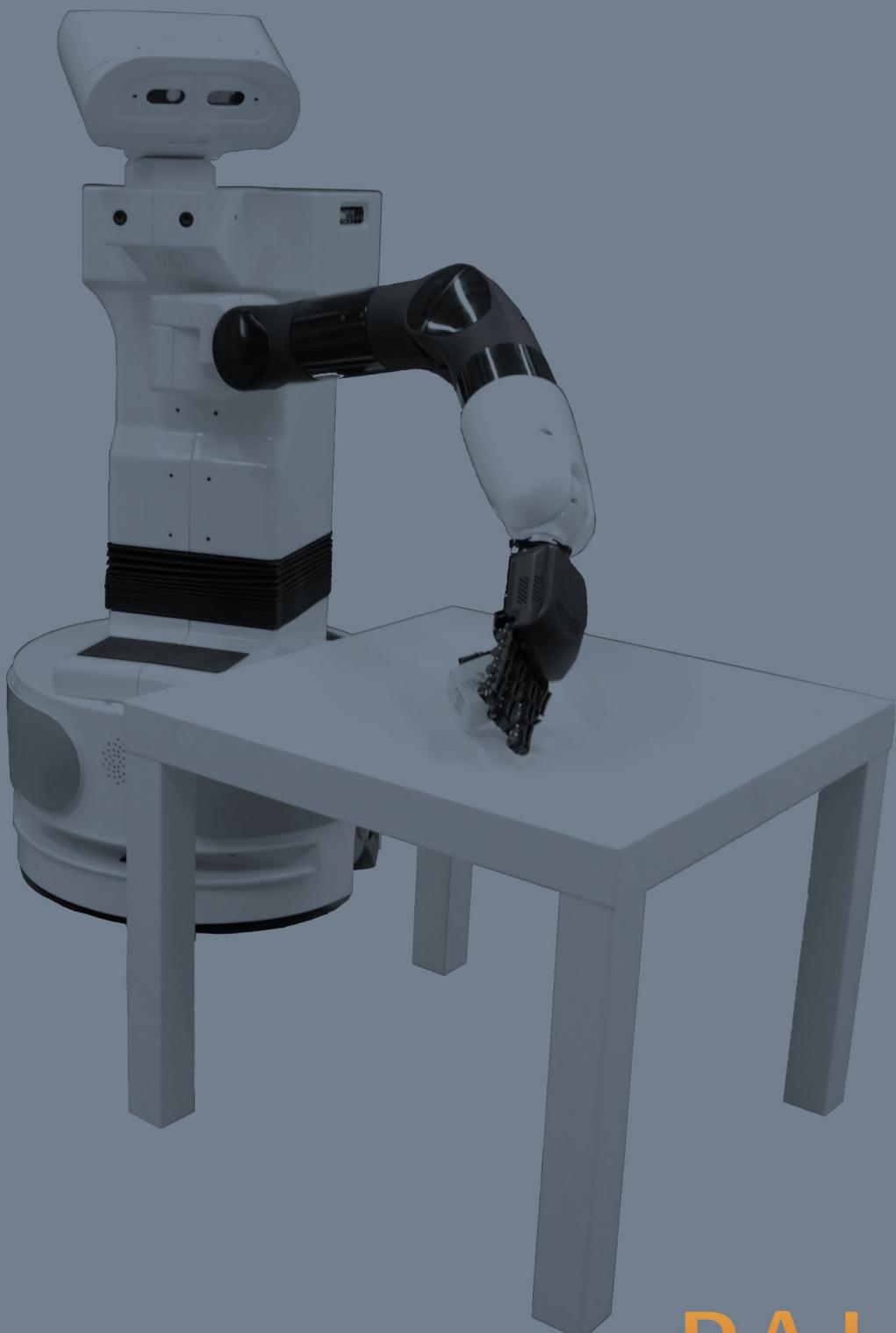


**TIA GO**

# Handbook





**TIA Go**

## **Handbook**



**Barcelona  
2020**

©2020 PAL Robotics S.L.  
*All rights reserved*

## Document revisions

All notable changes to this manual will be documented in this file.

### 1.23.0

**Date** 2020/01/24

- First release including complete documentation of TIAGo++

### 1.22.0

**Date** 2019/09/25

- Update the documentation of robot firmware update
- Added documentation for the Optris thermal camera add-on
- Added documentation for the new Velodyne VLP-16 add-on

### 1.21.0

**Date** 2019/07/16

- Add NTP section to development computer, links to Robot NTP section
- Document how to change the pal\_startup environment for adding new workspaces
- Add introspection controller section, that shows how to record data and visualize it using: PlotJuggler, introspection controller, rosbags.

### 1.20.0

**Date** 2019/05/07

- Add changelog
- Update documentation on how to change controllers of the robot
- Document how to change the motion action server for WMovementTab on WebCommander section

### 1.19.0

**Date** 2019/03/07

- Document new end effector exchange procedure
- Document new procedure for launching gazebo simulations
- Update how to install public simulation
- Update NTP robot configuration
- Update Jetson documentation

---

## 1.18.0

**Date** 2018/12/19

- Add documentation for TIAGo Base touch screen
- Add documentation for collision detector and low battery shutdown
- Fixes to Leap Motion documentation

# Contents

<b>1 Package contents</b>	<b>6</b>
1.1 Overview . . . . .	6
<b>2 Specifications</b>	<b>9</b>
2.1 Robot overview . . . . .	9
2.2 Mobile base . . . . .	10
2.2.1 Onboard computer . . . . .	11
2.2.2 Battery . . . . .	11
2.2.3 Power connector . . . . .	11
2.2.4 Laser range-finder . . . . .	12
2.2.5 Sonars . . . . .	12
2.2.6 IMU . . . . .	12
2.2.7 User panel . . . . .	13
2.2.8 Service panel . . . . .	14
2.2.9 Connectivity . . . . .	15
2.3 Torso . . . . .	15
2.3.1 Lifter . . . . .	15
2.3.2 Expansion panel . . . . .	16
2.3.3 Laptop tray . . . . .	16
2.4 Arm . . . . .	18
2.5 Force/Torque sensor . . . . .	19
2.6 End-effector . . . . .	20
2.6.1 Hey5 hand . . . . .	20
2.6.2 PAL gripper . . . . .	21
2.6.3 Schunk gripper WSG32 . . . . .	21
2.7 Head . . . . .	23
2.7.1 Pan-tilt mechanism . . . . .	23
2.7.2 Speaker . . . . .	23
2.7.3 Stereo microphones . . . . .	23
2.7.4 RGB-D camera . . . . .	24
2.8 Robot kinematics . . . . .	25
2.9 Electrical parts and components . . . . .	25
<b>3 Storage</b>	<b>29</b>
3.1 Overview . . . . .	29
3.2 Unpacking . . . . .	29
3.3 Storage cautions . . . . .	30

<b>4 Introduction to safety</b>	<b>33</b>
4.1 Overview . . . . .	33
4.2 Intended applications . . . . .	33
4.3 Working environment and usage guidelines . . . . .	33
4.4 Battery manipulation . . . . .	34
<b>5 Safety measures in practice</b>	<b>35</b>
5.1 Turning the robot on properly . . . . .	35
5.2 Shutting down the robot properly . . . . .	36
5.3 Emergency stop . . . . .	36
5.4 Measures to prevent falls . . . . .	36
5.4.1 Measure 1 . . . . .	36
5.4.2 Measure 2 . . . . .	37
5.4.3 Measure 3 . . . . .	37
5.4.4 Measure 4 . . . . .	37
5.4.5 Measure 5 . . . . .	38
5.5 Measures to prevent collisions . . . . .	38
5.5.1 Measure 1 . . . . .	38
5.5.2 Measure 2 . . . . .	38
5.6 How to proceed when an arm collision occurs . . . . .	39
5.7 Low battery shutdown . . . . .	39
5.8 Firefighting equipment . . . . .	40
5.9 Leakage . . . . .	40
<b>6 Robot identification</b>	<b>43</b>
<b>7 Default network configuration</b>	<b>47</b>
<b>8 Software recovery</b>	<b>51</b>
8.1 Overview . . . . .	51
8.2 Robot computer installation . . . . .	51
8.3 Development computer installation . . . . .	51
<b>9 TIAGo Robot's Internal Computers</b>	<b>57</b>
9.1 TIAGo LAN . . . . .	57
9.2 Users . . . . .	57
9.3 File system . . . . .	57
9.4 Internal DNS . . . . .	58
9.5 NTP . . . . .	58
9.6 System upgrade . . . . .	59
9.7 Firmware update . . . . .	59
9.8 Meltdown and Spectre vulnerabilities . . . . .	59

<b>10 Development computer</b>	<b>63</b>
10.1 Overview . . . . .	63
10.2 Computer requirements . . . . .	63
10.3 Setting ROS environment . . . . .	63
10.4 ROS communication with the robot . . . . .	63
10.5 Compiling software . . . . .	64
10.6 System Upgrade . . . . .	64
10.7 NTP . . . . .	64
<b>11 WebCommander</b>	<b>67</b>
11.1 Accessing the WebCommander website . . . . .	67
11.2 Overview . . . . .	67
11.3 Default tabs . . . . .	67
11.3.1 Startup tab . . . . .	67
11.3.2 Startup extras tab . . . . .	68
11.3.3 Diagnostics Tab . . . . .	68
11.3.4 Logs Tab . . . . .	69
11.3.5 General Info Tab . . . . .	70
11.3.6 Installed Software Tab . . . . .	70
11.3.7 Settings Tab . . . . .	70
11.3.8 Networking tab . . . . .	72
11.3.9 Video Tab . . . . .	74
11.3.10 Movements Tab . . . . .	76
11.3.11 Demos tab . . . . .	77
11.4 Tab configuration . . . . .	77
11.4.1 Parameter format . . . . .	78
11.4.2 Startup Plugin Configuration . . . . .	78
11.4.3 Diagnostics Plugin Configuration . . . . .	78
11.4.4 Logs Plugin Configuration . . . . .	79
11.4.5 JointCommander Plugin Configuration . . . . .	79
11.4.6 General Info Plugin Configuration . . . . .	79
11.4.7 Installed Software Plugin Configuration . . . . .	79
11.4.8 Settings Plugin Configuration . . . . .	79
11.4.9 Networking Embedded Plugin Configuration . . . . .	80
11.4.10 Video Plugin Configuration . . . . .	80
11.4.11 Movements Plugin Configuration . . . . .	80
11.4.12 Commands Plugin Configuration . . . . .	80

<b>12 Software architecture</b>	<b>85</b>
12.1 Overview . . . . .	85
12.2 Operating system layer . . . . .	85
12.3 ROS layer . . . . .	85
12.4 Software startup process . . . . .	86
<b>13 Deploying software on the robot</b>	<b>89</b>
13.1 Introduction . . . . .	89
13.2 Usage . . . . .	89
13.3 Notes . . . . .	89
13.4 Deploy tips . . . . .	90
13.5 Use-case example . . . . .	90
13.5.1 Adding a new ROS Package . . . . .	90
13.5.2 Adding a new controller . . . . .	92
13.5.3 Modifying an installed package . . . . .	93
<b>14 Modifying Robot Startup</b>	<b>97</b>
14.1 Introduction . . . . .	97
14.1.1 Application start configuration files . . . . .	97
14.1.2 Computer start lists . . . . .	98
14.1.3 Additional startup groups . . . . .	98
14.2 Startup ROS API . . . . .	99
14.3 Startup command line tools . . . . .	99
14.4 ROS Workspaces . . . . .	99
14.5 Modifying the robot's startup . . . . .	99
14.5.1 Adding a new application for automatic startup . . . . .	100
14.5.2 Modifying how an application is launched . . . . .	100
14.5.3 Adding a new workspace . . . . .	100
<b>15 Sensors</b>	<b>103</b>
15.1 Description of sensors . . . . .	103
<b>16 Sensors ROS API</b>	<b>103</b>
16.1 Laser range-finder . . . . .	103
16.1.1 Topics published . . . . .	103
16.2 Sonars . . . . .	104
16.2.1 Topics published . . . . .	104
16.3 Inertial Measurement Unit . . . . .	104
16.3.1 Topics published . . . . .	104
16.4 RGB-D camera . . . . .	104

16.4.1 Topics published . . . . .	104
16.4.2 Services advertised . . . . .	104
16.5 Force/Torque sensor . . . . .	105
16.5.1 Topics published . . . . .	105
<b>17 Sensor visualization</b>	<b>105</b>
<b>18 Power status</b>	<b>109</b>
18.1 ROS API . . . . .	109
18.2 Description . . . . .	109
<b>19 Text-to-Speech synthesis</b>	<b>113</b>
19.1 Overview of the technology . . . . .	113
19.2 Text-to-Speech node . . . . .	113
19.2.1 Launching the node . . . . .	113
19.2.2 Action interface . . . . .	113
19.3 Examples of usage . . . . .	114
19.3.1 WebCommander . . . . .	114
19.3.2 Command line . . . . .	115
19.3.3 Action client . . . . .	115
<b>20 Base motions</b>	<b>119</b>
20.1 Overview . . . . .	119
20.2 Base motion joystick triggers . . . . .	119
20.2.1 Forward/backward motion . . . . .	119
20.2.2 Rotational motion . . . . .	120
20.2.3 Changing the speed of the base . . . . .	120
20.3 Mobile base control ROS API . . . . .	120
20.4 Mobile base control diagram . . . . .	121
<b>21 Torso motions</b>	<b>125</b>
21.1 Overview . . . . .	125
21.2 Torso motion joystick triggers . . . . .	125
21.3 Torso control ROS API . . . . .	125
21.3.1 Topic interfaces . . . . .	125
21.3.2 Action interfaces . . . . .	125
<b>22 Head motions</b>	<b>129</b>
22.1 Overview . . . . .	129
22.2 Head motion joystick triggers . . . . .	129
22.3 Head motions with rqt GUI . . . . .	129
22.4 Head control ROS API . . . . .	130
22.4.1 Topic interfaces . . . . .	130
22.4.2 Action interfaces . . . . .	130

<b>23 Arm motions</b>	<b>133</b>
23.1 Overview . . . . .	133
23.2 Arm motions with rqt GUI . . . . .	133
23.3 Arm control ROS API . . . . .	133
23.3.1 Topic interfaces . . . . .	133
23.3.2 Action interfaces . . . . .	134
<b>24 Hand motions</b>	<b>137</b>
24.1 Overview . . . . .	137
24.2 Hand motion joystick triggers . . . . .	137
24.3 Hand motions with rqt GUI . . . . .	137
24.4 Hand control ROS API . . . . .	138
24.4.1 Topic interfaces . . . . .	138
24.4.2 Action interfaces . . . . .	138
<b>25 PAL gripper motions</b>	<b>141</b>
25.1 Overview . . . . .	141
25.2 Gripper motion joystick triggers . . . . .	141
25.3 Gripper motions with rqt GUI . . . . .	141
25.4 Gripper control ROS API . . . . .	142
25.4.1 Topic interfaces . . . . .	142
25.4.2 Action interfaces . . . . .	142
25.4.3 Service interfaces . . . . .	143
<b>26 Schunk gripper motions</b>	<b>147</b>
26.1 Overview . . . . .	147
26.2 A note on the Schunk gripper position control . . . . .	147
26.3 Gripper motion joystick triggers . . . . .	147
26.4 Gripper motions with rqt GUI . . . . .	148
26.5 Gripper control ROS API . . . . .	148
26.5.1 Topic interfaces . . . . .	148
26.5.2 Action interfaces . . . . .	149
<b>27 End-effector exchange</b>	<b>153</b>
27.1 Overview . . . . .	153
27.2 Changing the end-effector software configuration . . . . .	153
27.3 Mounting the Hey5 hand . . . . .	153
27.3.1 Unmounting the gripper . . . . .	153
27.3.2 Mounting the Hey5 hand . . . . .	154
27.3.3 Validation . . . . .	157
27.4 Mounting the parallel gripper . . . . .	157

---

<b>28 Force-Torque sensor calibration</b>	<b>161</b>
28.1 Overview . . . . .	161
28.2 Running the calibration . . . . .	161
28.2.1 Action Interface . . . . .	161
28.3 Parameter customization . . . . .	161
<b>29 Upper body motions engine</b>	<b>165</b>
29.1 Motions library . . . . .	165
29.2 Motions specification . . . . .	166
29.3 Using predefined motions safely . . . . .	167
29.4 ROS interface . . . . .	167
29.4.1 Action interface . . . . .	167
29.5 Action clients . . . . .	168
<b>30 Navigation</b>	<b>171</b>
30.1 Overview . . . . .	171
30.2 Navigation architecture . . . . .	171
30.3 Navigation ROS API . . . . .	173
30.3.1 Topic interfaces . . . . .	173
30.3.2 Action interfaces . . . . .	173
30.3.3 Service interface . . . . .	173
30.4 SLAM and path planning in simulation . . . . .	174
30.4.1 Mapping . . . . .	174
30.4.2 Saving the map . . . . .	175
30.4.3 Localization and Path Planning . . . . .	176
30.5 SLAM and path planning on the robot . . . . .	178
30.5.1 Saving the map on the robot . . . . .	179
30.5.2 Localization and path planning . . . . .	179
30.5.3 Changing the active map on the robot . . . . .	179
30.6 Map Editor . . . . .	180
30.6.1 Launching the Map Editor . . . . .	181
30.6.2 Creating a map . . . . .	181
30.6.3 Managing maps . . . . .	182
30.6.4 Defining Points of Interest . . . . .	183
30.6.5 Defining Zones of Interest . . . . .	184
30.6.6 Defining Virtual Obstacles . . . . .	184
30.6.7 Defining Groups of POIs . . . . .	185
30.6.8 Modifying map meta-data . . . . .	186
30.7 Obstacle avoidance with the RGB-D camera . . . . .	187
30.7.1 Disabling obstacle avoidance with the RGB-D camera . . . . .	188
30.7.2 Enabling obstacle avoidance with the RGB-D camera . . . . .	189

<b>31 Dock station</b>	<b>193</b>
31.1 Overview . . . . .	193
31.2 The dock station hardware . . . . .	193
31.3 Installation . . . . .	193
31.4 Docking algorithm . . . . .	193
31.5 Usage . . . . .	194
31.5.1 Dock/Undock using rviz plugins . . . . .	194
31.5.2 Dock/Undock using action client . . . . .	195
31.5.3 Dock/Undock code example . . . . .	195
<b>32 Motion planning with <i>MoveIt!</i></b>	<b>199</b>
32.1 Overview . . . . .	199
32.2 Getting started with the <i>MoveIt!</i> graphical user interface in simulation . . . . .	199
32.2.1 The planning environment of <i>MoveIt!</i> . . . . .	200
32.2.2 End test . . . . .	201
32.3 <i>MoveIt!</i> with the real robot . . . . .	201
<b>33 Facial perception</b>	<b>205</b>
33.1 Overview . . . . .	205
33.2 Facial perception ROS API . . . . .	205
33.2.1 Topic interfaces . . . . .	205
33.2.2 Service interface . . . . .	206
33.3 Face perception guidelines . . . . .	206
<b>34 Speech recognition</b>	<b>209</b>
34.1 Overview . . . . .	209
34.2 Requirements . . . . .	209
34.3 Speech Recognition ROS API . . . . .	209
34.3.1 Action interface . . . . .	209
34.4 Recognizer behaviour . . . . .	210
34.5 Special keyword . . . . .	210
34.6 Configuration . . . . .	210
34.7 Google Cloud Speech account creation . . . . .	211
<b>35 Whole Body Control</b>	<b>215</b>
35.1 Overview . . . . .	215
35.2 WBC through ROS interface . . . . .	215
35.3 WBC upper-body teleoperation with leap motion . . . . .	219
35.4 WBC upper-body teleoperation with joystick . . . . .	221
35.5 WBC with rviz interactive markers . . . . .	223

---

<b>36 Change controllers</b>	<b>227</b>
36.1 Controller manager services . . . . .	227
36.2 Change controllers action . . . . .	228
<b>37 Introspection controller</b>	<b>231</b>
37.1 Start the controller . . . . .	231
37.2 Record and reproduce the data . . . . .	231
37.3 Record new variables . . . . .	232
<b>38 Simulation</b>	<b>235</b>
38.1 Overview . . . . .	235
38.1.1 Empty world . . . . .	235
38.1.2 Office world . . . . .	235
38.1.3 Table with objects world . . . . .	236
<b>39 LEDs</b>	<b>239</b>
39.1 ROS API . . . . .	239
39.1.1 Available services . . . . .	239
<b>40 Modifying the base touch screen</b>	<b>243</b>
40.1 Introduction . . . . .	243
40.2 Configuration file structure . . . . .	243
40.3 Menu types . . . . .	243
40.3.1 ConfigurableMenu . . . . .	243
40.3.2 EmergencyMenu . . . . .	244
40.3.3 Examples . . . . .	244
<b>41 Tutorials</b>	<b>249</b>
41.1 Overview . . . . .	249
41.2 Installing pre-requisites . . . . .	249
41.3 Downloading source code . . . . .	249
41.4 Building the workspace . . . . .	249
41.5 Running the tutorials . . . . .	249
<b>42 NVIDIA Jetson TX2</b>	<b>253</b>
42.1 Overview . . . . .	253
42.2 Installation . . . . .	253
42.3 Connection . . . . .	255
42.4 Provided example . . . . .	255
42.4.1 Object detection API . . . . .	255
42.4.2 How to test the example . . . . .	256

<b>43 Velodyne VLP-16</b>	<b>261</b>
43.1 Overview . . . . .	261
43.2 Configuring Velodyne . . . . .	261
43.3 Installation . . . . .	262
43.4 Starting velodyne drivers . . . . .	263
<b>44 Optris Thermal Camera</b>	<b>267</b>
44.1 Overview . . . . .	267
44.2 Installation . . . . .	267
44.3 Starting optris drivers . . . . .	268
<b>45 PAL gripper camera add-on</b>	<b>273</b>
45.1 Overview . . . . .	273
45.2 Add-on installation . . . . .	274
45.3 Running the camera driver . . . . .	277
45.4 Visualizing the camera image . . . . .	277
<b>46 Demos</b>	<b>281</b>
46.1 Learning-by-demonstration . . . . .	281
<b>47 Troubleshooting</b>	<b>285</b>
47.1 Overview . . . . .	285
47.2 Startup issues . . . . .	285
47.3 Navigation issues . . . . .	286
47.4 Upperbody movements issues . . . . .	287
47.5 Text-to-speech issues . . . . .	288
47.6 Network issues . . . . .	288
<b>48 Customer service</b>	<b>291</b>
48.1 Support portal . . . . .	291
48.2 Remote support . . . . .	292

**TIA Go**

**Welcome**





## Welcome

Thank you for choosing PAL Robotics. This Handbook contains 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 (whether under the law of contract, the law of torts or otherwise) in relation to the contents of, or use of, or otherwise in connection with, this Handbook:

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

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

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

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

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

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

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

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

# 1 Package contents

## 1.1 Overview

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



(a) TIAGo



(b) Battery charger



(c) Joystick



(d) Installation pendrive



(e) Dock station  
(optional)



(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's main parts are depicted in figure 2, and its main specifications are summarized in table 1.



Figure 2: TIAGo's main components

Dimensions	Height	110 – 145 cm
	Weight	72 Kg
	Base footprint	Ø 54 cm
Degrees of freedom	Mobile base	2
	Torso lift	1
	Arm	4
	Wrist	3
	Head	2
	Hey5 hand	19 (3 actuated)
	PAL gripper	2
Mobile base	Drive system	Differential
	Max speed	1 m/s
Torso	Lift stroke	35 cm
Arm	Payload	2 Kg
Electrical features	Reach	87 cm
	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: The robot's main specifications

## 2.2 Mobile base

TIAGo's mobile base is provided with a differential drive mechanism and contains an onboard computer, batteries, power connector, laser-range finder, three rear sonars, a user panel, a service panel and two WiFi networks to ensure wireless connectivity. Furthermore, the version of TIAGo with a docking station has a charging plate on the front.



Figure 3: Mobile base front view



Figure 4: Mobile base rear view

### 2.2.1 Onboard computer

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

Component	Description
CPU	Intel i5 / i7
RAM	8 / 16 GB
Hard disk	250 / 500 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 two batteries. In this case, the *total Nominal capacity is 1440 Wh*.

### 2.2.3 Power connector

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

**Connection** Insert the charging connector with the metal lock facing up and push it, as shown in figure 5b until you hear a 'click'.

**Disconnection** Once the charge is completed, the connector can be removed. In order to remove it, press the metal lock and pull the connector firmly, see figure 5c.

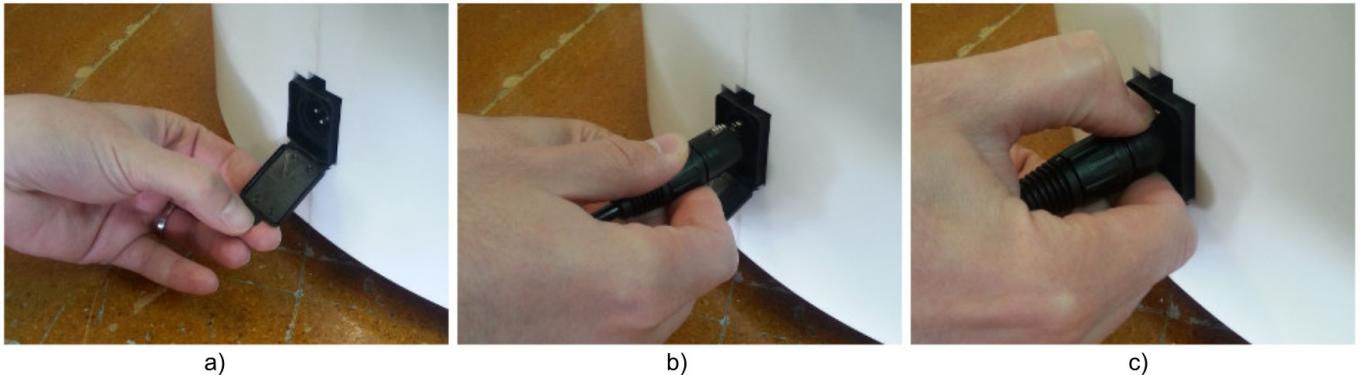


Figure 5: a) connector entry b) connector insertion procedure c) connector removal procedure.

#### 2.2.4 Laser range-finder

The specifications of the laser on the front part of the mobile base depend on the configuration options you have ordered. The lasers supported are shown in table 4.

Manufacturer	Hokuyo	Manufacturer	SICK	Manufacturer	SICK
Model	URG-04LX-UG01	Model	TIM561-2050101	Model	TIM571-2050101
Range	0.02 - 5.6 m	Range	0.05 - 10 m	Range	0.05 - 25 m
Frequency	10 Hz	Frequency	15 Hz	Frequency	15 Hz
Field of view	180°	Field of view	180°	Field of view	180°
Step angle:	0.36°	Step angle:	0.33°	Step angle:	0.33°

Table 4: Lasers range-finder specifications

#### 2.2.5 Sonars

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

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

Table 5: Sonar's 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 attitude. The specifications are presented in table 6.

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

Table 6: IMU's main specifications

### 2.2.7 User panel

The user panel is on the top, rear part of TIAGo mobile base. It provides the buttons to power up and shutdown the robot, and a screen to give visual feedback on the robot's status. All the specific elements of the user panel are shown in figure 6 and the description of each element is presented in table 7.

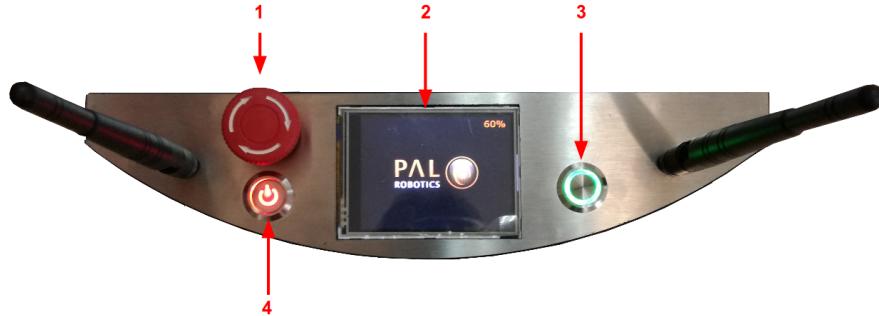


Figure 6: User panel

Number	Name / Short description
1	Emergency stop
2	Information display
3	On/Off button
4	Electric switch

Table 7: User panel description

**Electric switch** The electric switch is the main power control switch. Before turning TIAGo ON make sure first that this switch is ON, i.e. its red light indicator is ON. On the other hand, when TIAGo is not going to be used for a long period, please press the switch so that its red light indicator turns OFF. Note that this switch should not be turned OFF before using the On/Off button to turn OFF the onboard computer of the robot. Turning OFF this switch will cut instantaneously the power supply to all the robot components, including the onboard computer. Do not use this switch as emergency stop. For the emergency stop please refer to the next section.

**Emergency stop** When pushed, motors are stopped and disconnected. The green indicator of the On/Off button will blink fast in order to notify the user of the emergency state.

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

**Information display** A 320x240 Color TFT display shows the battery level on the top right corner.

**On/Off button** The standby control button is a pushbutton with a green light that indicates the system's current 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 the main power is connected, i.e. the electric switch is ON, see figure 6, the user must press this button for one second in order to start TIAGo.

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

### 2.2.8 Service panel

It is possible to access the service panel by removing the cover behind the laser, see figure 7.

This service panel gives the user access to video, usb and the On/Off button of the robot's computer. It can be used for reinstallation or debug purposes.

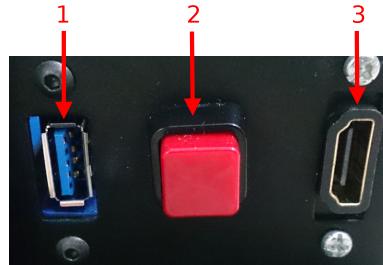


Figure 7: Service panel

Number	Name / Short description
1	USB 3.0
2	On/Off button for the 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 has a 802.11g interface.

There are two Gigabit Ethernet ports, ports 2 and 3 in figure 10, that can be used to connect to the robot's internal network. 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 robot's services.*

## 2.3 Torso

TIAGo 's torso is the structure that supports the robot's arm and head, and is equipped with an internal lifter mechanism which allows the user to change the height of the robot. Furthermore, it features an expansion panel and a laptop tray.

### 2.3.1 Lifter

The lifter mechanism is placed underneath the industrial bellows, shown in figure 8. The lifter is able to move at 50 mm/s and has a stroke of 350 mm. The minimum and maximum height of the robot is shown in figure 9.



Figure 8: Industrial bellows of the lifting torso

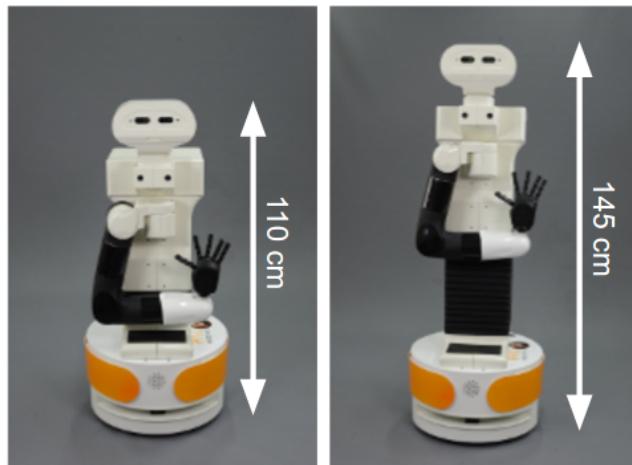


Figure 9: Height range of the robot

### 2.3.2 Expansion panel

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



Figure 10: 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 2.0 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 11 is the flat surface on top of the torso just behind the robot's head, see figure 11. It has mounting points to add new equipment, supporting to 5 kg, 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.

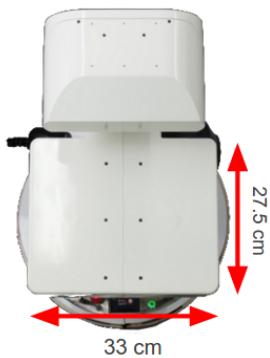


Figure 11: Laptop tray dimensions



Figure 12: Laptop placed on the rear tray of the robot

## 2.4 Arm

TIAGo's arm is composed of four M90 modules and one 3 DoF wrist, M3D, as shown in figure 13. The main specifications of the arm and wrist are shown in table 11.

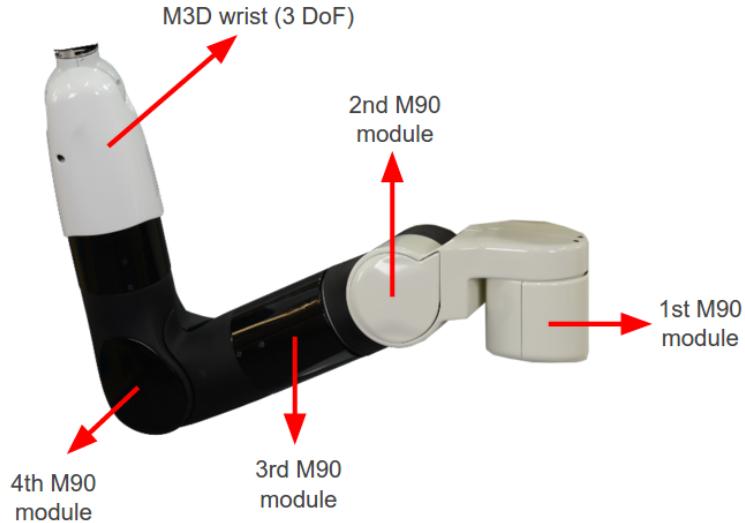


Figure 13: Arm components

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 an ATI mini45, see figure 14. The main specifications of the sensor are summarized in table 12.

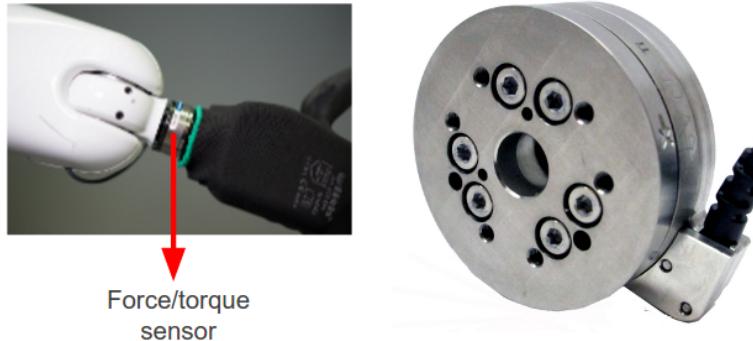


Figure 14: Force/torque sensor placement and close view

Physical Specs	Weight	Diameter	Height
	0.0917 Kg	45 mm	15.7 mm
Sensing ranges	Fx, Fy	Fz	Tx, Ty
Resolution	290 N	580 N	10 Nm
	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's end-effector is one of the modular features of the robot. TIAGo can be used with three interchangeable end-effectors: the Hey5 hand, the PAL parallel gripper and the Schunk WSG32 industrial gripper.

**Warning** Since September 2019 the Schunk WSG32 gripper is no longer available as the manufacturer has discontinued this product. Documentation about this end-effector is kept in this handbook as reference for customers already owning it.

### 2.6.1 Hey5 hand

The Hey5 hand is shown in figure 15. The main specifications of this underactuated, self-contained hand are summarized in table 13.



Figure 15: 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+right+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](http://NaturalMachineMotionInitiative.com) .

### 2.6.2 PAL gripper

The PAL parallel gripper is shown in figure 16. The gripper contains two motors, each controlling one of the fingers. Each finger has a linear range of 4 cm.



Figure 16: PAL gripper

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

Table 14: PAL gripper main specifications

### 2.6.3 Schunk gripper WSG32

The Schunk gripper WSG32 is an industrial parallel gripper that has been electro-mechanically integrated in TIAGoby PAL Robotics. The gripper has been integrated in the CAN bus of the robot's arm. The integrated version of the gripper is shown in figure 17. The gripper contains a single motor and is compatible with two different commercial fingers:

- ABF WSG32/50-DV: with integrated force sensing
- ABF WSG32/50-GV: no force sensors integrated

Furthermore, users can design their own fingers.



Figure 17: Schunk WSG32 gripper

Model	WSG 032-068-C
Weight	550 g
Payload	2 kg
Interchangeable fingers	Yes
Stroke per jaw	34 mm
Min. gripping force	5 N
Permanent gripping force (100% continuous duty)	50 N
Max. speed	400 mm/s

Table 15: Schunk WSG32 gripper main specifications

## 2.7 Head

TIAGo's head is equipped with a pan-tilt mechanism, i.e. 2 DoF, and is equipped with stereo microphones, a speaker and an RGB-D camera. Furthermore, on top of the head there is a flat surface with mounting points to allow the user to add new sensors or equipment. Note that the head has a payload of 0.5 kg when adding new equipment. Figure 18 shows the location of each component and the two joints of the pan-tilt mechanism.

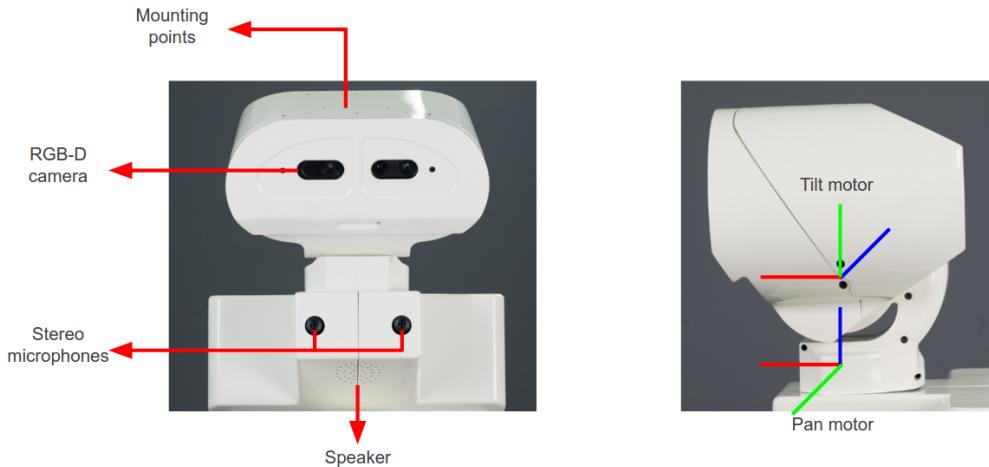


Figure 18: Head parts and components

### 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 16: Head pan-tilt main specifications

### 2.7.2 Speaker

The speaker specified in table 17 is provided below the robot's head.

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

Table 17: Speaker main specifications

### 2.7.3 Stereo microphones

The Andrea SuperBeam Stereo Array Microphone is integrated just below the head of the robot. The specifications of the audio card are presented in table 18 and the specifications of the Stereo Array Microphone are detailed in table 19.

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 sound 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	0-70°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 18: 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 19: USB external audio card main specifications

#### 2.7.4 RGB-D camera

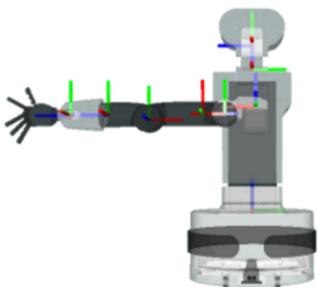
TIAGo's head includes an RGB-D camera, specified in table 20.

Manufacturer	Orbbec
Model	Astra S
Field of view	60° H, 49.5° V, 73° D
Interface	USB 2.0
Color stream modes	QVGA 320x240 @ 30 fps, VGA 640x480 @ 30 fps, 1280x960 @ 10 fps
Depth stream modes	QVGA 320x240 @ 30 fps, VGA 640x480 @ 30 fps, 160x120 @ 30 fps
Depth sensor range	0.4 - 2 m

Table 20: RGB-D main specifications

## 2.8 Robot kinematics

Figure 19 presents the kinematic specifications of TIAGo's 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 19: Upper body joint specification

## 2.9 Electrical parts and components

Neither TIAGo nor any of its electrical components or mechanical parts are connected to external ground. The chassis and all electromechanical components are physically isolated from the ground by the isolation rubber under its feet. Avoid touching any metal parts directly to prevent discharges and damage to TIAGo's electromechanical parts.

### 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 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

This section contains information relating to the storage of TIAGo .

### 3.2 Unpacking

This section explains how to unbox TIAGo safely. TIAGo is shipped with the flightcase shown in figure 20.



Figure 20: TIAGo flightcase

The flightcase **MUST be always transported vertically** to ensure the robot's safety. In order to move the flightcase, pull the handle on the back, as shown in figure 21. To place the flightcase in a given location use one of your feet to help you carefully set the flightcase in an upright position.



Figure 21: Moving the flightcase

Open the door of the crate, see figure 22a, and unfold the ramp, as shown in figure 22b. Remove the foam wedges holding the mobile base in place, as shown in figure 22c. Finally, remove the robot from the crate by pulling on the upper part of its back and, if necessary, from its shoulders, as shown in figure 22d. **Do not pull on any part of the mobile base cover**, as it could cause damage to you or the robot.

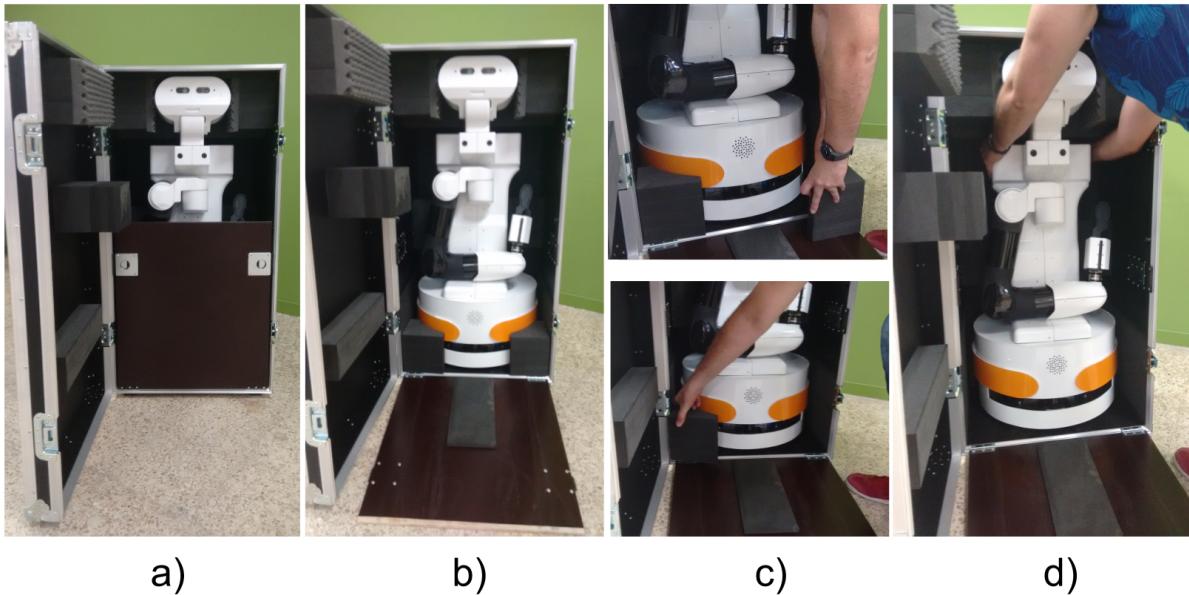


Figure 22: Unboxing procedure

### 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 the storage period exceeds two weeks.
- It is recommended to charge the battery to 50% when storing it for more than two weeks.
- Avoid the use or presence of water near TIAGo.
- Avoid any generation of dust close to TIAGo.
- Avoid the use or presence of magnetic 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 maintain safety, and 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 designed to be operated in a controlled environment, under supervision by trained staff at all times

- The hardware and software of TIAGo enables research and development of 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°C ~ +35°C

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

Maintain a safe environment:

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

- Do not allow the robot to come into 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

The following guidelines must be respected when handling the robot in order to prevent damage to the robot's internal batteries.

- 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 all circumstances, opening the internal battery case.
- Do not expose the robot to ambient temperatures above 49°C for over 24 hours.
- Do not store the robot at temperatures below -5°C for more than 7 days.
- For long-term storage (more than 1 month) charge the battery to 50%.
- Do not use TIAGo's batteries for any other purpose.
- Do not use any devices except 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

### WARNING

This Section presents important information that must be taken into consideration when using the robot. Read carefully the instructions to ensure the safety of the people on the surroundings and to prevent damages to the environment and to the robot. Follow these instructions **everytime the robot is used.**

### 5.1 Turning the robot on properly

### WARNING

The procedure described in this section requires to have a clearance of about 1.5 m in front of the robot and at each of its sides in order to execute the required movements safely.

When the robot is started the arm will be lying on top of the mobile base. If the robot is turned on and left on that pose the heat from the arm motors may be transferred, after some time, to the paint of the robot's lap cover which may end up melting it and causing aesthetical damage to these covers. In order to prevent this the procedure depicted in figure 23 and hereafter explained details a safe way to get the arm out of this initial pose, preventing self-collisions:

- a) In order to get the arm out of this position, either press the button **Get out of collision** in the **Demos** tab of the WebCommander, see Section 11, 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 using the joystick
- c) Execute the **Offer Gripper** or (or Offer Hand in case of having the Hey5 hand mounted) movement using, for instance, the Movements tab of the WebCommander
- d) Execute the **Home** motion in order to fold back the arm and the torso into a safe configuration where no contacts occur with the lap of the robot.

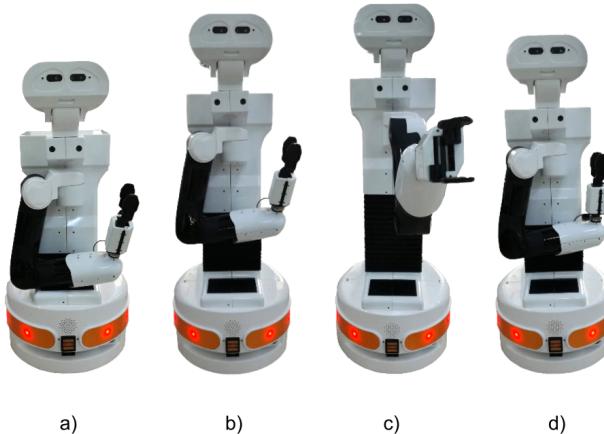


Figure 23: Procedure to start moving the arm safely

## 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 or the floor, the following procedure must be followed, as depicted in figure 24:

- Execute the **Home** predefined motion using, for instance, the WebCommander
- Hold the robot's wrist
- Press the On/Off button for 1 second and then wait until the green light indicator stops blinking
- Keep holding the wrist until the robot is turned off. You may then guide the wrist to its resting position on top of the mobile base

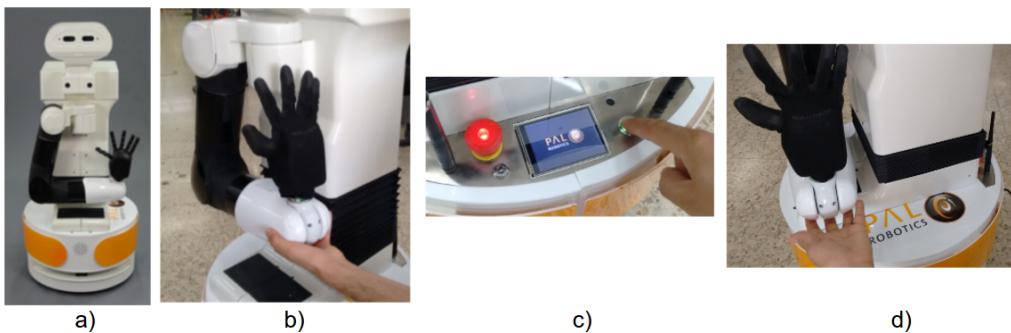


Figure 24: Shutdown procedure

## 5.3 Emergency stop

The emergency stop button can be found on the back of the robot between the power button and the battery level display. As the name implies, this button should only be used only in exceptional cases, when an 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 because the motors will be switched OFF and the arm will fall down, while the computer remains on.
  - After releasing the emergency stop button, the user has to re-start the robot by pressing the On/Off button until it stops blinking. After this operation, the robot's status should be restored to that prior to pressing the emergency button in few seconds.

## 5.4 Measures to prevent falls

TIAGo has been designed to be **statically stable**, even when the arms holding its maximum payload in its most extreme kinematic configuration. Nevertheless, some measures need to be respected in order to **avoid the robot from tipping over**.

### 5.4.1 Measure 1

Do not apply external downward forces to the arm when it is extended in the direction shown in figure 25

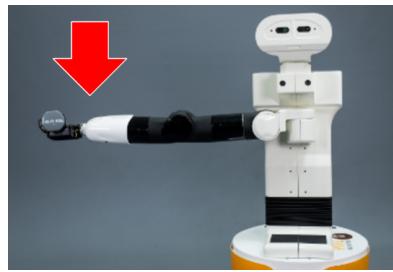


Figure 25: Fall prevention measure 1

#### 5.4.2 Measure 2

Do not navigate when the arms are extended, especially when the torso is also extended, see figure 26



Figure 26: Fall prevention measure 2

#### 5.4.3 Measure 3

TIAGo has been designed to navigate in flat floor conditions. Do not navigate on floors with unevenness higher than 5%, see figure 27.



Figure 27: Navigation in ramps is not recommended

#### 5.4.4 Measure 4

Avoid navigating close to downward stairs, as TIAGo's laser range-finder will not detect this situation and the robot may fall down the stairs.

### 5.4.5 Measure 5

In order to maintain safety, it is highly recommended to navigate 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 robot's footprint, lowering the probability that the arm collides with the environment
- Ensures the robot's stability as the center of mass is close to the center of the robot and it is kept low



Figure 28: Fall prevention measure 4

## 5.5 Measures to prevent collisions

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

### 5.5.1 Measure 1

Make sure that there are no obstacles in the robot's surroundings when playing back a predefined motion or moving the joints of the arm. Provided that the maximum reach of the arm is 86 cm without the end-effector, a safe way to move the arm is to prevent having obstacles closer than this distance, as shown in figure 29.

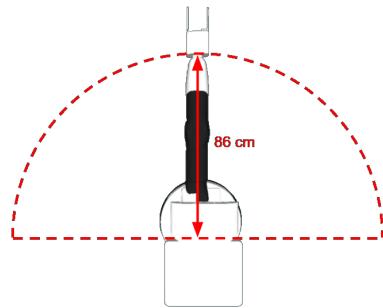


Figure 29: Area of influence of the arm

### 5.5.2 Measure 2

Another active measure that could mitigate damage due to collisions is the `collision_detector` node from the startup extras, that pretends to be an example on how to implement safety using feedforward current control. When enabled, this node monitors the current consumption of the seven joints of the arm, and

if a joint is consuming above a given threshold it stops the motion. Nevertheless, **when this node 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 abort the specific motion. Moreover, **activate the gravity compensation mode generates some noise in the current**, which makes that when going back to position control, if the **collision detector** is activated, some of the motions could be aborted due that they overpass the specific threshold. Furthermore, this safety measure is **neither available when Whole Body Control is activated**. It is important to remark, that this node **should not be used as a final safety measure**, since it just pretends to be an example for the user.

## 5.6 How to proceed when an arm collision occurs

When the arm collides with the environment, see figure 30, the motors of the arm will continue exerting force, which may cause potential damage 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-heating and over-current which switches 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 31:

- Press the emergency button to power off the robot's motors. The arm will fall down, so be ready to hold the wrist while the emergency button remains activated
- 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 safe and operational



Figure 30: Example of arm collision onto a table

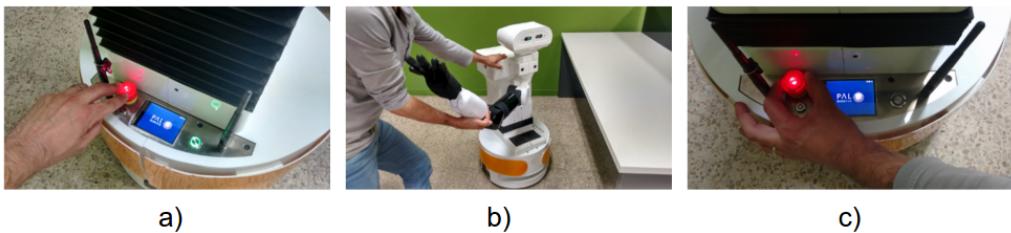


Figure 31: How to proceed when a collision occurs

## 5.7 Low battery shutdown

If the battery falls below a certain critical level, the current consumption is progressively reduced in order to make the arm fall down slowly and avoid any damage in the robot due to a blackout. Nevertheless, we

recommend the user to avoid working when the battery is very low because when the arm falls down, even if slowly, it may collide with the environment.

## 5.8 Firefighting equipment

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

If a fire occurs, please follow these instructions:

1. Call the firefighters.
2. Push the emergency stop button, as long as you can do so without any risk.
3. Only tackle a fire in its very early stages.
4. Always put your own and others' safety first.
5. Upon discovering the fire, immediately raise an alarm.
6. Make sure the exit remains clear.
7. Fire extinguishers are only suitable 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 runs out, get yourself and everyone else out of the building immediately, closing all doors behind you as you go. Then ensure the fire brigade are on their way.

## 5.9 Leakage

The battery is the only component of the robot that is able to leak. To avoid leakage of any substance from the battery, follow the instructions defined in section 3, to ensure the battery is manipulated and used correctly

**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 32) contains:

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



Figure 32: Identification label



**TIA Go**

**Default network configuration**





## 7 Default network configuration

When shipped or when a fresh re-installation is performed, the robot is configured as an access point. The information about the robot's network is provided in table 21. Note that the SSID ends with the serial number of the robot, i.e. in the given example the s/n is 0.

SSID	tiago-0
Channel	1
Mode key	WPA-PSK
Password	P@L-R0b0t1cs
Robot IP	10.68.0.1

Table 21: Access point default configuration

The address range 10.68.0.0–24 has been reserved. The robot computer name is **tiago-Xc**, where X is the serial number without the “0s” on the left. The alias **control** is also defined in order to refer to the computer name of the robot when connecting to it when it is set as access point or when using a direct connection, i.e. an Ethernet cable between the robot and the development computer.



**TIA Go**

**Software recovery**





## 8 Software recovery

### 8.1 Overview

This section explains the System and Software reinstall procedure for TIAGo .

### 8.2 Robot computer installation

To begin the installation process, plug a monitor and USB keyboard into the HDMI connector. See section 2.2.8 for information about the service panel.

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

- Turn on the robot and press F2 repeatedly. Wait until the BIOS menu appears.
- Enter *Advance Mode* by pressing F7.
- In the *Advanced > CPU Configuration* menu:
  - Set *Intel Virtualization Technology* to *Disabled*.
- In the *Advanced > CPU Configuration > CPU Power Management Configuration* menu:
  - Set *Intel(R) SpeedStep (tm)* to *Disabled*.
  - Set *CPU C-states* to *Disabled*.
- In the *Boot > Boot Configuration* menu:
  - Set *Wait for 'F1' if Error* to *Disabled*.
- Go to *Exit* and select *Save Changes & Reset*.
- Shut down the robot.

**Installation** The installation is performed using the Software USB drive provided with TIAGo.

- Insert the Software USB drive.
- Turn on the robot and press F2 repeatedly. Wait until the BIOS menu appears.
- Enter the *Boot Menu* by pressing F8 and select the Software USB drive.
- The *Language* menu will pop up. Select *English*.

The menu shown in Figure 33 appears next.

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

### 8.3 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.

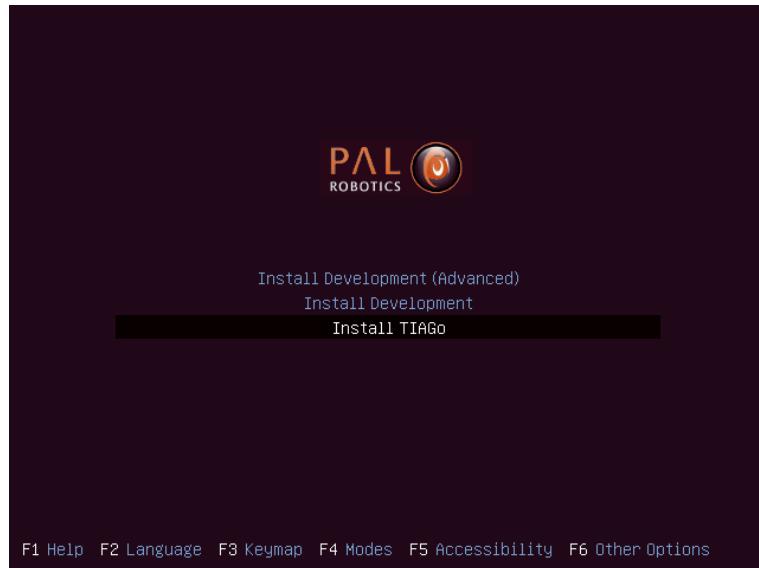


Figure 33: System installation menu

**Software Installation** The installation is performed using the Software USB drive provided with TIAGo.

- Insert the Software USB drive.
- Turn on the computer, access the BIOS and boot the Software USB drive.
- The *Language* menu will pop up. Select *English*.

The menu shown in Figure 34 will appear next.

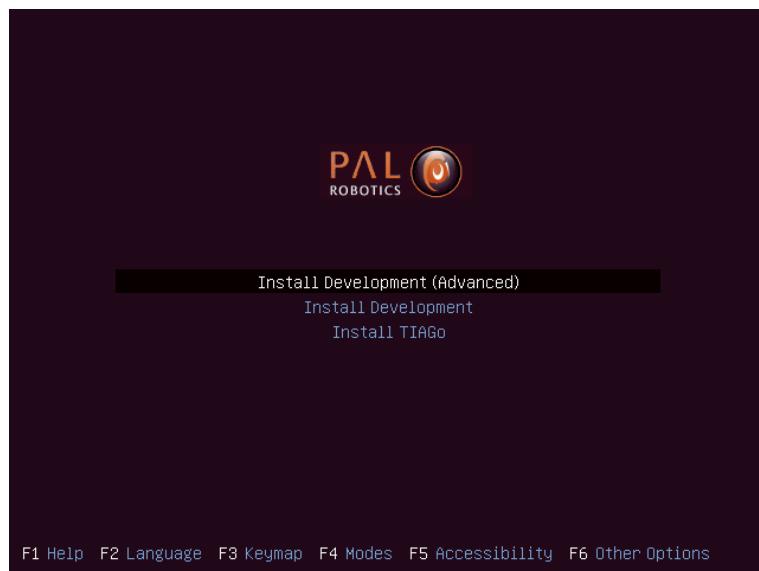


Figure 34: System installation menu

- Choose *Install Development (Advanced)* if you wish to select a target partition in the development computer. Using the “Install Development” option will **delete all partitions** in the disk and automatically install the system in a new partition.
- Select the keyboard layout by following the instructions.

**Default users** The following users are created by default:

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



**TIA Go**

**Robot computer**





## 9 TIAGo Robot's Internal Computers

### 9.1 TIAGo LAN

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

In order to connect of the robot, use ssh as follows:

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

### 9.2 Users

The users and default passwords in the TIAGo computers are:

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

### 9.3 File system

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

These partitions are created:

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

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

```
root@tiago-0c:~# rw
Remounting as rw...
Mounting /ro 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
Unbinding system files
```

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

## 9.4 Internal DNS

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

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

```
root@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, exit the command *delLocalDns*:

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

Example: `addLocalDns -u terminal`

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

## 9.5 NTP

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

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

```
server 192.168.1.6 iburst
```

Restart the ntp daemon to test your servers.

```
sudo systemctl restart ntp.service
```

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

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

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

```
sudo systemctl status ntp.service
```

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

```
server tiago-Xc iburst
```

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

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

Restart the `timesyncd` daemon.

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

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

## 9.6 System upgrade

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

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

Reboot after upgrade is complete.

## 9.7 Firmware update

To update firmware, use the application described in section 9.6. Check for updates for the `pal-ferrum-firmware-*` packages and install them.

Before running the script, place the arm in a safe position with a support underneath it, as during the installation of the script, the arm can tumble.

Run the `update_firmware.sh` script, as shown below. The update will take a few minutes.

```
pal@tiago-0c:~# rosrun firmware_update_robot update_firmware.sh
```

Finally, shut it down completely, power off with the electric switch and then power up the robot, as described in section 2.2.7.

## 9.8 Meltdown and Spectre vulnerabilities

Meltdown and Spectre exploit critical vulnerabilities in modern processors.

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

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

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

On this [website](#) the different tunables for disabling mitigation controls are displayed.

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

These changes need to be made in the persistent partition, as indicated in 9.3

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



**TIA Go**

**Development computer**





## 10 Development computer

### 10.1 Overview

The operating system used in the SDE Development Computer is based on the Linux Ubuntu 16.04 LTS distribution. Any documentation related to this specific Linux distribution applies to SDE. This document only points out how the PAL SDE differs from standard the Ubuntu 16.04.

### 10.2 Computer requirements

A computer with 8 CPU cores is recommended. A powerful graphics card with resolution of at least 1920x1080 pixels is recommended in order to have a better user experience when using visualization tools like rviz and the Gazebo simulator. The development computer ISO provides support for Nvidia cards. In case of upgrading the kernel of the development computer PAL Robotics cannot ensure proper support for other graphic cards.

### 10.3 Setting ROS environment

In order to use the ROS commands and packages provided in the development ISO the following command needs to be executed when opening a new console

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

A good way to spare the execution of this command everytime is to append it at the `/home/pal/.bashrc` file.

### 10.4 ROS communication with the robot

When developing applications for robots based on ROS, it is typical 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 vice versa. Otherwise, ROS messages will not be properly exchanged and unexpected behavior will occur.

Do the following checks before starting to work with a development computer running ROS nodes that point 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 9.4. Otherwise, you may export the environmental variable `ROS_IP` - the IP of the development computer that is visible from the robot. For example, if the robot is set as access point and the development computer is connected to it and it has been given IP 10.68.0.128 (use `ifconfig` to figure it out), use the following command in all terminals used to communicate with the robot:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
```

All ROS commands sent will then use the computer's IP rather than the hostname.

## 10.5 Compiling software

The development computer includes the ROS messages, system headers and our C++ open source headers necessary to compile and deploy software to the robot.

Some of the software APIs that we have developed are proprietary, and their headers are not included by default. If you require them you can contact us through our customer service portal and after signing a non disclosure agreement, they will be provided. These APIs are for accessing advanced features not available through a ROS API.

## 10.6 System Upgrade

In order to upgrade the software of the development computers, you have to use the `pal_upgrade_chroot.sh` command. Log in as root and execute:

```
root@development:~# /opt/pal/ferrum/lib/pal_debian_utils/pal_upgrade_chroot.sh
```

Notifications will appear whenever software upgrades are available.

## 10.7 NTP

Please follow the instructions on 9.5

**TIA Go**

**WebCommander**

**PAL** ROBOTICS

The logo for PAL Robotics features the word "PAL" in a large, orange, sans-serif font. To the right of "PAL" is the word "ROBOTICS" in a smaller, black, sans-serif font. To the right of "ROBOTICS" is a circular emblem. The emblem has a dark brown or black background with a thin orange border. Inside the circle is a stylized, glowing orange "C" shape that curves upwards and to the right, resembling a stylized eye or a flame.



## 11 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 several monitoring and configuration tasks that require a Graphical User Interface (GUI).

### 11.1 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, when the robot is acting as an access point, you can also use:

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

### 11.2 Overview

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

### 11.3 Default tabs

TIAGo comes with a set of preprogrammed tabs that are described in this section, these tabs can also be modified and extended, as explained in the 11.4 section. Each tab is an instantiation of a web commander plugin.

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

#### 11.3.1 Startup tab

**Plugin** Startup

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

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

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

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

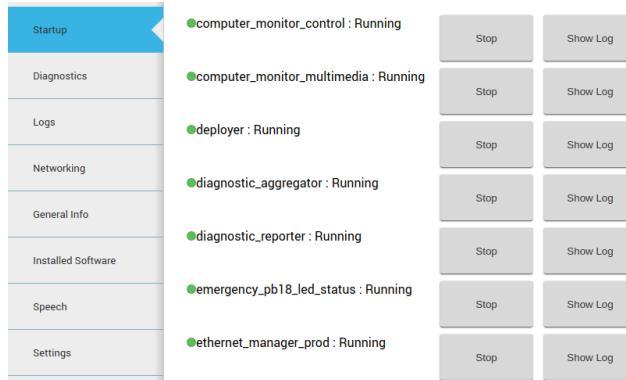


Figure 35: The Startup Tab displays the launched applications and their dependencies

### 11.3.2 Startup extras tab

**Plugin** Startup

**Description** This tab is optional, if present it will contain a list of PAL software which is not started by default during the boot up of the robot. These are optional features that need to be manually executed by the user.

### 11.3.3 Diagnostics Tab

**Plugin** Diagnostics

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

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

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

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

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

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

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

An example of this display is shown in Figure 36. The status of a particular category can be expanded by clicking on the “+” symbol to the left of the name of the category. This will provide information specific to the device or functionality. If there’s an error, an error code will be shown.

Default tabs	WebCommander
<p>The Diagnostics tab displays the status of various hardware and software components. Components listed include:</p> <ul style="list-style-type: none"> <li>Hardware : Error (Battery, Emergency Stop Button, joysticks)</li> <li>CAN buses : OK</li> <li>Cpu Control : OK</li> <li>Disk Control : OK</li> <li>Inclinometer : OK</li> <li>Laser Base : OK</li> <li>Laser Torso : OK</li> <li>Memory Control : OK</li> <li>Motor : OK (Network Control error)</li> <li>Wifi : OK</li> <li>joystick : Error (Joystick Driver Status error)</li> <li>Navigation : OK</li> <li>bumper_to_cloud : OK</li> <li>hokuyo : OK</li> <li>ir_to_cloud : OK</li> </ul>	

Figure 36: The Diagnostics tab displays the status of the hardware and software components of TIAGo

#### 11.3.4 Logs Tab

##### Plugin Logs

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

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

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

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

WebCommander		Default tabs																																																					
<table border="1"> <thead> <tr> <th>Timestamp</th> <th>Topic</th> <th>Message Content</th> </tr> </thead> <tbody> <tr><td>9/10/2013 15:25:51:791</td><td>/check_diagnostics</td><td>Looking for diagnostic name containing palCorbaDeployer-xenomai</td></tr> <tr><td>9/10/2013 15:25:48:410</td><td>/play_motion</td><td>could not get list of controllers from controller manager</td></tr> <tr><td>9/10/2013 15:25:46:422</td><td>/check_diagnostics</td><td>Looking for diagnostic name containing palCorbaDeployer-xenomai</td></tr> <tr><td>9/10/2013 15:25:43:410</td><td>/play_motion</td><td>could not get list of controllers from controller manager</td></tr> <tr><td>9/10/2013 15:25:41:115</td><td>/check_diagnostics</td><td>Looking for diagnostic name containing palCorbaDeployer-xenomai</td></tr> <tr><td>9/10/2013 15:25:38:410</td><td>/play_motion</td><td>could not get list of controllers from controller manager</td></tr> <tr><td>9/10/2013 15:25:35:809</td><td>/check_diagnostics</td><td>Looking for diagnostic name containing palCorbaDeployer-xenomai</td></tr> <tr><td>9/10/2013 15:25:32:410</td><td>/play_motion</td><td>could not get list of controllers from controller manager</td></tr> <tr><td>9/10/2013 15:25:30:503</td><td>/check_diagnostics</td><td>Looking for diagnostic name containing palCorbaDeployer-xenomai</td></tr> <tr><td>9/10/2013 15:25:27:410</td><td>/play_motion</td><td>could not get list of controllers from controller manager</td></tr> <tr><td>9/10/2013 15:25:25:197</td><td>/check_diagnostics</td><td>Looking for diagnostic name containing palCorbaDeployer-xenomai</td></tr> <tr><td>9/10/2013 15:25:21:410</td><td>/play_motion</td><td>could not get list of controllers from controller manager</td></tr> <tr><td>9/10/2013 15:25:19:890</td><td>/check_diagnostics</td><td>Looking for diagnostic name containing palCorbaDeployer-xenomai</td></tr> <tr><td>9/10/2013 15:25:15:410</td><td>/play_motion</td><td>could not get list of controllers from controller manager</td></tr> <tr><td>9/10/2013 15:25:14:584</td><td>/check_diagnostics</td><td>Looking for diagnostic name containing palCorbaDeployer-xenomai</td></tr> <tr><td>9/10/2013 15:25:10:410</td><td>/play_motion</td><td>could not get list of controllers from controller manager</td></tr> <tr><td>9/10/2013 15:25:00:278</td><td>/check_diagnostics</td><td>Looking for diagnostic name containing palCorbaDeployer-xenomai</td></tr> </tbody> </table>		Timestamp	Topic	Message Content	9/10/2013 15:25:51:791	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai	9/10/2013 15:25:48:410	/play_motion	could not get list of controllers from controller manager	9/10/2013 15:25:46:422	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai	9/10/2013 15:25:43:410	/play_motion	could not get list of controllers from controller manager	9/10/2013 15:25:41:115	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai	9/10/2013 15:25:38:410	/play_motion	could not get list of controllers from controller manager	9/10/2013 15:25:35:809	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai	9/10/2013 15:25:32:410	/play_motion	could not get list of controllers from controller manager	9/10/2013 15:25:30:503	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai	9/10/2013 15:25:27:410	/play_motion	could not get list of controllers from controller manager	9/10/2013 15:25:25:197	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai	9/10/2013 15:25:21:410	/play_motion	could not get list of controllers from controller manager	9/10/2013 15:25:19:890	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai	9/10/2013 15:25:15:410	/play_motion	could not get list of controllers from controller manager	9/10/2013 15:25:14:584	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai	9/10/2013 15:25:10:410	/play_motion	could not get list of controllers from controller manager	9/10/2013 15:25:00:278	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai
Timestamp	Topic	Message Content																																																					
9/10/2013 15:25:51:791	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai																																																					
9/10/2013 15:25:48:410	/play_motion	could not get list of controllers from controller manager																																																					
9/10/2013 15:25:46:422	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai																																																					
9/10/2013 15:25:43:410	/play_motion	could not get list of controllers from controller manager																																																					
9/10/2013 15:25:41:115	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai																																																					
9/10/2013 15:25:38:410	/play_motion	could not get list of controllers from controller manager																																																					
9/10/2013 15:25:35:809	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai																																																					
9/10/2013 15:25:32:410	/play_motion	could not get list of controllers from controller manager																																																					
9/10/2013 15:25:30:503	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai																																																					
9/10/2013 15:25:27:410	/play_motion	could not get list of controllers from controller manager																																																					
9/10/2013 15:25:25:197	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai																																																					
9/10/2013 15:25:21:410	/play_motion	could not get list of controllers from controller manager																																																					
9/10/2013 15:25:19:890	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai																																																					
9/10/2013 15:25:15:410	/play_motion	could not get list of controllers from controller manager																																																					
9/10/2013 15:25:14:584	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai																																																					
9/10/2013 15:25:10:410	/play_motion	could not get list of controllers from controller manager																																																					
9/10/2013 15:25:00:278	/check_diagnostics	Looking for diagnostic name containing palCorbaDeployer-xenomai																																																					

Figure 37: The Log Tab displays the log messages as they are being published in the robot

### 11.3.5 General Info Tab

**Plugin** General Info

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

### 11.3.6 Installed Software Tab

**Plugin** Installed Software

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

### 11.3.7 Settings Tab

**Plugin** Settings

**Description** The settings tab allows to change the behaviour of TIAGo .

Currently it allows to configure the language of TIAGo for speech synthesis. It is possible to select one from a drop down list. Changing the text-to-speech language will change the default language when sending sentences to be spoken by TIAGo (see Section 19 for further details).

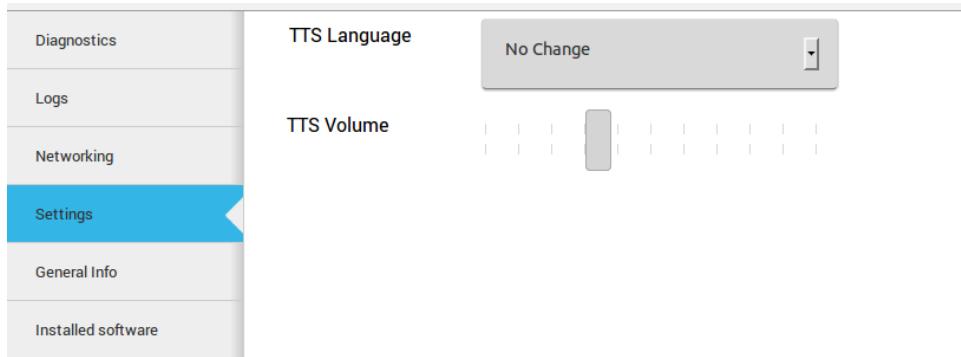


Figure 38: The Settings tab allows to modify the behaviour of TIAGo

**Hardware Configuration** The Settings tab allows the user to configure the hardware of the robot. Hardware configuration will let the user to disable/enable the different motors, enable/disable the Arm module, choose different End Effector configuration, also also enable/disable the mounted F/T sensor.

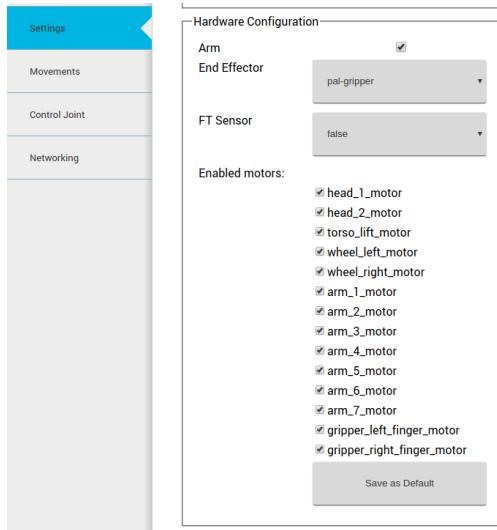


Figure 39: TIAGo Hardware Configuration

For instance, to disable the “head\_1\_motor”, untick the head\_1\_motor checkbox in the “Enabled motors” options. If you want to switch to a different end-effector, then in the “End Effector” drop down, select the end effector that you are going to install, and click the “Save as Default” button at the bottom of the section. Reboot the robot for the above selected configuration to be taken into effect.

**Remote Support** The Settings tab is equipped with the remote support connection widget. A technician from PAL Robotics can give remote assistance to the robot by connecting through this widget. Using an issue in the support portal, the PAL technician will provide the IP Address and the Port, this information need to be filled in the respective fields of the widget and then pressing the **Connect** button will allow for the remote assistance. If the robot needs to be rebooted, the customer has to activate the remote support after each reboot because it is not persistent.

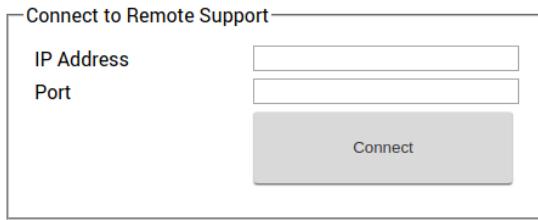


Figure 40: Remote support widget for TIAGo

At any point of time after the connection had established, the remote connection can be terminated by clicking the **Disconnect** button.

**Note:** After clicking the **Connect**, if the widget pops back to the normal, instead of showing the connection status, then it means that the robot is either not connected to internet (or) there should be some network issue.

### 11.3.8 Networking tab

**Plugin** NetworkingEmbedded

**Description** Figure 41 shows the networking tab. By default, the controls for changing the configuration are not visible in order to avoid access by multiple users.

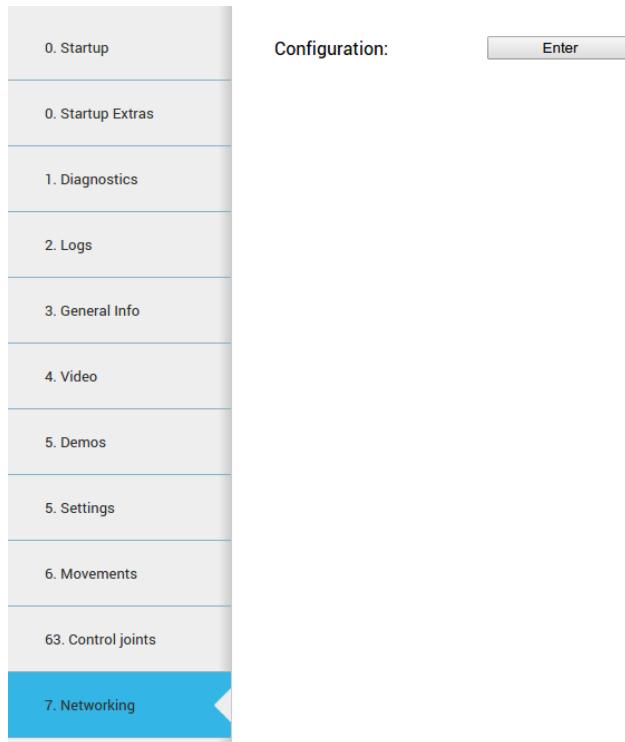


Figure 41: Networking configuration

If the *Enter* button is pressed, the tab connects to the network configuration system and the controls shown in figure 42 will appear.

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

The screenshot shows the 'Networking' configuration page in the WebCommander interface. The left sidebar lists various configuration sections: 0. Startup, 0. Startup Extras, 1. Diagnostics, 2. Logs, 3. General Info, 4. Video, 5. Demos, 5. Settings, 6. Movements, 63. Control joints, and 7. Networking. The '7. Networking' tab is currently active and highlighted in blue. The main content area is titled 'Configuration' and contains several configuration blocks:

- Wifi:** Includes fields for Mode (Client), SSID (rlan), Channel (1), Mode Key (WPA-PSK), and Password (\*\*\*\*\*).
- Ethernet:** Includes a Mode dropdown set to 'Internal'.
- IPv4:** Includes checkboxes for 'Enable DHCP Wifi' (checked) and 'Enable DHCP Ethernet' (unchecked), and fields for Address (192.168.1.212), Network (255.255.255.0), and Gateway (192.168.1.1).
- DNS:** Includes fields for Server (192.168.1.1), Domain (reem), and Search (reem).
- NTP:** Includes a Server field set to 192.168.1.1.
- VPN:** Includes checkboxes for 'Enable VPN' and 'Enable Firewall', and fields for Address (192.168.1.6) and Port (1194).

At the bottom right are 'Apply change' and 'Save' buttons.

Figure 42: Networking configuration controls

Configurations are separated in different blocks:

- WiFi:
  - Mode: Can be selected whether WiFi 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 the robot is in access point mode, use this channel.
  - Mode Key: Encryption of the connection. For more specific configurations select *manual*. In this case it is used the file */etc/wpa\_supplicant.conf.manual* that can be manually created in the robot.
  - Password: Password for the WiFi connection
- Ethernet:
  - Mode: Can be selected whether the ethernet connection works as an internal LAN or external connection(see Section 2.3.2).
- IPv4
  - Enable DHCP Wifi: Enables DHCP client in WiFi interface.
  - Enable DHCP Ethernet: Enables DHCP client in the external ethernet port.
  - Address, Network, Gateway: In client mode, the manual values of the building's network are used by the Wi-Fi interface. This is the same for the external ethernet port.
- DNS

- Server: DNS server.
- Domain: Domain to use in the robot.
- Search: Domain to use in the search.
- VPN
  - Enable VPN: If the customer has a PAL basestation, the robot can be connected to the customer's VPN.
  - Enable Firewall: When activating the VPN, a firewall can be connected to avoid an incoming connection from outside the VPN.
  - Address: Building network IP address of the basestation.
  - Port: Port of the basestation where the VPN server is listening.

No changes are set until the *Apply change* button is pressed.

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

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

Changes to the WiFi between client and access point could require a reboot of the computer in order to be correctly applied.

Using the diagnostic tab, it is possible to see the current state of the WiFi connection.

**Connecting TIAGo to a LAN** In order to connect TIAGo to your own LAN follow the steps below.

First of all you need to access the WebCommander via the URL <http://tiago-0c:8080> and go to the Networking tab. Press *Enter* button and then follow the instructions shown in figure 43.

Once you have filled in the right configuration and pressed the **Apply change** button it is **very important** to wait until you are able to ping the new robot IP in your own LAN. If it does not happen you might have to **reboot** the robot as the configuration changes have not been saved yet. The robot will reboot with its previous networking configuration, allowing you to repeat the process properly.

When the new configuration allows you to detect the robot in your own LAN then you may proceed to enter the WebCommander again and press the *Save* button and then the *Confirm* button.

**Setting TIAGo as an Access Point** In order to configure TIAGo as access point open the WebCommander via the URL <http://tiago-0c:8080> and go to the Networking tab. Press *Enter* button and then follow the instructions shown in figure 44.

Once you have filled in the right configuration and pressed the **Apply change** button it is **very important** to wait until the new Wi-Fi network is detected. A smartphone, a tablet or a computer provided with a WiFi card can be used for this purpose. If it does not happen you might have to **reboot** the robot as the configuration changes have not been saved yet. The robot will reboot with its previous networking configuration, allowing you to repeat the process properly.

When the new configuration allows you to detect the robot's Wi-Fi then you may proceed to enter the WebCommander after connecting to the Wi-Fi of the robot and press the *Save* button and then the *Confirm* button.

### 11.3.9 Video Tab

**Plugin** Video

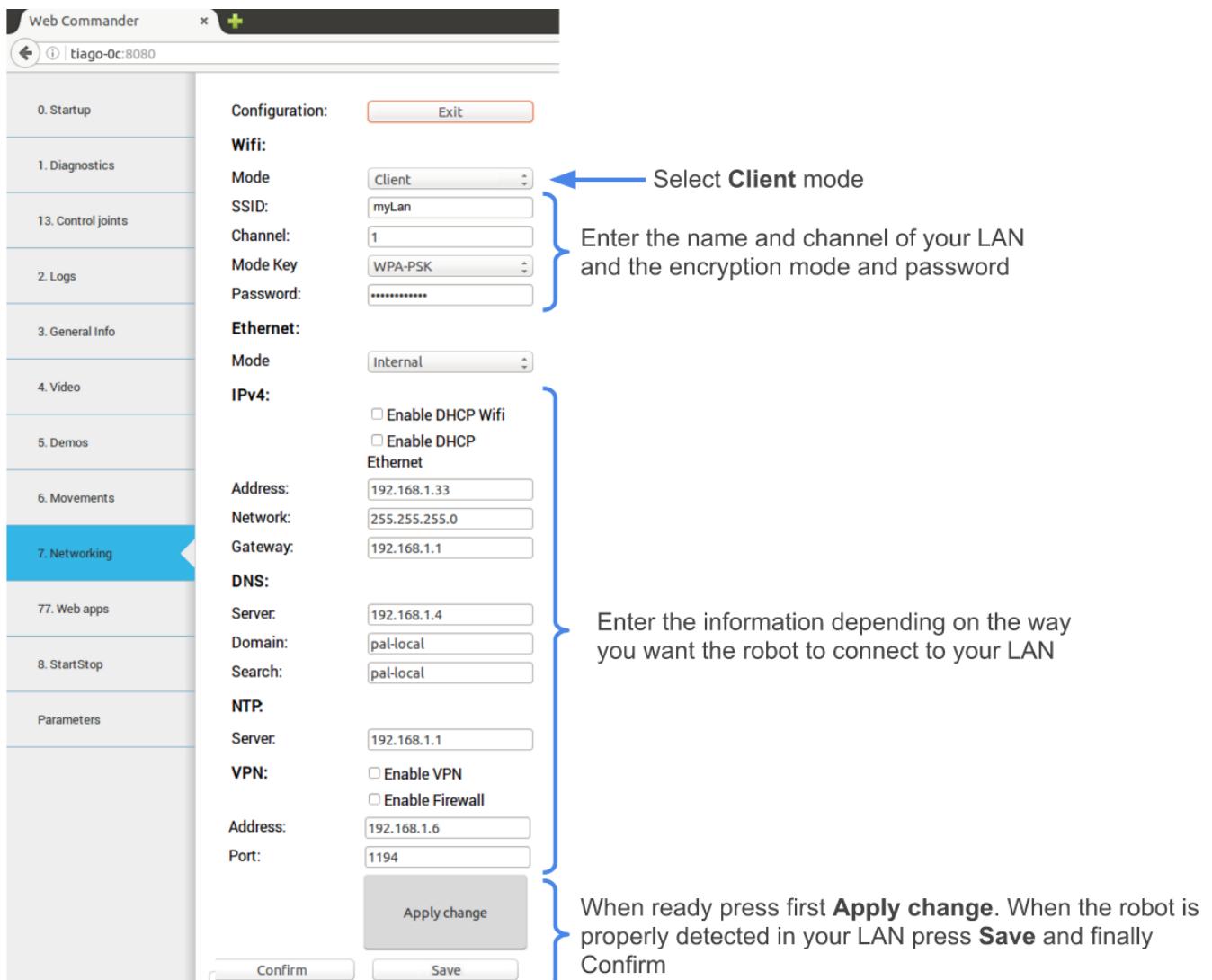


Figure 43: Configuring TIAGO to connect to a LAN

**Description** Displays the images from a ROS topic in the WebCommander

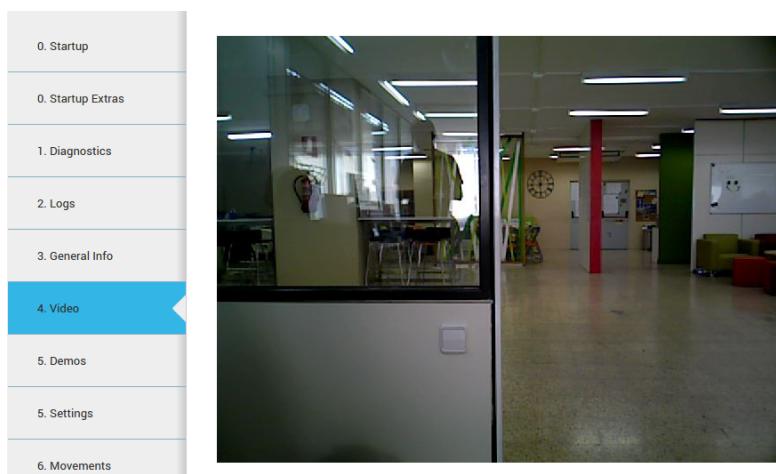


Figure 45: The Video Tab displays live video stream from the robot's camera

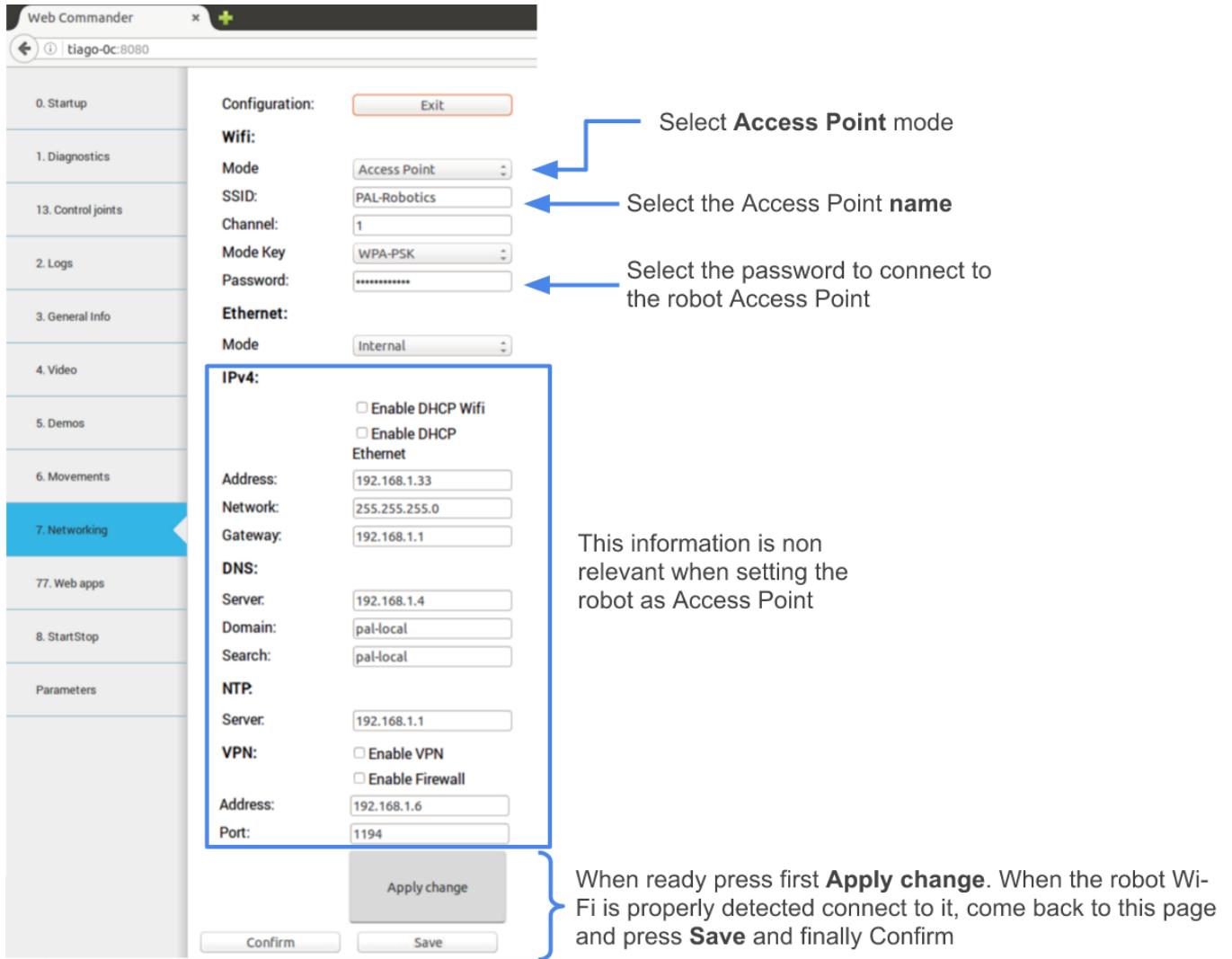


Figure 44: Configuring TIAGO as Access Point

### 11.3.10 Movements Tab

#### Plugin Movements

**Description** Enables playing pre-recorded motions on TIAGO .

The movement tab that can be seen in figure 46 allows a user to send upper body motion commands to the robot. Clicking on a motion will execute it immediately in the robot. Make sure the arms have enough room to move before sending a movement, to avoid possible collisions.

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

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

To disable these safety features, the “Safety is enabled” button must be clicked, it will then turn red and the next movement command sent will not take into account the sensors information. For security reasons, the

disable safety will only affect the next movement command sent. Sending multiple unsafe movements requires pressing the “Safety is enabled” button once before each command.

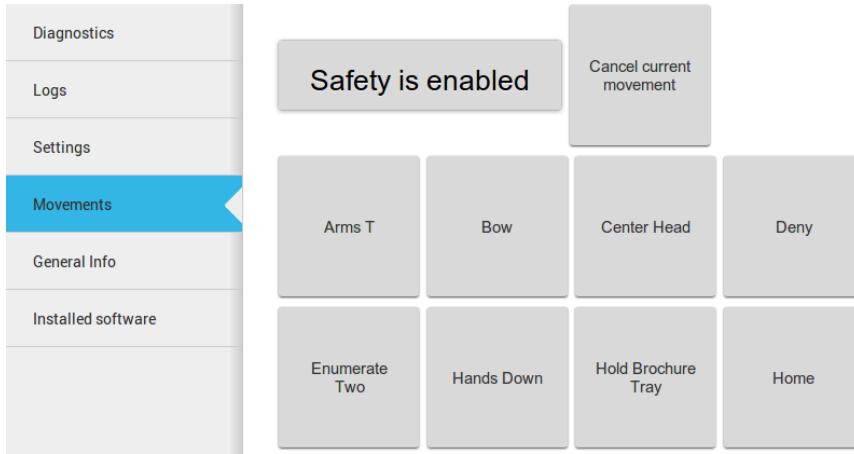


Figure 46: The Movement tab allows to send upper body motions to TIAGo

### 11.3.11 Demos tab

#### Plugin Commands

**Description** This tab provides buttons to start and stop different demos and synthesize voice messages with TIAGo, using either predefined buttons or by typing new messages in a text box.

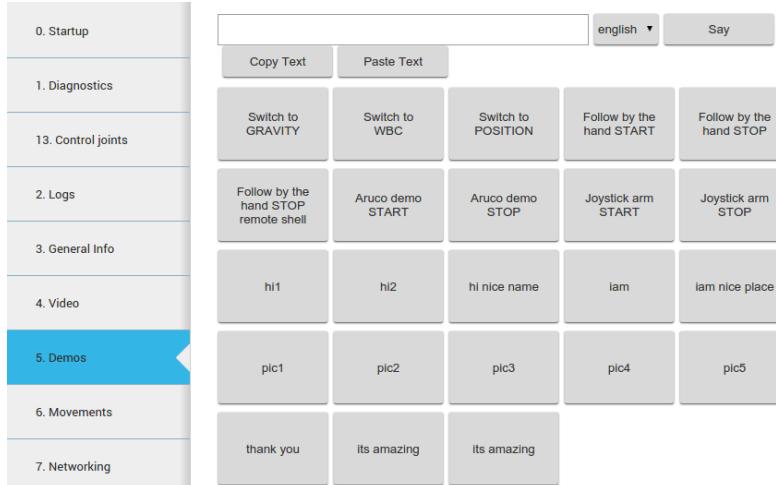


Figure 47: The demos tab provides buttons to start/stop demos and synthesize voice messages

## 11.4 Tab configuration

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

### 11.4.1 Parameter format

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

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

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

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

### 11.4.2 Startup Plugin Configuration

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

#### Parameters

**startup\_ids** A list of strings that contains the startup groups handled by the instance of the plugin. See section 14.1.3 on page 98.

### 11.4.3 Diagnostics Plugin Configuration

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

**Parameters** None required

#### 11.4.4 Logs Plugin Configuration

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

**Parameters** None required

#### 11.4.5 JointCommander Plugin Configuration

**Description** This tab provides slides to move each joint of TIAGo 's upper body.

**Parameters** A list of the joints groups to be controlled. Each element of the list must be a dictionary containing "name", "command\_topic" and "joints". Where "name" is the name that will be displayed for this group, "command\_topic" is the topic where the joint commands will be published and "joints" is a list containing the joint names to be commanded.

```
"8. Control Joint":  
    type: "JointCommander"  
    parameters:  
        - name: Arm  
          command_topic: /arm_controller/safe_command  
          joints: [arm_1_joint, arm_2_joint, arm_3_joint, arm_4_joint, arm_5_joint]  
        - name: Torso  
          command_topic: /torso_controller/safe_command  
          joints: [torso_lift_joint]  
        - name: Head  
          command_topic: /head_controller/command  
          joints: [head_1_joint, head_2_joint]
```

Example:

#### 11.4.6 General Info Plugin Configuration

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

**Parameters** None required

#### 11.4.7 Installed Software Plugin Configuration

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

**Parameters** None required

#### 11.4.8 Settings Plugin Configuration

**Description** The settings tab allows to change the behaviour of TIAGo .

**Parameters** None required

#### 11.4.9 Networking Embedded Plugin Configuration

**Description** This tab allows to change the network configuration.

**Parameters** None required

#### 11.4.10 Video Plugin Configuration

**Description** Displays the images from a ROS topic in the WebCommander

**Parameters**

**topic** Name of the topic to read images from, for instance: /xtion/rgb/image\_raw/compressed

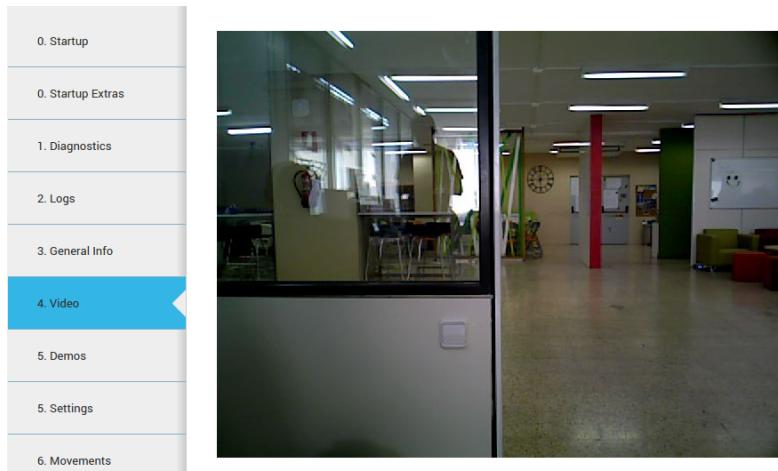


Figure 48: The Video Tab displays live video stream from the robot's camera

#### 11.4.11 Movements Plugin Configuration

**Description** Enables playing pre-recorded motions on TIAGo .

**Parameters**

**goal\_type** Either “play\_motion” or “motion\_manager”. Determines which action server will be used for sending the motions.

#### 11.4.12 Commands Plugin Configuration

**Description** Contains buttons that can be programmed through parameters to perform actions in the robot.

## Parameters

**buttons** A list of buttons, where each button is a dictionary with 2 fields. The `name` field is the text displayed on the button, and the second field `name` determines the type of button and is a dictionary with the configuration of the button.

```
wt:
    "Example Buttons":
        type: "Commands"
        parameters:
            buttons:
                - name: "Say some text"
                    say:
                        text: "This is the text that will be said"
                        lang: "en_GB"
                - name: "Greetings"
                    say_tts:
                        section: "macro"
                        key: "greetings"
                        lang: ""
                - name: "Wave"
                    motion:
                        name: "wave"
                        safe: True
                        plan: True
                - name: "Change to Localization"
                    remote_shell:
                        cmd: "rosservice call /pal_navigation_sm \"input: 'LOC'\""
                        target: "control"
```

There are 4 types of buttons: `say`, `say_tts`, `motion` and `remote_shell`

**say** Sends a text to the Text-To-Speech engine. It requires a `text` field containing the text to be said, and `lang` containing the language in the format `language_country` specified in the RFC 3066 .

**say\_tts** Sends text as a section/key pair to the Text-To-Speech engine. It requires a `section` and `key` field as specified in Section 19.2.2. The `lang` field can be left empty, but if it is specified it must be in the RFC 336 format.

**motion** Sends a motion to the motion manager engine. Requires a `name` field specifying the name of the motion, and two boolean fields `plan` and `safe` that determine to check for self-collisions and collisions with the environment respectively. For safety reasons they should always be set to True.

**remote\_shell** Enables the execution of a bash command in one of the robot's computers. Requires a `cmd` field containing a properly escaped, single line bash command, and a `target` field that can either be `control` or `multimedia`, indicating to execute the command in the control computer or in the multimedia computer of the robot.

Both **say** and **say\_tts** require that the robot is running PAL Robotics' TTS software.



**TIA Go**

**Software architecture**





## 12 Software architecture

### 12.1 Overview

The software provided in TIAGo is summarized in figure 49.

- Operating system:

Ubuntu LTS 64-bit



RT-Preempt



real-time control  
software

- Robotics middleware:

ROS LTS



PAL distribution



ROS packages developed by PAL Robotics

Figure 49: Tiago software summary

### 12.2 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 Orocoss for real-time, safe communication between processes.

### 12.3 ROS layer

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

- Packages belonging to the official ROS distribution melodic.
- Packages specifically developed by PAL Robotics, which are included in the company's own distribution, called ferrum.
- Packages developed by the customer.

The three categories of packages are installed in different locations of the SSD, as shown in figure 50. The ROS melodic packages and PAL ferrum packages are installed in a read-only partition as explained in section 9.3. 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 13. The same deployment tool can be used to install ROS packages in the user space.

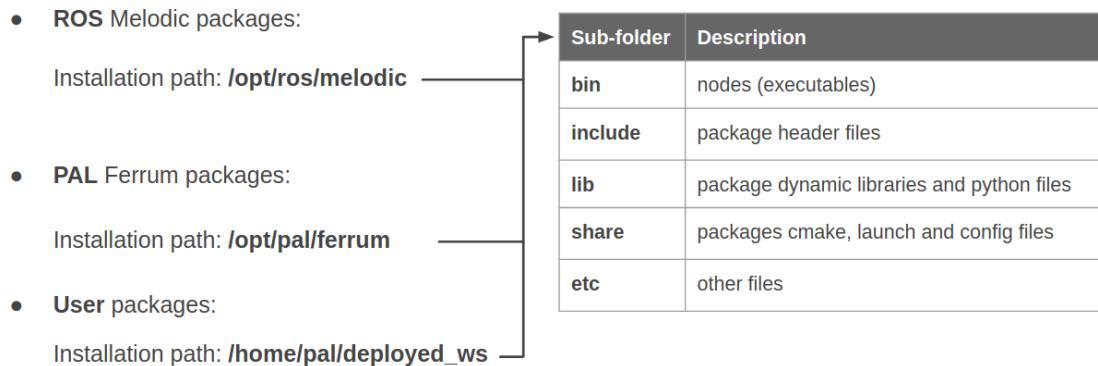


Figure 50: PAL Software overlay structure

## 12.4 Software startup process

When the robot boots up, the software required for its operation starts automatically. The startup process can be monitored in the WebCommander, as shown in figure 51.

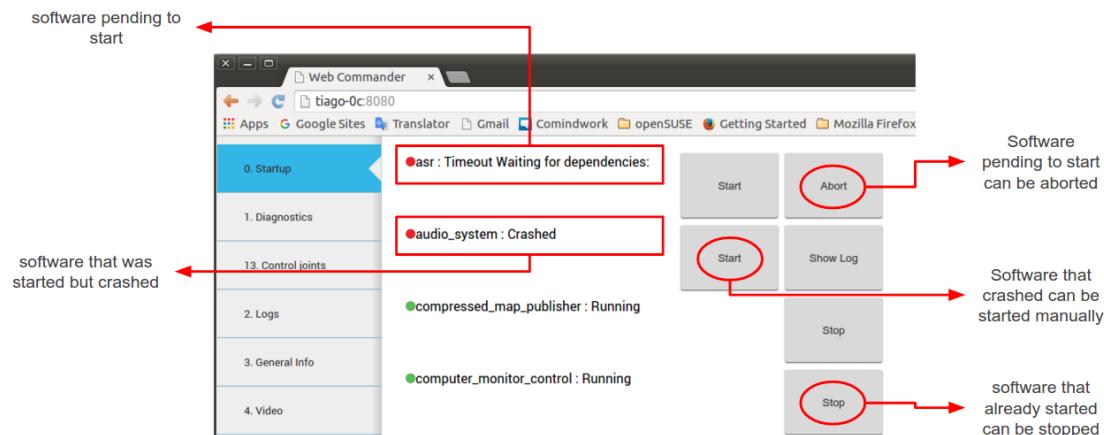


Figure 51: The startup tab displays the launched applications and their dependencies

**TIA Go**

## Deployment





## 13 Deploying software on the robot

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

The `deploy` tool can be used to:

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

### 13.1 Introduction

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

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

### 13.2 Usage

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

Deploy built packages to a robot. The default behavior is to deploy *all*
packages from any found workspace. Use --package to only deploy a single package.

positional arguments:
  robot                hostname to deploy to (e.g. tiago-0c)

optional arguments:
  -h, --help            show this help message and exit
  --user USER, -u USER  username (default: pal)
  --yes, -y             don't ask for confirmation, do it
  --package PKG, -p PKG deploy a single package
  --install_prefix INSTALL_PREFIX, -i INSTALL_PREFIX
                        Directory to deploy files
  --cmake_args CMAKE_ARGS, -c CMAKE_ARGS
                        Extra cmake args like
                        --cmake_args="-DCMAKE_BUILD_TYPE=Release"
e.g.: deploy.py tiago-0c -u root -p pal_tts -c="-DCMAKE_BUILD_TYPE=Release"
```

### 13.3 Notes

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

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

## 13.4 Deploy tips

- You can use an alias ( you may want to add it to your `.bashrc`) to ease the deploy process:  
`alias deploy="rosrun pal_deploy deploy.py"`
- You can omit the `--user pal` as it is the default argument
- You may deploy a single specific package instead of the entire workspace:  
`deploy -p hello_world tiago-0c`
- You can deploy multiple specific packages instead of the entire 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 be able to return to your previous state, if required.

## 13.5 Use-case example

### 13.5.1 Adding a new ROS Package

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

```
source /opt/pal/ferrum/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 52.

```
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 52: Hello world CMakeLists.txt

Edit the `src/hello_world_node.cpp` file with the contents in figure 53.

```
// 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 53: Hello world C++ source code

### Build the workspace

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

The expected output is shown in figure 54.

```
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 54: Build output of hello world package

### Deploy the package to the robot:

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

The deploy tool will build the entire workspace in a separate path and, if successful, it will request confirmation in order to install the package on the robot, as shown in figure 55.

```
[clean] No buildspace exists, no CMake caches to clear.
Preparing install space
Creating build space directory, '/home/pal/example_ws/build_pal_deploy'

Profile:                                pal_deploy
Extending:      [env] /home/pal/example_ws/devel:/opt/pal/dubnium:/opt/ros/indigo
workspace:     [exists] /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=/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 55: Deployment of hello world package

Press **Y** so that the package files are installed on the robot computer. Figure 56 shows the files that are copied for the hello world package, according to the installation rules specified by the user in the `CMakeLists.txt`.

```
Syncing binaries with robot...
The authenticity of host 'tiago-0c (192.168.1.33)' can't be established.
RSA key fingerprint is 4c:0e:17:4c:39:48:f7:a:f: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 56: Installation of the hello world package to the robot

Then connect to the robot:

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

And run the new node as follows:

```
rosrun hello_world hello_world_node
```

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

### 13.5.2 Adding a new controller

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

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

```
export ROS_MASTER_URI=http://tiago-0c:11311
rosservice call /controller_manager/list_controller_types | grep HelloController
```

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

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

```
cd tiago_ws
catkin_make # -j5 #optional
source devel/setup.bash # to get this workspace into the development environment
rosrun pal_deploy deploy.py --package ros_controllers_tutorials tiago-0c
```

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

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

```
export ROS_MASTER_URI=http://tiago-0c:11311
rosservice call /controller_manager/list_controller_types | grep HelloController
```

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

```
/reem_ws$ rosservice call /controller_manager/list_controller_types | grep HelloController
types: ['controller_manager_tests/EffortTestController', 'controller_manager_tests/MyDummyController', 'diff_drive_controller/DiffDriveController', 'effort_controllers/GripperActionController', 'effort_controllers/JointEffortController', 'effort_controllers/JointPositionController', 'effort_controllers/JointTrajectoryController', 'effort_controllers/JointVelocityController', 'force_torque_sensor_controller/ForceTorqueSensorController', 'imu_sensor_controller/IMuSensorController', 'joint_state_controller/JointStateController', 'pal_ros_controllers/CurrentLimitController', 'position_controllers/GripperActionController', 'position_controllers/JointPositionController', 'position_controllers/JointTrajectoryController', 'reemc_tutorial_controllers/CombinedResourceController', 'reemc_tutorial_controllers/ForceTorqueController', 'reemc_tutorial_controllers/HelloController', 'reemc_tutorial_controllers/ImuController', 'reemc_tutorial_controllers/JointController', 'velocity_controllers/JointVelocityController', 'walking_force_control/ArmTorqueControlComand', 'walking_force_control/CartesianControlAccelerationBasedHsu', 'walking_force_control/CartesianControlAccelerationBasedHsuThreeDOF', 'walking_force_control/CartesianControlAccelerationBasedSimplifiedNoPreM', 'walking_force_control/CartesianControlAccelerationBasedSimplifiedPreM', 'walking_force_control/CartesianControlForceBasedGauss', 'walking_force_control/CartesianControlImpedanceOtt', 'walking_force_control/CartesianControlVelocityBasedNakamura', 'walking_force_control/GravityCompensation', 'walking_force_control/InverseDynamics', 'walking_force_control/LocalJointInverseDynamics', 'walking_force_control/LocalJointPID', 'walking_force_control/LocalJointPIDGravity', 'walking_force_control/OldArmGravityCompensationControl', 'walking_force_control/TorqueControlComand', 'wbc/WholeBodyControlDynamicController', 'wbc/WholeBodyControlKinematicController', 'wbc/WholeBodyControlKinematicControllerRPC']
```

Figure 57: List of controller types

### 13.5.3 Modifying an installed package

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

Go to [https://github.com/ros-controls/ros\\_controllers](https://github.com/ros-controls/ros_controllers), open a new terminal and execute the following commands:

```
cd tiago_ws/src
git clone https://github.com/ros-controls/ros_controllers
# Fix bugs in controller
cd ..
catkin_make # -j5 #optional
source devel/setup.bash # to get this workspace into the development environment
rosrun pal_deploy deploy.py --package joint_state_controller tiago-0c
```

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



**TIA Go**

**Startup**





## 14 Modifying Robot Startup

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

### 14.1 Introduction

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

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

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

#### 14.1.1 Application start configuration files

These files are placed inside the `apps` directory within `config`.

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

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

The required attributes are:

- One of `roslaunch`, `rosrun` or `bash`: used to determine how to start the application. The value of `roslaunch`, `rosrun` or `bash` is the rest of the commands that you would use in a terminal (you can use bash magic inside such as '`rospack find my_cfg_dir`'). There are also some keywords that are replaced by the robot's information in order to make scripts more usable. `@robot@` is replaced by the robot name as used in our ROS packages (ie REEMH3 is `reem`, REEM-C is `reemc`, ...)
- `dependencies`: a list of dependencies that need to be running without error before starting this application. Dependencies can be seen in the diagnostics tab on page 68. If an application has no dependencies, it should be set to an empty list `[]`.

Optional attributes:

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

Examples:

`localization.yaml`

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

`web_commander.yaml`

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

### 14.1.2 Computer start lists

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

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

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

Examples:

`pal_startup_base/config/control/core.yaml`

```
# Core
- ros_bringup
- diagnostic_aggregator
- web_commander

# Deployers
- deployer_xenomai

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

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

### 14.1.3 Additional startup groups

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

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

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

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

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

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

## 14.2 Startup ROS API

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

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

**/pal\_startup\_{startup\_id}/stop** Arguments are **app** (name of the application as written YAML files for the application start configuration files). Returns a string containing if the app was stopped successfully.

**/pal\_startup\_{startup\_id}/get\_log** Arguments are **app** (name of the application as written YAML files for the application start configuration files) and **nlines** (number of lines of the log file to return). Returns up to the last nlines of logs generated by the specified app.

**/pal\_startup\_{startup\_id}/get\_log\_file** Arguments are **app** (name of the application as written YAML files for the application start configuration files). Returns the path of the log file of the specified app.

## 14.3 Startup command line tools

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

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

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

## 14.4 ROS Workspaces

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

- /home/pal/deployed\_ws (see 13)
- /opt/pal/ferrum
- /opt/ros/melodic

## 14.5 Modifying the robot's startup

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

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

### 14.5.1 Adding a new application for automatic startup

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

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

For instance:

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

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

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

```
- new_app
```

### 14.5.2 Modifying how an application is launched

To modify how the application “foo\_bar” is launched, copy the contents from the original foo\_bar.yaml file in the pal\_startup\_base package and perform the desired modifications.

### 14.5.3 Adding a new workspace

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

**TIA Go**

**Sensors**

**PAL** ROBOTICS

The logo for PAL Robotics features the word "PAL" in a large, orange, sans-serif font. To the right of "PAL" is the word "ROBOTICS" in a smaller, black, sans-serif font. To the right of "ROBOTICS" is a circular emblem. The emblem has a dark brown or black background with a thin orange border. Inside the circle, there is a stylized, glowing orange and white graphic that resembles a eye or a sensor array.



## 15 Sensors

This section contains an overview of the sensors included in TIAGo, as well as their ROS and C++ API.

### 15.1 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 a 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 the signal that returns to the sensor. Bad measurements can be caused by either blocking the sensor (it 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 provide the attitude.

- Head:

**RGB-D camera** This camera is mounted inside TIAGo's head 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** There are 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 TIAGo's end-effector.

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

## 16 Sensors ROS API

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

### 16.1 Laser range-finder

#### 16.1.1 Topics published

/scan ([sensor\\_msgs/LaserScan](#))

Laser scan data of the laser scanner.

## 16.2 Sonars

### 16.2.1 Topics published

/sonar\_base ([sensor\\_msgs/Range](#))

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

## 16.3 Inertial Measurement Unit

### 16.3.1 Topics published

/base\_imu ([sensor\\_msgs/Imu](#))

Inertial data from the IMU.

## 16.4 RGB-D camera

### 16.4.1 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.

### 16.4.2 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.

## 16.5 Force/Torque sensor

### 16.5.1 Topics published

/wrist\_ft ([geometry\\_msgs/WrenchStamped](#))

Force and torque vectors currently detected by the Force/Torque sensor.

## 17 Sensor visualization

Most of TIAGo's sensor readings can be visualized in rviz. In order to start the rviz GUI with a pre-defined configuration, execute the following from the development computer:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun rviz rviz -d `rospack find tiago_bringup`/config/tiago.rviz
```

Make sure to use your robot's serial number when exporting the `ROS_MASTER_URI` variable and to set your development computer's IP when exporting `ROS_IP` as explained in Section 10.4.

An example of how the laser range-finder is visualized in rviz is shown in figure 58.

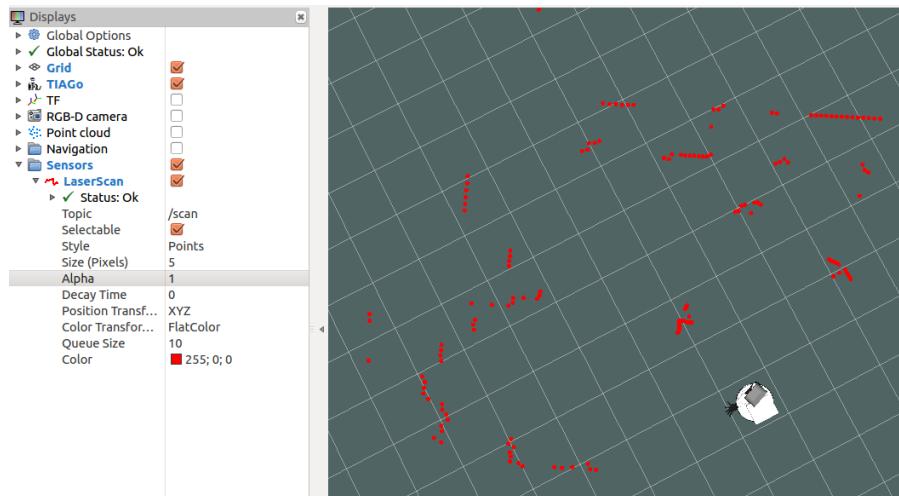


Figure 58: Visualization of the laser range-finder

Figure 59 shows an example of visualization of the RGB image, depth image and point cloud provided by the RGB-D camera.

Finally, figure 60 presents an example of force vector detected by the force/torque sensor.

## Sensor visualization

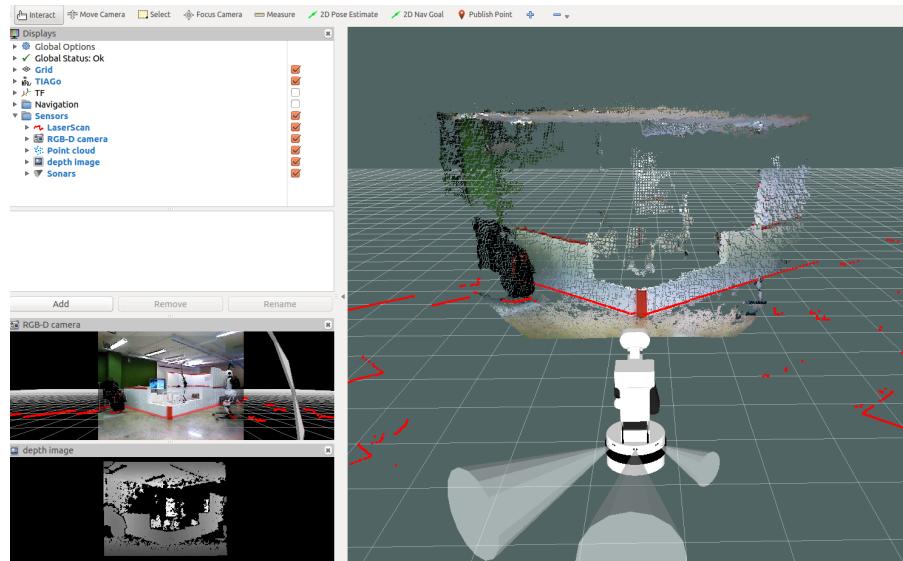


Figure 59: Visualization of the RGB-D camera



Figure 60: Visualization of force/torque sensor

**TIA Go**

**Power status**

**PAL**   
ROBOTICS



## 18 Power status

This section contains an overview of the power-related status data reported by TIAGo, as well as the ROS API and a brief description of the information available.

### 18.1 ROS API

The robot's power status is reported in the `/power_status` ROS topic.

*NOTE: This node is launched by default on startup.*

### 18.2 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**





## 19 Text-to-Speech synthesis

### 19.1 Overview of the technology

TIAGo incorporates the Acapela Text-to-Speech from [Acapela Group](#). 

The technology used in this engine is the one that leads the market of synthetic voices. It is based on unit selection and allows to produce highly natural speeches in formal styles. The system is able to generate speech output, based on a input text utterance<sup>1</sup>. It does the phonetic transcription of the text, predicts the appropriate prosody for the utterance and finally generates the signal waveform.

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

### 19.2 Text-to-Speech node

#### 19.2.1 Launching the node

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

System diagnostics described in Section 11 allow to check the status of the TTS service running in the robot. These services are started by default on start-up, so normally there is no need to start them manually. To start/stop them, the following commands can be executed in a terminal opened in the multimedia computer of the robot:

```
pal-start sound_server
pal-stop sound_server
```

#### 19.2.2 Action interface

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

- Goal definition fields:

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

- Result definition fields:

```
string text
string msg
```

- Feedback message:

---

<sup>1</sup>In spoken language analysis an utterance is a smallest unit of speech. It is a continuous piece of speech beginning and ending with a clear pause.

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

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

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

### Sending a raw text goal

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

```
string text
string lang_id
```

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

### Sending a I18nText goal

The `text` field of type `I18nText` has the following format:

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

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

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

In the ROS [manual](#) you can find examples about how to create an action client that uses these message definitions to generate a speech in TIAGo .

## 19.3 Examples of usage

### 19.3.1 WebCommander

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

Additionally buttons can be programmed to say predefined sentences, see the 11.4.12 for details.

Several buttons corresponding to different predefined sentences are provided in the lower part of the `Demos` tab, as shown in figure 61.

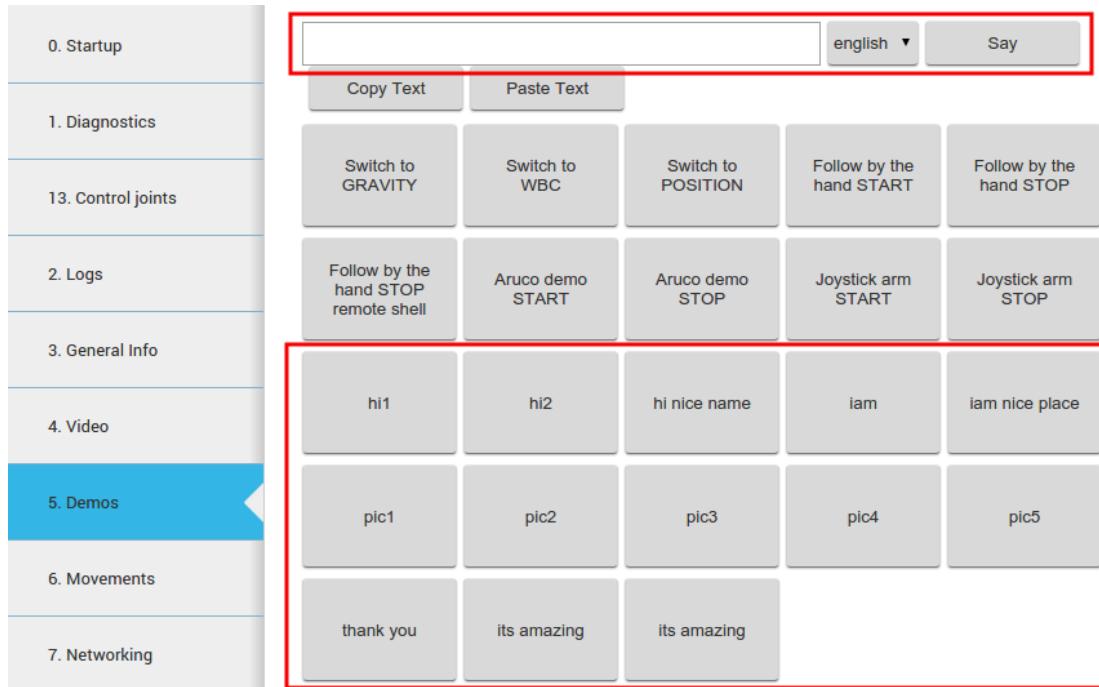


Figure 61: Voice synthesis in a commands tab of the WebCommander

### 19.3.2 Command line

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

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

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

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

### 19.3.3 Action client

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

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

```
export ROS_IP=DEV_PC_IP
```

The GUI shown in Figure 62 can be run as follows:

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

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

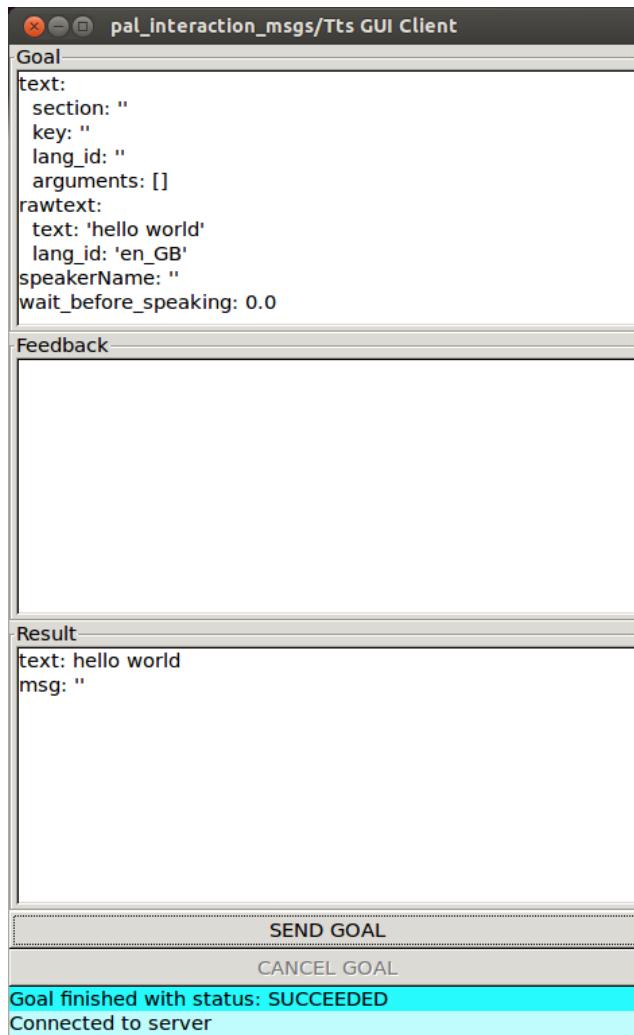


Figure 62: Voice synthesis using the GUI from actionlib

**TIA Go**

**Base motions**





## 20 Base motions

### 20.1 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 71. First, the motion triggers implemented in the joystick will be exposed, then the underlying ROS API to access the base controller will be presented.

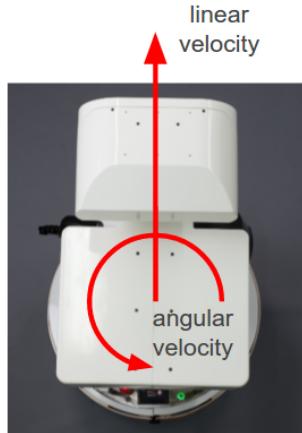


Figure 63: Mobile base velocities that can be commanded

### 20.2 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 64. Release the priority by pressing the same button.



Figure 64: Taking priority with the joystick

#### 20.2.1 Forward/backward motion

To move the base forward, use the left analog stick, as shown in figure 72.



Figure 65: Base linear motion with the joystick

### 20.2.2 Rotational motion

In order to make the base rotate on its Z axis, the right analog stick has to be operated, as shown in figure 66



Figure 66: Base rotational motion with the joystick

### 20.2.3 Changing the speed of the base

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

- a) To increase linear speed, see figure 67a
- b) To decrease linear speed, see figure 67b
- c) To increase angular speed, see figure 67c
- c) To decrease angular speed, see figure 67d



Figure 67: Joystick button combinations to change speed

## 20.3 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 of the two drive wheels.

## 20.4 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 68, 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, there are commands from the `move_base` node, which is used by the navigation pipeline that is presented in section 30.

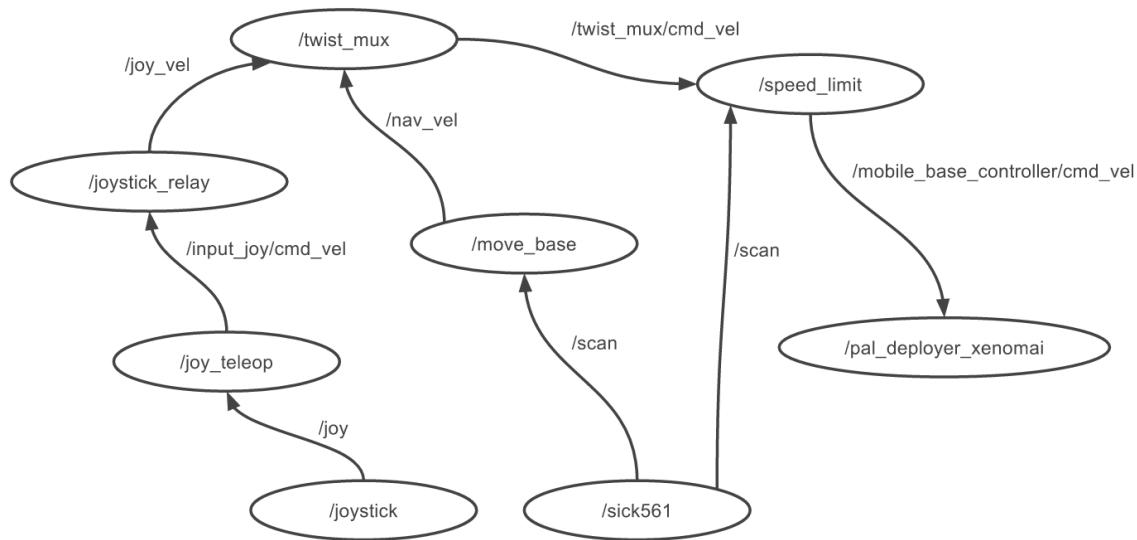


Figure 68: Mobile base control diagram



**TIA Go**

**Torso motions**





## 21 Torso motions

### 21.1 Overview

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

### 21.2 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 20.2. Move the torso by using the LB and LT buttons of the joystick, see figure 69. Press LB to raise the torso and LT to move it downwards, see figure 70.



Figure 69: Buttons to move the torso



Figure 70: Torso vertical motions triggered with the joystick

### 21.3 Torso control ROS API

#### 21.3.1 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.

#### 21.3.2 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.

**TIA Go**

**Head motions**

**PAL** ROBOTICS



## 22 Head motions

### 22.1 Overview

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

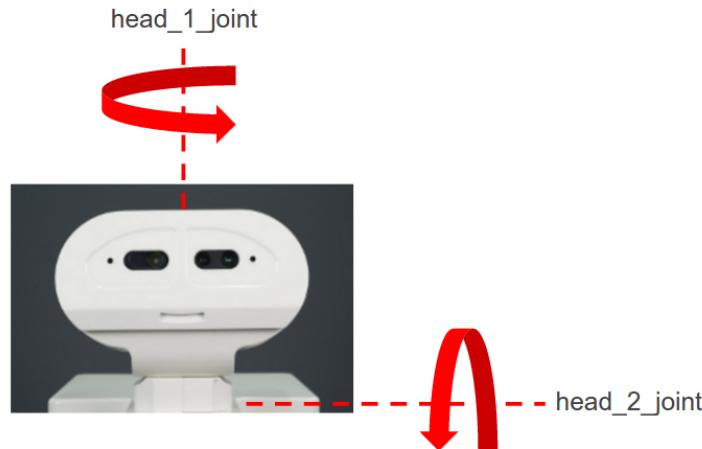


Figure 71: Rotational joints of the head

### 22.2 Head motion joystick triggers

The head is moved by using the X, Y, A and B buttons on the right hand side of the joystick, see figure 72.



Figure 72: Buttons to move the head

### 22.3 Head motions with rqt GUI

The joints of the head can be moved individually using a GUI implemented on the rqt framework that can be launched from a terminal.

In case of running this example with the real robot, i.e. not in simulation, open the terminal in a development computer and first run:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
```

Make sure to use your robot's serial number when exporting the `ROS_MASTER_URI` variable and to set your development computer's IP when exporting `ROS_IP` as explained in Section 10.4.

The GUI is launched in the same terminal as follows:

```
rosrun rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```

The GUI is shown in figure 73. Note that other groups of joints, i.e. head, torso, hand and gripper, can also be moved using this GUI. Furthermore, this is equivalent to using the control fonts tab in the WebCommander, as explained in section 11.4.5. In order to move the head joints, select `/controller_manager` in the combo box on the left and the `head_controller` on the right. Sliders for the two actuated joints of the head will show up.

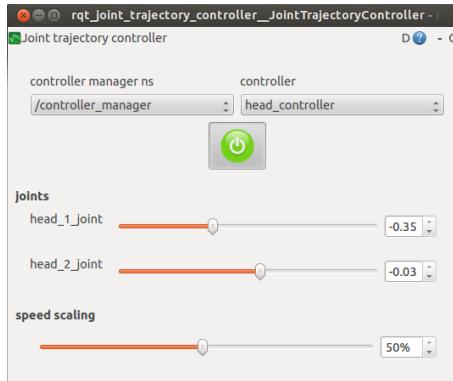


Figure 73: rqt GUI to move individual joints of the head

## 22.4 Head control ROS API

### 22.4.1 Topic interfaces

`/head_controller/command` ([trajectory\\_msgs/JointTrajectory](#))

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

### 22.4.2 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**

**PAL** ROBOTICS

The logo for PAL Robotics features the word "PAL" in a large, orange, sans-serif font. To the right of "PAL" is the word "ROBOTICS" in a smaller, black, sans-serif font. To the right of "ROBOTICS" is a circular emblem. The emblem has a dark brown or black background with a thin orange border. Inside the circle, there is a stylized, glowing orange "C" shape that curves upwards and to the right, resembling a stylized letter "G" or a gear.



## 23 Arm motions

### 23.1 Overview

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

### 23.2 Arm motions with rqt GUI

The joints of the arm can be moved individually using a GUI implemented on the rqt framework.

In case of running this example with the real robot, i.e. not in simulation, open the terminal in a development computer and first run:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
```

Make sure to use your robot's serial number when exporting the `ROS_MASTER_URI` variable and to set your development computer's IP when exporting `ROS_IP` as explained in Section 10.4.

The GUI is launched as follows:

```
rosrun rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```

The GUI is shown in figure 74. Note that other groups of joints, i.e. head, torso, hand, gripper, can be also moved using this GUI. This is equivalent to using the control joints tab in the WebCommander, as explained in section 11.4.5. 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 74: rqt GUI to move individual joints

### 23.3 Arm control ROS API

#### 23.3.1 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.

### 23.3.2 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**

**PAL**   
ROBOTICS



## 24 Hand motions

### 24.1 Overview

This section explains how to move the three motors of the Hey5 hand using the joystick or the rqt GUI, and explains the ROS API of the hand.

### 24.2 Hand motion joystick triggers

There are joystick triggers to close and open the hand using buttons RT and BT, respectively, as shown in figure 75.



Figure 75: Joystick buttons to trigger hand close/open motions

### 24.3 Hand motions with rqt GUI

The joints of the hand can be moved individually using a GUI implemented on the rqt framework.

In case of running this example with the real robot, i.e. not in simulation, open the terminal in a development computer and first run:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
```

Make sure to use your robot's serial number when exporting the `ROS_MASTER_URI` variable and to set your development computer's IP when exporting `ROS_IP` as explained in Section 10.4.

The GUI is launched as follows:

```
rosrun rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```

The GUI is shown in figure 76. Note that other groups of joints, i.e. head, torso, hand and gripper can also be moved using this GUI. This is equivalent to using the control joints tab in the WebCommander, as explained in section 11.4.5. In order to move the hand joints, select `/controller_manager` in the combo box on the left and the `hand_controller` on the right. Sliders for the three actuated joints of the hand will show up.

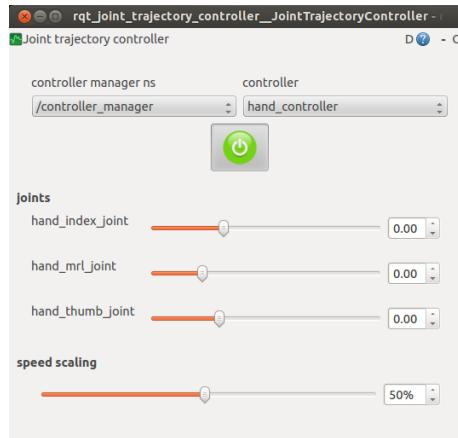


Figure 76: rqt GUI to move individual joints of the hand

## 24.4 Hand control ROS API

### 24.4.1 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.

### 24.4.2 Action interfaces

/hand\_controller/follow\_joint\_trajectory ([control\\_msgs/FollowJointTrajectory Action](#))

This action encapsulates the `trajectory_msgs/JointTrajectory` message.

**TIA Go**

**PAL gripper motions**





## 25 PAL gripper motions

### 25.1 Overview

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

### 25.2 Gripper motion joystick triggers

There are joystick triggers order to close and open the gripper using buttons RT and BT, respectively, as shown in figure 77.



Figure 77: Joystick buttons to close/open the gripper

### 25.3 Gripper motions with rqt GUI

In case of running this example with the real robot, i.e. not in simulation, open the terminal in a development computer and first run:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
```

Make sure to use your robot's serial number when exporting the `ROS_MASTER_URI` variable and to set your development computer's IP when exporting `ROS_IP` as explained in Section 10.4.

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

```
rosrun rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```

Select the controller manager namespace available. Select `gripper_controller` and then two sliders, one for each gripper joint, will appear, as shown in figure 78. One slide controls the position of each finger.

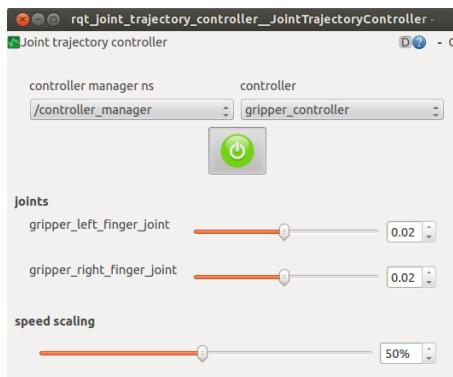


Figure 78: rqt GUI to move the gripper

## 25.4 Gripper control ROS API

### 25.4.1 Topic interfaces

/parallel\_gripper\_controller/command ([trajectory\\_msgs/JointTrajectory](#))

This topic is used to specify the desired distance between the robot's 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 the fingers becomes 3 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: [0.03]
  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. Position 0 corresponds to a closed gripper and 0.04 corresponds to an open gripper. An example to set the fingers to different positions using 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
```

### 25.4.2 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.

### 25.4.3 Service interfaces

/parallel\_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 in order to hold the object, while not overheating the motors.

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

```
rosservice call /parallel_gripper_controller/grasp
```



**TIA Go**

**Schunk gripper motions**





## 26 Schunk gripper motions

### 26.1 Overview

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



Figure 79: Schunk WSG32 gripper

### 26.2 A note on the Schunk gripper position control

The gripper is equipped with an integrated detector for parts to be gripped and a grip monitor, which are used in the usual operation way for industrial applications, where the size of the parts to be gripped is known beforehand. This is not the most common case for service robots where usually the robot has to grasp objects of unknown geometry and varying sizes. Therefore, for sake of generality PAL Robotics has integrated the Schunk WSG32 gripper in `ros_control` by exposing a position controller. From the user point of view the aperture of the gripper can be controlled using this position controller. Nevertheless, internally the commands sent to the gripper specify that an object of the size matching the aperture given has to be grasped. As result, if there is no such object or the actual size of the object is much different of the one specified with the target aperture, the gripper integrated detector will raise a grasp error which causes that the motor is not actuated until a new grasp command is received.

The internal parameters shown in table 22 of the gripper firmware have been set in order to maximize as much as possible the controllability of the gripper in position mode.

Parameter	Description	Value set
Part Width Tolerance	The tolerance of the specified nominal gripper width which is sent to the gripper with the grasp command. This is measured as the relative gap between two fingers moving towards one another.	30 mm
Clamping Travel	If a part to be gripped has been detected, the gripper will attempt to apply the required gripping force by moving the fingers within this range. This is measured as the relative gap between two fingers moving towards one another.	10 mm

Table 22: Schunk gripper internal parameters

### 26.3 Gripper motion joystick triggers

There are joystick triggers order to close and open the gripper incrementally using buttons RT and BT, respectively, as shown in figure 80.



Figure 80: Joystick buttons to close/open the Schunk gripper

## 26.4 Gripper motions with rqt GUI

In case of running this example with the real robot, i.e. not in simulation, open the terminal in a development computer and first run:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
```

Make sure to use your robot's serial number when exporting the `ROS_MASTER_URI` variable and to set your development computer's IP when exporting `ROS_IP` as explained in Section 10.4.

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

```
rosrun rqt_joint_trajectory_controller rqt_joint_trajectory_controller
```

Select the controller manager namespace available. The `gripper_controller` and one slider, as shown in figure 81, controlling the aperture of the gripper fingers.

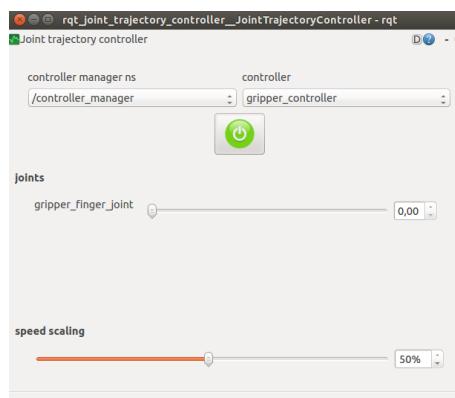


Figure 81: rqt GUI to move the gripper

## 26.5 Gripper control ROS API

### 26.5.1 Topic interfaces

`/gripper_controller/command` ([trajectory\\_msgs/JointTrajectory](#))

This topic is used to specify the stroke of each finger of the gripper. For example, a value of 0.034 m specifies that the fingers will be completely open providing an aperture of 0.068 m (68 mm). A value of 0 m will get the fingers closed. A single target position or a sequence of positions can be specified. For instance, the following command moves the gripper motor so that the distance between the fingers becomes 68 mm:

```
rostopic pub /gripper_controller/command trajectory_msgs/JointTrajectory " 
header: 
  seq: 0 
  stamp: 
    secs: 0 
    nsecs: 0 
  frame_id: '' 
joint_names: ['gripper_finger_joint'] 
points: 
- positions: [0.034] 
  velocities: [] 
  accelerations: [] 
  effort: [] 
  time_from_start: 
    secs: 1 
    nsecs: 0" --once
```

/gripper\_current\_limit\_controller/state ([pal\\_control\\_msgs/ActuatorCurrentLimit](#))

This topic provides the limited maximum force that the gripper can exert on a part. This value ranges from 0 to 1.0, being the proportion of the maximum attainable force which is 50 N. In order to retrieve the value the following command line can be used:

```
rostopic echo -n 1 /gripper_current_limit_controller/state
```

which reports by default

```
actuator_names: ['gripper_motor'] 
current_limits: [1.0]
```

/gripper\_current\_limit\_controller/command ([pal\\_control\\_msgs/ActuatorCurrentLimit](#))

The user can establish the limited maximum force of the gripper by publishing in this topic the proportion of the maximum attainable force which is 50 N. For example, to limit the maximum force to 5 N, i.e. 10% of the maximum attainable force, the limit has to be set to 0.1 as follows:

```
rostopic pub /gripper_current_limit_controller/command 
pal_control_msgs/ActuatorCurrentLimit " 
actuator_names: ['gripper_motor'] current_limits: [0.1]" --once
```

## 26.5.2 Action interfaces

/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.



**TIA Go**

**End-effector exchange**





## 27 End-effector exchange

### 27.1 Overview

This section explains how to exchange the robot's end-effectors, replace the parallel gripper with the Hey5 hand and vice versa.

### 27.2 Changing the end-effector software configuration

In order to change the configuration of the robot, go to the robot's web commander and to the Settings tab. Select the end effector from the drop-down menu (Please refer to 11.3.7 for further details) that you are going to install and Reboot the robot before proceeding with the following steps.

### 27.3 Mounting the Hey5 hand

The procedure to exchange the parallel gripper with the Hey5 hand is shown in the following video [Demounting and mounting TIAGo's end-effectors](#). The steps are summarized below.

#### 27.3.1 Unmounting the gripper

Make sure that the emergency button is pressed and the electric switch is turned off in order to ensure maximum safety during the procedure.

Locate the fastener that locks the end-effector in the end-effector clamp, as shown in figure 82

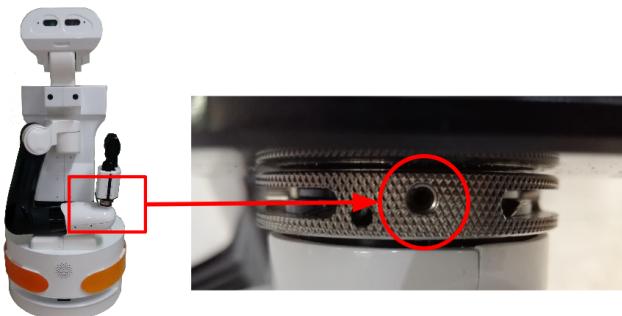


Figure 82: Screw locking the clamp of the end-effector

Use an appropriate Allen key to loosen the locking screw of the clamp, as shown in Figure 83

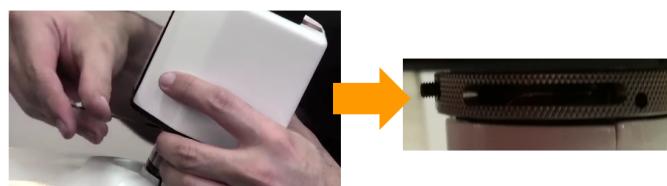


Figure 83: Loosen the locking-screw

Unlock the clamp by rotating it counterclockwise as shown in figure 84.

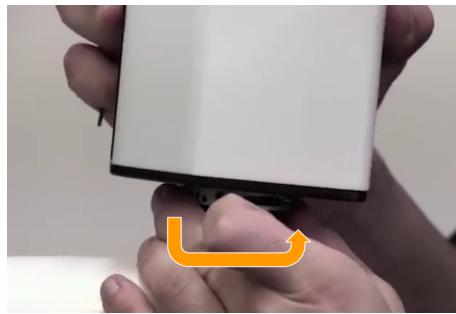


Figure 84: Unlocking the clamp with the hand

If the clamp cannot be unlocked by hand, use the tool provided with the spare end-effector, see figure 85. Insert the tool in the groove edge of the clamp and rotate it counterclockwise in order to unlock it, as shown in figure 86.



Figure 85: Tool for end-effector clamp



Figure 86: Unlocking the clamp using the tool provided

Once the clamp is unlocked, unmount the end-effector by lightly shaking and pulling it at the same time, see figure 87.

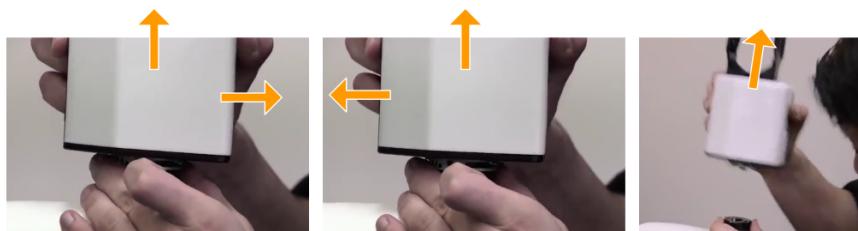


Figure 87: Unmounting the gripper

### 27.3.2 Mounting the Hey5 hand

Align the Hey5 hand and the clamp, as shown in figure 88.

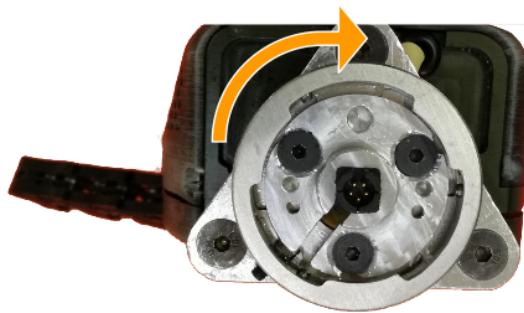


Figure 88: Correct alignment of the clamp

Position the Hey5 hand on top of the wrist mounting plate so the plate pin fits in the corresponding hole of the end-effector plate, see figure 89.



Figure 89: Alignment of the end-effector with the wrist mounting plate

Insert the end-effector by pressing until the mounting plate of the wrist is in contact with the end-effector plate, as shown in figure 90



Figure 90: Insertion of the end-effector

Use the tool to twist the clamp clockwise in order to tighten it, as shown in figure



Figure 91: Tightening Hey5 clamp

Now tighten the locking screw, as shown in figure 92.



Figure 92: Tightening Hey5 clamp

### 27.3.3 Validation

In order to validate that the Hey5 hand has been correctly assembled and its software properly activated, press the electric switch , release the emergency button and hold the On button of the mobile base for two seconds.

When the robot has correctly started up, go to the Diagnostics tab of the WebCommander, and under the Motors sections check that the Hey5 hand motors are shown in green.

## 27.4 Mounting the parallel gripper

The procedure to exchange the Hey5 hand with the parallel gripper is the same.



**TIA Go**

**Force-Torque Sensor calibration**





## 28 Force-Torque sensor calibration

### 28.1 Overview

This section explains how to use the Force-Torque (F/T) calibration procedure for the F/T sensor on the wrist. When reading the values of the F/T sensor there is typically a certain amount of noise, some caused by the sensor itself, some caused by the fact that the end-effector exerts a force on the sensor due to its weight.

The weight of the end-effector can easily be computed, and its effect compensated without much difficulty, but, in order to calibrate the offsets of the sensor, an experimental method must be used. This package provides an experimental method to compute said offsets in a simple, convenient way.

It is important to note that the offsets of the F/T sensor may slightly change during the operation of the arm, so it is suggested to perform this procedure every four or five hours of continued use.

### 28.2 Running the calibration

**Warning** The calibration procedure will extend the arm in the forward direction to its maximum length. For this reason it is important that the robot is, at least, more than one meter away from any obstacles. It is also important not to touch the end effector during the procedure, as this would result in incorrect offsets.

The calibration is controlled by an Action interface, which will perform a series of motions with the arm, compute the offsets of the F/T sensor, and set them up. Once the procedure is completed, if successful, the corrected readings, compensating both offset and end-effector weight, will be published in `/wrist_ft/corrected`.

#### 28.2.1 Action Interface

```
/wrist_ft/calibrate (tiago_ft_offsets_msgs/CalibrateWristOffsets Action)
```

The goal, feedback and result messages of this action are empty messages.

The calibration can also be triggered from terminal by running:

```
$ rostopic pub /wrist_ft/calibrate/goal
tiago_ft_offsets_msgs/CalibrateWristOffsetsActionGoal "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: ''
goal_id:
  stamp:
    secs: 0
    nsecs: 0
  id: ''
goal: {}"
```

### 28.3 Parameter customization

There are several parameters that can be changed to modify the procedure. Correct values are provided for the Hey5 hand, the PAL parallel gripper and the Schunk parallel gripper, they would only need to be updated in case of integrating a different end-effector.

- `end_effector_weight_kg`: Weight of the end-effector in Kg.
- `CoM_respect_wrist_ft_frame/offx`: X component of the position of the center of mass of the end-effector, expressed in meters with respect to the F/T sensor frame.
- `CoM_respect_wrist_ft_frame/offy`: Y component of the position of the center of mass of the end-effector, expressed in meters with respect to the F/T sensor frame.
- `CoM_respect_wrist_ft_frame/offz`: Z component of the position of the center of mass of the end-effector, expressed in meters with respect to the F/T sensor frame.

These parameters can be found in the `tiago_ft_wrist_calibration/config/end_effector_params.yaml` file, where `end-effector` can be `pal-gripper`, `pal-hey5` or `schunk-wsg`.

**TIA Go**

**Upper body motions engine**

**PAL** ROBOTICS

The logo for PAL Robotics features the word "PAL" in a large, orange, sans-serif font. To the right of "PAL" is the word "ROBOTICS" in a smaller, black, sans-serif font. To the right of "ROBOTICS" is a circular emblem. The emblem consists of a dark orange circle with a thin white border. Inside the circle is a stylized, white, swirling graphic that resembles both a lowercase 'c' and a lowercase 'g', positioned symmetrically on either side of a vertical axis.



## 29 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](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 in charge of commanding different groups of joints. Figure 93 shows the role of `play_motion` in order to play back predefined upper body motions.

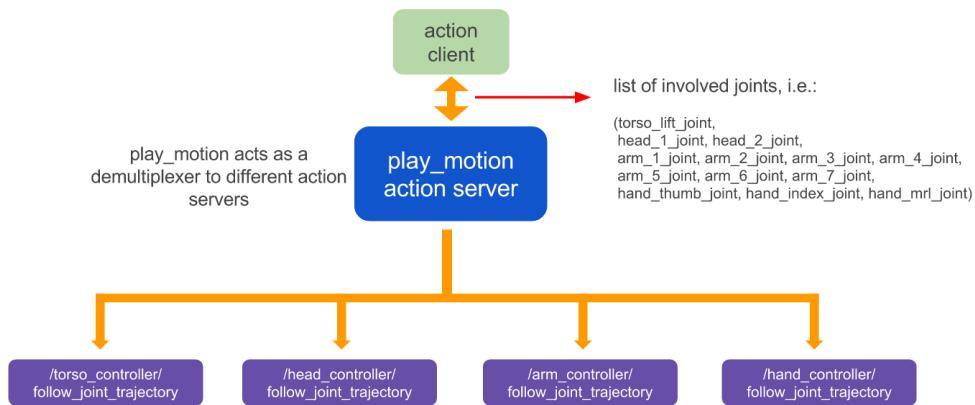


Figure 93: `play_motion` action server role

The different groups of actuated joints defined in TIAGo are those shown in table 23.

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 23: Groups of joints

The motions that `play_motion` is able to play back are based on a sequence of joint positions that need to be reached within given time intervals.

### 29.1 Motions library

The motion library is stored in:

`tiago_bringup/config/motions/tiago_motions_X.yaml` where `X` refers to the type of end-effector installed. 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 sorting the predefined motions file can be edited as follows:

```

roscore tiago_bringup
cd config/motions
vi tiago_motions_X.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 the `tiago_bringup` package.

## 29.2 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 the 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 also needs to include `head_2` joint, as they both belong to the same group. See table 23.
- **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 94.

As can be seen, the joints included in the motion are those 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 94: wave motion definition

## 29.3 Using predefined motions safely

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 minimize the risk of self-collisions at the beginning of the motion. As the upper body joints can be in any arbitrary position before starting the motion, the joint movements that must be made in order to reach the first predefined positions in the `points` tuple are the most dangerous. 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 matter of seconds. The user can disable this feature, as shown in the next section. Nevertheless, it is **strongly recommended** not to do so unless the position of the joints before executing the motion are well known, and a straight trajectory of the joints towards the first intermediate position is safe.

## 29.4 ROS interface

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

### 29.4.1 Action interface

`/play_motion` ([play\\_motion\\_msgs/PlayMotion Action](#))

This action plays back a given predefined upper body motion.

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. This parameter should be set to `False` by default to minimize the risk of self-collisions, as explained in section 29.3.
- **priority**: unimplemented feature. For future use.

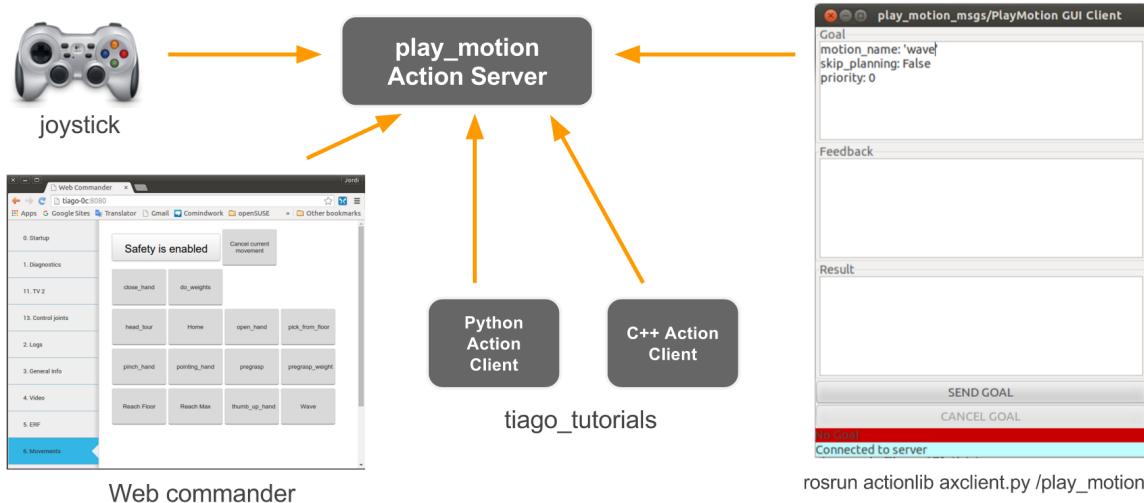


Figure 95: play\_motion action clients provided

## 29.5 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 95.

Note that predefined motions can be executed from the:

- **Joystick:** several combinations of buttons are already defined to trigger torso and hand/gripper motions.
- **WebCommander:** 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.

**TIA Go**

**Navigation**





## 30 Navigation

### 30.1 Overview

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

### 30.2 Navigation architecture

The navigation software provided with TIAGo can be seen as a black box with the inputs and outputs shown in figure 96.

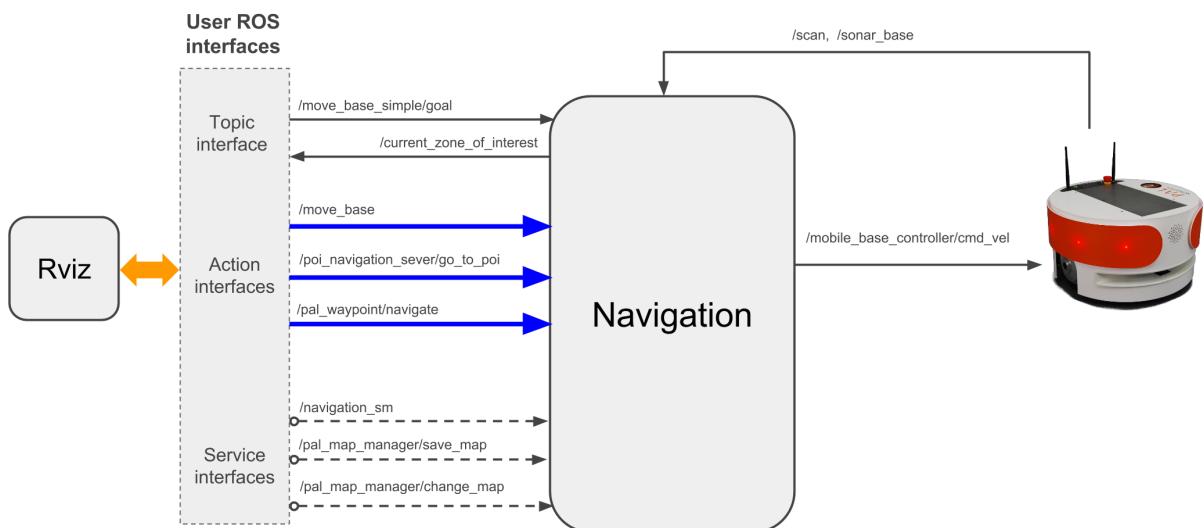


Figure 96: TIAGo navigation black box

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 comprising the navigation architecture are shown in figure 97. Note that nodes communicate using topics, actions and services, but also using parameters in the ROS param server (note depicted in the figure).

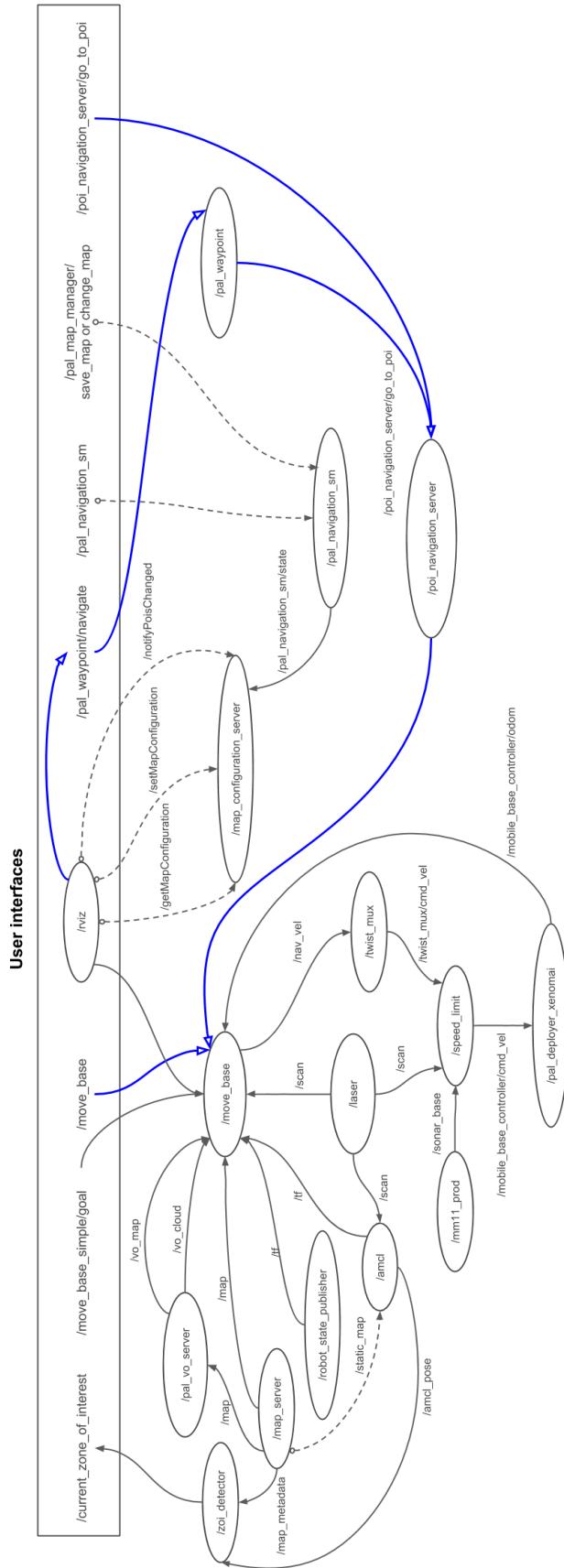


Figure 97: TIAGo navigation nodes overview

### 30.3 Navigation ROS API

#### 30.3.1 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<sup>2</sup> ([pal\\_zoi\\_detector/CurrentZoi](#))

Topic to print the name of the zone of interest where the robot is at present, if any.

#### 30.3.2 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 wants to be notified when the goal has been reached or if something fails in the process.

/poi\_navigation\_server/go\_to\_poi<sup>3</sup> ([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 30.6.

/pal\_waypoint/navigate<sup>3</sup> ([pal\\_waypoint\\_msgs/DoWaypointNavigationAction](#))

Action to make the robot visit all the POIs of a given group or subset. POIs and POI groups can be defined using the Map Editor, see section 30.6.

#### 30.3.3 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:

```
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 on the map.

/pal\_map\_manager/save\_map ([pal\\_navigation\\_msgs/SaveMap](#))

Service to save the map with a given name. 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/tiago_maps/configurations`.

/pal\_map\_manager/change\_map ([pal\\_navigation\\_msgs/Acknowledgment](#))

Service to choose the active map. Example:

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

<sup>2</sup>The publisher of this topic is included in the Advanced Navigation package.

<sup>3</sup>This software is included in the Advanced Navigation package

## 30.4 SLAM and path planning in simulation

### 30.4.1 Mapping

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

Run the simulator and mapping functionality with the following launch file :

```
source /opt/pal/ferrum/setup.bash
roslaunch tiago_0_gazebo tiago_mapping.launch
```

The Gazebo window shown in figure 98 will open and the robot should be visible in an office-like environment. Furthermore, a Rviz window will open where the robot model, sensor and map being built will be visualized.

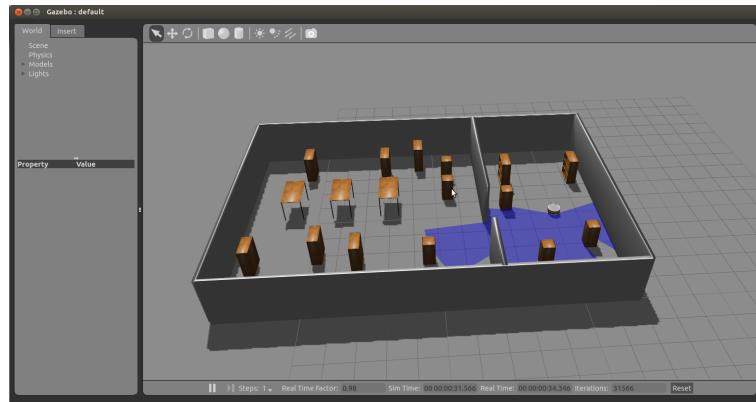


Figure 98: Small office world simulated in Gazebo

If you have a USB joystick plugged in to 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:

```
rosrun key_teleop key_teleop.py
```

While moving the robot around the environment, the map will start to appear in Rviz, as shown in the sequence of snapshots in figure 99.

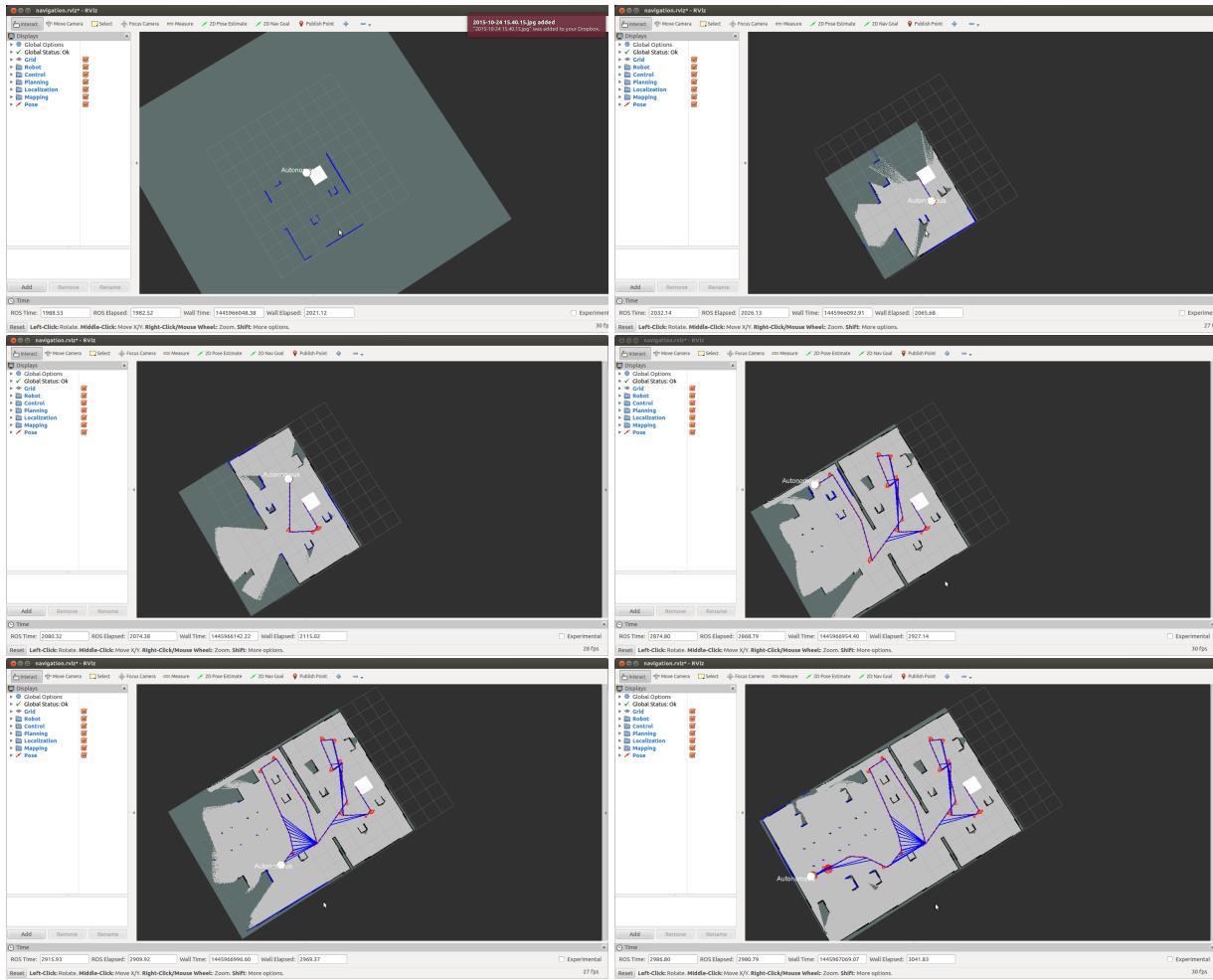


Figure 99: Partial maps built by the SLAM algorithm, starting (top left) and moving until a successful map is built (bottom right). The SLAM graph is represented by the nodes (red dots) and edges (blue segments).

### 30.4.2 Saving the map

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

```
rosrun map_server map_saver
```

This command saves two files (as shown in Figure 100): a *map.pgm* image file and a *map.yaml* configuration file. They will both be saved in the folder where the terminal is opened.

```
~$ rosrun 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 100: Output of map\_saver call

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

To save the map 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 101: Output of map\_saver call

The map will be named as the date and time it was saved, as in figure 101

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

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

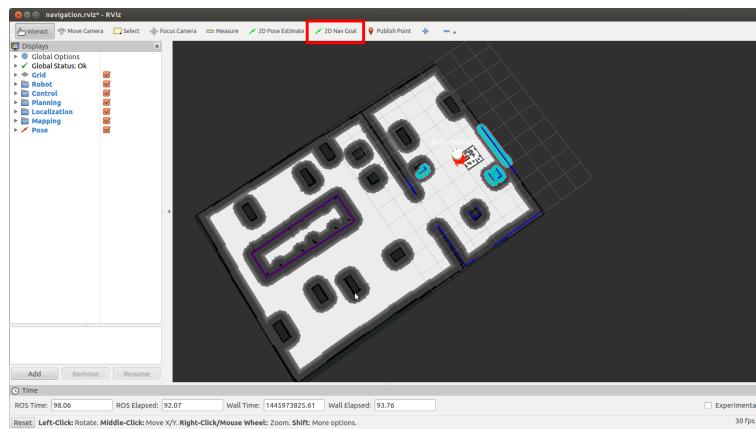
```
rosservice call /pal_navigation_sm "input: 'LOC'"
```

### 30.4.3 Localization and Path Planning

To run a simulation with the robot in localization and path planning mode, run:

```
roslaunch tiago_0_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 102 will show the map, localization particles, localized robot model, laser sensor readings and some virtual obstacles.

Figure 102: Small office world simulated navigation as visualized in rviz 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 the left mouse button), as shown in figure 103 then click in the map to select the target location the robot has to reach.

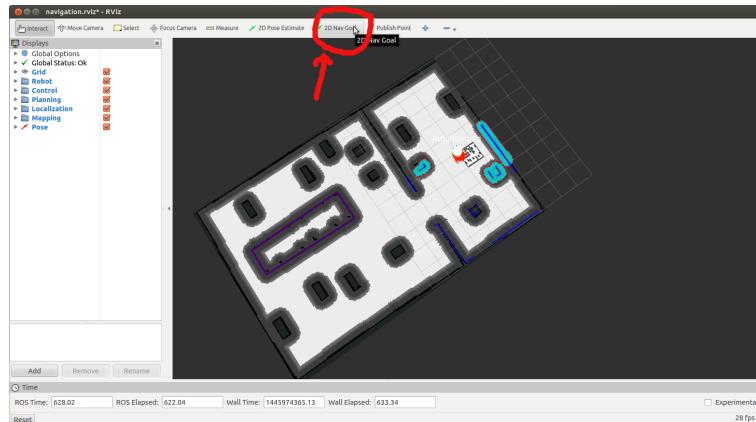


Figure 103: “2D Nav Goal” tool in rviz should be clocked before selecting in the map the target location the robot has to reach.

The plan will be generated and the robot will start moving along the trajectory towards the goal, as illustrated in the sequence of snapshots in figure 104.

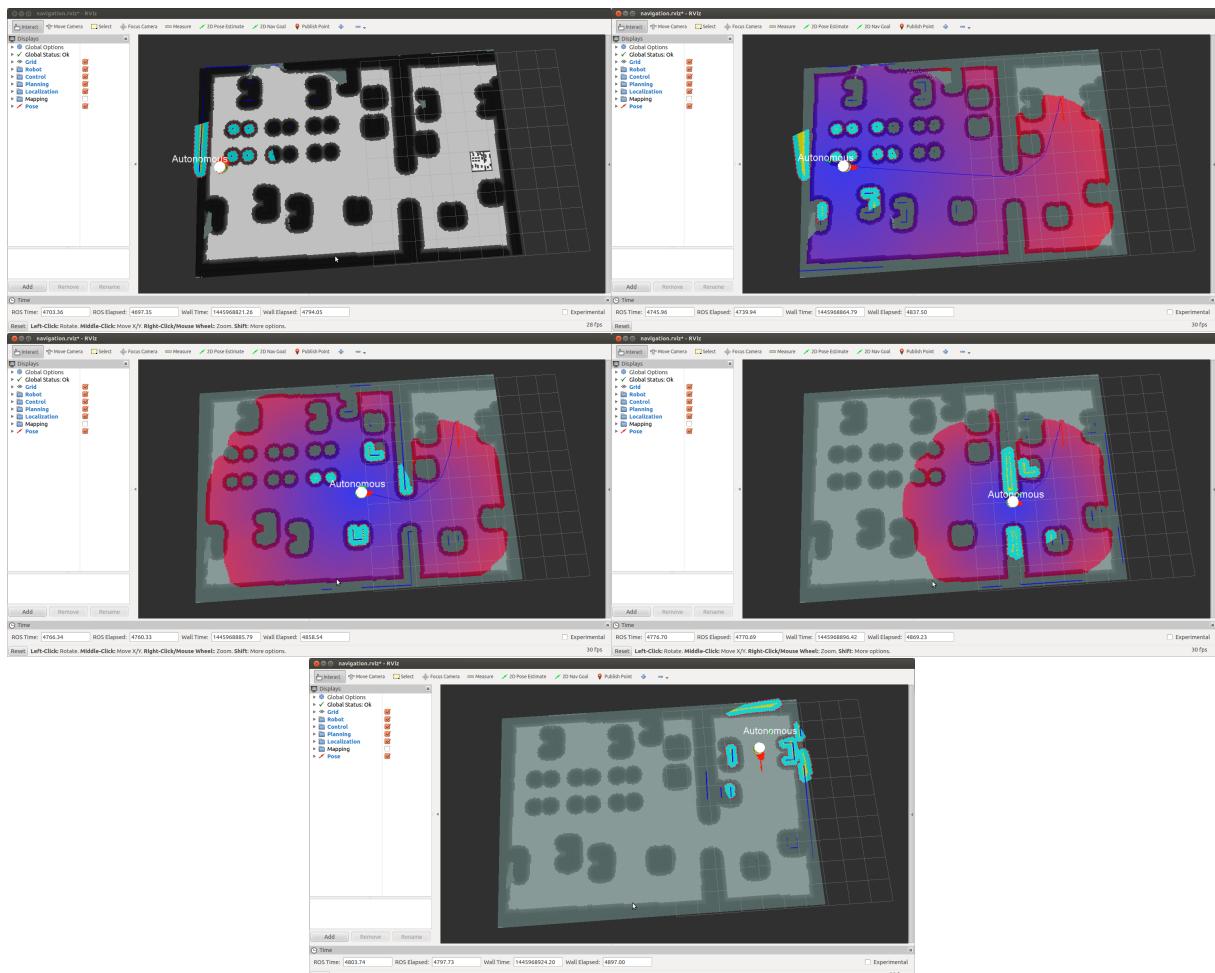


Figure 104: Autonomous navigation: the robot continuously localizes itself in the map, plans the path for reaching the target location and executes velocity commands for following the trajectory and avoiding obstacles. Costmaps and inflated obstacles are visible.

## 30.5 SLAM and path planning on the robot

### IMPORTANT NOTES

Before continuing with the instructions of this section make sure that the robot computer is able to resolve the development computer hostname, as explained in Section 10.4. Otherwise, some commands will not work with the real robot due to communication failures between the robot's computer and the development computer. For sake of clarity and without loss of generality hereafter we will assume that the development computer's IP is 10.68.0.128, which will be set in the `ROS_IP` environmental variable when needed according to Section 10.4. The user will have to adapt the examples below to set the right IP address. Furthermore, make sure that the robot is not plugged to the charger, otherwise it will not move.

Note that when the robot back connector is plugged to the charger the Navigation functionality is paused for safety. The status of the functionality can be checked in the `Webcommander` diagnostics tab, see figure 105. Alternatively, the status can be checked in `/pause_navigation` topic, which reports a boolean specifying whether the functionality is paused.

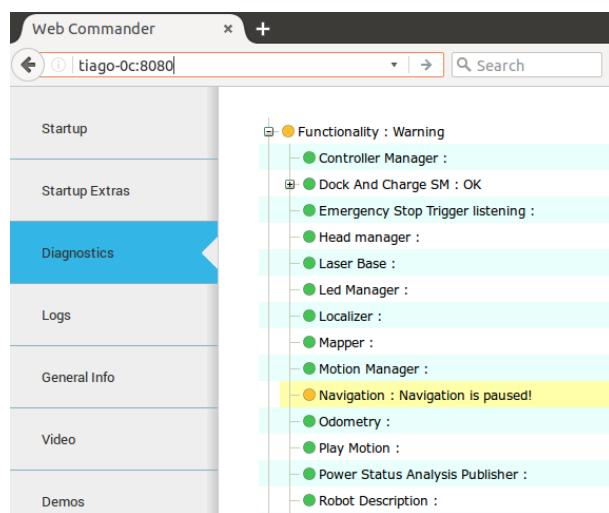


Figure 105: Navigation functionality paused due to charger connected

When the robot boots, the navigation pipeline starts automatically. Furthermore, the localization mode based on the last map will be active.

The procedure for the real robot is almost the same, just without 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
export ROS_IP=10.68.0.128
rosservice call /pal_navigation_sm "input: 'MAP'"
```

Then just follow the steps described above for the simulation case.

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

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun rviz rviz -d `rospack find tiago_2dnav`/config/rviz/navigation.rviz
```

In order to ensure that `rviz` works properly, make sure that the robot computer is able to resolve the development computer hostname, as explained in section 10.4.

### 30.5.1 Saving the map on the robot

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

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun 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
export ROS_IP=10.68.0.128
rosservice call /pal_map_manager/save_map "directory: ''"
```

The `save_map` service will store all the map files in the path

`$HOME/.pal/tiago_maps/configurations` in the robot's file system. The current map in use is the one pointed to by the symbolic link `$HOME/.pal/tiago_maps/config`.

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

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosservice call /pal_navigation_sm "input: 'LOC'"
```

### 30.5.2 Localization and path planning

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

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun 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 103 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 in figure 104.

If the robot does not move, make sure that the `ROS_IP` variable has been set with the right IP address of the development computer.

Navigation goals can be sent programmatically to the Action Server `/move_base_simple`.

### 30.5.3 Changing 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, use the following command:

```
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.

## 30.6 Map Editor

As seen in the previous sections with rviz, we can easily visualize the mapping process. In the localization mode, it is then possible to send the robot to any point of the map and provide a new localization estimate when necessary.

The PAL Map Editor<sup>4</sup> is a rviz plugin that provides advanced navigation functionalities to rviz, including:

- Downloading and uploading maps
- Re-naming maps
- Changing the active map
- Definition of Zones Of Interest (ZOIs)
- Definition of Points of Interest (POIs) and groups of POIs
- Definition of Virtual Obstacles (VOs)

The information about POIs, ROIs and VOIs are stored along with the map as meta-data.

A graphical joystick is also provided to remotely teleoperate TIAGo's mobile base.

In order to launch rviz with the Map Editor plugins, do the following:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun rviz rviz \
-d `rospack find tiago_2dnav`/config/rviz/advanced_navigation.rviz
```

In order to ensure that rviz works properly, make sure that the robot computer is able to resolve the development computer hostname, as explained in section 10.4.

The different plugins added to rviz are shown in figure 106.

---

<sup>4</sup>Software provided in the Advanced Navigation package

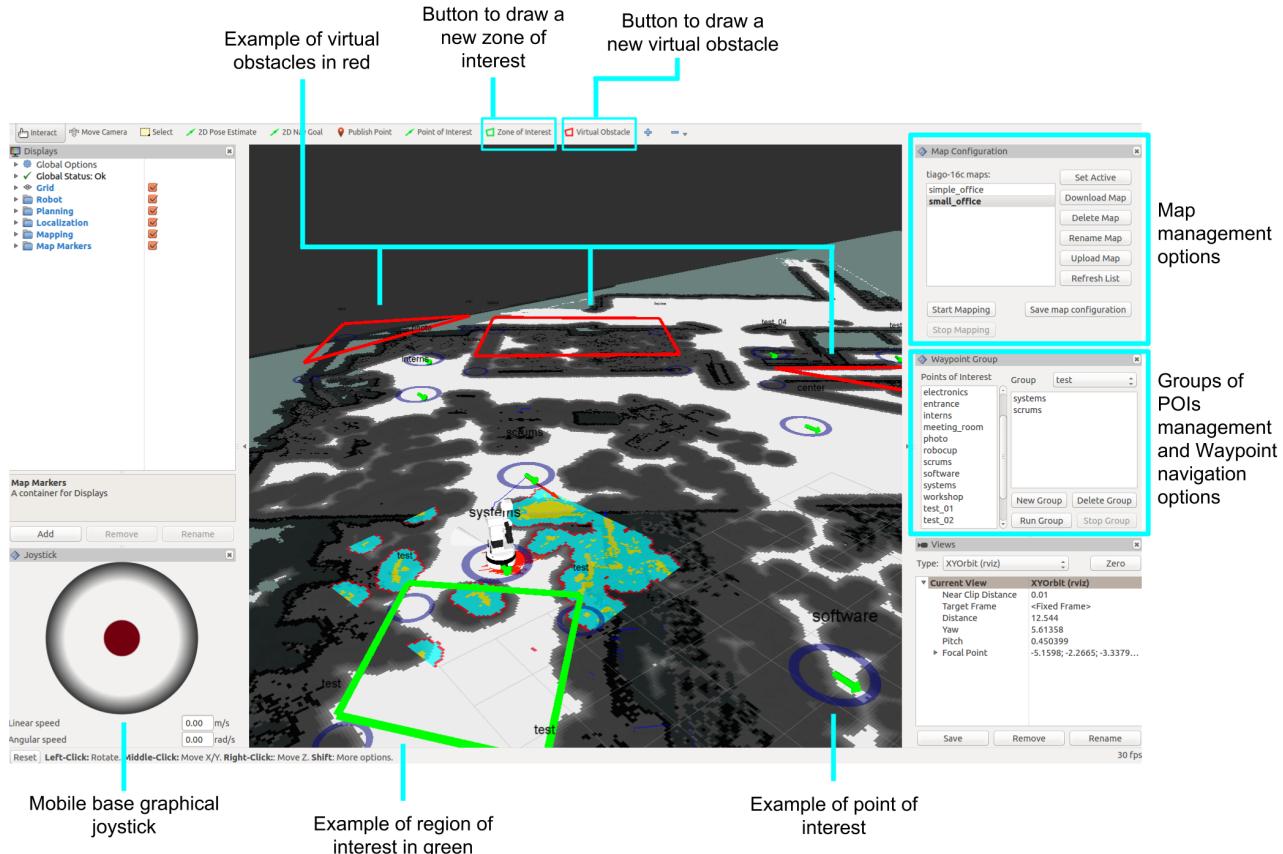


Figure 106: Rviz with the Map Editor plugins

### 30.6.1 Launching the Map Editor

In order to run the Map Editor in a development computer the following commands can be used in simulation and with a real robot.

#### Simulation

```
roslaunch tiago_0_gazebo tiago_navigation.launch
```

#### Real robot

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun rviz rviz -d `rospack find tiago_2dnav`/config/rviz/advanced_navigation.rviz
```

In both cases, when running the Map Editor for the first time an example of map will be loaded.

### 30.6.2 Creating a map

In order to create a new map the button `Start Mapping`, in the `Map Configuration` panel, must be pressed. The robot base can be then tele-operated using the robot's joystick, the `key_teleop` package or the graphical joystick of the Map Editor. Once the robot has mapped all the area of interest the user has to press the `Stop Mapping`. This will create a new map on the robot with a name consisting of a time-stamp.

Figure 107 shows the process of mapping in the Map Editor: on the top-right picture the previous active map is shown. At its right the status after starting mapping. The lower row shows the map under creation and the complete map after pressing the Stop Mapping.

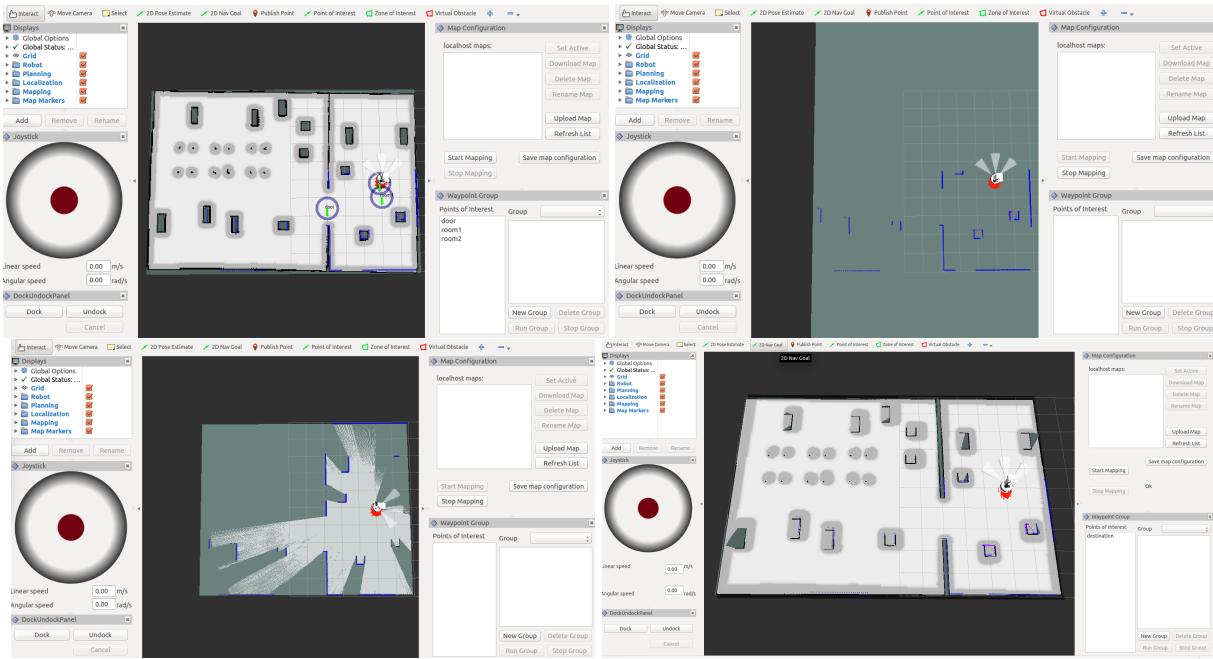


Figure 107: Map creation using the Map Editor

### 30.6.3 Managing maps

In order to list all the maps stored in the robot the button Refresh List of the Map Configuration panel must be pressed. The list of available maps will appear in the panel as shown in figure 108. Note that the one printed in bold is the active one.

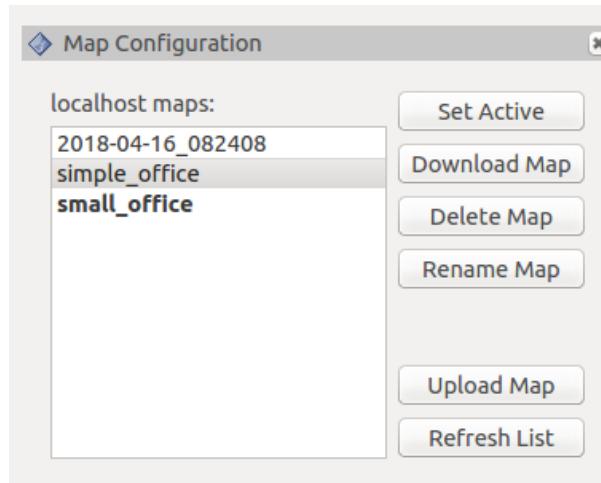


Figure 108: Management of maps

Map operations available are:

- Set Active: the map selected in the list will be loaded and used for navigation.

- Download Map: this button saves the selected map in the list in the path of the local computer running the Map Editor specified by the user.
- Delete Map: removes the selected map in the robot's computer.
- Rename Map: change the name of the selected map.
- Upload Map: copy a map from the local computer to the robot's computer.

### 30.6.4 Defining Points of Interest

A Point of Interest is a pose in the map defined by a set of coordinates and orientation. Once a POI is defined and stored in the map meta-data the robot can be sent to the POI using the `go_to_poi` action interface explained in section 30.3.2.

To create a new POI the button `Point of Interest` on the top bar of the Map Editor must be pressed. Then move the mouse to the desired map location, press the mouse's left button and drag the mouse to define the orientation of the POI. Release the mouse's button when the green arrow that will appear points to the desired direction and sense. Afterwards, a dialog requesting the name of the POI will show up as shown in figure 109. After accepting it the POI will be created. In order to get these meta-data permanently stored press `Save map configuration`.

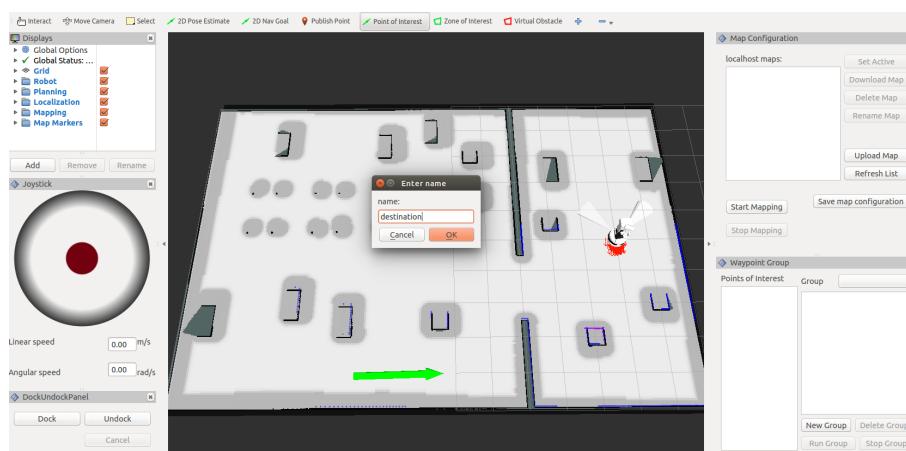


Figure 109: Definition of POIs

### 30.6.5 Defining Zones of Interest

Zones of Interest provide a simple way to have topological localization, i.e. obtain the name of the zone of the map where the robot is located. A ZOI can be defined by pressing the `Zone of Interest` button on the top bar of the Map Editor. By clicking on a map point the user specifies the central position of the ZOI. Afterwards, a dialog requesting the name of the zone and then another requesting how many points will be used to define the zone will appear, see figure 110. A green polygon with the selected number of vertices will appear on the selection map point. The user can then drag the vertices with the mouse to define the final placement of the Zone of Interest. Note that in order to move a vertex of the zone the mouse icon must be placed on top of the blue circle, the color of the circle will vary slightly, before clicking on it and dragging it.



Figure 110: Definition of ZOIs

After all the ZOIs are defined the `Save map configuration` will store the corresponding map meta-data. Any time the robot enters any of the ZOIs, the topic `/current_zone_of_interest` will print the name of the ZOI.

### 30.6.6 Defining Virtual Obstacles

Virtual obstacles are of key importance to prevent some dangerous situations during navigation, like falling downstairs or colliding with obstacles not clearly visible for the robot's sensors, and to label forbidden areas where the robot is not allowed to enter. The VOs are created the same way than the ZOIs by pressing the `Virtual Obstacle` button on the top bar of the Map Editor. VOs are represented with red polygons, see figure 111. In order to store the meta-data defining the VOs the user must press `Save map configuration`.

## Map Editor

## Navigation

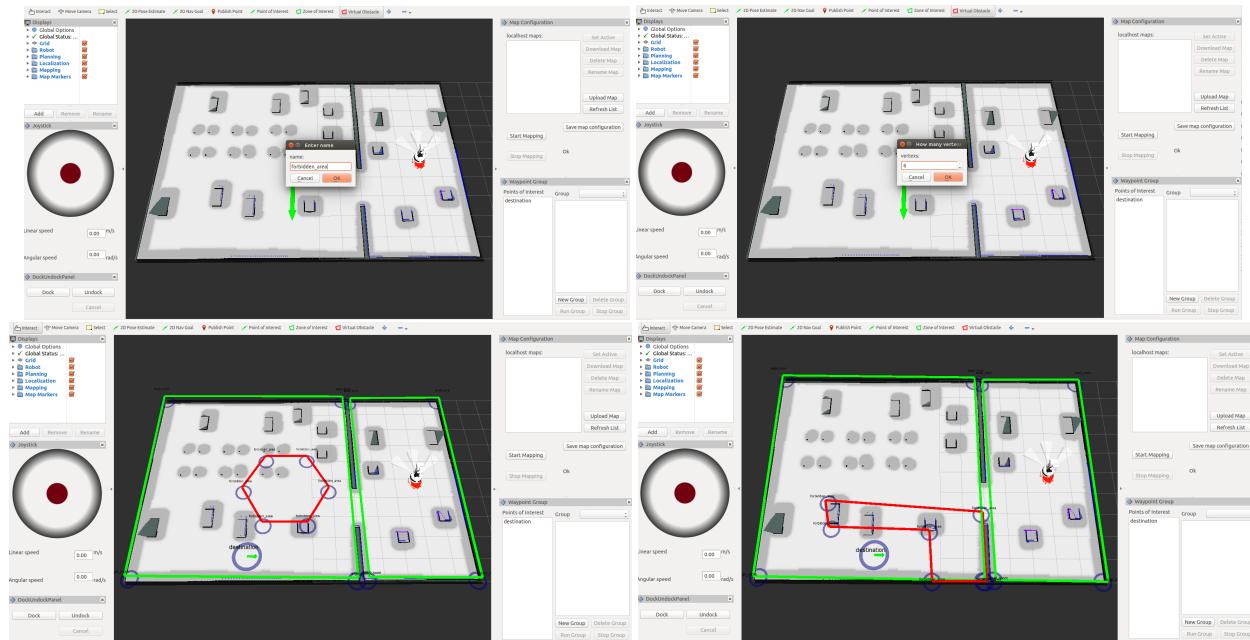


Figure 111: Definition of Virtual Obstacles

### 30.6.7 Defining Groups of POIs

After defining POIs the user can define groups of different POIs. A group of POIs can be run so that the robot visits all the POIs in the group in sequence. To create groups of POIs the panel **Waypoint Group** is provided. The list on the left of the panel shows all the POIs created. First press **New Group** to define a new group of POIs. A dialog requesting the name of the group will appear, see figure 112. In order to add POIs first select the POI on the left list and drag it to the panel of its right. The same POI can be set to a group multiple times. When the group contains all the desired POIs press the **Save map configuration**.

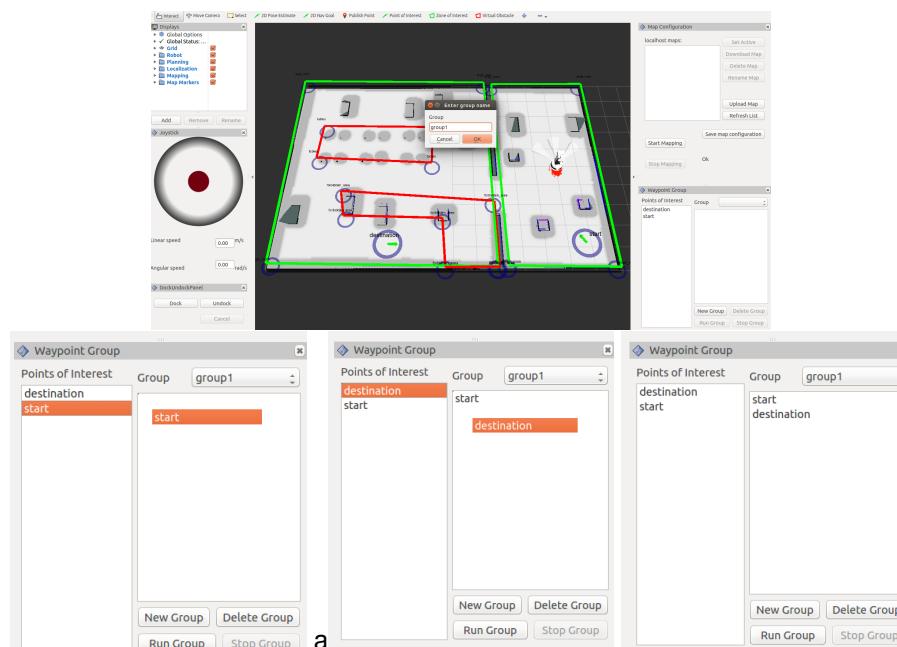


Figure 112: Definition of POI groups

The robot will visit in sequence all the POIs in a group when the group is selected in the dropdown list named **Group** and when pressing the **Run Group**. The task can be cancelled at any time by pressing the **Stop group**. Figure 113 shows the robot running a group of 2 POIs and avoiding VOs. In order to run a group of POIs the action interface `/pal_waypoint/navigate` is available.

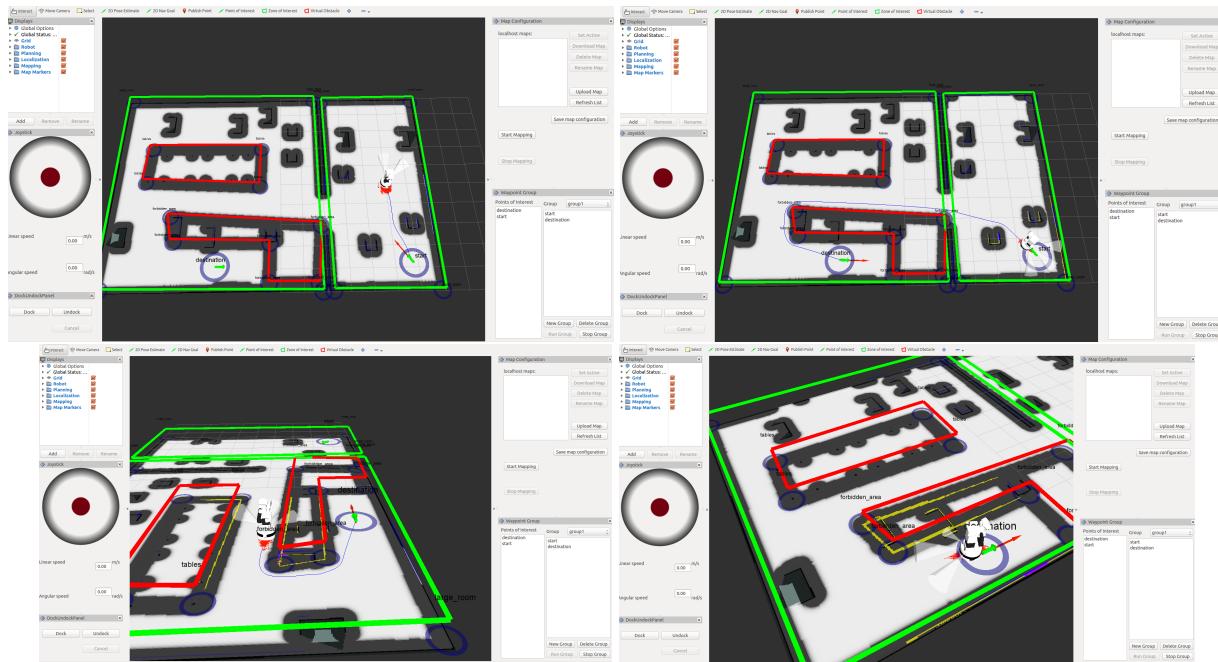


Figure 113: Running a POI group

### 30.6.8 Modifying map meta-data

The POIs, ZOIs and VOs can be modified at any time or removed by placing the mouse icon on top of their blue circle and clicking on the right button. A popup menu will appear providing different options: removing the POI or in case of ZOIs and VOs adding a new vertex, removing a vertex or removing the whole ZOI or VO.

After any modification of the POIs, ZOIs or VOs remember to press **Save map configuration** to store the changes permanently in the map.

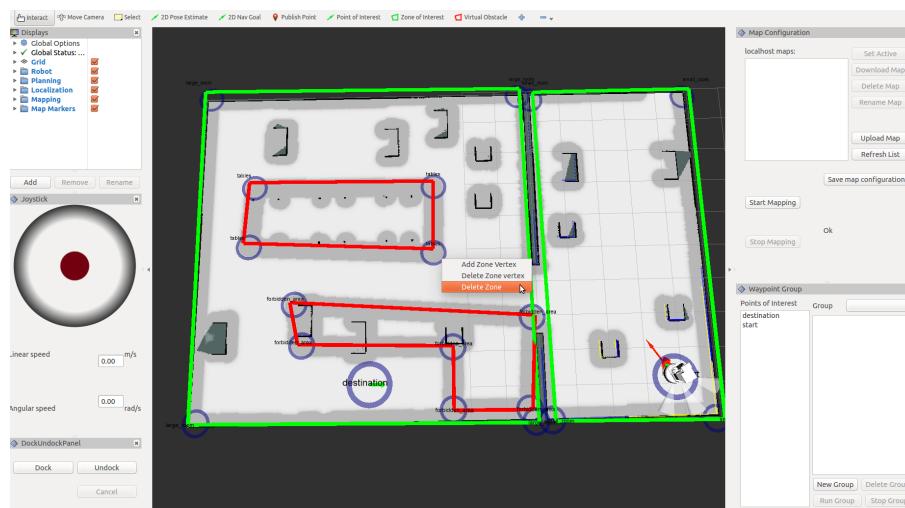


Figure 114: Example of meta-data modification

### 30.7 Obstacle avoidance with the RGB-D camera

The Advanced Navigation package provides obstacle avoidance using the depth information of the RGB-D camera on TIAGo's head. This is very important to safely navigate by avoiding obstacles that can not be detected by the laser scanner but by the camera. A good example of objects that can compromise the navigation safety are tables. Note that the laser can only detect the leg(s) of the table and the robot may collide with it when navigation close to it. Furthermore, a navigation plan crossing the table can be even generated by the global planner.

The obstacle avoidance using the RGB-D camera offers a good solution to these problems and that is why it is enabled by default. The feature is implemented as follows: any time that the robot is requested to go to a map point the following behavior starts:

- The `pal_head_manager` node is disabled to prevent undesired head motions during the navigation
- The torso is raised a bit and the head is lowered so that potential obstacles in front of the robot fall inside the camera field of view, see figure 115
- The point cloud provided by the camera will be continuously integrated in the laser scan in order that obstacles detected by the camera are taken into account
- Both the torso and the camera will remain in this configuration until the robot reaches the goal position in the map. Then, the `pal_head_manager` node is enabled again.



Figure 115: Robot pose adopted to improve RGB-D based obstacle avoidance performance

Figure 116 presents an example in simulation where using the camera for obstacle avoidance is mandatory. Note that the robot is in front of a table that initially only the legs are detected by the laser. The robot is sent to a map point at the other side of the table. The first planned path involves crossing the table, but when the camera detects the table plane the obstacle is added to the costmap and the planner looks for an alternative path. This kind of navigation example fails when using only the laser for obstacle avoidance.

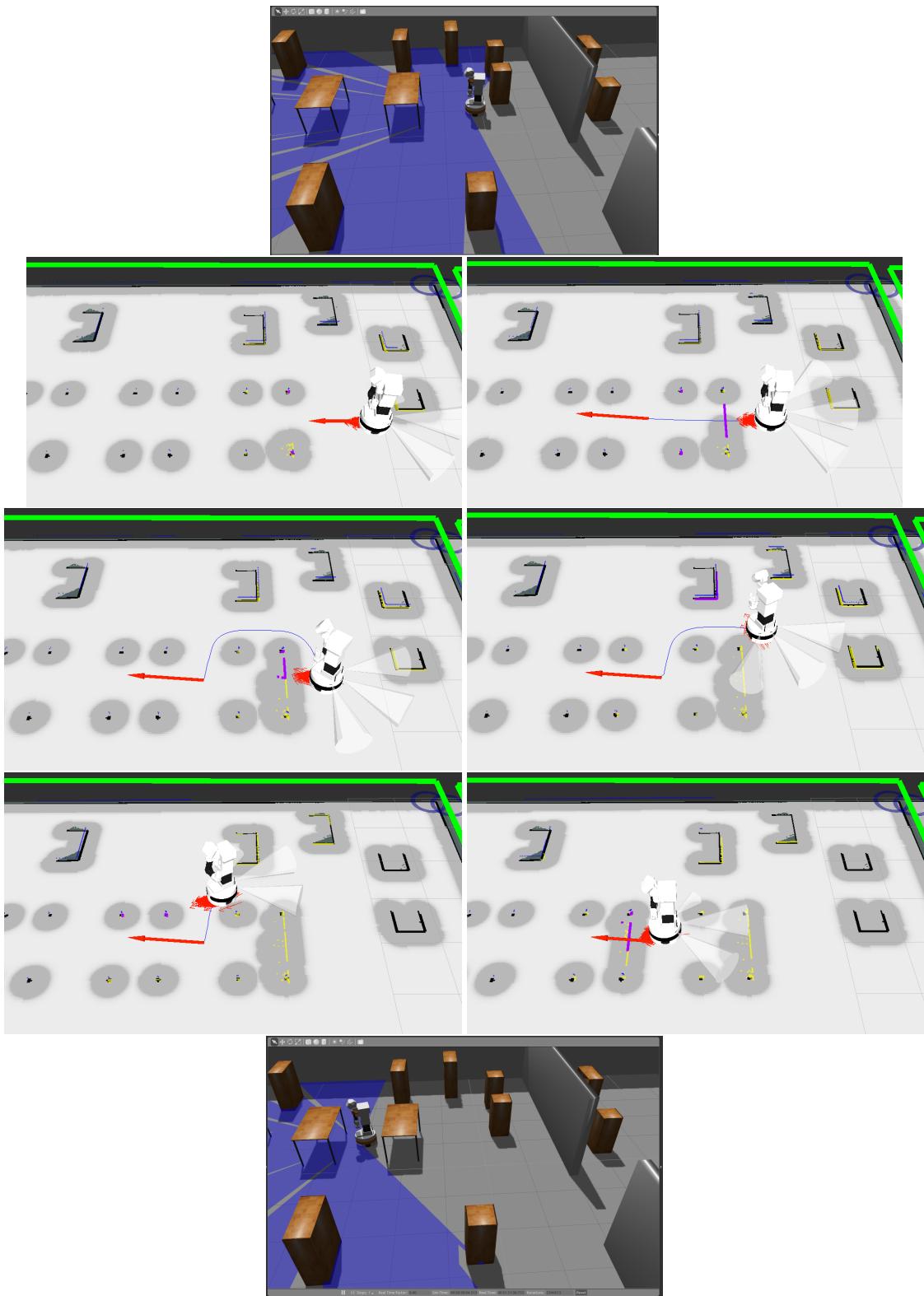


Figure 116: Obstacle avoidance using the camera point cloud

### 30.7.1 Disabling obstacle avoidance with the RGB-D camera

In case that the user needs to disable this feature the following commands can be executed in the robot:

```
ssh pal@tiago-0c
pal-stop navigation_camera_mngr
rosrun dynamic_reconfigure dynparam set \
/move_base/local_costmap/obstacle_rgbd_layer enabled false
rosrun dynamic_reconfigure dynparam set \
/move_base/global_costmap/obstacle_rgbd_layer enabled false
```

### 30.7.2 Enabling obstacle avoidance with the RGB-D camera

In order to re-enabling this feature the following commands can be executed in the robot:

```
ssh pal@tiago-0c
pal-start navigation_camera_mngr
rosrun dynamic_reconfigure dynparam set \
/move_base/local_costmap/obstacle_rgbd_layer enabled true
rosrun dynamic_reconfigure dynparam set \
/move_base/global_costmap/obstacle_rgbd_layer enabled true
```



**TIA Go**

**Dock station**

**PAL** ROBOTICS

The logo for PAL Robotics features the word "PAL" in a large, orange, sans-serif font. To the right of "PAL" is the word "ROBOTICS" in a smaller, black, sans-serif font. Next to "ROBOTICS" is a circular emblem. The emblem has a dark brown or black background with a thin orange border. Inside the circle, there is a stylized, glowing orange "C" shape that curves around a central point, resembling a robotic eye or a sun.



## 31 Dock station

### 31.1 Overview

Depending on the version acquired, your TIAGo may include a docking station that allows the robot to recharge itself automatically. This section describes the components of the docking station and how to integrate it with your algorithms.

### 31.2 The dock station hardware

The dock is composed of a metal structure containing a pattern detected by TIAGo's LIDAR sensor, a connection to the external charger and the power contacts that will transmit energy to the robot, as shown in figure 117. The dock also has tabs that can be screwed to the floor or wall to fix it in place, although this is not required. It is important to emphasize that although the charger possesses several security protections, the user should not touch or meddle with the power contacts, for safety reasons.

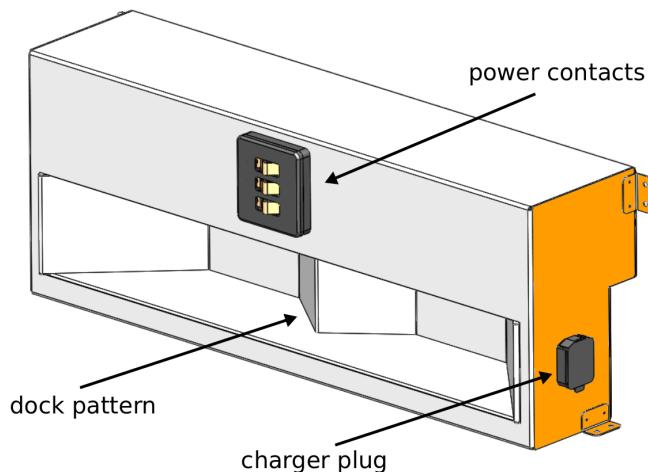


Figure 117: The dock station

### 31.3 Installation

The docking station should preferably be mounted against a hard surface, to avoid the robot displacing it while performing a docking manoeuvre. The power charger must be plugged in the respective plug. The user must also ensure that no objects are present in the dock's surroundings that could interfere with the docking manoeuvre.

### 31.4 Docking algorithm

When the robot is asked to go to the docking station, it activates two services in parallel. The first is responsible for pattern detection, while the second performs the servoing to reach the power contacts:

- pattern detector: the robot is capable of detecting the pattern up to one meter from the LIDAR sensor and with an orientation angle of  $\pm 10^\circ$ .

- servoing manoeuvre: comprises of two steps. First the robot aligns itself with the power contacts and then it advances until the contact is made or a timeout occurs (if the docking station is not powered, or the contact fails, for example).

Figure 118 illustrates the requirements for the docking manoeuvre.

Once the robot is docked, it will block most velocity commands sent to the base, in order to avoid manoeuvres that could potentially damage the robot or the docking station. There are only two ways of moving the robot after it is docked: by performing an undock manoeuvre, or by using the gamepad as it can override all velocity commands.

**WARNING: It is the sole responsibility of the user to operate the robot safely with the gamepad after the robot has reached the docking station.**

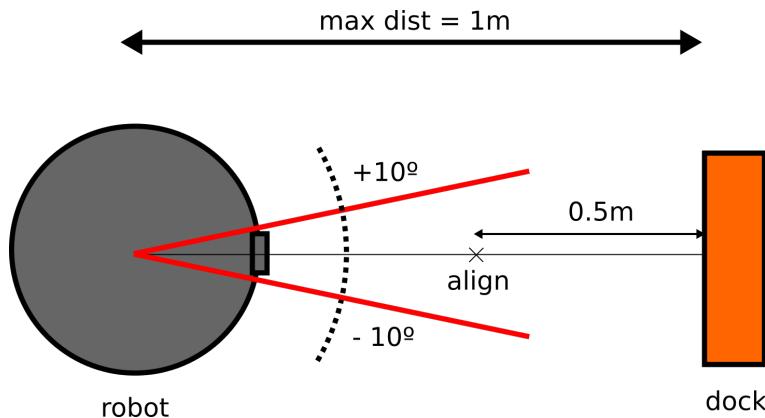


Figure 118: The docking specifications

## 31.5 Usage

The dock/undock manoeuvres are available through two different action servers that can be activated by using the provided rviz plugin or directly through the action server interface.

### 31.5.1 Dock/Undock using rviz plugins

A dock/undock panel is available as an rviz plugin that can be added to any rviz configuration by going to the menu **Panels -> Add New Panel** and then choosing the DockUndockPanel. There is also a preconfigured rviz file that is shipped with the robot and can be loaded with the following command:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun rviz rviz -d `rospack find tiago_2dnav`/config/rviz/advanced_navigation.rviz
```

Make sure to use your robot's serial number when exporting the `ROS_MASTER_URI` variable and to set your development computer's IP when exporting `ROS_IP` as explained in Section 10.4.

Figure 119 shows the layout of the panel.

Once the user has positioned the robot within the tolerances specified previously, they can click the **Dock** button to perform a docking manoeuvre. It is possible to cancel the docking manoeuvre at any time by clicking the **Cancel** button. Similarly, the robot can be moved out from the dock by clicking the **Undock** button. A status message will be shown beside the **Cancel** button, informing the user of the status of the action requested. Note: the robot will only accept an undock order if it was previously docked, otherwise the action request will be rejected.

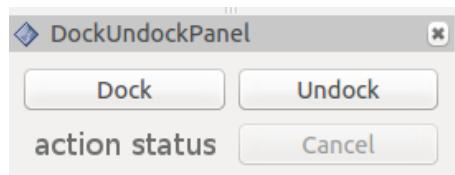


Figure 119: rviz plugin

### 31.5.2 Dock/Undock using action client

ROS provides an action client interface that can be used to communicate with the action servers responsible for the dock and undock manoeuvres. To run the action client, enter the following command to perform the docking manoeuvre:

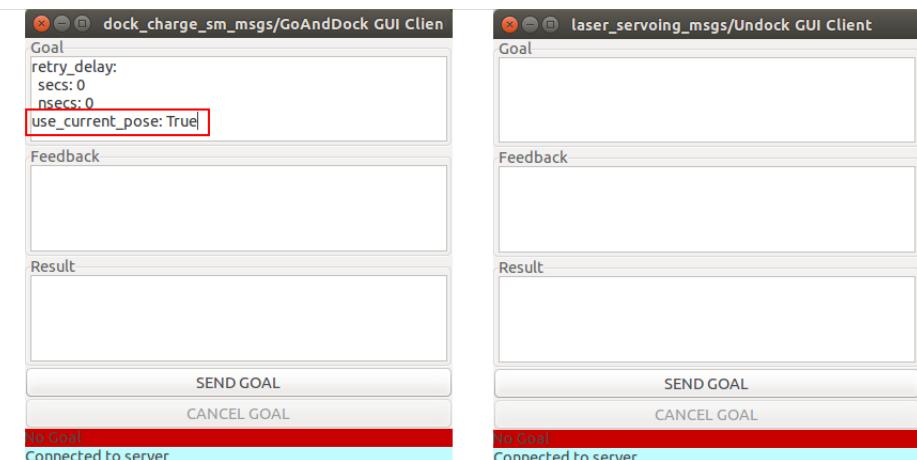
```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun actionlib axclient.py /go_and_dock
```

and for the undocking manoeuvre:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun actionlib axclient.py /undocker_server
```

After any of the previous commands are executed, a panel will pop up. Figure 120 shows both the **/go\_and\_dock** and the **/undocker\_server** panels.

Note: for the **docking action client**, the field **use\_current\_pose** should be set to **True**, otherwise the action will fail (this field is not needed for the **/undocker\_server**). In this interface, the button **SEND GOAL** will start the docking (or undocking) manoeuvre. As before, the **CANCEL GOAL** button will abort the action, and the status of both the server and the goal will be displayed in the bottom of the panel.

Figure 120: The action client for the **docking** and **undocking** manoeuvres

### 31.5.3 Dock/Undock code example

Finally, the user can interact with the action servers directly by code, in either Python or C++. There are plenty of examples of the use of action clients in ROS Wiki. Below is a very simple example in Python code that connects and sends a goal to the **/go\_and\_dock** server. Note that the field **goal.use\_current\_pose** (line 19) is set to **False**, as in the previous example.

```
1 #! /usr/bin/env python
2 import rospy
3 import rospkg
4 import actionlib
5
6 from dock_charge_sm_msgs.msg import GoAndDockAction, GoAndDockGoal
7 from std_msgs.msg import Bool
8
9 class SimpleDock():
10     def __init__(self):
11
12         rospy.init_node('simple_dock')
13         self.dock_checker_sub = rospy.Subscriber("/power/is_docked", Bool, self.is_docked_cb)
14         self.is_docked = False
15
16     def go_and_dock_client(self):
17
18         goal = GoAndDockGoal()
19         goal.use_current_pose = True
20         self.dock_client = actionlib.SimpleActionClient("go_and_dock", GoAndDockAction)
21         self.dock_client.wait_for_server()
22         self.dock_client.send_goal(goal)
23         rospy.loginfo ("goal_sent_to_go_and_dock_server")
24
25     def is_docked_cb(self, is_docked):
26         if (is_docked.data):
27             self.dock_checker_sub.unregister()
28             rospy.loginfo ("simple_docker: the robot is docked!")
29             quit ()
30
31 if __name__ == '__main__':
32     try:
33         sd = SimpleDock()
34         sd.go_and_dock_client()
35         rospy.spin()
36     except rospy.ROSInterruptException:
37         print ("program_interrupted_before_completion")
```

**TIA Go**

**Motion planning**





## 32 Motion planning with *MoveIt!*

### 32.1 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>.

### 32.2 Getting started with the *MoveIt!* graphical user interface in simulation

This subsection provides a brief introduction to some basic use cases of *MoveIt!*. For testing, a TIAGo simulation is recommended. In order to run it with the real robot please refer to Section 32.3.

1. Start a simulation:

```
roslaunch tiago_0_gazebo tiago_gazebo.launch 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 121 shows that by selecting different planning groups, the GUI shows only the relevant “flying end-effectors”.

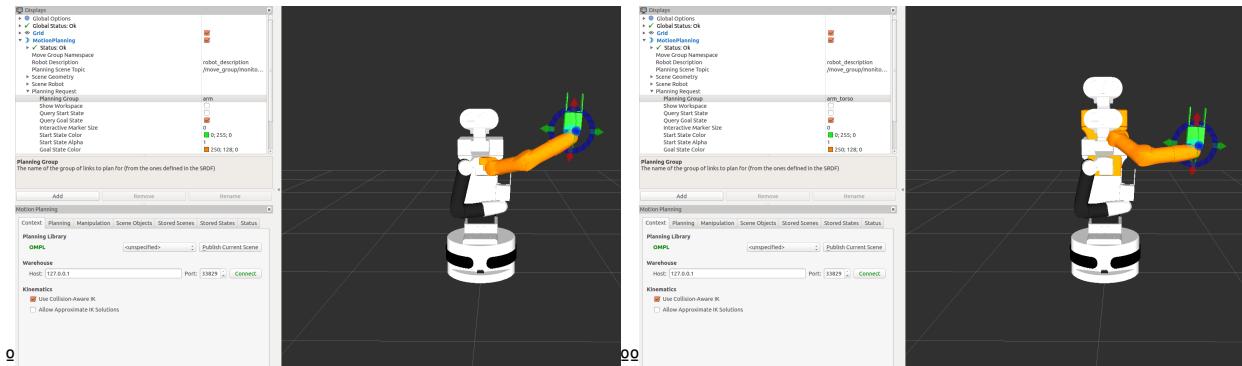


Figure 121: *MoveIt!* graphical interface in rviz. TIAGo with the planning group *arm* (left) and with the planning group *arm\_torso* (right).

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, as depicted in figure 122.
4. By clicking the “Update” button of the Goal State, new poses can be randomized. Figure 123 shows a few randomly generated poses.
5. The sequence of images in figure 124 shows the result of clicking the “Update” goal pose with random inrviz, then clicking “Plan and Execute” in the simulator. Rviz will visualize the plan before executing.
6. The GUI also allows the operator to define goal poses using the “flying end-effector” method. As shown in figure 125, the 6 DoF pose of the end-effector can be defined by using the visual marker attached. The red, green and blue arrows define the translation of x, y and z coordinates respectively, and the colored rings define the rotation around these axes following the same logic.

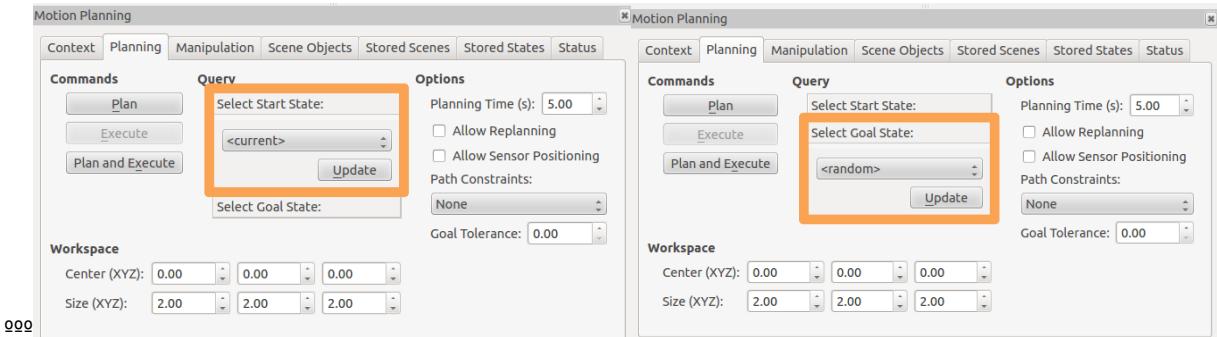


Figure 122: Tab to set start state (left) and tab to set goal state (right).

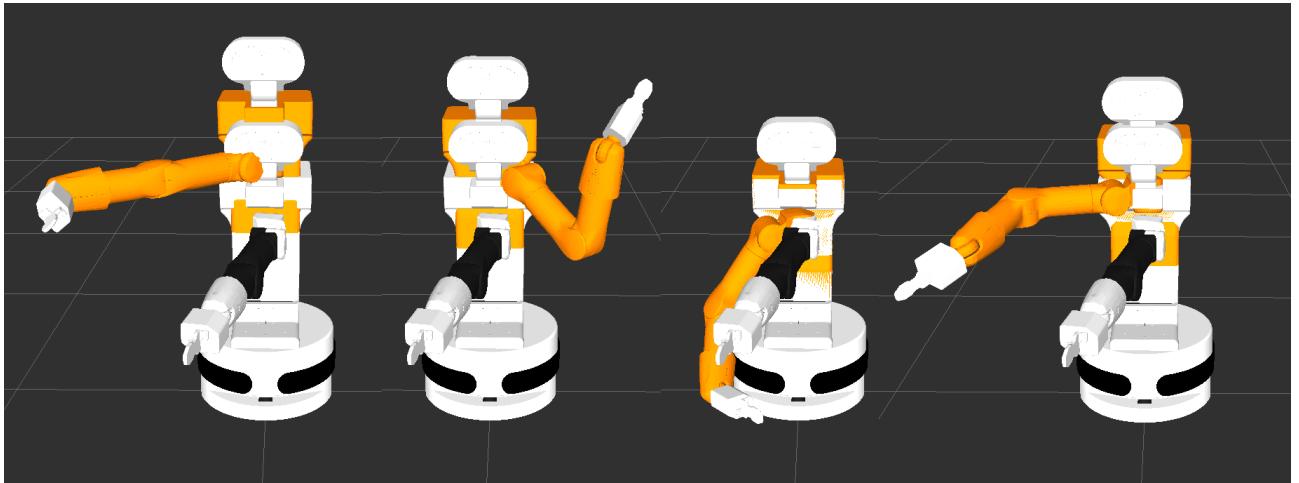


Figure 123: Random poses generated with the GUI

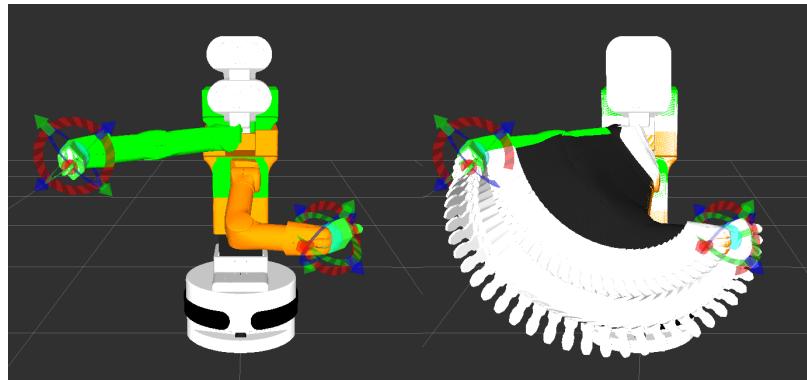


Figure 124: Sequence of plan

### 32.2.1 The planning environment of *MoveIt!*

For a complete description, see [http://moveit.ros.org/wiki/Environment\\_Representation/Overview](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 `meshes/head` folder, as shown in figure 126.

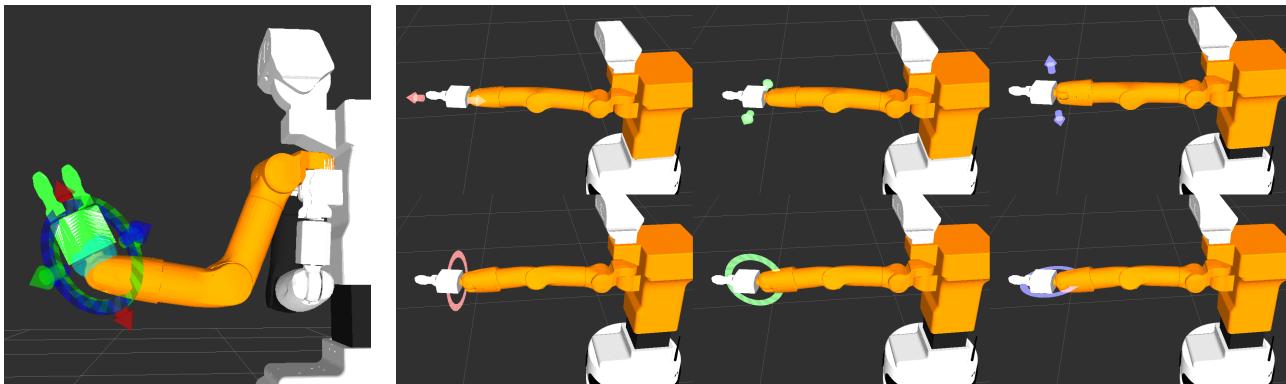


Figure 125: End-effector tool (left) and the usage of each DoF (right)

2. Place the mesh close to 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 on the right of figure 127. Make sure that no body part is in red, which would mean that *MoveIt!* detected a collision in the goal position. Goals which result in a collision state will not be planned at all by *MoveIt!*.
4. (Optional) To execute such a motion, go the “Context” tab and click “*Publish current scene*”. This is necessary for the planning environment in the GUI to be moved to the one used by *MoveIt!* 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 re-planning until it is successful. A succesful plan for the current example is shown in figure 128.

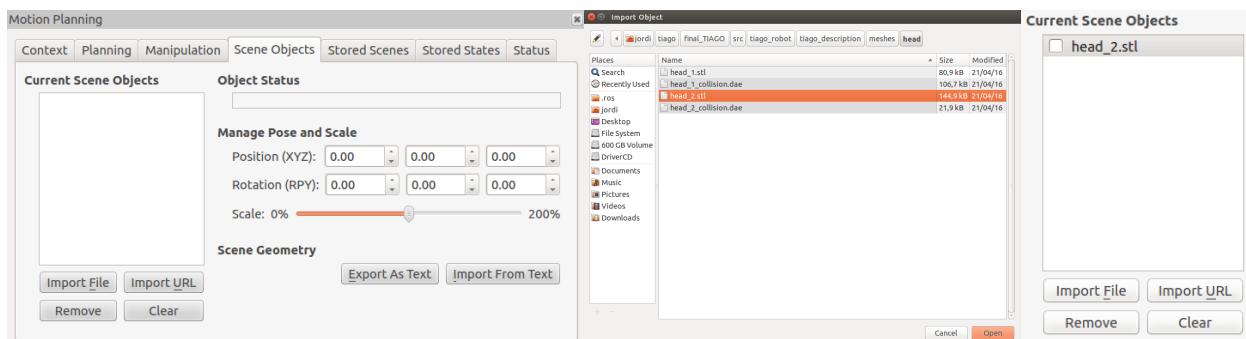


Figure 126: Scene objects tab (left), select the mesh (center), and the object listed in scene objects (right)

### 32.2.2 End test

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

## 32.3 MoveIt! with the real robot

In order to run the examples explained above with the real robot the MoveIt! graphical interface has to be launched from a development computer as follows:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
roslaunch tiago_moveit_config moveit_rviz.launch config:=true
```

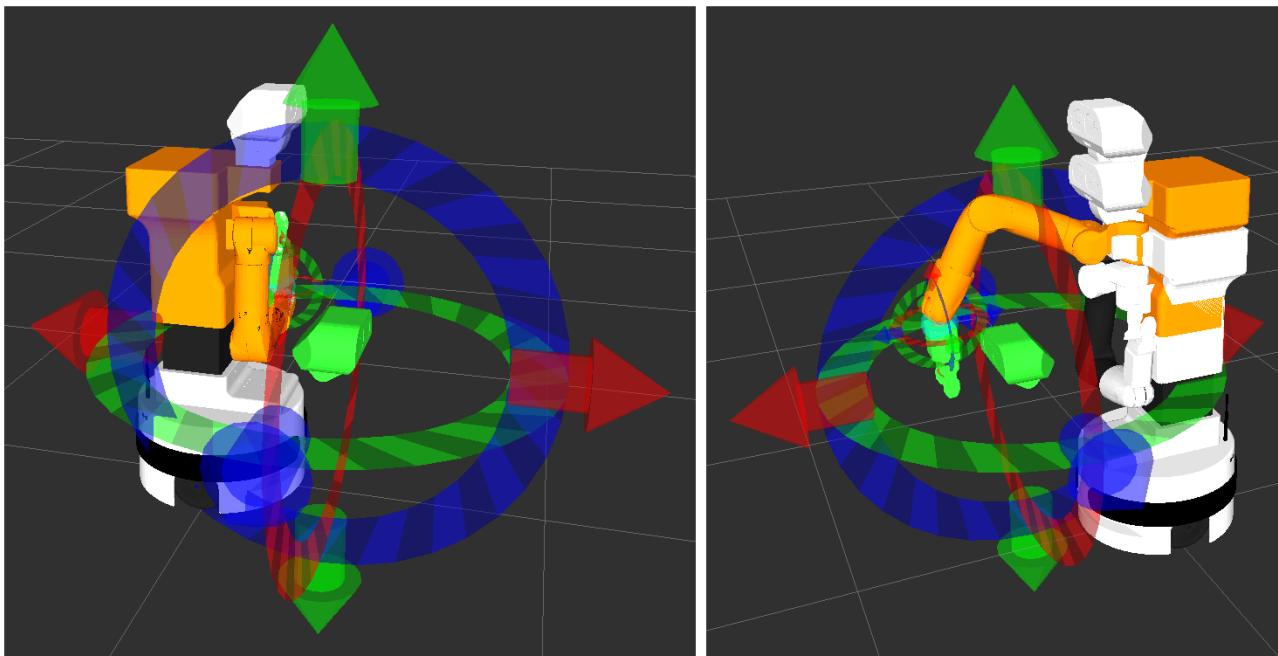


Figure 127: Example with an object in the scene. Initial pose (left) and goal pose (right).

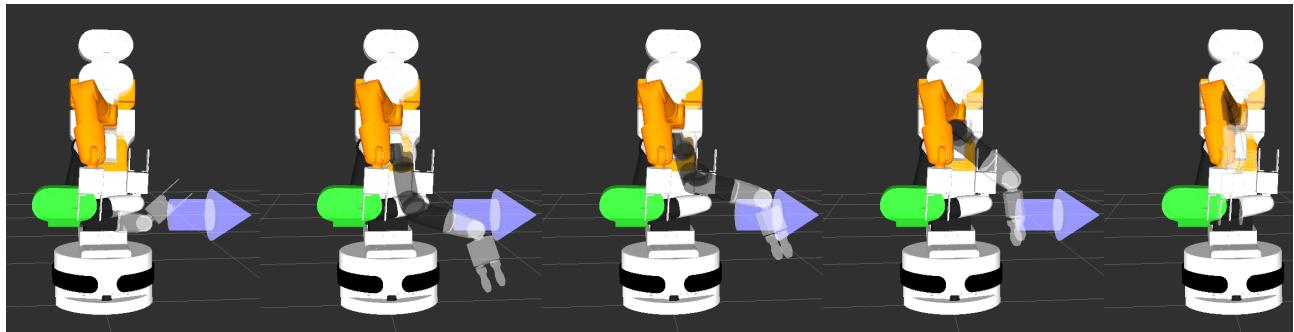


Figure 128: Example of plan found with the object in the scene.

Make sure to use your robot's serial number when exporting the `ROS_MASTER_URI` variable and to set your development computer's IP when exporting `ROS_IP` as explained in Section 10.4.

**TIA Go**

**Facial perception**





## 33 Facial perception

### 33.1 Overview

This chapter presents the software package for face and emotion recognition included in TIAGo<sup>5</sup>.

Face and emotion recognition are implemented on top of the [Verilook Face SDK](#) provided by Neurotechnolgy.

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

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



Figure 129: Face processing example

### 33.2 Facial perception ROS API

#### 33.2.1 Topic interfaces

`/pal_face/faces` ([pal\\_detection\\_msgs/FaceDetections](#))

Array of face data found in the last processed image

`/pal_face/debug` ([sensor\\_msgs/Image](#))

Last processed image with overlayed face data: face ROIs, gender, facial features, name and emotion.

In order to visualize the debug image, enter the following command from a development computer:

```
export ROS_MASTER_URI=http://tiago-0c:11311
rosrun image_view image_view image:=/pal_face/debug
_image_transport:=compressed
```

<sup>5</sup>This software is included in the Facial Perception Premium Software package

### 33.2.2 Service interface

/pal\_face/set\_database ([pal\\_detection\\_msgs/SetDatabase](#))

Service to select the active database in the robot.

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

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

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

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

/pal\_face/start\_enrollment ([pal\\_detection\\_msgs/StartEnrollment](#))

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

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

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

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

/pal\_face/stop\_enrollment ([pal\\_detection\\_msgs/StopEnrollment](#))

Service to stop learning a new face:

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

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

/pal\_face/recognizer ([pal\\_detection\\_msgs/Recognizer](#))

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

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

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

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

In order to disable the recognizer:

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

### 33.3 Face perception guidelines

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

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

**TIA Go**

**Speech recognition**





## 34 Speech recognition

### 34.1 Overview

This chapter presents the software package for online speech recognition included in TIAGo<sup>6</sup>.

When enabled, the ROS package that implements speech recognition captures audio from the robot's microphones and sends it to the recognizers for processing. It returns recognized speech.

It has the following features:

- Continuous speech recognition.
- Recognition after hearing a special keyword.
- Ability to use multiple speech recognizers.
- Current recognizers implemented:
  - [Google Cloud Speech API](#) in 80 languages, accepts [hint phrases](#) to the recognizer for commonly spoken phrases.
  - [DeepSpeech](#) a TensorFlow implementation of Baidu's DeepSpeech architecture trained on American English.

### 34.2 Requirements

- Speech Recognition Premium Software Package.
- An appropriate level of ambient noise. The noisier the environment, the worse the recognition results will be.
- Google Cloud Speech requirements:
  - TIAGo must be connected to the internet and able to reach Google's servers.
  - A valid Google Cloud Speech account. See Section 34.7.
- DeepSpeech requirements:
  - NVIDIA Jetson TX2

### 34.3 Speech Recognition ROS API

#### 34.3.1 Action interface

/kw\_activated\_asr (speech\_multi\_recognizer/KeywordActivatedASR.action)

This action starts the speech recognition, with optional keyword triggering.

The goal message includes the following data:

- **language:** A [BCP47 language tag](#). For instance en-US.

---

<sup>6</sup>This software is included in the Speech Recognition Premium Software package

- **skip\_keyword:** If false (default value), will wait until the special keyword is understood using an offline recognition engine. After detecting the keyword, it will listen and perform one recognition with the online engines.
- **preferred\_phrases:** A list of phrases that are most likely to be recognized (provided to the online recognizer, if it supports it).

The feedback message includes the following data:

- **recognition\_results:** A list of strings with the recognition, if any. If multiple recognizer engines are configured, it will contain one entry per recognizer that performed a successful recognition.

There is no result message - the action ends when the user aborts. While active, it will perform continuous speech recognition.

**Note:** DeepSpeech recognizer only supports the speech recognition in English, so the language param in the action interface does no effect to the recognition.

## 34.4 Recognizer behaviour

The following pseudo code illustrates how the speech recognizer operates.

```
while goal == active:
    if not skip_keyword:
        wait_for_keyword()    <--- OFFLINE (free)
        listen_to_one_sentence()
        recognize_sentence()      <--- ONLINE (paid)
        send_action_feedback()
```

## 34.5 Special keyword

The keyword to enable the speech recognition by default is "Hey Tiago".

## 34.6 Configuration

This system can be configured via yaml files in the multimedia computer.

The application will try to load the file at:

```
$HOME/.pal/pal_audio_cfg_base/speech_recognition.yaml
```

If it doesn't exist, it will look for the file at:

```
`rospack find pal_audio_cfg_base`/config/speech_recognition.yaml
```

This file contains the following parameters:

**device\_name** Name of the device to use for recording (ie: hw:0,0)

**keyword\_model** Absolute path to the [Snowboy](#) model, or the name of a file inside the *config* directory of the *snowboy* catkin package.

**keyword\_recognized\_text** Text the robot will say after recognizing the keyword.

**keyword\_sensitivity** Sensitivity of the keyword recognizer, lower means more false positive, higher can make it more difficult to detect.

### 34.7 Google Cloud Speech account creation

At the time this manual was written, Google offers 60 minutes of online speech recognition per month, free of charge. After that, your account will be billed according to your use of their recognition engine. For more information, refer to <https://cloud.google.com/speech/pricing>

1. Go to <https://cloud.google.com/speech/> .
2. Click on “Try it free”.
3. Log in and agree to the terms and conditions.
4. Complete your personal/company information and add a payment method.
5. You will be presented with your Dashboard. If not, click here (<https://console.cloud.google.com/home/dashboard>).
6. Enter “Speech API” in the search bar in the middle of the screen and click on it.
7. Click on Enable to enable the Google Cloud Speech API.
8. Go to the Credentials page (<https://console.cloud.google.com/apis/credentials>) .
9. Click on “Create credentials,” and on the dropdown menu select: “Service account key”.
10. You’ll be asked to create a service account. Fill in the required info and set **Role** to Project -> **Owner**.
11. A .json file will be downloaded to your computer. **STORE THIS FILE SECURELY: IT IS THE ONLY COPY.**
12. Copy this file to TIAGo’s /home/pal/.pal/gcloud\_credentials.json Google will now use these credentials.



**TIA Go**

**Whole Body Control**





## 35 Whole Body Control

### 35.1 Overview

The Whole Body Control (WBC)<sup>7</sup> is PAL's implementation of the Stack of Tasks<sup>8</sup>. It includes a hierarchical quadratic solver, running at 100 Hz, able to accomplish different tasks with different priorities assigned to each. In order to accomplish the tasks, the WBC takes control of all TIAGo's upper-body joints.

In TIAGo's WBC package, the following Stack of Tasks have been predefined (from highest to lowest priority):

- Joint limit avoidance: to ensure joint limits are never reached.
- Self-collision avoidance: to prevent the arm from colliding with any other part of the robot while moving.

These two tasks are automatically managed by the WBC, and should be always active with the highest priority. Not using them may be potentially dangerous for the robot because it could damage himself.

Then the user may push new tasks to move the end-effector to any spatial configuration and make the robot look to any spatial point. These goals can be changed dynamically as we will see in the following subsections.

The difference between using WBC and other inverse kinematic solvers is that the WBC finds online solutions, automatically preventing self-collisions and ensuring joint limit avoidance.

### 35.2 WBC through ROS interface

When creating new functionalities in TIAGo involving WBC, the user may need to send goals for the arm and head control programmatically. In order to do so, start the WBC as explained in the following steps.

#### Starting the WBC

First, make sure there are no obstacles in front of the robot, as the arm will extend when activating WBC.

The easiest way to start whole body kinematic controller would be to push the button on the web commander. Go to the WBC Demos section from the web commander and push on WBC button as shown in 130

The second option is to change the controllers via rosservice call.

First 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'}"
```

<sup>7</sup>Included in the corresponding Premium software package.

<sup>8</sup>N. Mansard, O. Stasse, P. Evrard, A. Kheddar. A versatile generalized inverted kinematics implementation for collaborative working humanoid robot: the stack of tasks. ICAR 2009.

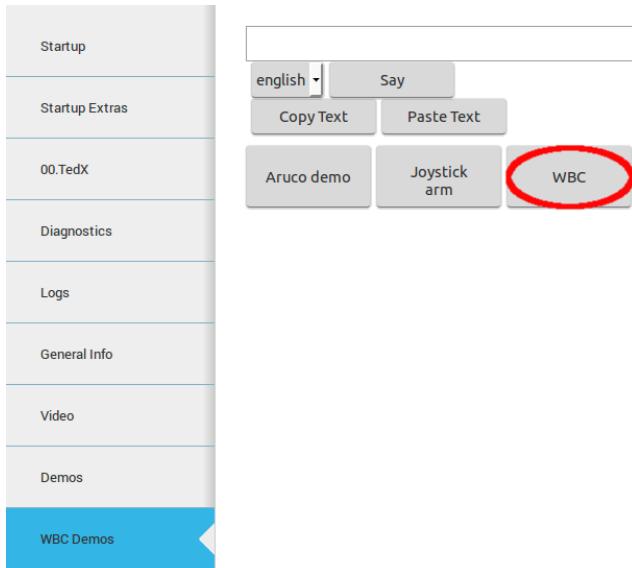


Figure 130: Start WBC

And launch it:

```
roslaunch tiago_wbc tiago_wbc.launch
```

This starts WBC with the basic stack presented in the previous subsection.

If the user only wants to load the wbc controller just run:

```
roslaunch tiago_wbc tiago_wbc.launch spawn:=false
```

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

Once WBC is loaded the user could start it by executing on a terminal connected to the robot:

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

This last step won't work if there are other controllers active that share resources such as arm\_controller, torso\_controller or head\_controller.

### Push tasks to the stack

To push new tasks to the stack there is a specific service to do it:

```
rosservice call /whole_body_kinematic_controller/push_task
"push_task_params:
params: ''
respect_task_id: ''
order:
order: 0
blend: false"
```

Although it requires to define all the params of the task.

The easiest way to push a task to command a specific link to a desired pose, and another one to command the head of the robot to gaze a specific point is to run:

```
roslaunch tiago_wbc push_reference_tasks.launch
source_data_arm:=topic_reflexx_typeII source_data_gaze:=topic
```

By default this will command the arm\_tool link frame. If for example the user wants to command another link execute:

```
roslaunch tiago_wbc push_reference_tasks.launch
source_data_arm:=topic_reflexx_typeII source_data_gaze:=topic tip_name:=arm_7_link
```

After those steps there will be three new tasks that:

- Command the position of /arm\_tool\_link in cartesian space: to allow the end-effector to be sent to any spatial position.
- Command the pose /xtion\_optical\_frame link in cartesian space: to allow the robot to look towards any direction.
- Command the orientation of /arm\_tool\_link in cartesian space: to allow the end-effector to be sent to any spatial orientation.

There are now the following ROS topics:

/whole\_body\_kinematic\_controller/arm\_tool\_link\_goal ([geometry\\_msgs/PoseStamped](#))

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

/whole\_body\_kinematic\_controller/gaze\_objective\_xtion\_optical\_frame\_goal  
([geometry\\_msgs/PoseStamped](#))

Topic to specify the spatial point that the camera on 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_controller/arm_tool_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\_tool\_link to the desired pose, 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\_tool\_link is also specified using a quaternion.

If the orientation or position sent to the robot is unreachable, the robot will try to go to the closest distance or orientation possible.

### Example to send a goal for the head

A goal can be sent through command line as follows:

```
rostopic pub \
  /whole_body_kinematic_controller/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 robot's head will move so that the camera looks towards the point at which the arm had been sent using the previous example.

### Get the current stack

At any time that WBC is active it could be checked the current stack by calling:

```
rosservice call /whole_body_kinematic_controller/get_stack_description
```

It should appear a list with the tasks described before.

### Remove a task from the stack

There is also the possibility to remove one of multiple tasks from the stack online. In a terminal connected from the robot run:

```
rosservice call /whole_body_kinematic_controller/pop_task
"name: 'orientation_arm_tool_link' blend: false"
```

The task to send the `/arm_tool_link` to a specific orientation will be removed.

It can be double checked by getting the current stack, or by trying to send the arm to a specific position with a specific orientation. The user will notice that the robot goes to the desired position with the orientation chosen by the optimizer.

Is very important to take care when removing the tasks. The tasks `self_collision` and `joint_limits` should be never removed since it could damage the robot.

### Stopping the WBC

To stop WBC, open a browser with the web commander and push the Default controllers button located at Demos section (see 131)

This will automatically stop WBC and start the head, torso and arm controllers.

The second option is to change the controllers via rosservice call using a terminal.

Connect to the robot through a terminal and switch the controllers:



Figure 131: Stop WBC

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

### 35.3 WBC upper-body teleoperation with leap motion

Using WBC, the robot can be teleoperated with a leap motion camera, as soon as the robot boots up. This leap motion camera tracks the position and orientation of the hand, as well as the different hands gestures. The user could rotate, move and control the end effector of the robot as his own hand.

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 to ensure 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's local DNS, as explained in section 10.4.

Then, in the development computer, open a terminal and execute the following commands:

Enter as a superuser:

```
sudo su
```

Enter your sudoer password and then execute:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
source /opt/LeapSDK/leap_setup.bash
roslaunch leap_motion_demo leap_motion_demo.launch
```

Make sure to use your robot's serial number when exporting the `ROS_MASTER_URI` variable and to set your development computer's IP when exporting `ROS_IP` as explained in Section 10.4.

Wait until the robot moves to the initial pose shown in figure 133, and until the terminal in the development computer shows the following messages:

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

At this point, the user can place his/her open hand above the leap motion camera, as shown in figure 133b. The terminal will start printing messages like the following:

```
[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's hand is properly tracked and teleoperation commands are being sent to the robot's WBC. You should see the robot's arm, torso and end-effector moving according to the user's hand movements. The leap motion camera is able to track the human hand at several heights, closed and open. The leap motion tracking works best when the hand is kept as flat as possible.

If the leap motion sensor is not tracking the hand, please make sure that the leap motion is connected, and that a red light is lit in the center as shown in figure 132.



Figure 132: Leap motion sensor started

If the light is not on, or the robot didn't move to the WBC default reference, close the previous terminal and restart the process. This issue is being addressed by PAL and should be fixed in future versions.

Once finished, to stop the demo, press **CTRL+C** in the terminal. The robot will then move back to the initial pose.

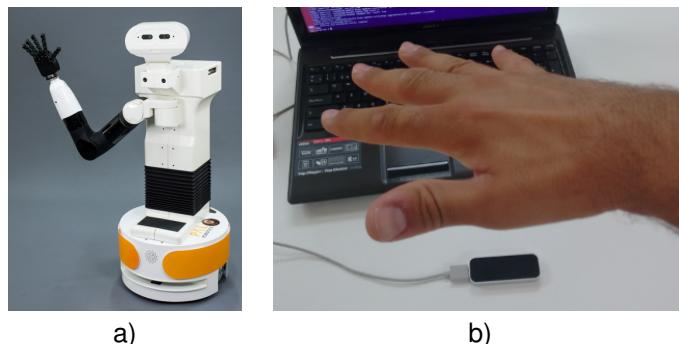


Figure 133: a) Robot initial pose of wbc teleoperation demo. b) Operator hand pose

As the robot won't change automatically from `whole_body_kinematic_controller` to position controllers when stopping the demo. It will be necessary to change the control mode back to position control, by pressing the `Default controllers` button in `Demos` of the WebCommander, as explained before.

### 35.4 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 automatically controlled in order to move and rotate the end-effector in cartesian space. In order to start it, open the WebCommander from your browser and press the the `Joystick arm` button in `WBC Demos` section. (see 134)

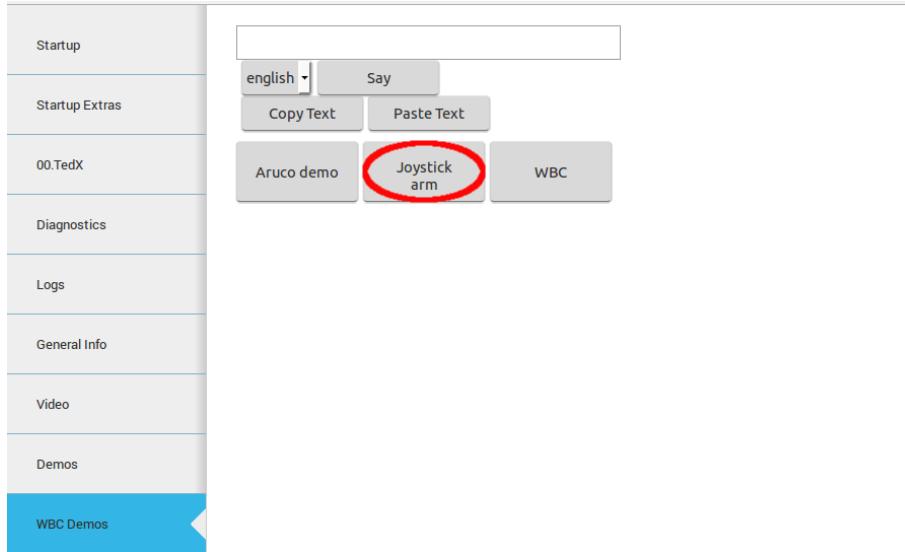


Figure 134: Start teleoperation with joystick

The robot's arm will extend, and the operator will then be able to use the joystick to command the pose. In order to teleoperate, the user has to gain the joystick priority by pressing the button `START`, as shown in figure 135. Note that pressing this button will give the operator control of the mobile base and the upper body alternatively. Press it once or twice until the mobile base is no longer controlled with the joystick.



Figure 135: Button to gain control of the arm

The user can control the position of the end-effector with respect to the `base_footprint` frame, which lies at the center of the robot's mobile base, as shown in figure 136.

The following motions can be controlled with the joystick:

- To move the end-effector along the `X` axis of the reference frame: press the left analog stick as shown in figure 137a. Pressing upwards will move the end-effector forward; pressing downwards will move the end-effector backward.
- To move the end-effector along the `Y` axis of the reference frame: press the left analog stick as shown in figure 137b. By pushing this stick left or right, the end-effector will move laterally in the same direction.
- To move the end-effector along the `Z` axis of the reference frame: press the right analog stick as shown in figure 137c. Pressing upwards will move the end-effector upwards; pressing downwards will move the end-effector downwards.

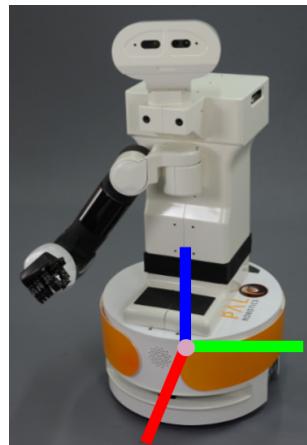


Figure 136: Reference frame of the joystick teleoperation demo. X axis in red, Y axis in green and Z axis in blue.

- To rotate the end-effector along the Y axis of the reference frame: press the left pad as shown in figure 138a.
- To rotate the end-effector along the Z axis of the reference frame: press the left pad as shown in figure 138b.
- To rotate the end-effector along the Y axis of the reference frame: press the LB and LT buttons from the joystick as shown in figure 138c.
- To go back to the initial pose press the BACK button from the joystick. as shown in figure 139.

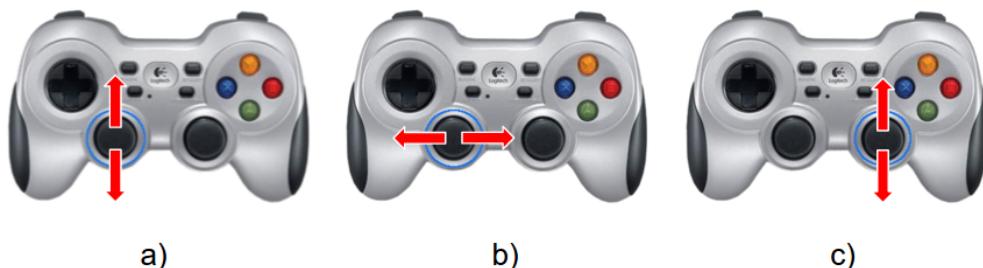


Figure 137: Buttons to move the end-effector in cartesian space

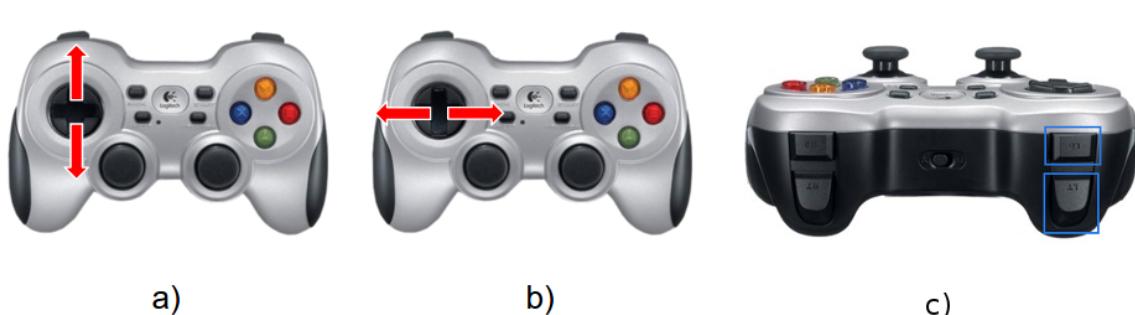


Figure 138: Buttons to rotate the end-effector in cartesian space



Figure 139: Button to go back to the initial position

In order to stop the demo click on the same button which now should appear in blue. This will automatically stop WBC and will restore the default controllers.

### 35.5 WBC with rviz interactive markers

An easy way to test the WBC's capabilities is by using rviz and moving the robot's head and arm by moving interactive markers.

Open a terminal from the development computer and connect to the robot:

Make sure there are no obstacles in front of the robot, as the arm will extend when the WBC is launched.

Launch WBC as stated in [Starting the WBC](#). The easiest way is to launch it from WebCommander.

Then open a new terminal connected to the robot and push the necessary tasks into the stack.

```
ssh pal@tiago-0c
roslaunch tiago_wbc push_reference_tasks.launch
```

Open a new terminal in the development computer and run rviz as follows:

```
source /opt/pal/ferrum/setup.bash
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun rviz rviz -d `rospack find tiago_wbc`/config/rviz/tiago_wbc.rviz
```

Make sure to use your robot's serial number when exporting the `ROS_MASTER_URI` variable and to set your development computer's IP when exporting `ROS_IP` as explained in [Section 10.4](#).

Rviz will show up, as in figure 140, and the user will be able to command the robot's head and end-effector by dragging and rotating the two interactive markers that will appear.

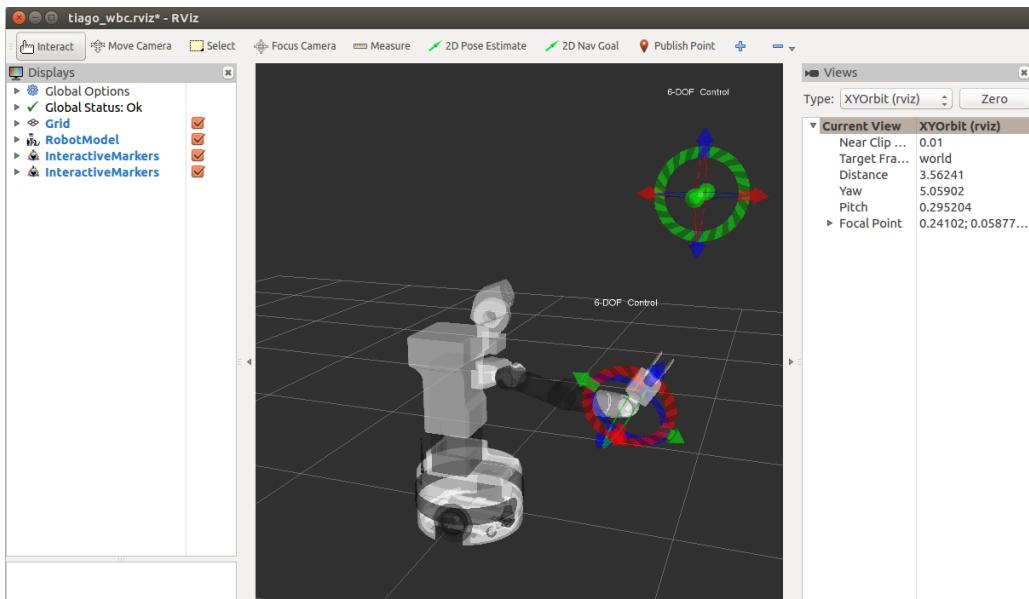


Figure 140: WBC demo with rviz

In order to stop the demo, close rviz by pressing `CTRL+C`, and start the default controllers as explained in Stopping the WBC subsection.

**TIA Go**

**Change controllers**





## 36 Change controllers

In order to change robot controllers, as for example to start whole body kinematic controller or gravity compensation controller, and stop position controllers there are two ways to do it.

- 1- Use the rosservice API with the controller manager services
- 2- Use the change controllers action

### 36.1 Controller manager services

There are three main services.

**List controllers** It lists all the loaded controllers, and shows which of them are active (running) and which of them are inactive (stopped).

```
ssh pal@tiago-0c
rosservice call /controller_manager/list_controllers
```

Also lists the resources used for every controller.

It is important to remark that ROS control doesn't allow two active controllers using a common resource.

**Load controllers** It loads a controller. The parameters for the specific controller must be loaded previously on the param server.

```
ssh pal@tiago-0c
rosservice call /controller_manager/load_controller "name:
'controller_name'"
```

It returns true if the controller has been loaded correctly, and false otherwise.

**Switch controllers** Starts and/or stops a set of controllers. In order to stop a controller this should be active. A controller must be loaded before start it.

```
ssh pal@tiago-0c
rosservice call /controller_manager/switch_controller "start_controllers:
- 'whole_body_kinematic_controller'
stop_controllers:
- 'head_controller'
- 'arm_controller'
- 'torso_controller'
strictness: 0"
```

It is recommended to start and stop the desired controllers in one service call.

In the case of the gravity\_compensation\_controller this is crucial, because once the controller is stopped, the current applied on the arm is zero so it falls down.

The service returns true if the switch has been successfully executed, false otherwise.

**Unload controllers** It unloads a controller. The controller must be stopped before being unload.

```
ssh pal@tiago-0c
rosservice call /controller_manager/unload_controller "name:
'controller_name'"
```

It returns true if the controller has been unloaded correctly, and false otherwise.

Once a controller has been unloaded, in order to restart it, it will be necessary to load it again.

## 36.2 Change controllers action

The change controllers action uses the rosservice API of the controller manager, but allows the change of controllers in a simple and intuitive way in a single call. The user doesn't need to know which controllers are active and which resources are being used at any time. The action automatically stops the active controllers that share resources with those who want to be loaded. Moreover it has different flags in the goal msg:

- switch\_controllers: If true it stops and starts the controller in a single switch service call. Otherwise, first it calls stop and then it calls start. By default this should be always True. Specially when gravity\_compensation\_controller needs to be stopped.

- load: Load the controller that is gonna be started

- unload: Unload the request stop controllers. This doesn't affect the controllers that are automatically stopped because they share resources with the controllers that are gonna be started.

This action is also optimized to avoid issues with PAL controllers, as for example the whole body kinematic controller needs to be unloaded and loaded every time.

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

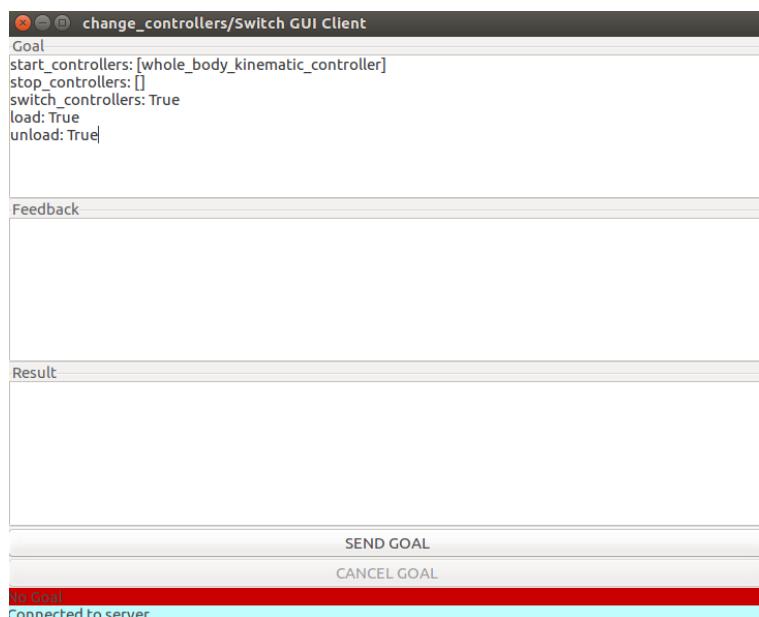


Figure 141: Change controllers action

In the robot the server is automatically launched on the startup. To run it on simulation execute.

```
roslaunch change_controllers change_controllers.launch
```

**TIA Go**

**Introspection controller**





## 37 Introspection controller

The introspection controller is a tool used at PAL to serialize and publish data on the real robot that could be recorded and used later for debugging.

### 37.1 Start the controller

The introspection controller doesn't use any resource, and it could be activated in parallel with any other controller.

In order to start it run:

```
ssh pal@tiago-0c
roslaunch introspection_controller introspection_controller.launch
```

Once the controller is started it will start publishing all the information on the topic: /introspection\_data/full

### 37.2 Record and reproduce the data

If you want to record the information from your experiment, it can simply be done using rosbag.

```
ssh pal@tiago-0c
rosbag record -O NAME_OF_THE_BAG /introspection_data/full
```

Once you are finished to record your experiment simply close it with Ctrl-C

Then copy this file in your development PC

```
ssh pal@tiago-0c
scp -C NAME_OF_THE_BAG.bag pal@development:PATH_TO_SAVE_IT
```

Once in your development PC you can reproduce it using PlotJuggler.

```
rosrun plotjuggler PlotJuggler
```

Once PlotJuggler is open load the bag: File -> Load Data and select the recorded rosbag.

For more information about PlotJuggler please visit: <http://wiki.ros.org/plotjuggler>

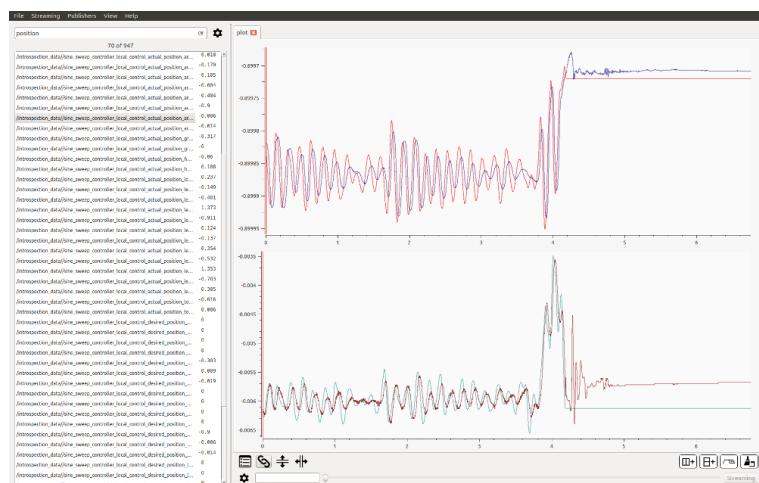


Figure 142: PlotJuggler

### 37.3 Record new variables

In order to record new variables it will be necessary to register them inside your code as follows.

```
#include <pal_statistics / pal_statistics .h>
#include <pal_statistics /pal_statistics_macros.h>
#include <pal_statistics / registration_utils .h>

...
double aux = 0;
pal_statistics :: RegistrationsRAII registered_variables_;
REGISTER_VARIABLE("/introspection_data", "example_aux", &aux, &registered_variables_);

Eigen::Vector3d vec(0,0,0);
REGISTER_VARIABLE("/introspection_data", "example_vec_x", &vec[0], &registered_variables_);
REGISTER_VARIABLE("/introspection_data", "example_vec_y", &vec[1], &registered_variables_);
REGISTER_VARIABLE("/introspection_data", "example_vec_z", &vec[2], &registered_variables_);
...
```

Take in account that the introspection controller only accepts one dimensional variables. For more information please check: [https://github.com/pal-robotics/pal\\_statistics](https://github.com/pal-robotics/pal_statistics)

**TIA Go**

**Simulation**

**PAL** ROBOTICS



## 38 Simulation

### 38.1 Overview

When installing a development computer, as explained in section 8.3, the user can run Gazebo simulations of TIAGo.

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

#### 38.1.1 Empty world

This is the default world that is loaded when the simulation is launched. The robot is spawned in an empty world with no objects, as shown in figure 143. In order to launch the simulation, the following instruction needs to be executed in a terminal:

```
source /opt/pal/ferrum/setup.bash
roslaunch tiago_0_gazebo tiago_gazebo.launch
```

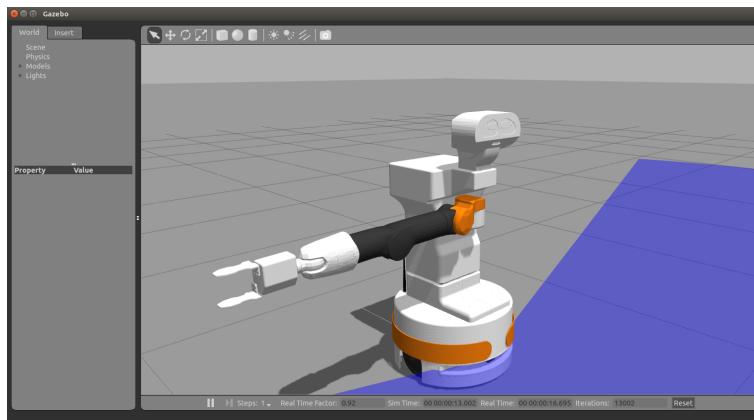


Figure 143: Empty world simulated in Gazebo

#### 38.1.2 Office world

The simple office world shown in figure 144 can be simulated with the following instruction:

```
source /opt/pal/ferrum/setup.bash
roslaunch tiago_0_gazebo tiago_gazebo.launch world:=small_office
```

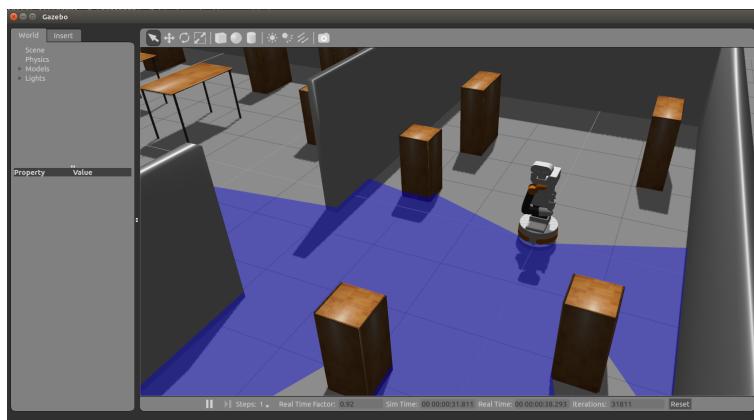


Figure 144: Small office world simulated in Gazebo

### 38.1.3 Table with objects world

In this simulation, TIAGo is spawned in front of a table with several objects on top, see figure 145. The instruction needed is:

```
source /opt/pal/ferrum/setup.bash  
roslaunch tiago_0_gazebo tiago_gazebo.launch world:=objects_on_table
```

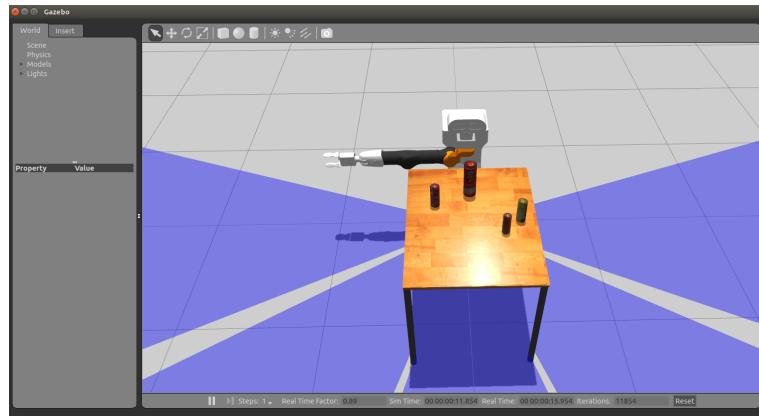


Figure 145: Tabletop scenario simulation

**TIA Go**

**LEDS**

**PAL**   
ROBOTICS



## 39 LEDs

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

### 39.1 ROS API

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

Keep in mind that there is an application running in TIAGo that changes the LED 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` script as follows.

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

#### 39.1.1 Available services

The following services are used for controlling the LED strips present in TIAGo. All 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 be 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 be 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 back and forth starting from the left, and 4 for pixels running back and forth 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**

**Touch Screen**

**PAL**   
ROBOTICS



## 40 Modifying the base touch screen

This section describes how the menu of the robot's base touch screen is implemented and how to modify it.

### 40.1 Introduction

TIAGo touch screen is configured via YAML files that are loaded as ROS Parameters upon robot startup.

The menu can display labels and buttons, and the buttons can be pressed to navigate to another menu, or to execute a command.

The application will look for a directory in `$HOME/.pal/touch_display_manager_cfg`, if it does not exist it will load the configuration in the `touch_display_manager_cfg` ROS package's `config` directory.

### 40.2 Configuration file structure

Below is an example configuration file

```
main_menu:
  type: ConfigurableMenu
  params:
    - text: "Example menu"
      entry_type: Label
    - text: "Mute"
      entry_type: ActionButton
      action:
        remote_shell:
          cmd: "rosparam set /pal/playback_volume 0"
          target: "control"
    - text: "Unmute"
      entry_type: ActionButton
      action:
        remote_shell:
          cmd: "rosparam set /pal/playback_volume 85"
          target: "control"
```

At the root of the file, there are the different menus, there must exist at least one menu with the name **main\_menu**, this is the menu loaded by default.

Each menu must specify its **type**, as well as parameters (**params**) that depend on it's type.

### 40.3 Menu types

#### 40.3.1 ConfigurableMenu

A configurable menu is a generic menu that can be programmed to display labels and buttons.

The parameters are a list of entries, each entry is a dictionary with at least:

**text** The text to be displayed in the label or button

**entry\_type** The type of the entry, can be Label, Button, ReturnButton or ActionButton

Each entry\_type has it's own parameters, that are described below.

**Label** This entry will display some text, and is not clickable. It doesn't require extra parameters.

**Button** This entry, when pressed, will navigate to another menu. It requires one parameter named **show\_menu**, which must contain the id of the menu as written in the yaml file.

**ReturnButton** This entry, when pressed, will return to the previous menu. It requires one parameter named **return\_text**, which is used internally in some situations. It can be set to an empty string.

**ActionButton** This entry, when pressed, will execute some action. It requires a parameter named **action**, syntax of this parameter is the same as the button syntax in 11.4.12.

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

### 40.3.2 EmergencyMenu

A menu that is activated when emergency button is pressed, and displays another menu.

It has a single parameter, **emergency\_submenu**, which should contain the name of the menu that must be displayed.

### 40.3.3 Examples

Emergency menu that displays that the emergency button has been displayed.

```
main_menu:
  type: EmergencyMenu
  params:
    emergency_submenu: emergency_submenu
emergency_submenu:
  type: ConfigurableMenu
  params:
    - text: "Emergency Pressed"
      entry_type: Label
    - text: ""
      entry_type: Label
    - text: "Exit"
      entry_type: ReturnButton
      return_text: "Exit"
```

Main menu to navigate to submenu, with a button for sending a sentence to the TTS engine:

```
main_menu:  
    type: ConfigurableMenu  
    params:  
        - text: "Main Menu"  
          entry_type: Label  
        - text: "Show submenu"  
          entry_type: Button  
          show_menu: submenu  
submenu:  
    type: ConfigurableMenu  
    params:  
        - text: "SubMenu"  
          entry_type: Label  
        - text: "Say something"  
          entry_type: ActionButton  
          action:  
              say:  
                  text: "This is the text that will be said"  
                  lang: "en_GB"  
        - text: "Exit"  
          entry_type: ReturnButton  
          return_text: "Exit"
```



**TIA Go**

**Tutorials**





## 41 Tutorials

### 41.1 Overview

A comprehensive set of tutorials are provided in the public ROS wiki of TIAGo in <http://wiki.ros.org/Robots/TIAGO/Tutorials>. The source code is hosted in PAL Robotics' github public repositories, under [https://github.com/pal-robotics/tiago\\_tutorials](https://github.com/pal-robotics/tiago_tutorials).

### 41.2 Installing pre-requisites

The following ROS package needs to be installed in the development computer:

```
sudo apt-get install ros-melodic-humanoid-nav-msgs ros-melodic-moveit-commander
```

### 41.3 Downloading source code

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

```
mkdir ~/tiago_public_ws
cd ~/tiago_public_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/aruco_ros.git
git clone https://github.com/pal-robotics/tiago_tutorials.git
```

### 41.4 Building the workspace

In order to build the workspace, do the following:

```
cd ~/tiago_public_ws
source /opt/pal/ferrum/setup.bash
catkin build
```

### 41.5 Running the tutorials

In order to run the different tutorials, refer to [and skip the first section referring to Tutorials Installation](#).

The tutorials in the ROS wiki are intended to be run with the public simulation model of TIAGo. In order to run them for your customized robot model, take into consideration to following:

- When running the tutorials in simulation the following arguments, which are stated in the ROS wiki, have to be removed in order to use your custom TIAGo version:
  - public\_sim:=true
  - robot:=steel
  - robot:=titanium

Also, `tiago_0_gazebo` must be used instead of `tiago_gazebo` or `tiago_2dnav_gazebo`. For example, when it suggests the operator runs the simulation like this:

```
roslaunch tiago_gazebo tiago_gazebo.launch public_sim:=true robot:=steel
```

Run the following command instead:

```
roslaunch tiago_0_gazebo tiago_gazebo.launch
```

- When running the tutorials against the actual robot, run them from the development computer with the `ROS_MASTER_URI` pointing to the robot computer, i.e.:

```
export ROS_MASTER_URI=http://tiago-0c:11311  
export ROS_IP=10.68.0.128
```

Make sure to use your robot's serial number when exporting the `ROS_MASTER_URI` variable and to set your development computer's IP when exporting `ROS_IP` as explained in Section 10.4.

The tutorials cover different areas, including control and motion generation, autonomous navigation, motion planning and grasping, and perception with OpenCV and PCL.

**TIA Go**

**Nvidia Jetson**

**PAL** ROBOTICS

The logo for PAL Robotics features the word "PAL" in a large, orange, sans-serif font. To the right of "PAL" is the word "ROBOTICS" in a smaller, black, sans-serif font. Next to "ROBOTICS" is a circular emblem. The emblem consists of a dark brown circle with a thin orange border. Inside the circle is a stylized, glowing orange "C" shape that curves upwards and to the right, resembling a flame or a gear.



## 42 NVIDIA Jetson TX2

### 42.1 Overview

The NVIDIA Jetson TX2 add-on <sup>9</sup> is a dedicated AI computing device developed by NVIDIA that can be integrated into TIAGo.

The TX2 provided with TIAGo is installed by PAL Robotics with an arm64 Ubuntu 16.04.



Figure 146: PAL Jetson add-on

### 42.2 Installation

The add-on device is meant to be attached to the back of TIAGo and connected to the expansion panel of the robot as shown in figure 147.



Figure 147: PAL Jetson add-on attached to TIAGo

<sup>9</sup>Included in the corresponding Premium hardware package.

In order to attach the add-on the following components are provided:

- NVIDIA Jetson TX2 encapsulated and with power supply cable, see figure 148a.
- Ethernet cable to connect the device to the expansion panel, see figure 148b.
- Specific fasteners DIN 912 M3x18 to attach the device to the back of the robot, see figure 148c.

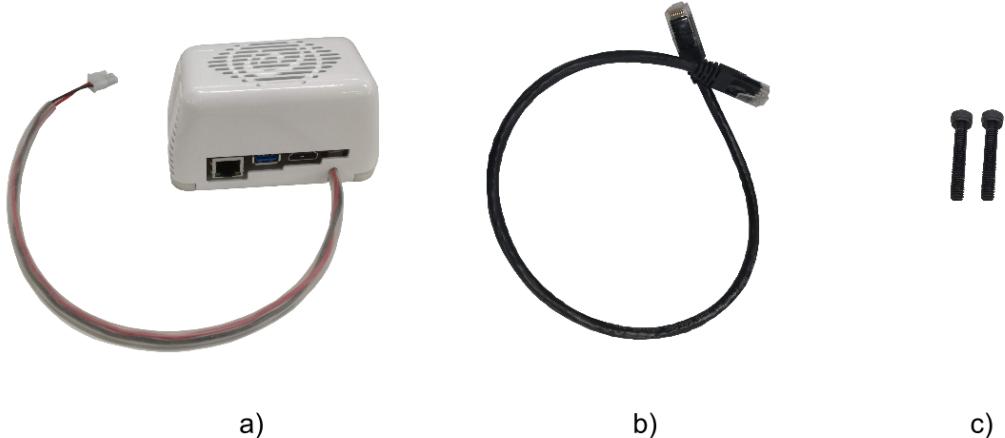


Figure 148: NVIDIA Jetson TX2 add-on contents

The installation steps are as follows:

- Remove the two fasteners of the covers of the back of TIAGo as shown in figure 157a.
- Use the DIN 912 M3x18 provided fasteners to attach the add-on using the holes of the previous fasteners, see figure 157b.
- Connect the power supply cable of the add-on to the power source connector of the expansion panel and use the ethernet cable provided to connect the device to one of the GigE ports of the expansion panel, see figure 157c.

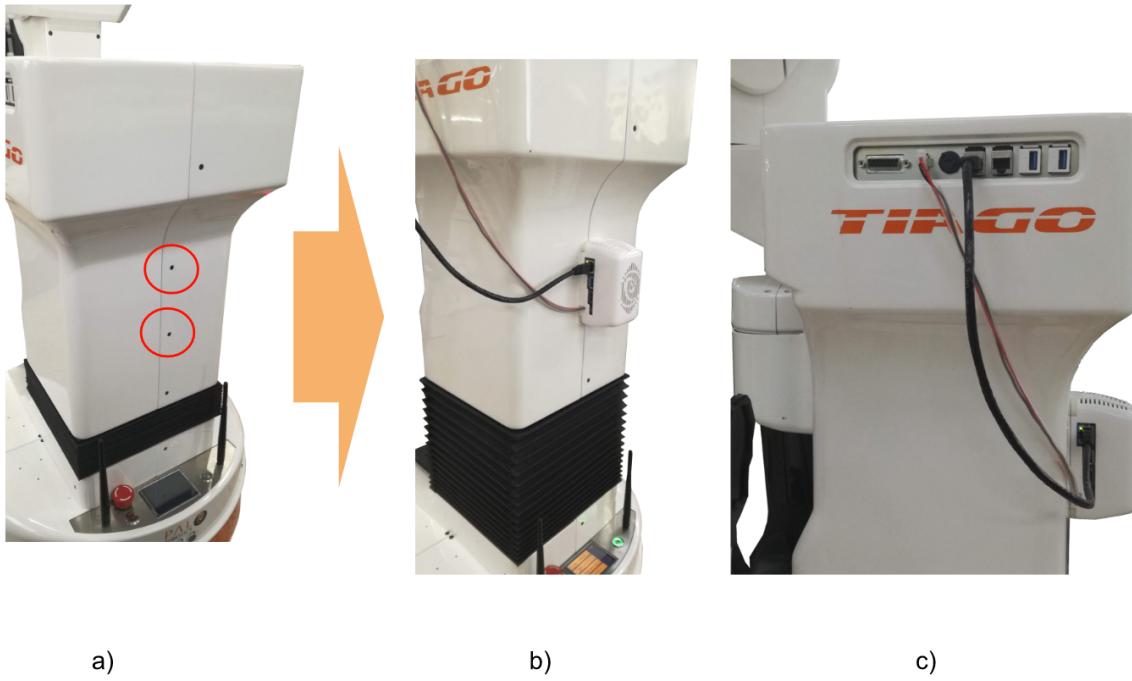


Figure 149: Steps to install the add-on

## 42.3 Connection

To access the TX2, the user must log into the robot's control computer and then connect to the TX2 using its hostname *tiago-0j*:

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

The default user is **pal**, with password **pal**.

The super user password is **palroot**.

## 42.4 Provided example

We have published and provide installed an example that wraps the **Tensorflow Object Detection** with a ROS action server.

[https://github.com/pal-robotics/inference\\_server](https://github.com/pal-robotics/inference_server)

With TIAGO, the example is provided inside a Docker image installed on the TX2, which isolates it from the potential changes to the installed libraries.

It is started automatically when the TX2 boots, and it can be used following the documentation on the example's repository.

Keep in mind that the TX2 is not accessible from outside the control pc. So all applications that interact with it need to be run inside *tiago-0c*.

### 42.4.1 Object detection API

The Object and Person Detection example starts an action server with the name */inference\_server* and of type

*inference\_server/InferenceAction* which has the following structure:

```
sensor_msgs/CompressedImage input_image
---
sensor_msgs/CompressedImage image
int16 num_detections
string[] classes
float32[] scores
sensor_msgs/RegionOfInterest[] bounding_boxes
---
```

To run it, send a goal and the server will consider the *input\_image* for the inference, if the input image is empty, then it capture an image from TIAGo's camera and return the following fields, sorted by their score:

**image** Resultant image after inference from Object and Person Detection API

**num\_detections** Number of detected objects in the inference image

**classes** Name of the class to which the object belongs (depends on the model used for the inference)

**scores** Detection scores or the confidence of the detection of the particular object as a particular class

**bounding\_boxes** Bounding box of each of the detected object

The node will also publish the */inference\_image/image\_raw* image topic displaying the detections.

#### 42.4.2 How to test the example

In order to test the Object detection action server provided the following instructions may be used in a development computer that can access the tiago-0c computer:

Open a terminal an run these instructions:

```
ssh pal@tiago-0c
rosrun topic_tools relay /inference_image/image_raw/compressed \
/jetson_detections/compressed
```

This command will take the image topic published by tiago-0jand republish it on the tiago-0cwhich is accessible from outside.

Open a second terminal and run an *image\_view* node that will show the image with the detected objects:

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun image_view image_view image:=/jetson_detections \
_image_transport:=compressed
```

Remember to assign the actual IP of the development computer to the *ROS\_IP* on the instructions above.

On a third terminal run the following commands in order to enable the object detection in the NVIDIA Jetson TX2:

```
ssh -X pal@tiago-0c
rosrun actionlib axclient.py /inference_server
```

when the GUI shows up, see figure 150, press the *SEND GOAL* button.

After sending the action goal the *image\_view* node started on the second terminal will refresh the image where the detected objects will be shown, see figure 151.

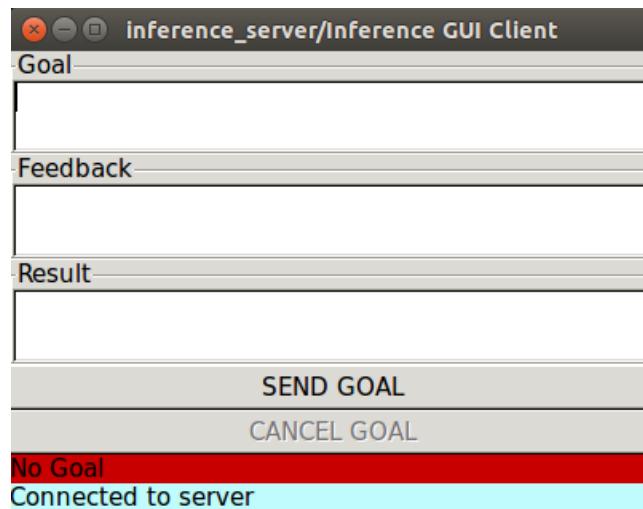


Figure 150: axclient GUI to send a goal to the object detector action server

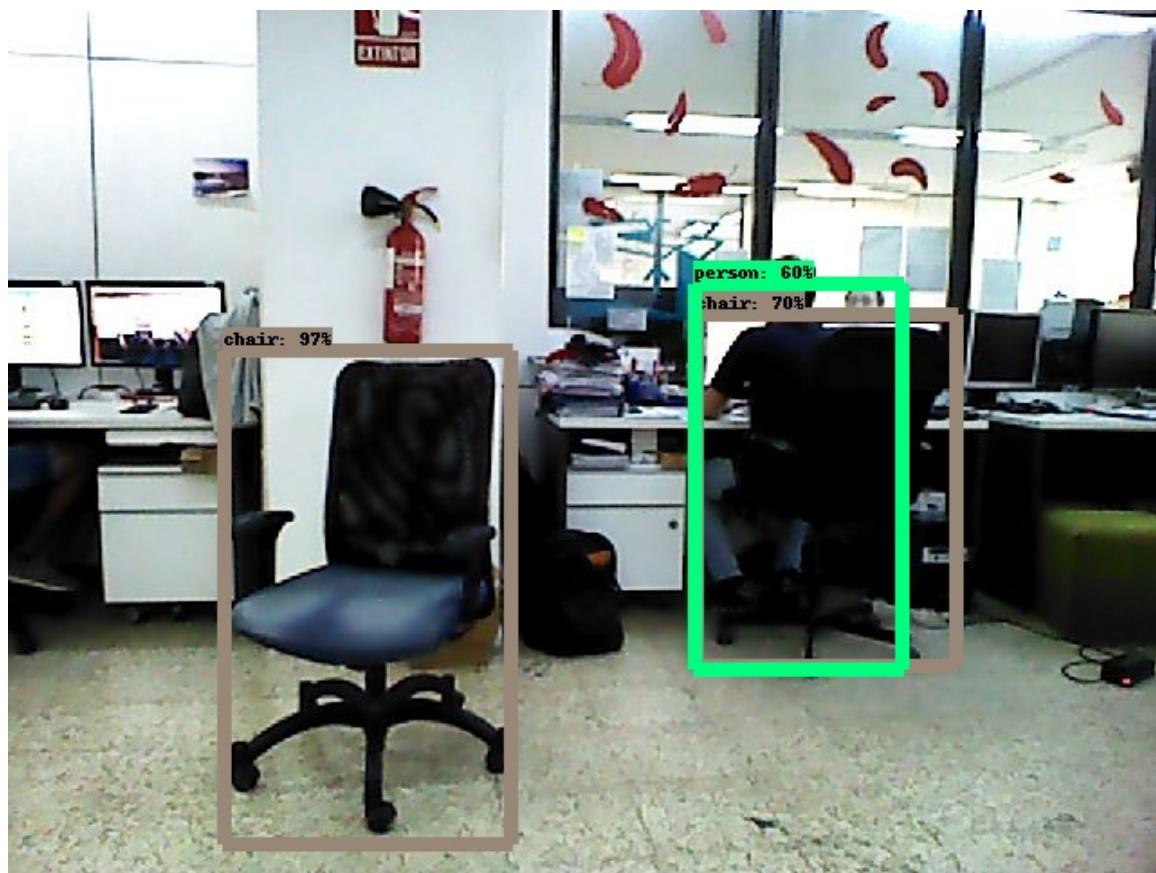


Figure 151: Object and person detection image example



**TIA Go**

**Velodyne VLP-16**

**PAL** ROBOTICS

The logo for PAL Robotics features the word "PAL" in a large, orange, sans-serif font. To the right of "PAL" is the word "ROBOTICS" in a smaller, black, sans-serif font. Next to "ROBOTICS" is a circular emblem containing a stylized orange "C" shape.



## 43 Velodyne VLP-16

### 43.1 Overview

The Velodyne VLP-16 add-on<sup>10</sup> is a 3D LiDAR sensor with a range of 100 m. It supports 16 channels with ~300,000 points/second. VLP-16 has 360° horizontal field of view and 30° vertical field of view(±15° up and down). Velodyne VLP-16 LiDAR Puck sensor has been developed by Velodyne that can be integrated into TIAGo.



Figure 152: Velodyne VLP-16 LiDAR add-on

### 43.2 Configuring Velodyne

The velodyne provided is configured with the IP 10.68.0.55 . In order to access the velodyne configuration page, the connection with the development computer should be configured as following:

#### Configure the development computer's IP address through the Gnome Interface

1. Access the Gnome Menu (Super key), type "Networks Connections" then run it. Select the connection's name and click on "edit". Choose the IPv4 Settings tab and change the "Method" field to "Manual" from the drop-down list.
2. Click on "add" and set the IP address field to 10.68.0.1 ("1" can be any number in a range between 1 and 254, except 55). As the 10.68.0.55 address has already been taken by the sensor.
3. Set the "Netmask" to 255.255.255.0 and the "Gateway" to 0.0.0.0.
4. The settings should look similar as shown in the Figure 153
5. To finalize the settings click on "Save".

#### Accessing Velodyne Configuration

After finalizing the network configuration, power up the velodyne and connect it to the development computer. In the network connection, select the previously configured network configuration to establish the connection with the Velodyne sensor.

To check the configuration, open a web browser and access the sensor's network address: 10.68.0.55. The following page as shown in the Figure 154. The configuration of the sensor such as the firmware update, Laser return type, Motor RPM, Field of View, Network settings etc. can be done in this page.

<sup>10</sup>Included in the corresponding Premium hardware package.

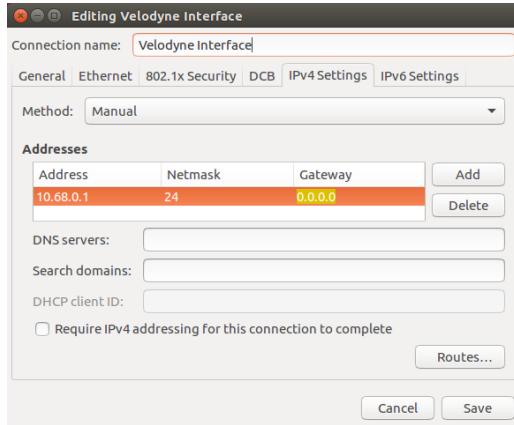


Figure 153: Velodyne network connection settings

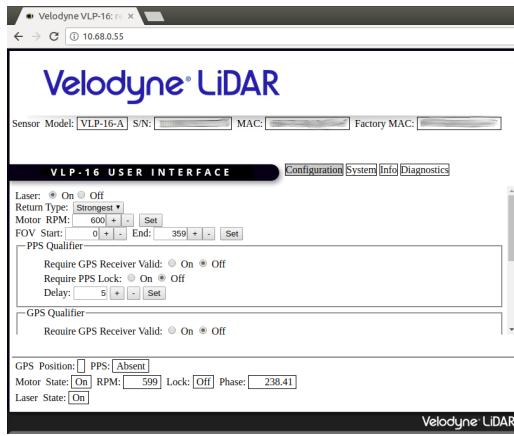


Figure 154: Velodyne configuration page

### 43.3 Installation

The add-on device is meant to be attached to the laptop tray of TIAGo and connected to the expansion panel of the robot as shown in figure 155.

In order to attach the add-on the following components are provided:

- Velodyne VLP-16 along with its mounting, see figure 156a.
- Ethernet cable and the power cable to connect the device to the expansion panel, see figure 156b.
- Specific fasteners DIN 912 M3x12 and washers DIN 9021 ø3 to attach the device to the back of the robot, see figure 156c.

The installation steps are as follows:

- Use the fasteners DIN 912 M3x12 and washers DIN 9021 ø3 provided to attach the add-on using the holes on the laptop tray of TIAGo, see figure 157.
- Connect the power supply cable of the add-on to the power source connector of the expansion panel and use the ethernet cable provided to connect the device to one of the GigE ports of the expansion panel, see figure 157.



Figure 155: Velodyne VLP-16 add-on attached to TIAGo

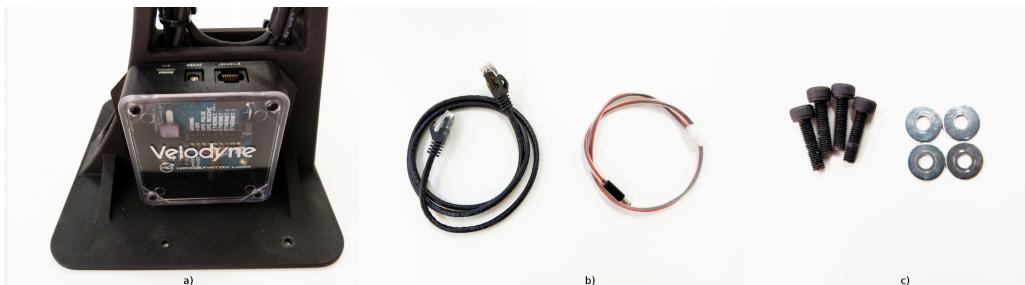


Figure 156: Velodyne VLP-16 kit contents

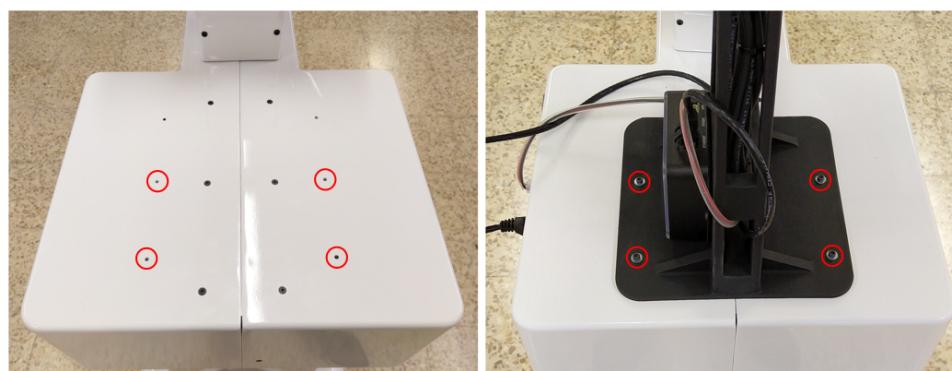


Figure 157: Steps to install the add-on

#### 43.4 Starting velodyne drivers

Once the velodyne is connected to the robot, connect to the robot and start the velodyne startup application manually as shown below to start receiving data from the sensor.

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

Once the velodyne application is started, velodyne data should be published in ROS. This could be verified by the following commands:

```
ssh pal@tiago-0c
rostopic list | grep velodyne
```

The laser scan information from the velodyne can be access from the topic `/scan_velodyne`.

**TIA Go**

**Optris Thermal Camera**

**PAL** ROBOTICS

The logo for PAL Robotics features the word "PAL" in a large, orange, sans-serif font. To the right of "PAL" is the word "ROBOTICS" in a smaller, black, sans-serif font. To the right of "ROBOTICS" is a circular emblem. The emblem consists of a dark brown circle with a thin orange border. Inside the circle, there is a stylized white "C" shape that also forms a partial letter "O".



## 44 Optris Thermal Camera

### 44.1 Overview

The Optris Thermal Camera add-on<sup>11</sup> is a thermographic camera with its thermal sensitivity of 40 mK, specifically suited for detection of slightest temperature differences, making it indispensable in quality control of products and in medical prevention. The measurement temperature ranges from -20 °C to 100 °C, 0 °C to 250 °C and 120 °C to 900 °C, and spectral range of 7.5 to 14 µm, with an accuracy of ±2°C or ±2% (whichever is greater). Optris 450 sensor has been developed by Optris infrared measurements, that can be integrated into TIAGo.



Figure 158: Optris PI450 add-on

### 44.2 Installation

The add-on device is meant to be attached on to the head of TIAGo and connected to the expansion panel of the robot as shown in figure 159.



Figure 159: Optris thermal camera add-on attached to TIAGO

In order to attach the add-on the following components are provided in the kit:

- Optris Thermal camera, see figure 160.

<sup>11</sup>Included in the corresponding Premium hardware package.

- Mounting plate, see figure 160
- USB cable to connect the device to the expansion panel, see figure 160.
- Specific fasteners DIN 912 M3x10 to attach the device to the head of the robot, see figure 160.



Figure 160: Velodyne VLP-16 kit contents

The installation steps are as follows:

- First, attach the camera to the mounting panel with the provided fasteners DIN 7991 M4x8, see figure 161.
- Use the fasteners DIN 912 M3x10 provided to attach the mounting plate along with the camera on top of the head of TIAGo, see figure 162.
- Connect the cable to the optris camera and then connect the USB cable end to the USB port of the expansion panel, see figure 162.

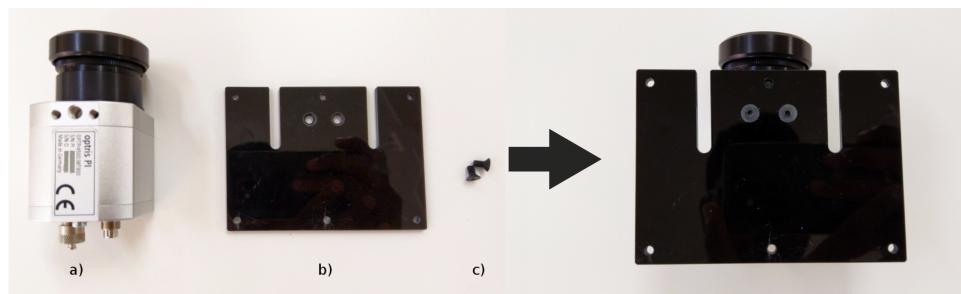


Figure 161: Steps to install the add-on

### 44.3 Starting optris drivers

Once the camera is connected to the robot, connect to the robot and start the thermal\_camera startup application manually as shown below to start receiving data from the sensor.



Figure 162: Steps to install the add-on

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

Once the `thermal_camera` application is started, thermographic information should be published in ROS. This could be verified by the following commands:

```
ssh pal@tiago-0c
rostopic list | grep optris
```

By default, the node will publish the thermal information in form of an image, but this cannot be visualized as the `color_conversion` node, which converts into displayable color representation is not used. In order to visualize the data, run the following commands inside the robot as shown below:

```
ssh pal@tiago-0c
pal-stop thermal_camera
roslaunch tiago_0_thermal_camera optris_camera.launch color_conversion:=true
```

Now, the data can be visualized from the topic : `/optris/thermal_image_view` as shown in the figure 163

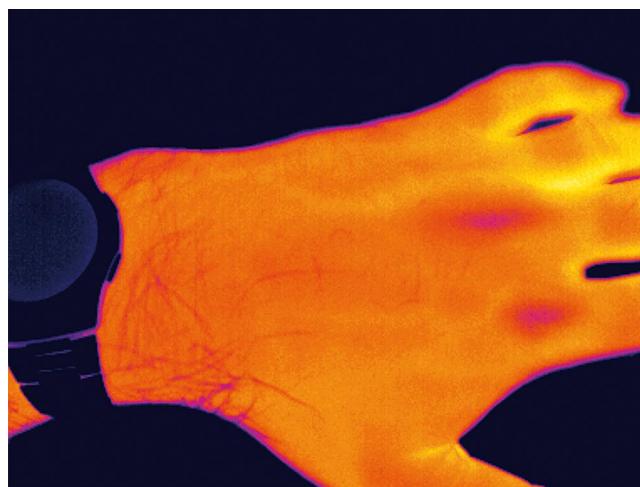


Figure 163: Steps to install the add-on



**TIA Go**

**PAL gripper camera add-on**





## 45 PAL gripper camera add-on

This chapter presents the add-on providing a small RGB camera on the PAL gripper end-effector shown in figure 164. For instance, the camera on the gripper is very useful in the latter stages of a grasping task for commanding the end-effector with more precision.



Figure 164: PAL gripper camera add-on mounted on TIAGo

### 45.1 Overview

The PAL gripper add-on is composed of an endoscopic RGB camera with a 2 m USB cable mounted on a 3D printed support to attach it to the gripper, see figure 165.



Figure 165: PAL gripper camera add-on

The cable has attached 4 fixation points as shown in figure 166.

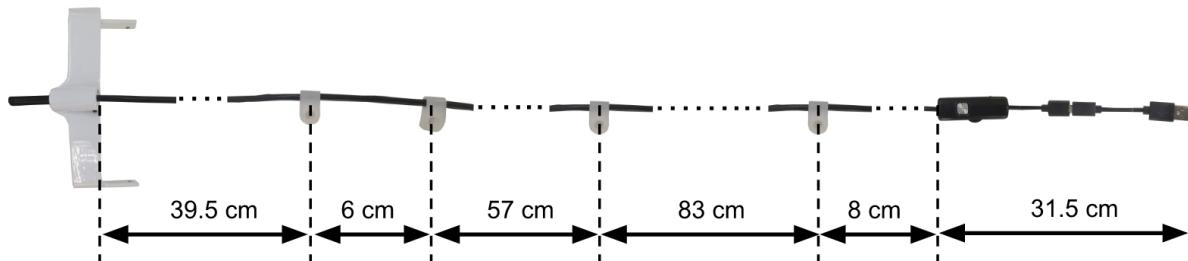


Figure 166: Specification of the gripper camera cable

## 45.2 Add-on installation

In order to mount the camera on the gripper a set of fasteners are provided:

- 4x M2,5x8 DIN 7991, see figure 167a, for the fixation points on the gripper.
- 2x M3x10 DIN 7991, see figure 167b, for the fixation points on the wrist.
- 1x M3x10 DIN 7991, see figure 167c, for the fixation point on the upper limb of the arm.
- 1x M3x16 DIN 7991, see figure 167c, for the fixation point on the lateral side of the upper part of the torso.

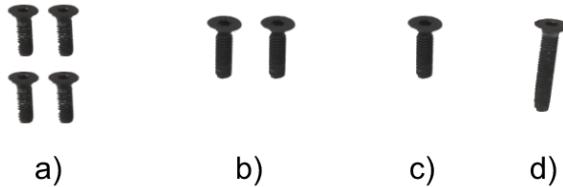


Figure 167: Fasteners to fix the camera gripper

Follow the procedure below to properly attach the camera and its cable to TIAGO:

- Run the `Offer Gripper` motion in order to have access to the 4 fasteners of the gripper cover in order to attach the camera mounting part as shown in figure 168, using the fasteners shown in figure 167a. The motion can be run from command line as follows:

```
ssh pal@tiago-0c
rosrun play_motion run_motion offer_gripper
```



Figure 168: Mounting points on the gripper.

- Fix the 2 mounting points on the wrist cover as shown in figure 169, using the fasteners shown in figure 167b. In order to make sure that the cable between the camera and these fixation points is loose enough to prevent breakage by running the following joint motions:

```
rosrun play_motion move_joint arm_6_joint -1.39 3.0
rosrun play_motion move_joint arm_6_joint 1.39 3.0
rosrun play_motion move_joint arm_7_joint -2.07 3.0
rosrun play_motion move_joint arm_7_joint 2.07 3.0
rosrun play_motion move_joint arm_6_joint -1.39 3.0
rosrun play_motion move_joint arm_7_joint -2.07 3.0
```



Figure 169: Mounting points on the wrist.

- Run the following motion to have access to the fixation point on the upper limb of the arm as shown in figure 170, and fix the next mounting point using the fastener shown in figure 167c :

```
rosrun play_motion move_joint arm_3_joint -2.0 3.0
```



Figure 170: Mounting point on the arm.

- Make sure that the cable is loose enough between the fixation point of the upper limb and the wrist by running the following motions:

```
rosrun play_motion move_joint arm_3_joint -3.45 3.0  
rosrun play_motion move_joint arm_6_joint 0.0 3.0  
rosrun play_motion move_joint arm_4_joint 2.3 3.0  
rosrun play_motion move_joint arm_5_joint 2.07 3.0  
rosrun play_motion move_joint arm_5_joint -2.07 3.0  
rosrun play_motion move_joint arm_4_joint -0.32 3.0  
rosrun play_motion move_joint arm_5_joint 2.07 3.0
```

- Run the following motion to better install the fixation point lateral side of the upper part of the torso as shown in figure 171, using the fastener shown in figure 167c:

```
rosrun play_motion move_joint arm_3_joint 0.0 3.0
```

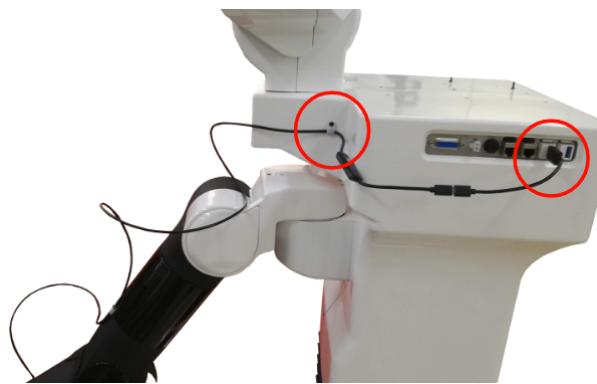


Figure 171: Mounting point on the torso and USB connection.

- Make sure that the cable is loose enough between the fixation point on the upper limb of the arm and the torso by running the following motions:

```
rosrun play_motion move_joint arm_3_joint -3.11 3.0
rosrun play_motion move_joint arm_4_joint 0.5 3.0
rosrun play_motion move_joint arm_1_joint 2.68 3.0
rosrun play_motion move_joint arm_2_joint 1.02 3.0
rosrun play_motion move_joint arm_2_joint -1.5 10.0
rosrun play_motion move_joint arm_1_joint 0.07 6.0
rosrun play_motion move_joint arm_4_joint 1.0 3.0
rosrun play_motion move_joint arm_2_joint 0.0 3.0
rosrun play_motion move_joint arm_4_joint 0.0 3.0
rosrun play_motion move_joint arm_3_joint 1.55 3.0
rosrun play_motion move_joint arm_3_joint -3.48 10.0
```

- Finally plug the USB connector to one of the ports on the expansion panel as shown in figure 171.

### 45.3 Running the camera driver

In order to start the driver of the camera run the following commands on a console:

```
ssh pal@tiago-0c
roslaunch tiago Bringup end_effector_camera.launch
```

The different camera topics will be published in the `/end_effector/camera` namespace.

### 45.4 Visualizing the camera image

In order to check that the camera is working properly the image can be visualized from a development computer as follows (make sure to set your development computer's IP when exporting `ROS_IP` as explained in Section 10.4):

```
export ROS_MASTER_URI=http://tiago-0c:11311
export ROS_IP=10.68.0.128
rosrun image_view image_view image:=/end_effector/camera/image_raw \
_image_transport:=compressed
```

An example of image provided by the gripper camera compared to the image of camera on the head is shown in figure 172.



Figure 172: Example of images provided by the camera on the robot's head (top-right) and the camera on the gripper (bottom-right)

**TIA Go**

**Demos**





## 46 Demos

### 46.1 Learning-by-demonstration

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

The details of the demo and the instructions on how to download the required packages, build them and how to run it are explained in the README.md file of the github's repository at [https://github.com/pal-robotics/learning\\_gui](https://github.com/pal-robotics/learning_gui).



**TIA Go**

## **Troubleshooting**





## 47 Troubleshooting

### 47.1 Overview

This chapter presents typical issues that may appear when using TIAGo and possible solutions. Please check the tables below carefully before reporting an issue to the support platform in order to find solutions faster.

### 47.2 Startup issues

#	Issue description	Possible solutions
1.1	The robot does not power up when pressing the On/Off button of the rear panel	Make sure that the electric switch is pressed, i.e. its red light is ON
1.2	After several days of not using the robot does not power up	Make sure that the battery is charged. Connect the charger and try to turn the robot on again after a couple of minutes
1.3	After several deays of not using the robot, the batteries, which were charged, are now empty	If the the electric switch was left pressed it is possible that the batteries have been completely discharged

Table 24: Startup issues

### 47.3 Navigation issues

#	Issue description	Possible solutions
2.1	The robot does not navigate	Make sure that the joystick has not taken the priority of the base. Press the <code>START</code> button of the joystick to release it. Check Section 20.2 for more details.
2.2	The robot does not navigate even if the joystick has not the priority	Check in <i>WebCommander -&gt; Diagnostics -&gt; Functionality</i> if <i>Navigation</i> is paused. If so, check issue 2.3. Otherwise, check issue 2.4
2.3	The robot does not navigate because Navigation is paused	Check if the charger is connected on the rear part of the robot. If so, disconnect the charger and Navigation should be unpause. Otherwise, it is possible that the robot was docked and someone has removed the robot from the docked station or removed the power supply to the dock station. In these latter cases you may run from a development computer: <b>rosrun actionlib axclient.py /undocker_server</b> to force the robot to undock properly. Note that this will cause the robot to move about half a meter backwards and then turn 180° about itself.
2.4	Even if Navigation is not paused, the robot does not navigate when sending navigation goals	Make sure that <b>move_base</b> is running, i.e. you may check the status of <i>WebCommander -&gt; Startup -&gt; move_base</i> . If the goals are sent from a development computer, either from Rviz or from command line or a node, make sure also that the <code>ROS_MASTER_URI</code> and <code>ROS_IP</code> environmental variables are properly set as explained in Section 10.4

Table 25: Navigation issues

## 47.4 Upperbody movements issues

#	Issue description	Possible solutions
3.1	Pre-defined movements do not work	Make sure in <code>WebCommander -&gt; Startup &gt; play_motion</code> that this node is running. Otherwise press the 'Start' button to re-launch it. If this is not the case, try to reboot the robot and check if the problem persists.
3.2	The torso does not move when trying to control it with the joystick	Make sure that the arm is not in collision with some other part of the robot. If so, try first running the pre-defined movement <code>Offer Gripper</code> or <code>Offer Hand</code>
3.3	I have defined a new movement for <code>play_motion</code> and it interrupts abruptly during its execution	Make sure that none of the joints of your motion is being controlled by other nodes. For example, if your movement includes the head joints, make sure to stop the <code>head_manager</code> node. Check Section 14.2 for details on how to stop this node.
3.4	I have defined a new movement for <code>play_motion</code> and when running it the arm collides with the robot	Make sure to set to false the <code>skip_planning</code> flag of the <code>play_motion</code> goal when running the motion. Otherwise the first waypoint of the movement will be executed without motion planning and self-collisions may occur.

Table 26: Upperbody movement issues

## 47.5 Text-to-speech issues

#	Issue description	Possible solutions
4.1	The robot does not speak when sending a voice command	Make sure that the volume is set appropriately. Check that <i>WebCommander -&gt; Settings -&gt; Playback Volume</i> has not been lowered

Table 27: Text-to-speech issues

## 47.6 Network issues

#	Issue description	Possible solutions
8	I have configured TIAGo's network to connect to my LAN but it does not work and I have no longer access to the robot via WiFi	Connect a computer to one of the Ethernet ports of the expansion panel and open a web browser to connect to <code>http://10.68.0.1:8080</code> to have access to the WebCommander. Check on the Network tab if the configuration is OK.
9	I have applied on TIAGo a new network configuration but after rebooting the robot the changes made have been reverted	To save permanently the new network configuration make sure to press 'Save' and 'Confirm' after pressing 'Apply change'. Otherwise the network configuration changes are not save permanently.

Table 28: Network issues

**TIA Go**

**Customer service**

**PAL** ROBOTICS

The logo for PAL Robotics features the word "PAL" in a large, orange, sans-serif font. To the right of "PAL" is the word "ROBOTICS" in a smaller, black, sans-serif font. To the right of "ROBOTICS" is a circular emblem. The emblem has a dark brown or black background with a thin orange border. Inside the circle, there is a stylized, glowing orange and white graphic that resembles a eye or a sensor array.



## 48 Customer service

### 48.1 Support portal

All communication between customers and PAL Robotics is made using tickets in a helpdesk software.

This web system can be found at <http://support.pal-robotics.com>. Figure 173 shows the initial page of the site.

New accounts will be created on request by PAL Robotics.

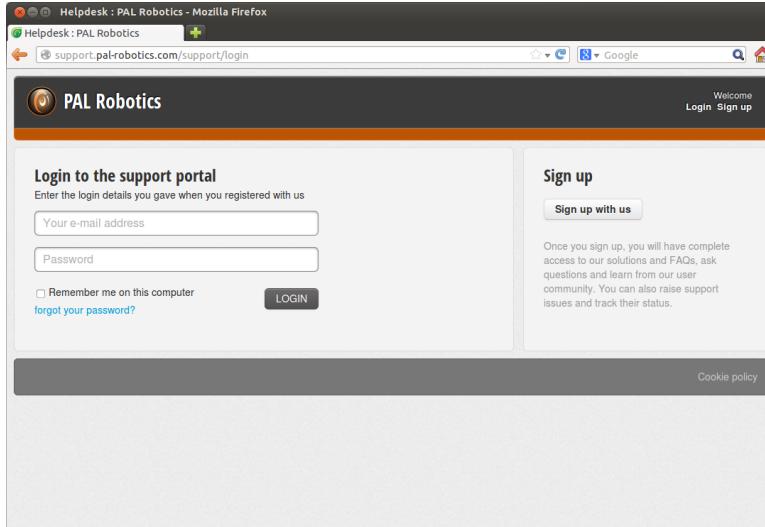


Figure 173: PAL Robotics support website

Once the customer has entered the system (Figure 174), two tabs can be seen: *Solutions* and *Tickets*.

The *Solution* section contains *FAQs* and *News* from PAL Robotics .

The *Tickets* section contains the history of all tickets the customer has created.

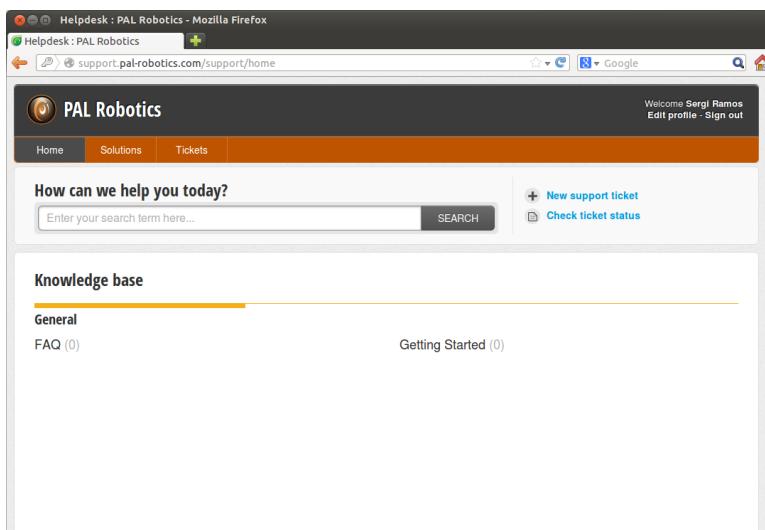


Figure 174: Helpdesk

Figure 175 shows the ticket creation webpage.

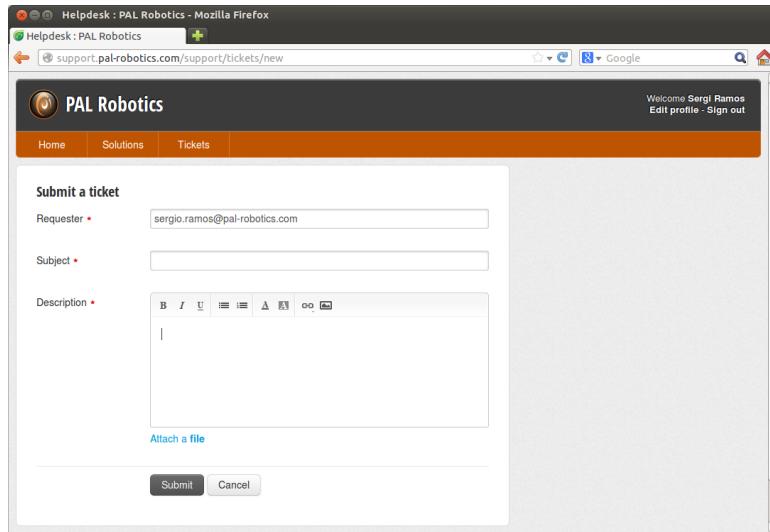
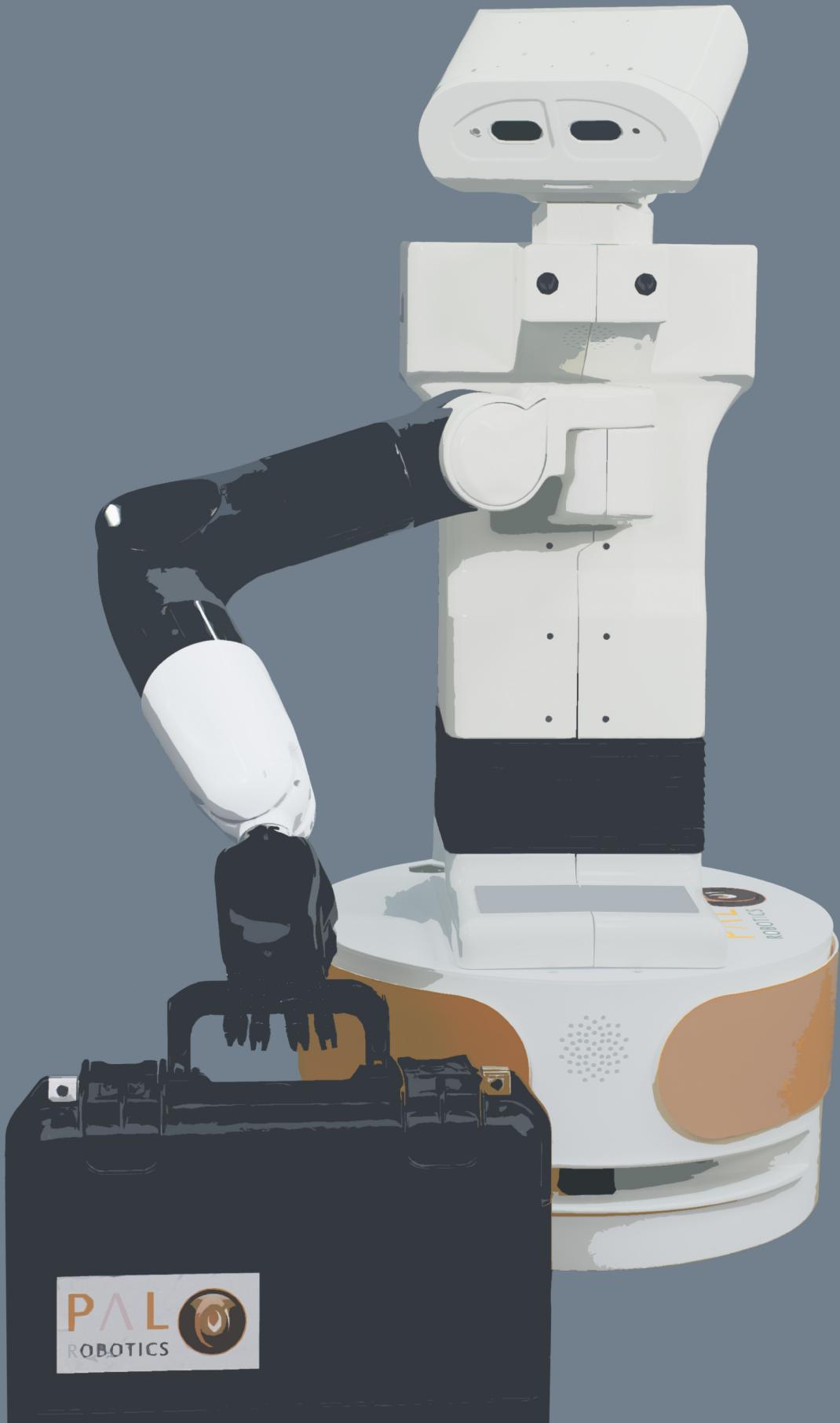


Figure 175: Ticket creation

## 48.2 Remote support

A technician from PAL Robotics can give remote support. This remote support is disabled by default, so the customer has to activate it manually (Please refer to 11.3.7 for further details).

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



PAL ROBOTICS S.L.

Pujades 77-79, 4º 4<sup>a</sup> Tel.: +34 934 145 347 info@pal-robotics.com  
08005 Barcelona, Spain Fax: +34 932 091 109 www.pal-robotics.com

