275735495761e-05]
                      time_from_start: 0.5986432903225807
        - positions: [0.2100182730215221, -0.19804383715043955, -2.201828529353112,
                      1.1506916903158508, -1.5699428404463194, 0.19996712869117406, -2.739275735495761e-05]
                      time_from_start: 0.897964935483871
        - positions: [0.21305559526914042, -0.20101979935265152, -2.1978861464357484,
                      1.1505996502477411, -1.5699428404463194, 0.19996712869117406, -2.739275735495761e-05]
                      time_from_start: 1.1972865806451614
        - positions: [0.20549296967279762, -0.19890287778612953, -2.20267222997745,
                      1.148804868919603, -1.5699428404463194, 0.19996712869117406, -2.739275735495761e-05]

```

Figure 109: Arm movements taught by a demonstrator

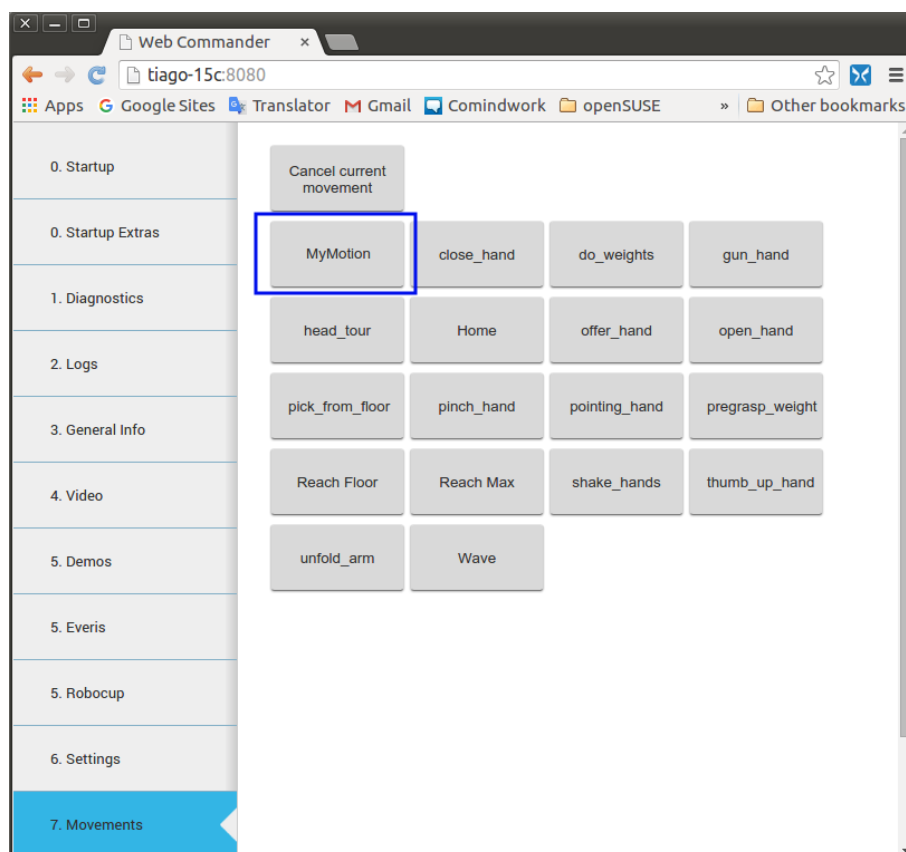


Figure 110: New motion in WebCommander



Customer Service



32 Customer service

32.1 Support portal

All communications between customers and PAL Robotics are made using tickets in a helpdesk software.

This web system can be found at <http://support.pal-robotics.com>. Figure 111 shows the initial web of the support web site.

New accounts will be created on request by PAL Robotics.

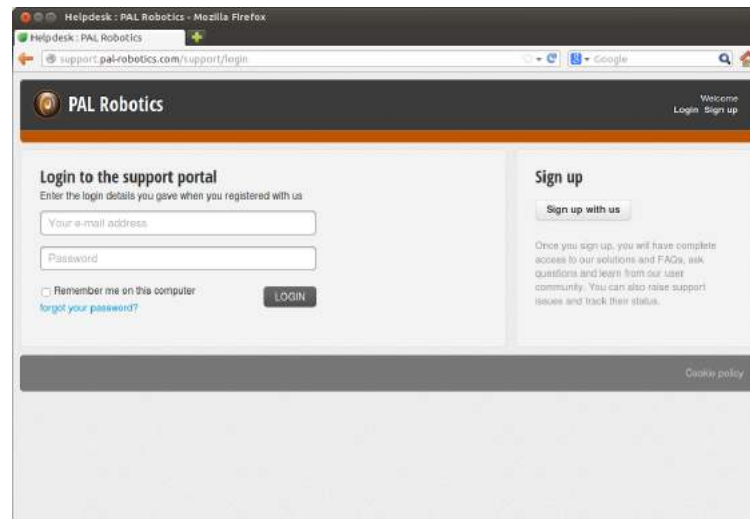


Figure 111: PAL Robotics support web

Once the customer has entered the system (Figure 112), two tabs can be seen: *Solutions* and *Tickets*.

In the Solution section there are *FAQs* and *News* that PAL Robotics gives to the customers.

In the *Tickets* section there is the history of all the tickets created by the customer.

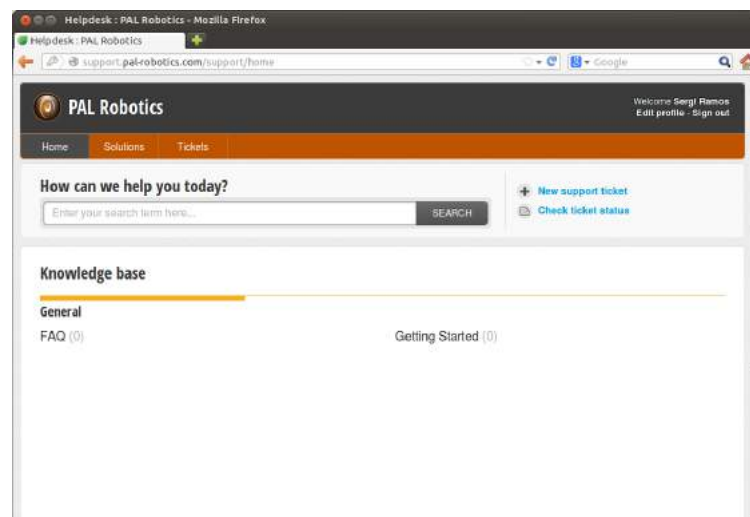


Figure 112: Customer connected

Figure 113 shows the ticket creation webpage.

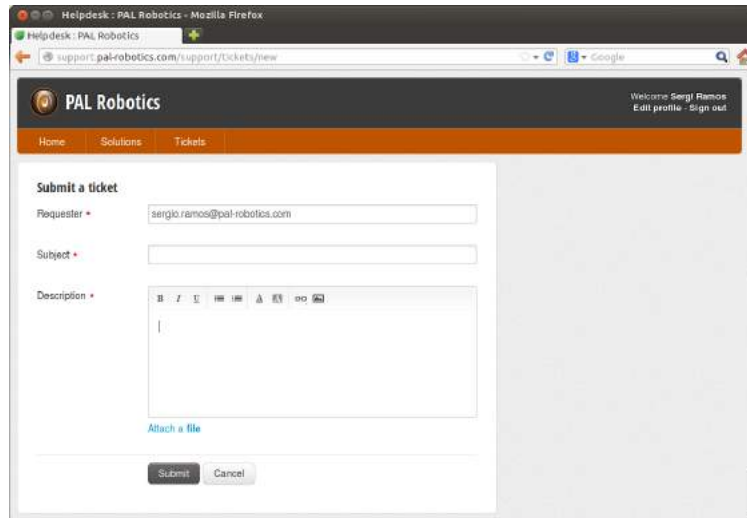


Figure 113: Ticket creation

32.2 Remote support

A technician from PAL Robotics can give remote support to the robot. This remote support is disabled by default, and the customer has to activate it manually. If the robot needs to be rebooted, the customer has to activate the remote support after each boot because it is not persistent.

The robot connects to the PAL remote support system using the command *remoteAssistanceConnect*:

```
root@tiago-0c:~# remoteAssistanceConnect ipAddress port
```

Using an issue in the support portal the PAL technician will provide the IP address and the port the customer has to use.



PAL ROBOTICS S.L.

Pujades 77-79, 4º 4ª

Tel.: +34 934 145 347

info@pal-robotics.com

08005 Barcelona, Spain Fax: +34 932 091 109

www.pal-robotics.com

