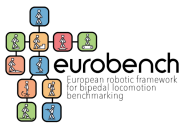


Deliverable Title	D3.2 Completion of manufacturing of REEM-C. Report describing the humanoid robot platforms that will be available to the users of the open call
Deliverable Lead:	PAL Robotics
Related Work Package:	WP3: HUMANOID ROBOTS BENCHMARKING
Related Task(s):	T3.2 Manufacturing of Humanoid Robot
Author(s):	Luca Marchionni, Sergi García García, Adriá Roig Moreno
Dissemination Level:	Confidential
Due Submission Date:	31/12/2018
Actual Submission:	04/03/2019
Project Number	779963
Instrument:	Research and Innovation Action
Start Date of Project:	01.01.2018
Duration:	48 months
Abstract	This deliverable concludes Task 3.5 on the manufacturing of the REEM-C biped humanoid robot which will be used for testing the benchmarks in T3.4.



This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 779963



D3.2 Completion of manufacturing of REEM-C. Report describing the humanoid robot platforms that will be available to the users of the open call



*This project has received funding from the European Union's Horizon
2020 research and innovation program under grant agreement No 779963*



Versioning and Contribution History

Version	Date	Contributors	Modification reason
v.01	25/02/19	Luca Marchionni, Sergi García García, Adriá Roig Moreno	Final Version
v.02	04/03/19	Diego Torricelli	Revision and approval of the Coordinator

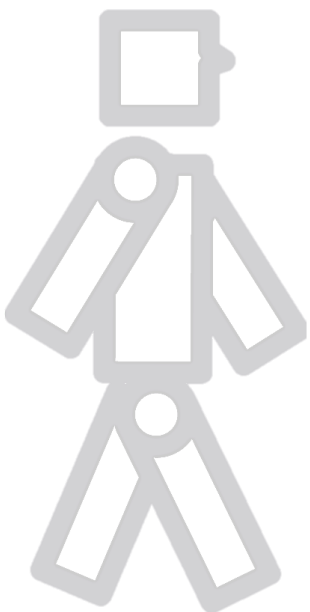


Table of Contents

- 1 Executive Summary.....5
- 2 Description of work & main achievements6
- 3 Deviations from the workplan.....7
- 4 Conclusion.....7
- 5 Annex: REEM C - Handbook.....8

1 Executive Summary

This deliverable concludes Task 3.5 on the manufacturing of the REEM-C biped humanoid robot which will be used for testing the benchmarks in T3.4. The deliverable is the Product Handbook and includes detailed information regarding the robot, specifications, sensors, handling instruction, set-up and mounting, general information for usage, software, webcommander and many more useful data for the proper use of the robot.



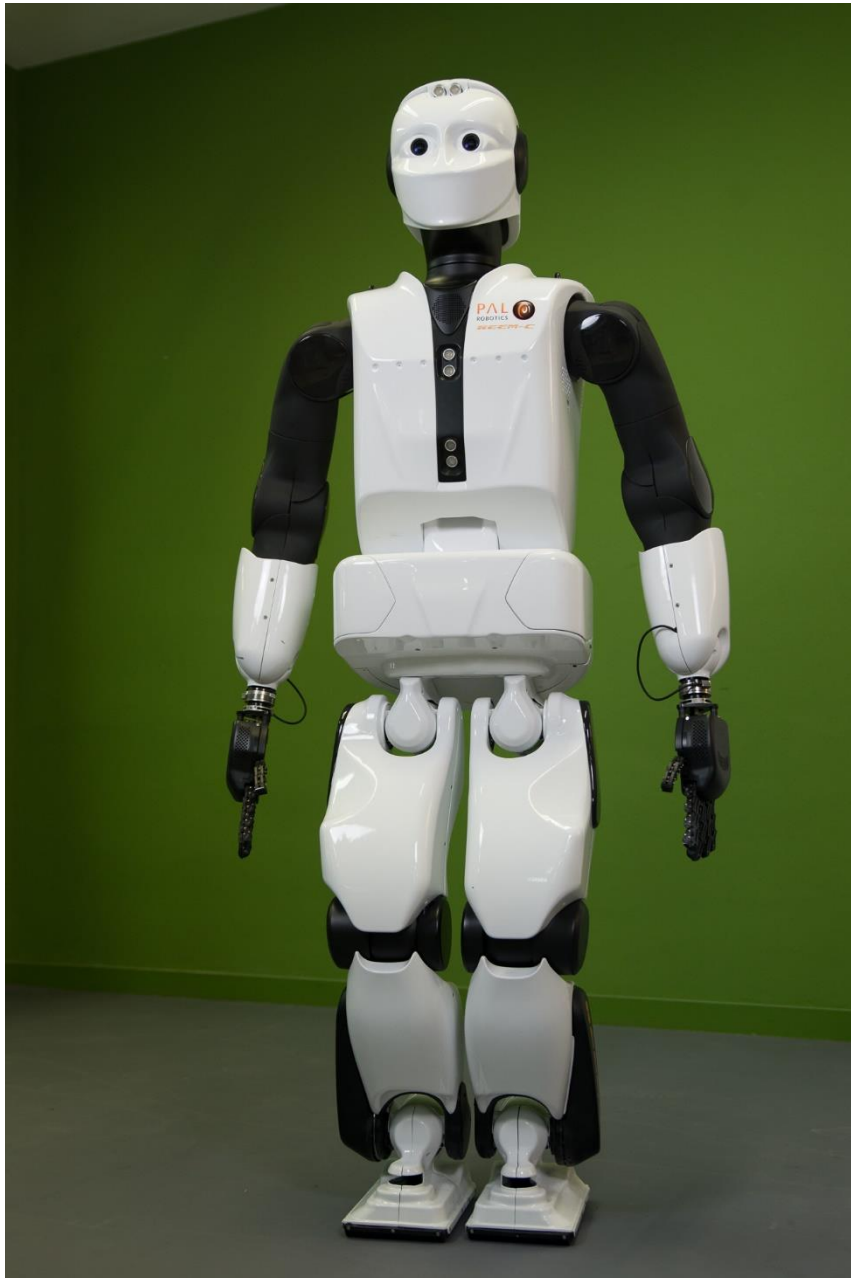
The Handbook is presented as an Annex to the Deliverable.



*This project has received funding from the European Union's Horizon
2020 research and innovation program under grant agreement No 779963*

2 Description of work & main achievements

The robot prototype has been manufactured on time and it is available at PAL Robotics premises for performing test trials of the proposed benchmarking scenarios; during the second and the third year of the project, the test will be performed at PAL Robotics lab while later tests will be performed in the facilities at ITT. In the last year of the project, the robot will be moved to IIT lab.



Compared with the previous versions of the robot and with the aim of improving its performances, the following modifications have been introduced in the new prototype:



D3.2 Completion of manufacturing of REEM-C. Report describing the humanoid robot platforms that will be available to the users of the open call

1. The RGB stereo camera has been substituted with and RGBD camera, in order to improve the perception capabilities of the robot. Do to this modifications, covers and wiring was redesigned.
2. More powerful computers have been introduced; the consequence was the redesign of the whole logic box in order to improve its thermal behaviour.
3. New motors for the legs, with the aim of increasing the walking speed.

All the technical information of all the robot components are included in the Hadnbook.

3 Deviations from the workplan

The deliverable has been submitted on M14 instead of M12, since PAL used this occasion to update the handbook in accordance with the modifications introduced in the new prototype.

4 Conclusion

The Handbook provides all the information needed for the proper use of the humanoid REEM C robot, both for the Consortium partners and for the participant of the FSTP actions. PAL Robotics will support the testing phases during all the duration of the project as well as the implementation of the Third Parties control algorithms on the REEM C platform.





D3.2 Completion of manufacturing of REEM-C. Report describing the humanoid robot platforms that will be available to the users of the open call

5 Annex: REEM C - Handbook



*This project has received funding from the European Union's Horizon
2020 research and innovation program under grant agreement No 779963*





REEM-C

Handbook

PAL
ROBOTICS



REEM C

Handbook



**Barcelona
2019**

Contents

1 Package contents	6
1.1 Overview	6
2 Specifications	9
2.1 Overall dimensions	9
2.2 Weight	9
2.3 Degrees of freedom	9
2.4 Payload	10
2.5 Kinematics	10
2.6 Dynamics	12
3 Sensors	18
3.1 Cameras	18
3.2 6 axes force sensors	19
3.3 Inertial Measurement Unit	20
3.4 Motors	20
3.5 Connectivity	21
3.6 Electrical parts and components	22
3.7 Service Panel	22
3.8 Extra Service Panels connectors	23
4 Storage	27
4.1 Overview	27
4.2 Transport	27
4.3 Storage cautions	27
5 Indications for handling	28
6 Set-up and mounting conditions	31
6.1 Overview	31
6.2 Robot crate	31
6.3 How to get the robot out of the crate	31
6.4 How to put the robot into its crate	33
7 Power supply	34
7.1 Battery assembly	34

8 Introduction to safety	39
8.1 Overview	39
8.2 Intended applications	39
8.3 Working environment and usage guidelines	39
8.4 Risks	40
8.5 Safety distance	40
8.6 Battery manipulation	41
9 Safety measures in practice	42
9.1 Control system functions	42
9.2 Emergency stop	42
9.3 Firefighting equipment	42
9.4 Leakage	43
10 General information for usage	47
10.1 User access	47
10.2 Power supply	47
10.3 Controls and protective devices	49
10.4 Fault identification and location	50
11 Network kick-off	50
11.1 Overview	50
11.2 Reserved IP address	50
11.3 Network Diagram	51
11.4 Basestation	51
11.5 Robot	53
11.6 Development computer	53
12 Getting started	54
12.1 Switch on the robot	55
12.2 Shutting down the robot	55
13 Nature and frequency of inspections	59
13.1 Cleaning	59
13.2 Daily inspection	59
13.3 Once every three months	59
13.4 Once every year (by technical service)	59
13.5 Once every three years (by technical service)	59
14 De-commission, dismantling and disposal	60

15 Robot identification	63
16 Software recovery	67
16.1 Overview	67
16.2 Basestation installation	67
16.3 Robot computers installation	68
16.4 Development computer installation	69
17 Generation of USB installation	71
17.1 Overview	71
17.2 Requirements	71
17.3 PAL-Robotics ISO	71
18 Basestation	75
18.1 Overview	75
18.2 Users	75
18.3 Network configuration	75
18.4 NTP server	75
18.5 OpenVPN	76
18.5.1 Add an element to the VPN	77
18.5.2 Remove an element from the VPN	77
18.5.3 Copying files between VPN users	78
18.6 DNS Server	78
18.7 PPTP Connections	79
18.7.1 Address pool	79
18.7.2 Security	79
18.7.3 DNS settings	80
18.8 L2TP/IPsec PSK	80
18.8.1 Address pool	80
18.8.2 Security	80
18.8.3 DNS settings	80
18.9 Software repositories	81
18.9.1 PAL Robotics 's repositories	81
18.9.2 Customer software repository	81
18.10 System upgrade	82
18.11 PAL Robotics 's Corporate Basestation	83
18.12 DHCP server	83

19 Development computer	87
19.1 Overview	87
19.2 Computer requirements	87
19.3 Setting ROS environment	87
19.4 ROS communication with the robot	87
19.5 System Upgrade	88
20 REEM-C Robot's Internal Computers	91
20.1 REEM-C LAN	91
20.2 Users	91
20.3 File system	91
20.4 Internal DNS	92
20.5 System upgrade	92
21 WebCommander	95
21.1 Accessing the WebCommander website	95
21.2 Overview	95
21.3 Default tabs	95
21.3.1 Startup tab	95
21.3.2 Diagnostics Tab	96
21.3.3 Logs Tab	97
21.3.4 General Info Tab	98
21.3.5 Installed Software Tab	98
21.3.6 Settings Tab	98
21.3.7 Networking Tab	99
21.3.8 Video Tab	102
21.3.9 Movements Tab	103
21.4 Tab configuration	104
21.4.1 Parameter format	104
21.4.2 Startup Plugin Configuration	104
21.4.3 Diagnostics Plugin Configuration	105
21.4.4 Logs Plugin Configuration	105
21.4.5 General Info Plugin Configuration	105
21.4.6 Installed Software Plugin Configuration	105
21.4.7 Settings Plugin Configuration	105
21.4.8 Networking Plugin Configuration	105
21.4.9 Video Plugin Configuration	105
21.4.10 Movements Plugin Configuration	106
21.4.11 Commands Plugin Configuration	106

22 Camera publishing	108
22.1 RGBD camera	108
23 Camera subscription	108
23.1 Command line	108
23.1.1 Image visualization	108
23.2 C++ API	108
23.2.1 ROS subscribers	108
23.2.2 pal_camera_client package	109
24 Sensors	110
24.1 Description of sensors	110
24.2 ROS API	110
24.2.1 Published topics	111
25 Walking	112
25.1 Walking_controller	112
25.1.1 Action API	112
25.1.2 Topic API	113
25.1.3 Published topics	113
25.1.4 Service API	113
25.2 Walking parameters	113
26 Whole Body Control	116
26.1 Overview	116
26.2 Task	116
26.2.1 Center of mass	117
26.2.2 GoTo	117
26.2.3 Fixed Constraint	117
26.2.4 Gaze	117
26.2.5 Self collision	117
26.2.6 Joint Limit Task	117
26.3 Stack	117
26.3.1 Parameters in Yaml file	118
26.3.2 Launch file	118

27 Joint Trajectory Controller	119
27.1 Overview	119
27.1.1 Trajectory representation	119
27.1.2 Hardware interface type	119
27.1.3 Other features	119
27.2 Sending trajectories	119
27.2.1 Available interfaces	119
27.2.2 Preemption policy	120
27.2.3 Trajectory replacement	120
27.2.4 Trajectory replacement example: basics	120
27.2.5 Trajectory replacement example: effects of trajectory start time	121
27.3 ROS API	121
27.3.1 Action interface	121
27.3.2 Subscribed topics	121
27.3.3 Published topics	121
27.3.4 Services	122
27.3.5 Parameters	122
28 ros_control	123
28.1 Introduction to ros_control	123
28.1.1 How to load and unload controllers	124
28.1.2 How to start and stop controllers	124
28.2 How to create a custom controller	124
28.2.1 Hello controller	125
28.2.2 Moving a position controlled joint	126
28.2.3 Inertial measurement unit (IMU)	127
28.2.4 Force torque sensors	128
28.2.5 Combining different resources (hardware interfaces) in a single controller	129
28.3 Controlling <i>REEM-C</i> in effort	132
29 Deploying software on the robot	133
29.1 Introduction	133
29.2 Usage	133
29.3 Notes	133
29.4 Deploy tips	134
29.5 Use-case example	134
29.5.1 Adding a new ROS Package	134
29.5.2 Adding a new controller	136
29.5.3 Modifying an installed package	137

30 Modifying Robot Startup	138
30.1 Introduction	138
30.1.1 Application start configuration files	138
30.1.2 Computer start lists	139
30.1.3 Additional startup groups	139
30.2 Startup ROS API	140
30.3 Startup command line tools	140
30.4 Modifying the robot's startup	140
30.4.1 Adding a new application for automatic startup	140
30.4.2 Modifying how an application is launched	141
31 Motion planning with <i>Movel!</i>	142
31.1 Getting started with the <i>Movel!</i> graphical user interface	142
31.2 End test	144
32 SLAM map building	145
32.1 Overview	145
32.2 Start SLAM in simulation	145
32.3 Save the map	146
32.4 Stop mapping	146
32.5 Start SLAM on the robot	147
32.5.1 Filter the crane during mapping or navigation	147
33 Localization in a known map	147
33.1 Overview	147
33.2 Start localization in simulation	147
33.3 Start localization on the robot	149
34 Control source switch	149
34.1 Overview	149
34.2 Change between Manual and Autonomous control sources	149
34.3 Change the control source priorities	150
35 Path Planning in a known map	151
35.1 Overview	151
35.2 Start path planning in simulation	152
35.2.1 Send a goal to the robot	152
35.3 Start path planning on the robot	152
35.4 Virtual Obstacle Avoidance	153
35.5 Changing the map	153

36 Navigation State Machine	153
36.1 Overview	153
36.2 Start the State Machine in simulation	153
36.3 Change from localization to mapping	154
36.4 Change from mapping to localization	154
36.5 Start the State Machine in the robot	155
37 Map Editor	155
37.1 Overview	155
37.2 Glossary	156
37.3 Start	156
37.4 Download OGM from the robot	157
37.5 Load and Align Building Plan	158
37.6 Add and remove virtual elements to the map	159
37.6.1 Points of Interest	159
37.6.2 Zones of Interest	159
37.6.3 Virtual Obstacles	159
37.6.4 Remove elements	159
37.7 Upload Map to the robot	160
37.8 Save and Load the map to the disk	160
38 Overview	165
38.1 REEM-C's ROS environment	165
38.2 Source code	165
39 Manually building the tutorials (optional)	166
40 Setup	167
40.1 In a simulated environment	167
40.1.1 Start a simulated REEM-C	167
40.1.2 Connect a terminal to the robot	167
40.2 On the real robot	167
40.2.1 Start a real REEM-C	167
40.2.2 Connect a terminal to the robot	168
40.2.3 A note on switching from simulated to real robot deployments (advanced)	168
41 Device State Notifications	168
41.1 Accessing the device state notification website	168
41.2 Diagnostics Tab	168
41.3 Logs Tab	169

42 Joints initialization	169
42.1 Usage	170
42.2 Example	170
42.3 End test	170
43 Rviz Basics	171
43.1 Setup	171
44 Rqt Basics	172
44.1 Setup	173
44.1.1 Default Rviz configuration	173
44.1.2 Launch the GUI	173
44.1.3 GUI details	173
44.2 End test	174
45 Modifying the simulation world	174
45.1 Setup	174
45.2 Modifying the simulation world	174
45.3 Sensor readings	175
46 Inspect the kinematic workspace of REEM-C	176
46.1 Setup	176
46.2 Moving joints with the sliders interface	177
46.3 End test	177
47 Read sensors	177
47.1 Setup	177
47.2 Test the sensors	177
47.2.1 Test	177
47.2.2 End of test	179
48 Moving individual joints – command line	180
48.1 Usage	180
48.2 Example	180
48.2.1 Setup	180
48.2.2 Run the example	180
48.3 A note on safe execution	181
48.4 Testing all joints	181
48.4.1 Upper body	181
48.4.2 Lower body	182
48.5 End test	183

49 Moving individual joints – graphical user interface	183
49.1 Setup	183
49.1.1 Launch the GUI	183
49.2 End test	184
50 Setting current limits – graphical user interface	184
50.1 Setup	184
50.1.1 Launch the GUI	184
50.2 End test	185
51 Perform simple upper-body motions.	186
51.1 Setup	186
51.2 Querying and executing motions from the command line	186
51.3 Triggering motions with the joystick	187
51.4 Example motion sequence	188
51.5 A note on unexpected hand contacts	188
51.6 Note on ABORTED hand goals	190
51.7 End test	190
52 Grasping objects	190
52.1 Usage	190
52.2 Graspable objects	191
52.3 Example	191
52.4 End test	193
53 Control REEM-C with keyboard or joystick input	193
53.1 Setup	193
53.2 End test	195
54 Leg inverse kinematics library	195
54.1 Example code	195
54.2 Build and execute the example	195
55 Walking safety library	196
55.1 Example code	196
55.2 Build and execute the example	198

56 Walking examples	198
56.1 Setup	199
56.1.1 Notes when running on the real robot	199
56.1.2 Starting the walking controller	199
56.2 Walking using topic interface	199
56.2.1 Setup	200
56.3 Walking using a service interface	200
56.3.1 Setup	200
56.3.2 Setup	200
56.4 Walking using an action interface	200
56.4.1 Setup	200
56.5 End test	200
56.6 Whole Body Control	201
56.6.1 REEM-C with interactive markers	201
56.6.2 Dynamic simulation in gazebo	201
56.6.3 REEM-C dancing	201
56.6.4 Dancing in dynamic simulation	201
56.6.5 Dancing with real robot	202
57 Customer service	205
57.1 Support portal	205
57.2 Remote support	206

REEM C

Welcome



Welcome

Thank you for choosing PAL Robotics. This Handbook contains information related to the REEM-C 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

REEM-C 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 REEM-C 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 REEM-C 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 REEM-C 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 REEM-C 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 REEM-C . Make sure you can find all of them: transportation box and battery (Figure 1), connectivity (Figure 2), and peripherals (Figure 3).



(a) Transport case



(b) Battery and charger

Figure 1: Transportation box and battery



(a) Wireless access point



(b) Basestation

Figure 2: Connectivity



Figure 3: Peripherals

REEEM C

Specifications



2 Specifications

2.1 Overall dimensions

- Height: 1640 mm
- Width: ~720 mm
- Depth: ~360 mm



Figure 4: Full robot view

2.2 Weight

The total weight of REEM-C is about 80 kg, including batteries.

2.3 Degrees of freedom

REEM-C is provided with a total of 68 degrees of freedom, following the distribution, shown in Table 1.

	Qty. of DoF
Head	2
Left Arm	7
Right Arm	7
Left Hand	19
Right Hand	19
Waist	2
Left Leg	6
Right Leg	6

Table 1: Degrees of freedom

2.4 Payload

REEM-C is capable to grasp objects and sustain them. A payload is up to 1kg per hand¹, with arm completely stretched, and 10 kg sustained between torso and arms.

2.5 Kinematics

The kinematic structure of REEM-C is depicted in Figures 5 and 6. Each depicted frame is located at the origin of a joint. All joints are revolute, and their rotation axes coincide with the z axis of its associated frame. The base link of the robot is located close to its center of mass, in the waist.

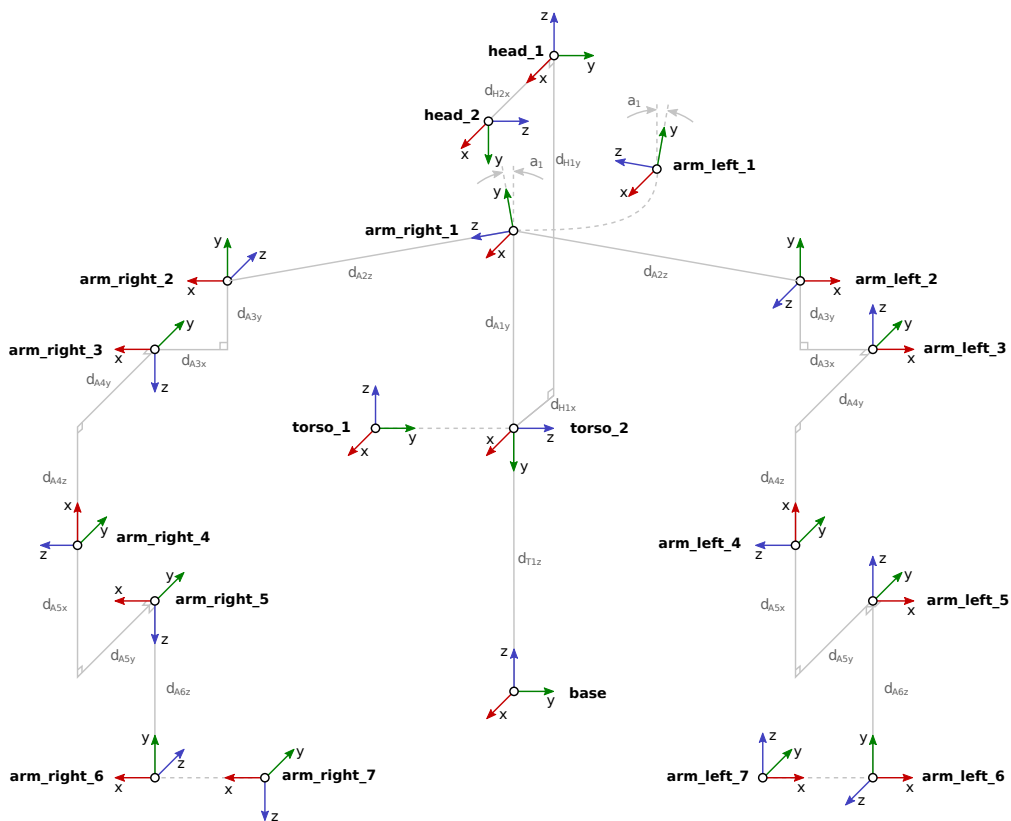


Figure 5: REEM-C upper body joints

¹Refer to the Hey5 hand user manual for a detailed specification of the hands.

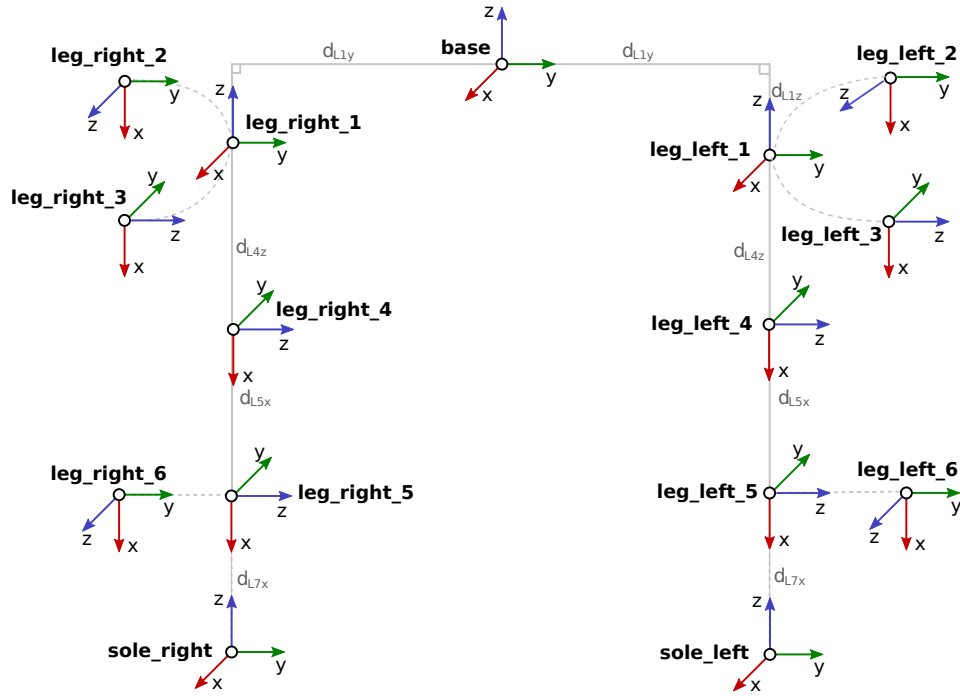


Figure 6: REEM-C lower body joints

Parameter	Value (mm)
d_{T1z}	143.53
d_{H1x}	0
d_{H1y}	423.16
d_{H2x}	44.5
d_{A1y}	318.27
a_1	15
d_{A2z}	217.49
d_{A3y}	142
d_{A3x}	20
d_{A4y}	20
d_{A4z}	88
d_{A5x}	88
d_{A5y}	20
d_{A6z}	150
d_{L1y}	75
d_{L1z}	143.53
d_{L4z}	300
d_{L5x}	300
d_{L7x}	115

Table 2: REEM-C links

Table 4 lists the range of motion of each joint and details how to compute the location of each joint frame. CAD models are used both for visualization and collision checking purposes. The distances are specified in Table 2.

A few notes concerning the notation used in Table 4. Rotations are represented using two conventions:

- **Axis-angle**, e.g., $R_x(\alpha)$ is a rotation of α degrees about the x axis.
- **Roll, pitch, yaw sequence** about the fixed axes, that is

$$R(\text{roll}, \text{pitch}, \text{yaw}) = R_z(\text{yaw})R_y(\text{pitch})R_x(\text{roll})$$

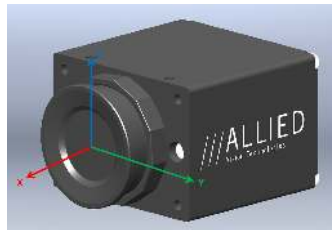
Rigid transformations of the form $H = \begin{bmatrix} R(\cdot) & T(\cdot) \\ 0 & 1 \end{bmatrix}$ are represented in the table as “ $R(\cdot) T(\cdot)$ ” for brevity.

Table 3 lists a collection of sensors installed in REEM-C, as well as their locations. When creating sensor frames, the convention depicted must be used.

Link	Parent Link	Rotation	Origin Translation
camera_link	head_2	$R(90, 0, 0)$	$T(18.9, -187.9, 0)$
sole_right (force sensor)	leg_right_6	$R(0, -90, 0)$	$T(117, 0, 0)$
sole_left (force sensor)	leg_left_6	$R(0, -90, 0)$	$T(117, 0, 0)$
imu	base	$R(0, 0, 0)$	$T(30.25, 4, -25.68)$

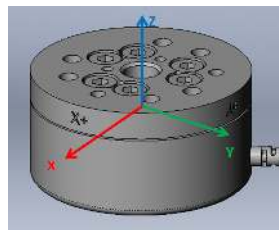
Table 3: Sensors location for REEM-C

The reference frames for the sensors are depicted in Figures 7 and 8.

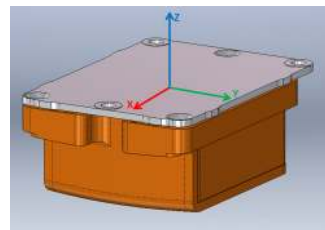


camera

Figure 7: Camera sensor



force sensor



IMU sensor

Figure 8: Force and IMU sensors

2.6 Dynamics

This subsection provides the dynamics of the robot. Table 4 shows the details for the output torque, speed and the range of motion for each DoF. There is a cam mechanism that limits the range of motion of the head when moving forward. Table 5 specifies what this limit is depending on the other head DoF angle.

The table 6 provides the specifications for the weight and the center of mass of REEM-C.

Name joint	Parent joint	Translation			Rotation			Joint range			Maximum angular velocity [RPM]	
		x	y	z	x(roll)	y(pitch)	z(jaw)	Min	Max	Nominal Torque[Nm]		Max Torque [Nm]
Legs												
leg_left_1_joint	base	0	75	-143.525	0	0	0	-45	30	42.7	42.7	22.5
leg_left_2_joint	leg_left_1_joint	0	0	0	0	90	0	-15	30	34	64	21.7
leg_left_3_joint	leg_left_2_joint	0	0	0	-90	0	0	-100	45	47	55.7	34.8
leg_left_4_joint	leg_left_3_joint	300	0	0	0	0	0	0	150	108	138.3	30.8
leg_left_5_joint	leg_left_4_joint	300	0	0	0	0	0	-75	45	49	80.9	23.3
leg_left_6_joint	leg_left_5_joint	0	0	0	90	0	0	-30	15	34	64	21.7
leg_right_1_joint	base	0	-75	-143.525	0	0	0	-30	45	42.7	42.7	22.5
leg_right_2_joint	leg_right_1_joint	0	0	0	0	90	0	-30	15	34	64	21.7
leg_right_3_joint	leg_right_2_joint	0	0	0	-90	0	0	-100	45	47	55.7	34.8
leg_right_4_joint	leg_right_3_joint	300	0	0	0	0	0	0	150	108	138.3	30.8
leg_right_5_joint	leg_right_4_joint	300	0	0	0	0	0	-75	45	49	80.9	23.3
leg_right_6_joint	leg_right_5_joint	0	0	0	90	0	0	-15	30	34	64	21.7
Torso and head												
torso_1_joint	base	0	0	143.525	0	0	0	-75	75	78	87	25.8
torso_2_joint	torso_1_joint	0	0	0	-90	0	0	-15	45	78	89	25.8
head_1_joint	torso_2_joint	0	-423.16	0	90	0	0	-75	75	3.6	4	26.7
head_2_joint	head_1_joint	44.5	0	0	-90	0	0	-15	45 ²	3.6	4	26.7
Arms												
arm_left_1_joint	torso_2_joint	0	-318.27	0	165	0	0	-45	180	39	44.6	25.8
arm_left_2_joint	arm_left_1_joint	0	0	-217.49	90	75	90	-15	120	22.3	22.3	35
arm_left_3_joint	arm_left_2_joint	20	-142	0	-90	0	0	-135	157.5	17.9	17.9	43.8
arm_left_4_joint	arm_left_3_joint	0	-20	-88	0	-90	0	0	130	17.9	17.9	43.8
arm_left_5_joint	arm_left_4_joint	-88	20	0	0	90	0	-120	120	3	3	18.7
arm_left_6_joint	arm_left_5_joint	0	0	-150	90	0	0	-81	81	6.6	6.6	16.9
arm_left_7_joint	arm_left_6_joint	0	0	0	-90	0	0	-120	120	6.6	6.6	16.9
arm_left_tool_joint	arm_left_7_joint	0	0	-46	90	0	0	-	-	-	-	-
wrist_left_tool_joint	arm_left_tool_joint	7.85	0	0	90	0	90	-	-	-	-	-
wrist_left_ft_joint	wrist_left_tool_joint	0	0	12.73	-90	0	0	-	-	-	-	-
arm_right_1_joint	torso_2_joint	0	-318	0	-165	0	0	-45	180	39	44.6	25.8
arm_right_2_joint	arm_right_1_joint	0	0	217.49	-90	-75	90	-15	120	22.3	22.3	35
arm_right_3_joint	arm_right_2_joint	20	-142	0	90	0	0	-135	157.5	17.9	17.9	43.8
arm_right_4_joint	arm_right_3_joint	0	-20	88	0	90	0	0	130	17.9	17.9	43.8
arm_right_5_joint	arm_right_4_joint	-88	20	0	0	-90	0	-120	120	3	3	18.7
arm_right_6_joint	arm_right_5_joint	0	0	150	-90	0	0	-81	81	6.6	6.6	16.9
arm_right_7_joint	arm_right_6_joint	0	0	0	90	0	0	-120	120	6.6	6.6	16.9
arm_right_tool_joint	arm_right_7_joint	0	0	46	90	0	0	-	-	-	-	-
wrist_right_ft_joint	arm_right_tool_joint	7.85	0	0	90	0	90	-	-	-	-	-
wrist_right_tool_joint	wrist_right_ft_joint	0	0	12.73	-90	0	0	-	-	-	-	-

Table 4: Joint properties. Joint torque/speed represents the expected performance for a fully charged battery.

² Depends on head_1_joint angle (see Table 5)

head_1_joint angle	head_2_joint max. range of motion
0°	45°
10°	45°
20°	39°
30°	27°
40°	18°
50°	14°
60°	14°
70°	14°
75°	14°

Table 5: head_2_joint limit (moving forward) depending on head_1_joint angle

Link	Parent link	Mass [g]	Center of mass location w.r.t. link frame [mm]		
			x	y	z
Legs					
base_link	world	13468.39	-15.7	-4.1	20
leg_left_1_link	base_link	1087.51	-25.7	0.0	25.3
leg_left_2_link	leg_left_1_link	721.51	0.6	8.5	-11.2
leg_left_3_link	leg_left_2_link	4855.51	150.8	-17.9	18.7
leg_left_4_link	leg_left_3_link	3406.72	147.1	-4.1	19.7
leg_left_5_link	leg_left_4_link	705.33	-0.3	11.3	7.9
leg_left_6_link	leg_left_5_link	2289.9	63.0	3.4	-3.1
Torso and head					
torso_1_link	base_link	1598.1	-0.1	0.0	-6.5
torso_2_link	torso_1_link	13115.1	-18.5	-180.4	-0.7
head_1_link	torso_2_link	1119.7	18.2	2.5	-5.2
head_2_link	head_1_link	1631.3	11.8	-81.0	1.0
Arms					
arm_left_1_link	torso_2_link	1,536.1	7.0	-0.1	-186.3
arm_left_2_link	arm_left_1_link	1,603.9	14.4	-62.9	-10.4
arm_left_3_link	arm_left_2_link	1,642.9	6.1	-11.3	-57.5
arm_left_4_link	arm_left_3_link	1,202.2	-41.3	11.8	13.9
arm_left_5_link	arm_left_4_link	1,336.0	-0.5	0.0	-78.3
arm_left_6_link	arm_left_5_link	416.0	-0.1	-2.5	-0.1
arm_left_7_link	arm_left_6_link	67.0	0.0	0.0	-28.9

Table 6: Link properties

Figures 9, 10, 11 and 12, show the leg joint coordinate frames.



(a) base_link



(b) leg_left_1_joint

Figure 9: Left leg joints (I)



(c) leg_left_2_joint



(d) leg_left_3_joint

Figure 10: Left leg joints (II)



(e) leg_left_4_joint



(f) leg_left_5_joint

Figure 11: Left leg joints (III)



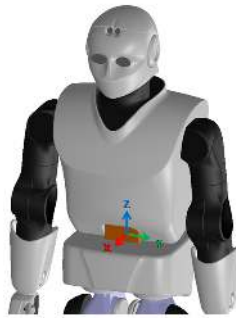
(g) leg_left_6_joint



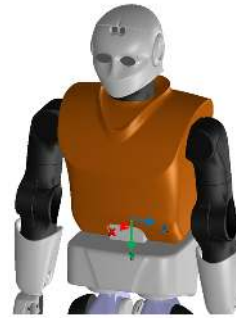
(h) sole_left_joint

Figure 12: Left leg joints (IV)

Figure 13 indicates the joint coordinates of the torso.



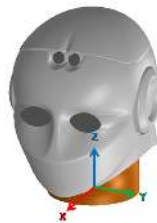
(a) torso_1_joint



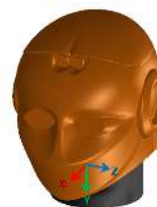
(b) torso_2_joint

Figure 13: Torso joints

Figure 14 indicates the joint coordinates of the head.



(a) head_1_joint



(b) head_2_joint

Figure 14: Head joints

Figures 15, 16, 17 and 18, indicate the joint coordinates of the arms.



(a) arm_left_1_joint



(b) arm_left_2_joint

Figure 15: Left arm joints (I)



(c) arm_left_3_joint



(d) arm_left_4_joint

Figure 16: Left arm joints (II)



(e) arm_left_5_joint



(f) arm_left_6_joint

Figure 17: Left arm joints (III)



(g) arm_left_7_joint

Figure 18: Left arm joints (IV)

3 Sensors

3.1 Cameras

There is a camera in the head of REEM-C that includes depth sensors and a RGB sensor.

FOV(HxVxD)	91degrees x 42 degrees x 77 degrees
	Dual Global shutter sensors for up to 90FPS depth stream
	Full HD RGB camera calibrated and synchronized to depth data

Table 7: Camera features

Format	Resolution	Frame Rate
Z [16 bits]	1280x720	6,15,30
	848x480	6,15,30,60,90
	640x480	6,15,30,60,90
	640x360	6,15,30,60,90
	480x270	6,15,30,60,90
	424x240	6,15,30,60,90
Y8 [8 bits] L_UYVY [16 bits] RY8_LY8 [16 bits]	1280x720	6,15,30
	848x480	6,15,30,60,90
	640x480	6,15,30,60,90
	640x360	6,15,30,60,90
	480x270	6,15,30,60,90
	424x240	6,15,30,60,90
YUY2	1920x1080	6,15,30
	1280x720	6,15,30,60,90
	848x480	6,15,30,60,90
	640x480	6,15,30,60,90
	640x360	6,15,30,60,90
	480x270	6,15,30,60,90
	424x240	6,15,30,60,90
	320x240	6,15,30,60,90
	320x180	6,15,30,60,90
Calibration [24 bits]	1280x720	25,15
	960x540	30,15
	1280x800	30,15
	640x400	30,15
RY8_LY8 [16 bits]	1920x1080	25,25
	1280x800	30,15

Table 8: Depth camera image format

Active Pixels	1920 x 1080
Sensor Aspect Ratio	16:9
Format	10-bit RAW RGB
F Number	f/2.0
Focal Length	1.93mm
Filter type	IR Cut Filter
Focus	Fixed
Shutter Type	Rolling Shutter
Vertical Field of View	69.4 degrees
Horizontal Field of View	42.5 degrees
Diagonal Field of View	77 degrees
Distorsion	<=1.5%

Table 9: Depth camera features

3.2 6 axes force sensors

On REEM-C there are two 6-axis force sensors in the ankles and two in the wrists. The specifications are shown in Table 10 and Table 11.

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

Table 10: Force Torque sensors mounted in the ankles

Fx,Fy range	290 N
Fz range	580 N
Tx,Ty range	10 Nm
Tz range	10 Nm
Fx,Fy resolution	1/8 N
Fz resolution	1/8 N
Tx,Ty resolution	1/188 Nm
Tz resolution	1/752 Nm

Table 11: Force Torque sensors mounted in the wrists

3.3 Inertial Measurement Unit

There is one inertial measurement unit placed in the hip. Close to the center of mass of REEM-C .

Gyro bias stability	18°/h
Roll/Pitch Static 1 σ RMS	0.2°
Roll/Pitch Dynamics 1 σ RMS	0.5°
Yaw	1.0°

Table 12: Inertial Measurement Unit

3.4 Motors

Every degree of freedom of REEM-C is powered by an electrical motor, except the fingers of the hands, which are underactuated (one motor moves more degrees of freedom). In Table 13 are shown the input specifications for every non-hand joint motor. These values also correspond to the respective joints of the right side.

Joint	Nominal voltage (V)	Power (W)	Torque constant (mN.m/A)	Cont. current (A) (RMS)
leg_left_1_joint	48	145	185.3	3.5
leg_left_2_joint	48	270	149.9	7
leg_left_3_joint	48	270	254.6	7
leg_left_4_joint	48	624	154.8	16
leg_left_5_joint	48	270	254.6	7
leg_left_6_joint	48	270	149.9	7
arm_left_1_joint	48	219	135.8	4.57
arm_left_2_joint	48	171	86.9	3.57
arm_left_3_joint	48	171	86.9	3.57
arm_left_4_joint	48	171	86.9	3.57
arm_left_5_joint	48	11	61.3	0.204
arm_left_6_joint	48	11	61.3	0.204
arm_left_7_joint	48	11	61.3	0.204
torso_1_joint	48	219	135.8	4.57
torso_2_joint	48	219	135.8	4.57
head_1_joint	48	23	69.8	0.401
head_2_joint	48	23	69.8	0.401

Table 13: Motors

3.5 Connectivity

REEM-C is equipped with a dual band 802.11 a/b/g/n and a 10/100/1000 Mbps Ethernet port.

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

- Wireless VPN connection

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

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

- Direct VPN connection

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

- Direct wired connection

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

- Direct wireless connection

Another possibility to connect a laptop directly to REEM-C is using the wireless connection in Access Point mode. This is essentially identical to plug to the wired service port.

3.6 Electrical parts and components

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

Electrical power supply and connectors

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

Battery

The specifications of the battery supplied with REEM-C are shown in Table 14.

Type	Li-Ion
V_nominal	44.4V
V_max	50.4V
V_cutoff	30V
Nominal capacity	27.2 Ah
Nominal energy	1225 Wh
Cont. discharge current	20A
Pulse discharge current	45A
Max. charging current	15A
Charging method	CC/CV
Weight	8.5 kg

Table 14: Battery specifications

Computers

There are 2 computers (PCs) in REEM-C , one is for real-time control applications and another one for multimedia applications. On the Service Panel the Control PC is referred as PC1, Multimedia PC is referred as PC2. Both PCs are controlled with the same power button. Both PCs have 2 USB ports and 1 VGA port accessible through the service panel.

3.7 Service Panel

There are 2 access levels to the connectors of the service panel (all detailed in Table 15):

- First level: access to power and emergency switch and basic communication ports (Figure 19).
- Second level: after removing the cover of service panel, the USB and VGA ports of the Control and Multimedia computers are accessible (Figure 20).

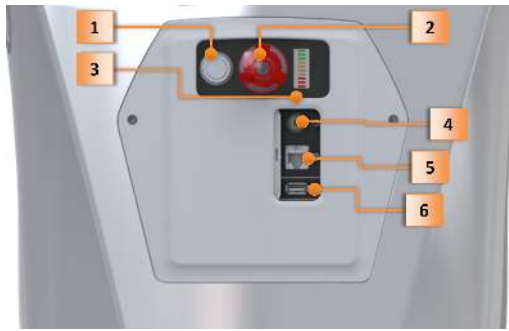


Figure 19: First level

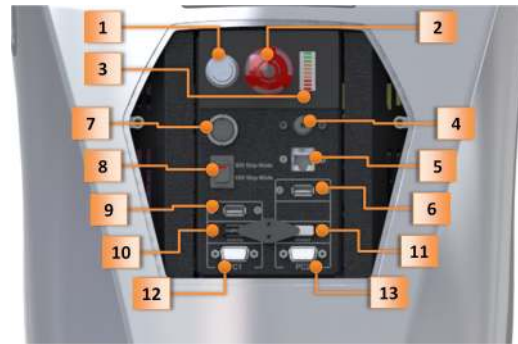


Figure 20: Second level

Number	Name / Short description
1	Power Button
2	Emergency Stop
3	Battery Level Indicator
4	Audio Connector
5	Ethernet RJ-45 Connector
6	USB port of Multimedia PC (e.g Kinect)
7	Service Connector
8	Hardware / Software Service Switch
9	USB port of Control PC
10	USB port for Joystick receiver (Control PC)
11	USB port for Headset receiver (Multimedia PC)
12	VGA Output Connector from Control PC
13	VGA Output Connector from Multimedia PC

Table 15: Service Panel description

3.8 Extra Service Panels connectors

Some REEM-C models have additional connectors in the service panel to ease integration of a external Gigabit device. The location of these connectors are depicted in figure 21.

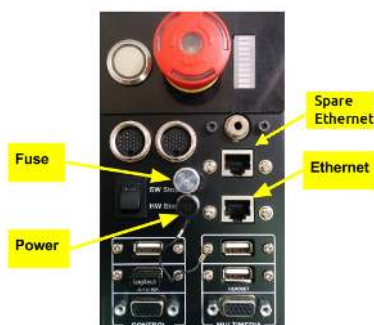


Figure 21: Extra connectors



Figure 22: Power connector

1. The power connector provides access to the robot's battery power when the robot is turned on. The output voltage can vary depending on the battery's state of charge. GND pin is marked red on the

connector's housing. Power connector shown in figure 22 is a LEMO ECG.0B.302.CLL model, mating part is a LEMO FGG.0B.302.Cxxx.

2. The output is protected with a 5A fuse. **WARNING: always replace the broken fuse with the same model. Do not use fuses with different form factor or higher current rate.** Fuse model is Littelfuse (it is not a typo) 0273005.
3. The connector labelled as *Ethernet* in figure 21 is connected to the internal robot Gigabit switch. It can be configured to access the robot using a wired connection.
4. The connector labelled as *Spare Ethernet* is connected to the multimedia computer directly.

REEM C

Handling



4 Storage

4.1 Overview

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

4.2 Transport

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

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

The batteries cannot be transported within the same crate as REEM-C.

4.3 Storage cautions

- Always store REEM-C where it will not be exposed to weather.
- The storage temperature range for REEM-C is between 0°C ~ +50°C.
- The storage temperature range for the batteries is between +10°C ~ +35°C.
- It is recommended to remove or disconnect the battery from REEM-C when storage period exceeds two weeks.
- It is recommended to charge the battery to 50% when storing it for more than two weeks.
- The storage temperature range for the Basestation is between 0°C ~ +60°C.
- If REEM-C has to be stored for a long time, it is highly recommended to keep it in its crate. Instructions explaining how to place the robot into the crate can be found in Section 6.4.
- REEM-C should be always hung using the hooks that stand out from the shoulders (Figure 23). To reduce the stress on the mechanical structure of the robot, make sure that the feet are clearly in contact with the floor, completely flat and the robot remains straight and vertical. Always use the carabiners and ropes supplied with REEM-C.



Figure 23: Hooks location

- Never hold REEM-C by any other part but the hooks over the shoulders (Figure 23).
- REEM-C should never stay out of its crate without being sustained by the hooks over the shoulders (Figure 23).
- Avoid the use or presence of water near REEM-C.
- Avoid any generation of dust close to REEM-C.
- Avoid the use or presence of magnetic devices or electromagnetic fields near REEM-C.

5 Indications for handling

In order to handle REEM-C safely, take the following into account:

- REEM-C may only be used in perfect technical conditions, in accordance with its designated use, and only by safety-conscious people who are fully aware of the risks involved during the operation.
- Never use REEM-C without being sustained by the hooks over the shoulders (Figure 23). Take the weight of the robot into account when dimensioning the holding device (see chapter “Specifications”, section “Weight”).
- While the robot is on, keep away all objects, obstacles and people, respecting the safety distances shown in Figure 33.
- Never drag REEM-C.
- Never pull or push REEM-C.
- Never lift REEM-C by hand.
- Never hit or drop REEM-C.
- Never use sharp objects against REEM-C.
- While handling REEM-C, do not wear watches or any kind of jewelry on hands or wrists. It is recommended to use gloves.
-
- Do not touch the camera’s lenses to avoid damaging or dirtying them.
- Avoid the use or presence of liquids near the robot.
- REEM-C contains numerous delicate electronic circuits and components which can become damaged as a result of electrostatic discharge (ESD). Prior to handling, carefully read and follow these procedures:
 - Avoid touching any metallic parts of REEM-C.
 - It is advisable to wear an electrostatic discharge (ESD) wrist strap when handling external metallic parts or make sure to discharge yourself of ESD.

REEM C

Assembly



6 Set-up and mounting conditions

6.1 Overview

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

6.2 Robot crate

Approximate external dimensions of the crate:

- Weight: 75 kg (empty), 155 kg (with REEM-C inside)
- Length: 1950 mm
- Width: 900 mm
- Height: 650 mm (including wheels)

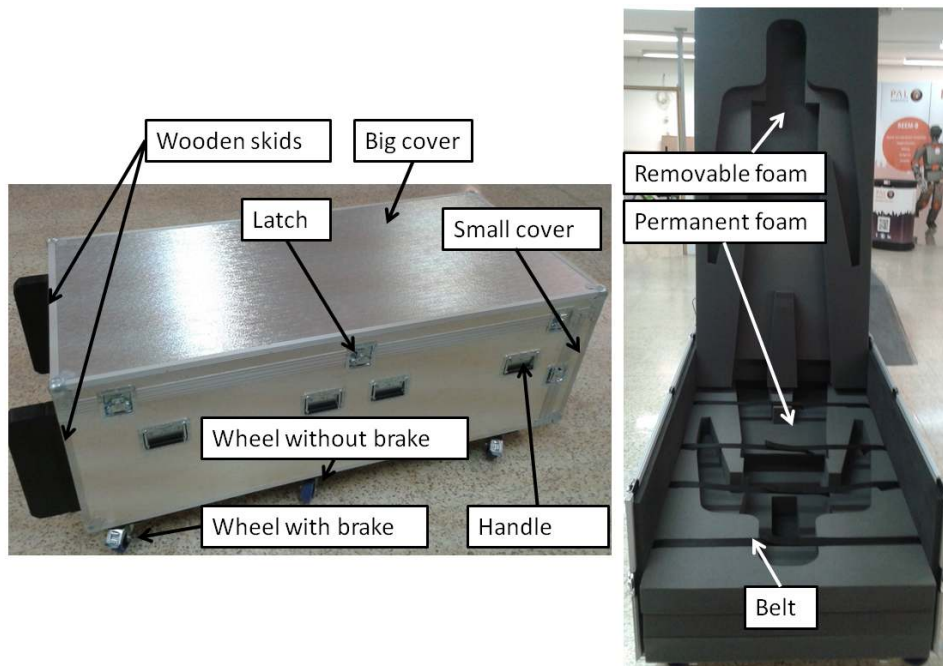


Figure 24: Crate parts

6.3 How to get the robot out of the crate

To get the robot out of its crate safely, it is needed:

- At least two people.
- A crane (i.e. Figure 25), with a maximum cradle height over 1900 mm, and horizontal legs height less than 150 mm.
- Two adjustable straps and the carabiners supplied with the robot (Figure 26) hanging on the cradle.



Figure 25: Crane



Figure 26: Carabiners

Instructions to get REEM-C out of the crate:

1. With the crate in horizontal position, remove the big cover.
2. Remove the small cover.
3. Remove the foam protector and make sure that the 4 belts that keep the robot secured are tight.
4. Using the corresponding handles at both sides of the box, lift the crate in such a way that the box will remain on top of the two wooden skids. Later, this two wooden parts will allow to place the legs of the crane below the crate. To make it safer it is recommended to release the crate wheels brakes.
5. Move the crane in front of the robot to such a position that the carabiners can be inserted in both black hooks over the shoulders of the robot and insert them. The horizontal legs of the crane will be located below the crate, between the wooden skids. Do not to hit the robot plastic covers with the carabiners to avoid damages.
6. Adjust both straps with the same length in such a way that allows the crane to lift up the robot completely. It is recommended to shorten the straps as much as possible but avoiding collisions between REEM-C head and the cradle. This will allow the operator to take more profit of the lift stroke.

7. Lift the robot using the crane until the straps are slightly tensioned.
8. Loosen all the belts that fix the robot to the crate (head, torso, hip and knees).
9. To separate the robot from the back foam protector lift the robot slowly at the same time that pulling the crane.
10. Now the robot can be completely moved away from the crane.
11. Move the robot with the crane carefully, avoiding eventual collisions.

6.4 How to put the robot into its crate

Required to get the robot into its crate either to store or transportation:

- At least two people.
- A crane (i.e. Figure 25), with a maximum cradle height over 1900 mm, and horizontal legs height less than 150 mm.
- Two adjustable straps and the carabines supplied with the robot hanging on the cradle.

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

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

7 Power supply

7.1 Battery assembly

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

1. Make sure that the “Hardware Stop Mode” is selected in the service panel and press the Emergency stop.
2. Remove both covers in each side of the robot pelvis, by loosening the screws (Figure 27, Figure 28).
3. The battery will be inserted from robot left side, as shown in Figure 29.
4. Handle carefully. The weight of the battery is about 8.5 kg.
5. Slide the battery inside the compartment up to the end (Figure 30).
6. Tighten the two M4 screws supplied with the battery (Figure 30).
7. Plug the connector, as shown in Figure 31. Tighten 2 fixing screws with a screwdriver.
8. Mount both side covers (Figure 32) and insert its M5 screws.



Figure 27: Release screws



Figure 28: Remove side covers

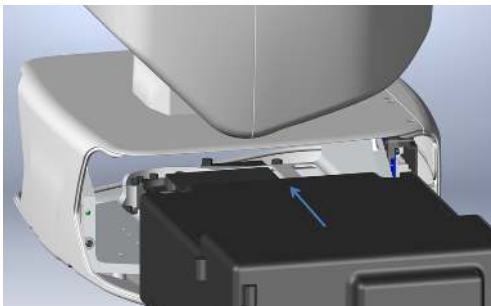


Figure 29: Battery assembly

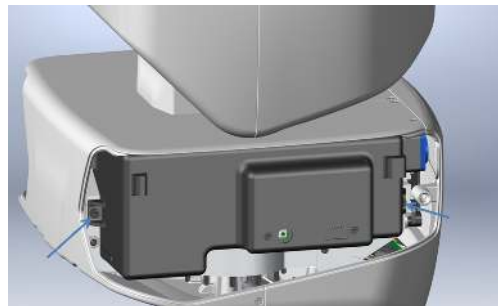


Figure 30: Insert screws

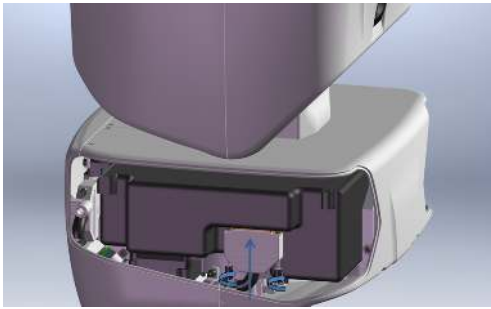


Figure 31: Plugging

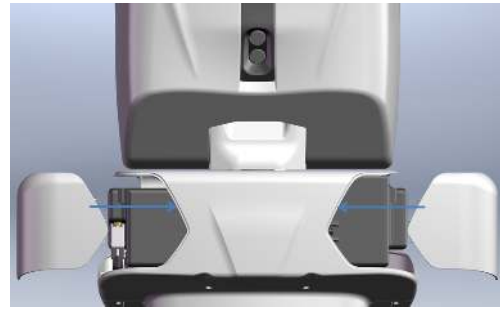


Figure 32: Attach side covers

REEM C

Safety



8 Introduction to safety

8.1 Overview

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

8.2 Intended applications

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

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

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

8.3 Working environment and usage guidelines

The working temperatures are:

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

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

Maintain a safe environment:

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

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

8.4 Risks

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

8.5 Safety distance

In order to keep both the robot and operators safe while REEM-C is working, everyone except authorized personnel should maintain the distance shown in Figure 33.

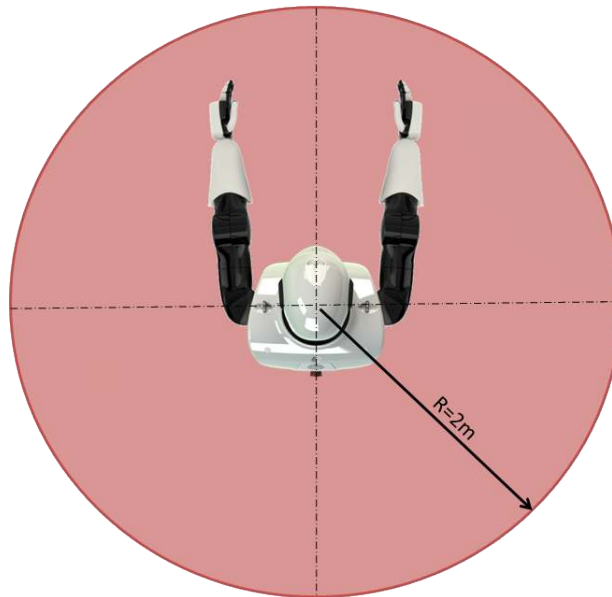


Figure 33: Safety distance

8.6 Battery manipulation

General guidelines when manipulating the battery:

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

9 Safety measures in practice

9.1 Control system functions

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

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

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

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

9.2 Emergency stop

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

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

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



Figure 34: Emergency button in normal state (left), emergency button in pressed state (right)

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

9.3 Firefighting equipment

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

If there is a fire, please follow these instructions:

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

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

9.4 Leakage

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

REEM C

Using REEM-C



10 General information for usage

This section contains information and procedures necessary for using the system.

10.1 User access

Computers of the robot, basestation and development computer have 3 users:

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

The default passwords of the elements can be found in Section 16.

10.2 Power supply

The robot is supplied with a 48V Lithium-Ion battery. This is the only power source and never be replaced for any other battery that had not been supplied with the robot or otherwise by PAL Robotics. When the battery is discharged use exclusively the battery charger supplied with the robot.



Figure 35: Battery and charger supplied

The charger has an output of 50.4V DC and it is suitable to give 6A. The device has a male security connector POWERCON NAC3FCA to plug-in into a female battery plug.

If the battery has to be recharged inside the robot, it is possible to plug the charger using the female connector located on the back of the waist. In case that the battery is out of the robot, verify that all environmental conditions are met. The connector plug is located in the battery as shown in Figure 36.



Figure 36: Connector of charger

The battery charger has 3 state LED indicators and one power switch. The table below describes the meaning of LED indicators.

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

Procedure of charging the battery:

1. Plug the battery charger to the electrical network being sure that the switch of the battery charger is in 0 position.
2. Plug the battery charger to the battery as shown in Figure 37.
3. Turn on the charger switching to position 1. The LED indicator will go to charging state.

When the battery is charged:

1. Turn off the battery charger with its switch.
2. Unplug the security connector from the battery as indicated in Figure 38.
3. Unplug the battery charger from the electrical network.



Figure 37: Plugging charger



Figure 38: Unplugging charger

10.3 Controls and protective devices

The robot has controllers on several levels.

Software controllers:

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

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

Hardware controllers:

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

Protective devices:

As protective devices, on the back of the robot, near the battery compartment are located 4 fuses as shown in Figure 39.

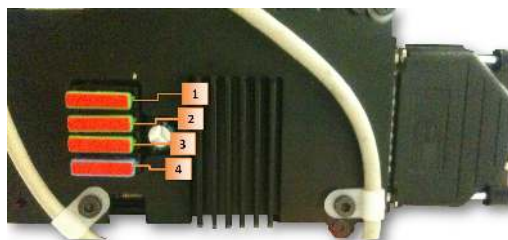


Figure 39: Fuses

These fuses are described below:

Fuse Number	Description	Value	Manufacturer / Model
Fuse 1	Right leg power	30A/80V	LITTELFUSE 166.7000.5306
Fuse 2	Torso power	30A/80V	LITTELFUSE 166.7000.5306
Fuse 3	Left leg power	30A/80V	LITTELFUSE 166.7000.5306
Fuse 4	Standby Power	3A/80V	LITTELFUSE 166.7000.4306

Always replace damaged fuse with the one indicated in the table. Never use a fuse with different characteristics because it can cause irreversible damage to the robot.

10.4 Fault identification and location

To avoid damage to any electrical, mechanical or structural component of the robot it is important to locate any fault that can occur during the use of the robot. Please, follow the instructions and steps described in Section 13.

In case of any malfunction, be sure that the issue is not produced by any mistake in your application. First of all, restore default parameters and configuration and restart the operation. If the problem persists, contact PAL Robotics and give us complete description, including:

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

In order to identify hardware problems consult the output of the WebCommander tool, documented in section 21.

11 Network kick-off

11.1 Overview

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

11.2 Reserved IP address

The majority of communications between components onboard the REEM-C happens via an onboard Ethernet network, referred to as the "Robot Internal Network".

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

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

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

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

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

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

The IP addresses used in the building network MUST not use any of these three ranges because can interfere with the services of the robot.

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

11.3 Network Diagram

The default setup for the network is depicted in figure 40. On the left hand side of the figure, the IP configuration of the robot devices are represented. On the right hand side, the network elements of the building network are shown.

Robot internal network The building network consist of two PCs, one switch and one router. The wifi router in the robot has two interfaces: an Ethernet interface and a Wifi interface. In the default setup, the robot is connected to the building network through the Wifi interface.

Building network The following elements comprises the following elements: access point, basestation and workstations. In the default setup, these three elements belongs to the same IP subnet: e.g. 192.168.1.X. Nevertheless, these elements can belong to different subnets as long as the robot and the workstation have access to the basestation.

11.4 Basestation

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

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

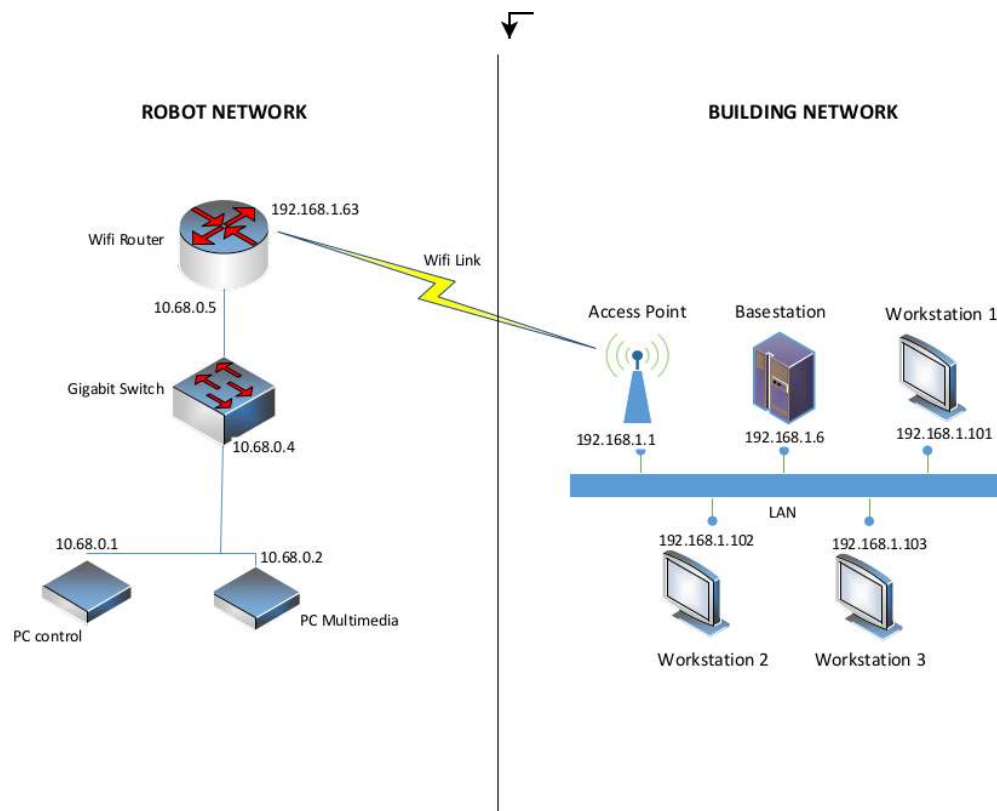
Other important network configuration is the NTP server.

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

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

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

The default ntp.conf has the following configuration:



ii

Figure 40: Network diagram

```

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

```

Listing 1: Default ntp configuration file

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

Register robots and devices For all the communications of the robots and devices (tablets and terminals) an encrypted VPN is created. The basestation acts as a server for this VPN, that means that all the robots and devices have to be registered in the basestation in order to be allowed to join it.

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

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

Here it is shown an example of how to register a REEM-C with serial number 1.

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

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

Here it is shown an example of how to register a development computer desktop1.

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

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

This command will be explained in more details when all the capabilities of the basestation are presented.

In this document it will be used as example a REEM-C robot with serial number 1.

11.5 Robot

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

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

Using a web browser the following address will show the Reem Commander of the robot.

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

Section 21.3.7 on page 99 explains how to modify the networking parameters.

11.6 Development computer

Register computer

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

To modify the hostname it is needed to edit the file */etc/hostname* and the file */etc/hosts* using user root. Then a reboot is required.

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

Network Configuration

Network configuration is set using the standard NetworkManager in Ubuntu.

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

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

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

```

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

```

Listing 2: Default */etc/hosts* configuration file

Change the IP address of *basestation_public* (line 4) for the current IP of the basestation.

Then, a restart of the VPN is needed:

```
service openvpn restart
```

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

```

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

```

Listing 3: Default ntp configuration file

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

```

1 server basestation

```

Listing 4: Default ntp configuration file

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

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

Another possibility is to use the basestation for reem domain and building DNS server for everything else. Figure 41 shows an example of this situation.

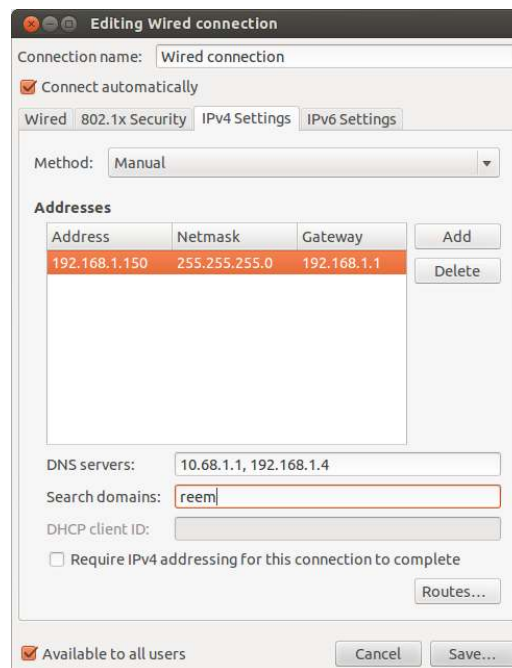


Figure 41: DNS configuration

12 Getting started

Before starting the robot make sure that the user is familiar with both the instructions of the Safety Section 8, and the usage information from Section 10.

12.1 Switch on the robot

1. Make sure that the emergency stop button is released, the robot is hanging on the crane and its feet do not touch the ground.
2. Switch on the robot by pressing the power button for 2 seconds.



Figure 42: Power button states: off(left) and on(right)

3. Fans should start spinning and the light of the power button should change to red.
4. Verify that the motors and sensors are running using the WebCommander. The figure 43 shows the motors status. If there is a problem in any of the motors, repeat the process rebooting the robot. If the problem persists contact PAL Robotics. Be aware that WebCommander can not be displayed with the Firefox web browser. So use Chrome or Chromium web browser.
5. Now the robot can be operated.

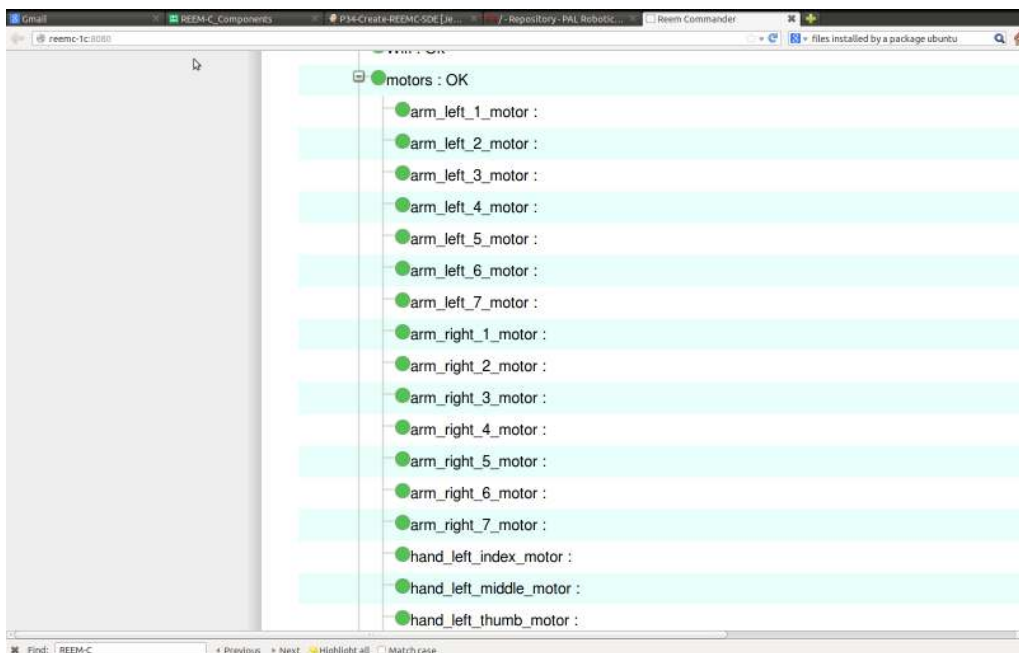


Figure 43: WebCommander motors status

12.2 Shutting down the robot

- First make sure that the robot is secured on the crane.

- Hold the power button until its light disappears. Now the robot is off.
- Extend the legs and leave the robot secured on the crane and make sure that its feet touch the ground. See Figure 44 for example positions.



Figure 44: Correct (left) and incorrect (right) shutdown positions

REEM C

Maintenance



13 Nature and frequency of inspections

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

13.1 Cleaning

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

13.2 Daily inspection

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

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

Check that each degree of freedom is running smoothly.

13.3 Once every three months

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

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

13.4 Once every year (by technical service)

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

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

13.5 Once every three years (by technical service)

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

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

14 De-commission, dismantling and disposal

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

Disposal and recycling of the Lithium-Ion battery:

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

REEM C

Marking



15 Robot identification

The robot is identified by physical labels containing:

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

A total of three labels of two different kinds will be found on the robot (Figure 45, Figure 46).

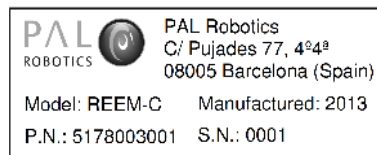


Figure 45: Label type A

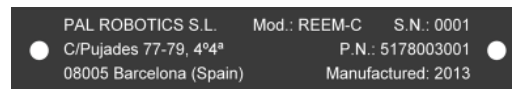


Figure 46: Label type B

Table 16 indicates the accessibility and location of the three labels:

	Type	Location	No. of bolts to access
1	A	Rear pelvis, cover	None
2	B	Pelvis, right, structure	1
3	B	Back, structure	8

Table 16: Labels location and accessibility

REEM C

Software Recovery

PALO 
ROBOTICS

16 Software recovery

16.1 Overview

The procedure of how to install from the beginning all the elements, that integrates REEM-C infrastructure, will be explained in this section.

16.2 Basestation installation

Hardware installation

Connect the computer, the mouse and the keyboard to the electrical plug. Do not connect the Ethernet cable because the installation is self-contained.

BIOS configuration

In this section the optimal BIOS configuration will be set and it will be configured to boot from the USB memory.

- Insert the USB memory in a USB slot.
- Turn on the computer pressing F2 or DEL. Wait until BIOS menu appears.
- Press F8 and select Sony Storage Media.
- *Go to Save & Exit and execute Save Changes & Exit.*

Operating System installation

If the BIOS has been configured as indicated in the previous section, the menu shown in Figure 47 will appear.

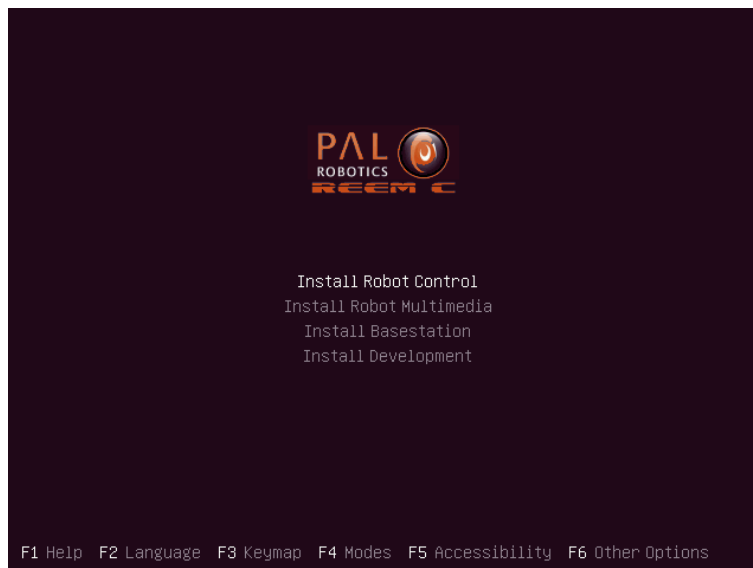


Figure 47: System installation menu

Select the “Install Basestation” option. Then a menu asks for the layout of the keyboard. Please choose the appropriate layout.

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

Default users

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

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

16.3 Robot computers installation

Hardware installation

REEM-C has two computers called Control and Multimedia. Control computer is labeled as CONTROL and multimedia computer is labeled as MULTIMEDIA in the robot.

During the installation process it is needed to plug a monitor in the VGA connector and an USB keyboard.

BIOS configuration for multimedia computer

In this section the optimal BIOS configuration will be set and it will be configured to boot from the USB memory.

- Insert the USB memory in a USB slot.
- Turn on the computer pressing F2. Wait until BIOS menu appears.
- Go to *Save & Exit* and execute *Restore Defaults*.
- Go to menu *Advanced*, then *ACPI Settings* and mark *Enable ACPI Auto as Enabled*.
- In the *Advanced* menu, select *CPU configuration* and mark *Hyper-threading* as *Disabled*.
- In the *Chipset* menu, go to *PCH-IO Configuration* and mark *Restore AC Power Loss* as *Power Off*.
- Go to menu *Boot*, then *Boot Option*. Make sure than *1st Boot* is the USB drive.
- Go to *Save and Exit* menu and exit using *Save Changes and Exit*.

BIOS configuration for control computer

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

- Insert the USB memory in a USB slot.
- Attach keyboard and turn on the computer pressing DEL key in keyboard. Wait until BIOS menu appears.
- Go to *Save & Exit* and execute *Restore Optimized Defaults*.
- In the *Advanced* menu, select *CPU configuration* and mark *Hyper-threading* as *Disabled* and *EIST* as *Disabled*.
- In the *Chipset* menu, go to second LAN Controller and mark *Disabled*.
- In the *Chipset* menu, go to *PCH-IO Configuration* and mark *Restore AC Power Loss* as *Power Off*.
- Go to menu *Boot*, then *Hard Disk Drive BBS Priorities*. Make sure that *1st Boot* is the USB drive .
- Go to *Save and Exit* menu and exit using *Save Changes and Exit*.

Operating System installation

If the BIOS has been configured as indicated in previous section, the menu shown in Figure 48 will appear.

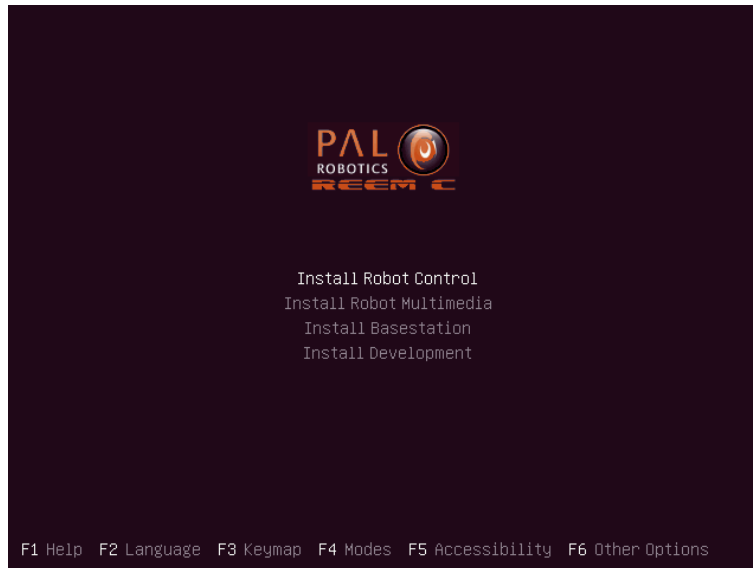


Figure 48: System installation menu

When the computer is selected it asks for the layout of the keyboard.

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

Default users

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

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

Operating system installation

- The menu, shown in Figure 48 will appear, select control or multimedia accordingly.

16.4 Development computer installation

Hardware installation

Connect the computer to the electric plug, the mouse and the keyboard. Do not connect the Ethernet cable because all the installation is self-contained.

Please, connect the monitor to the DVI connector. Do not use the VGA connector because this connector does not have graphic acceleration.

Operating System installation

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

- Insert the USB memory in a USB slot.
- Press PC Start button.
- Access the BIOS pressing the DEL or the F2 keyboard key.
- *Choose the USB memory* as the booting device.
- Once the computer boots from the USB, the menu shown in Figure 49 will appear.

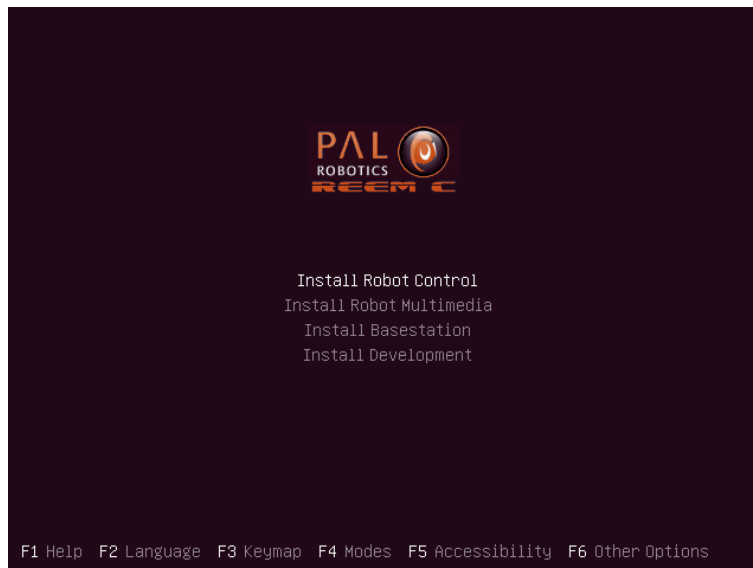


Figure 49: System installation menu

- Select the “Install Development” option. Then a menu asks for the layout of the keyboard. Please choose the appropriate layout.

Default users

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

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

17 Generation of USB installation

17.1 Overview

In this section it will be specified the procedure of generation of the USB for installation from the ISO files. Please, contact PAL-Robotics in order to download the ISOs from a PAL ftp server.

17.2 Requirements

The material needed for the generation of the USBs is:

- PAL-Robotics ISO: A file called *pal-robotics-software-X.XX-eurobench-amd64.iso*
- USB pendrive
- Ubuntu computer

17.3 PAL-Robotics ISO

This section explains how to generate an USB with PAL-Robotics ISO.

The generation is made using an application called *usb-creator-gtk*. As *root* user type in a terminal:

```
usb-creator-gtk
```

The window shown in Figure 50 will appear.

Then perform the following sequence:

- First erase all the contents in the USB memory. Select the USB in the field *Disk to use* and press *Erase Disk*.
- Press *Other* button and select *pal-robotics-software-X.XX-eurobench-amd64.iso* file.
- Now with the USB ready and the ISO selected press *Make Startup Disk*.
- A window will indicate the progress. When this action is completed the USB is ready.

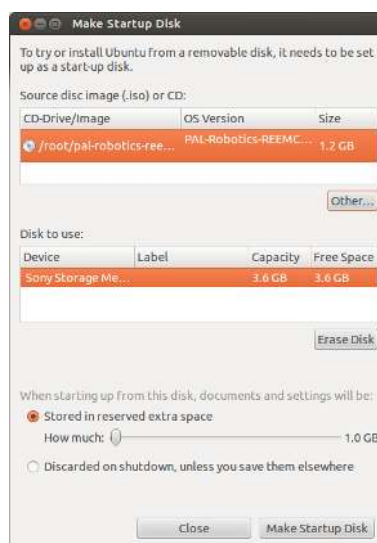


Figure 50: usb-creator-gtk application

REEM C

Basestation



18 Basestation

18.1 Overview

The Basestation is the central element that coordinates communication between the different elements of the REEM-C infrastructure.

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

18.2 Users

There are three users in the Basestation:

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

18.3 Network configuration

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

```

1 auto lo
2 iface lo inet loopback
3
4 auto eth0
5 iface eth0 inet static
6     address 192.168.1.6
7     netmask 255.255.255.0
8     gateway 192.168.1.1
9     dns-search reem
10    dns-nameservers 127.0.0.1

```

Listing 5: Default network configuration

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

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

18.4 NTP server

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

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

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

The default `ntp.conf` has the following configuration:

```

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

```

Listing 6: Default ntp configuration file

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

18.5 OpenVPN

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

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

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

By default the VPN uses an IP with the range 10.68.1.X/24³. It is possible to use other ranges, but the operator will need to contact PAL Robotics for more information.

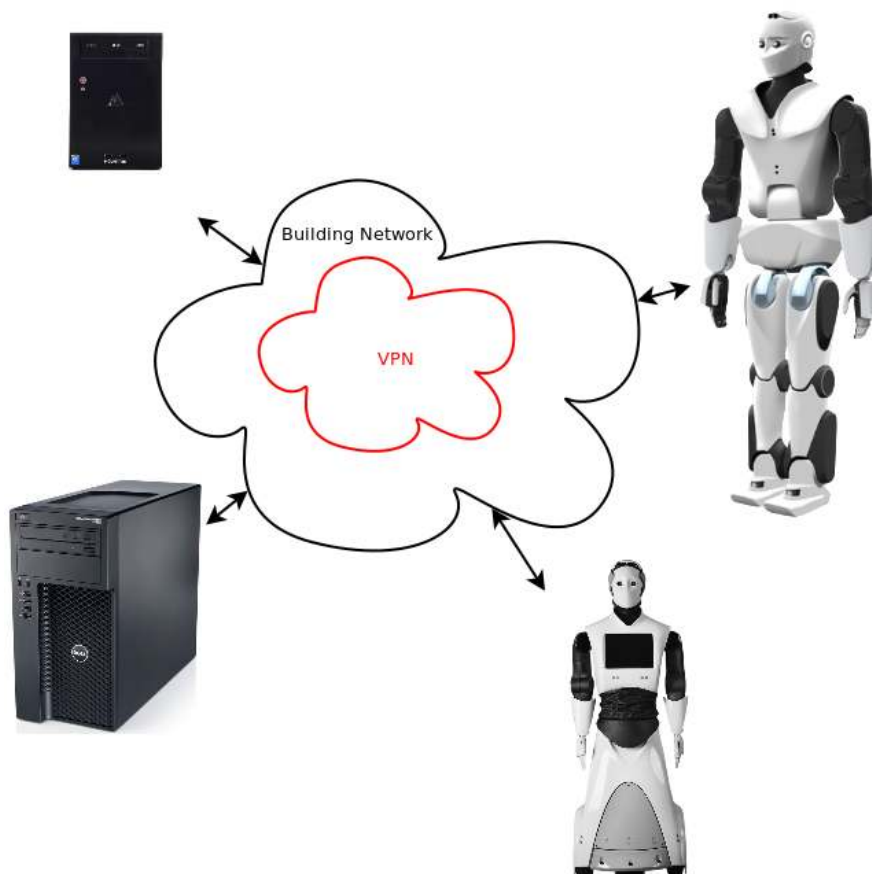


Figure 51: OpenVPN network

³This range is shared with the PPTP and L2TP/IPsec PSK connections. It has to be taken in consideration when assigning IP addresses.

18.5.1 Add an element to the VPN

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

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

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

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

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

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

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

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

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

18.5.2 Remove an element from the VPN

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

- Robot: use the command *delRobotVpn*:

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

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

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

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

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

Examples: delVpnUser -u desktop1.
```

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

18.5.3 Copying files between VPN users

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

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

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

There are two commands for making network copies:

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

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

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

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

18.6 DNS Server

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

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

These commands are:

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

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

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

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

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

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

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

18.7 PPTP Connections

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

18.7.1 Address pool

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

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

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

Listing 7: PPTP address pool

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

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

18.7.2 Security

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

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

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

Listing 8: PPTP default user

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

18.7.3 DNS settings

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

18.8 L2TP/IPsec PSK

It is also possible to connect to basestation using L2TP/IPsec PSK. It's been configured to allow access of Android or iOS devices.

18.8.1 Address pool

In */etc/xl2tpd/xl2tpd.conf* the pool of IP addresses that will be assigned dynamically is defined.

```
1 [lns default]
2 ip range = 10.68.1.250–10.68.1.253
3 local ip = 10.68.1.254
```

Listing 9: L2TP address pool

By default, there is a pool of 4 IP addresses (from 10.68.1.250 to 10.68.1.253). It can be modified by the user.

18.8.2 Security

Default shared IPsec password is *p@l-r0b0t1cs* and it can be found in */etc/ipsec.secrets*:

```
1 # RCSID $Id: ipsec.secrets.proto,v 1.3.6.1 2005/09/28 13:59:14 paul Exp $
2 # This file holds shared secrets or RSA private keys for inter-Pluto
3 # authentication. See ipsec_pluto(8) manpage, and HTML documentation.
4
5 # RSA private key for this host, authenticating it to any other host
6 # which knows the public part. Suitable public keys, for ipsec.conf, DNS,
7 # or configuration of other implementations, can be extracted conveniently
8 # with "ipsec showhostkey".
9 192.168.1.6 %any : PSK "p@l-r0b0t1cs"
```

Listing 10: IPsec default shared key

For L2TP user and password is defined in */etc/ppp/chap-secrets*:

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

Listing 11: L2TP default user

In line 3 is defined the default user: *reem* and the default password: *pal*.

18.8.3 DNS settings

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

18.9 Software repositories

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

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

18.9.1 PAL Robotics 's repositories

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

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

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

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

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

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

The repositories are:

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

18.9.2 Customer software repository

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

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

This is done using the command `refresh_repository`:

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

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

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

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

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

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

18.10 System upgrade

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

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

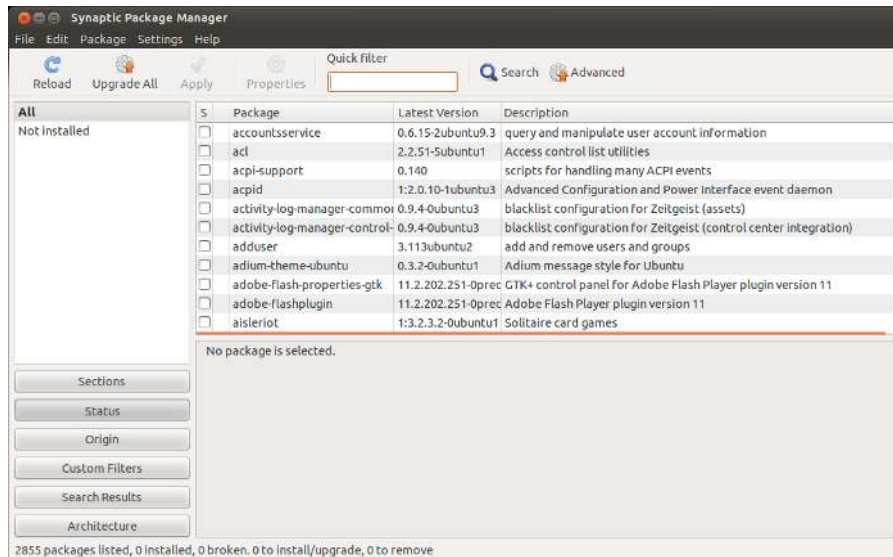


Figure 52: pal-upgrade-synaptic application

As shown in Figure 52, none of the debian packages included in the repositories explained in section 18.9 are installed.

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

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

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

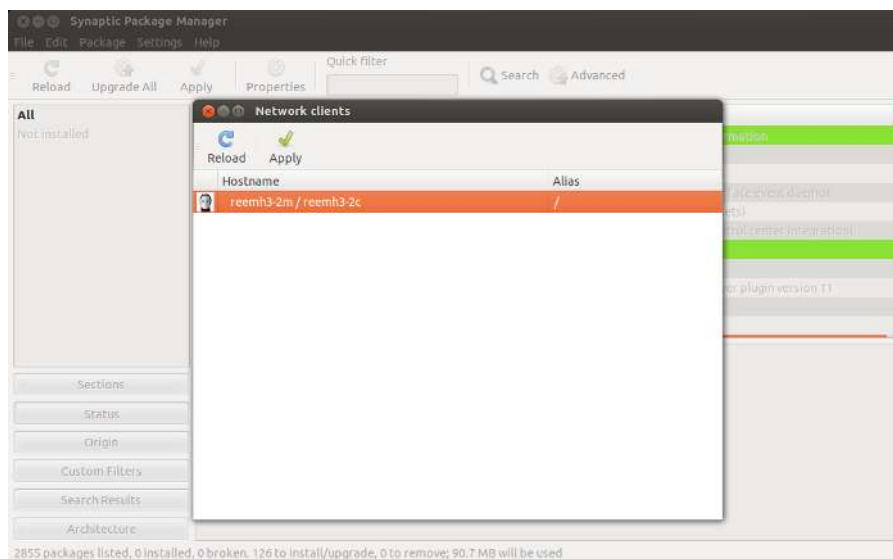


Figure 53: pal-upgrade-synaptic client selections

In this window (shown in Figure 53) select the robots and devices (use the Ctrl button for multiple selection).

The *Reload* button refreshes the list of connected clients.

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

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

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

For a general upgrade of all the debians, there is a button labeled *Upgrade All* (in Figure 52). When pressed, the network clients will open and multiple robots and/or devices can be selected. When *Apply* is selected (Figure 53) the same process will begin as before, but with all the debians installed.

For complete control of installation and/or de-installation of debian packages, use the *Advance* button. When this button is pressed a window like the one shown in Figure 53 will appear, but only one device at a time can be selected. When the *Apply* button is pressed, a remote synaptic of the selected device will open. In this synaptic, installations and de-installations can be performed. In robots, the synaptic of the multimedia computer will open first, and when it is closed the synaptic of the control computer will open.

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

18.11 PAL Robotics 's Corporate Basestation

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

This connection is used for the software updates indicated in section 18.9, as it allows remote support from PAL Robotics.

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

This allows both sides to authenticate the file.

- Receiving files from PAL Robotics:

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

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

- Sending files to PAL Robotics:

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

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

18.12 DHCP server

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

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

```
root@basestation:~# apt-get install dhcp3-server
```

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

REEM C

Development Computer



19 Development computer

19.1 Overview

The operating system used in the SDE Development Computer is based on the Linux Ubuntu 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.

19.2 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.

19.3 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/erbium/setup.bash
```

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

19.4 ROS communication with the robot

When developing applications for robots based on ROS, it is typical to have the `rosmater` 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://reemc-6c: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 reemc-6c
```

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

Then do the same from the robot:

```
ssh pal@reemc-6c
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 20.4. 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://reemc-6c:11311
export ROS_IP=10.68.0.128
```

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

19.5 System Upgrade

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

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

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

Notifications will appear whenever software upgrades are available.

REEM C

Robot Computers

PALO 
ROBOTICS

20 REEM-C Robot's Internal Computers

20.1 REEM-C LAN

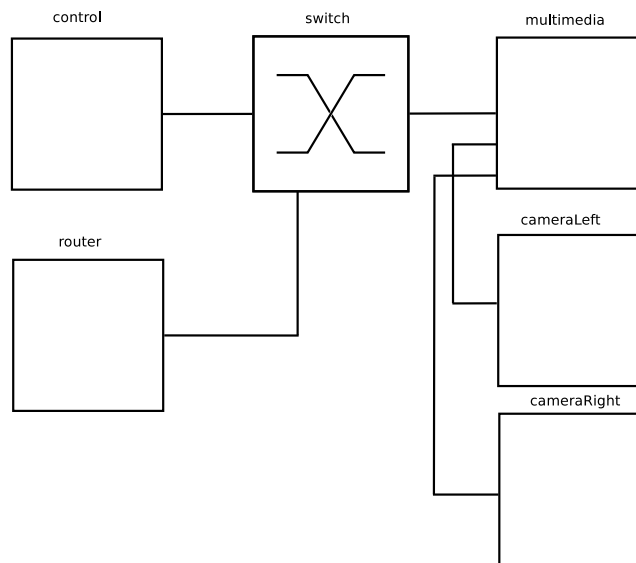


Figure 54: REEM-C LAN

Figure 54 shows the internal LAN of the robot. The name of every element is the network name used for the internal connections.

It has two computers, one is called *control* and the other one *multimedia*.

In REEM-C the control computer can be called inside this LAN as *reemcc* and the multimedia computer as *reemcm*.

Finally the VPN name is *reemc-SERIALNUMBERc* for control and *reemc-SERIALNUMBERm* for multimedia. These names are set as the hostnames of the computers and can be used in the internal LAN or in the VPN connections.

20.2 Users

The users and default passwords in the REEM-C computers are:

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

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

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

20.3 File system

The REEM-C 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@reemc-6c:~# rw
Remounting as rw...
Mounting /ro as read-write
Binding system files...
root@reemc-6c:~# 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@reemc-6c:~# exit
root@reemc-6c:~# 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.

20.4 Internal DNS

The control computer has a DNS server that is used for the internal LAN of the REEM-C 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@reemc-6c:~# 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@reemc-6c:~# delLocalDns -h
-h shows this help
-u DNSNAME dns name to remove

Example: delLocalDns -u terminal
```

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

20.5 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 REEM-C software available from the PAL repositories.

Reboot after upgrade is complete.

REEM C

Developer's Manual



21 WebCommander

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

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

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

```
http://reemc-6c:8080
```

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

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

21.2 Overview

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

21.3 Default tabs

REEM-C 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 21.4 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.

21.3.1 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.

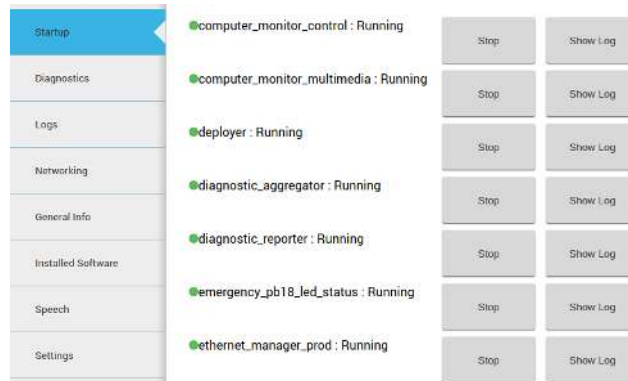


Figure 55: The Startup Tab displays the launched applications and their dependencies

21.3.2 Diagnostics Tab

Plugin Diagnostics

Description Displays the current status of REEM-C’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 REEM-C, 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 56. The status of a particular category can be expanded by clicking on the “+” symbol to the left of the name of the category. This will provide information specific to the device or functionality. If there’s an error, an error code will be shown.

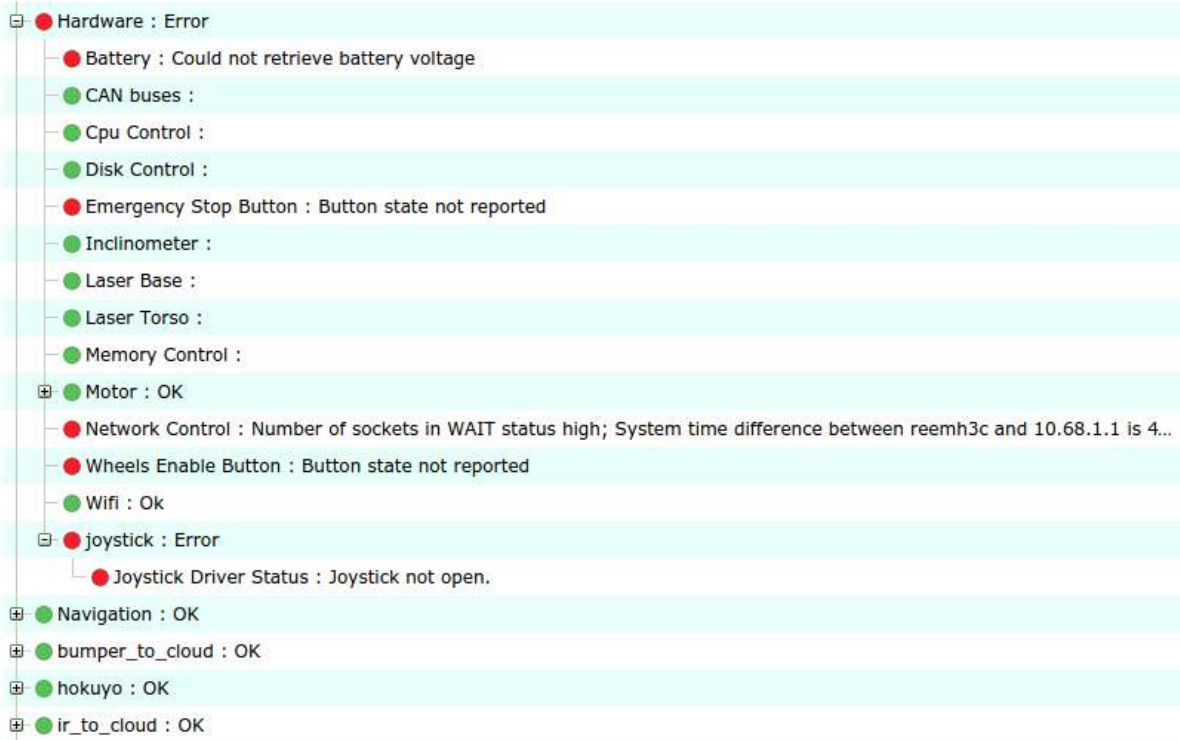


Figure 56: The Diagnostics tab displays the status of the hardware and software components of REEM-C

21.3.3 Logs Tab

Plugin Logs

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

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

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

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

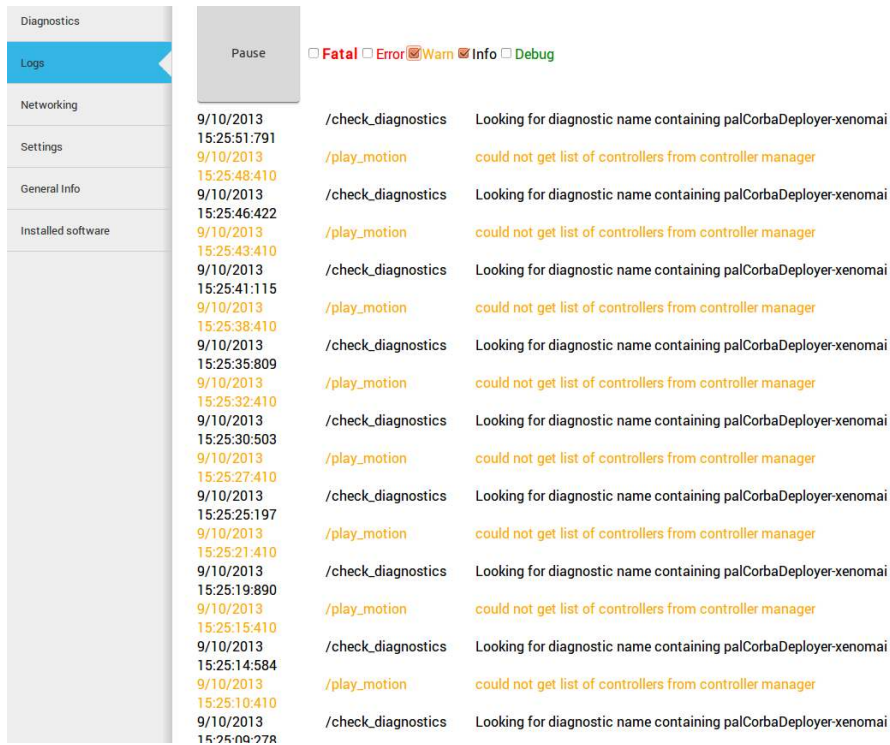


Figure 57: The Log Tab displays the log messages as they are being published in the robot

21.3.4 General Info Tab

Plugin General Info

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

21.3.5 Installed Software Tab

Plugin Installed Software

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

21.3.6 Settings Tab

Plugin Settings

Description The settings tab allows to change the behaviour of REEM-C .

Currently it allows to configure the language of REEM-C 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 REEM-C.

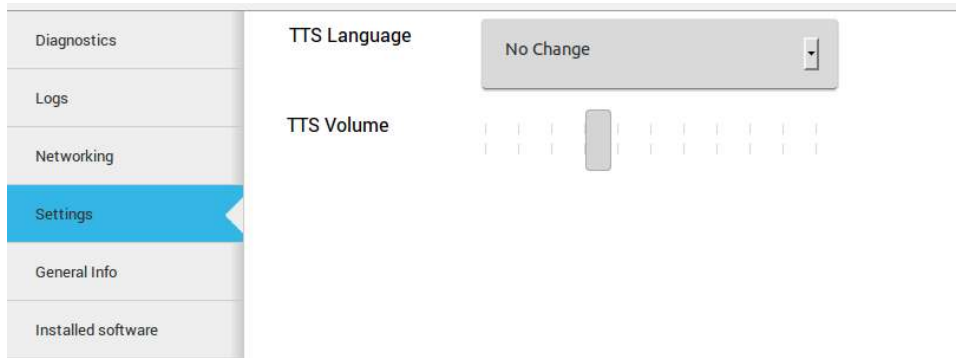


Figure 58: The Settings tab allows to modify the behaviour of REEM-C

21.3.7 Networking Tab

Plugin Networking

Description Enables the configuration of REEM-C's networking .

Figure 59 shows the Networking Tab. By default the controls for changing the configuration are not visible in order to avoid access by multiple users.

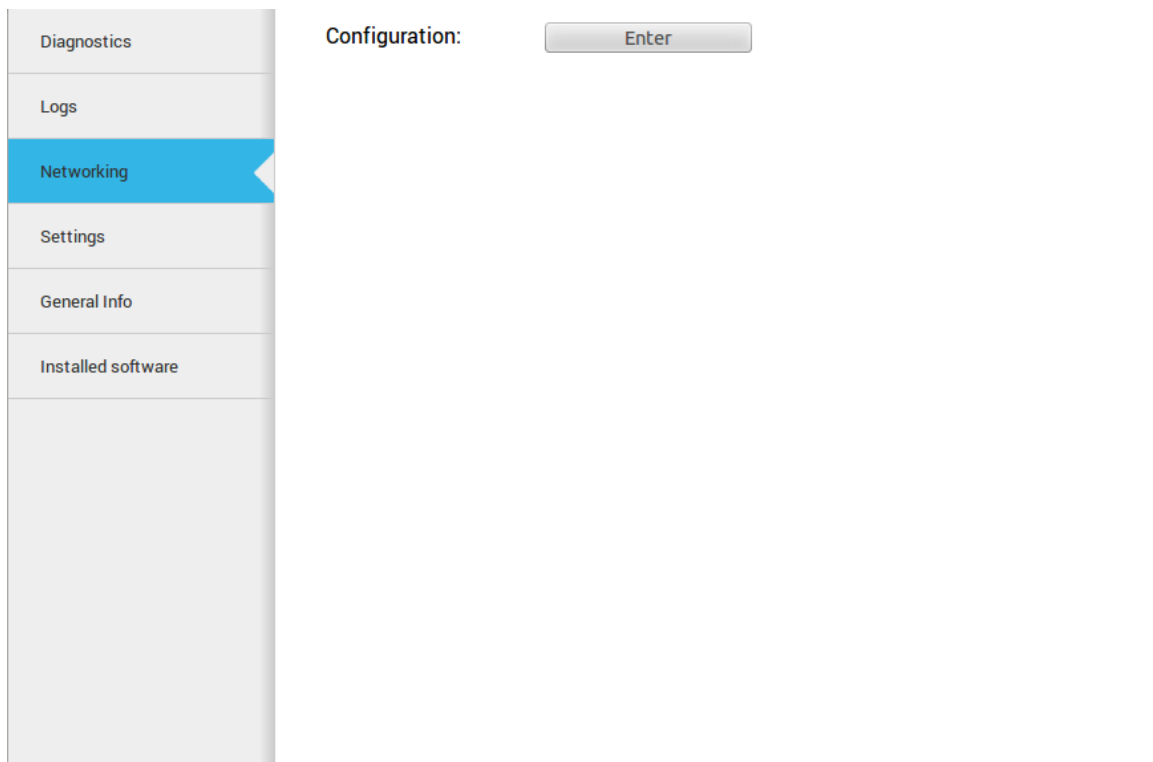


Figure 59: Networking configuration

If *Enter* button is pressed, the Tab connects to the network configuration system and the controls shown in Figure 60 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 has 'Networking' highlighted. The main content area is titled 'Configuration:' and contains the following sections:

- Wifi:**
 - Mode: Client (dropdown)
 - SSID: rlan
 - Band: 5ghz-onlyn
 - Frequency: 5180
 - Country: spain
 - Enable Mode key
 - Wpa:
 - Wpa2:
 - Enable DHCP
 - Address: 192.168.1.237/24
 - Network: 192.168.1.0
 - Gateway: 192.168.1.1
- Ethernet:**
 - Mode: Internal (dropdown)
 - Enable DHCP
 - Address: 192.168.1.211/24
 - Network: 192.168.1.0
 - Gateway: 192.168.1.1
- Vpn:**
 - Address: 192.168.1.6
 - Port: 1194

Buttons: 'Exit' (top right), 'Apply change' (bottom center), 'Save' (bottom left).

Figure 60: Networking configuration controls

Configurations are separated into three blocks:

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

- Wpa2: Password for WPA2 encryption.
 - Enable DHCP: In Client mode, use DHCP to obtain the IP address of the building network.
 - Address, Network, Gateway: In Client mode, manual values of the building network will be used by the Wifi interface.
- Ethernet:
 - Mode: Can be selected whether the Ethernet connection works as an Internal LAN or VPN connection.
 - Enable DHCP: In VPN mode, use DHCP to obtain the IP address of the building network.
 - Address, Network, Gateway: Manual values of the building network to be used by the Ethernet interface (when in VPN mode).
 - VPN
 - Address: Building network IP address of the Basestation.
 - Port: Port of the Basestation where the VPN server is listening.

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

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

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

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

As shown in Figure 61, click on *Components/hardware/Wifi* in the *Diagnostics* tab, to see the current Wifi state.

The screenshot shows the 'Diagnostics' tab selected in a sidebar menu. The main content area displays the following information:

```

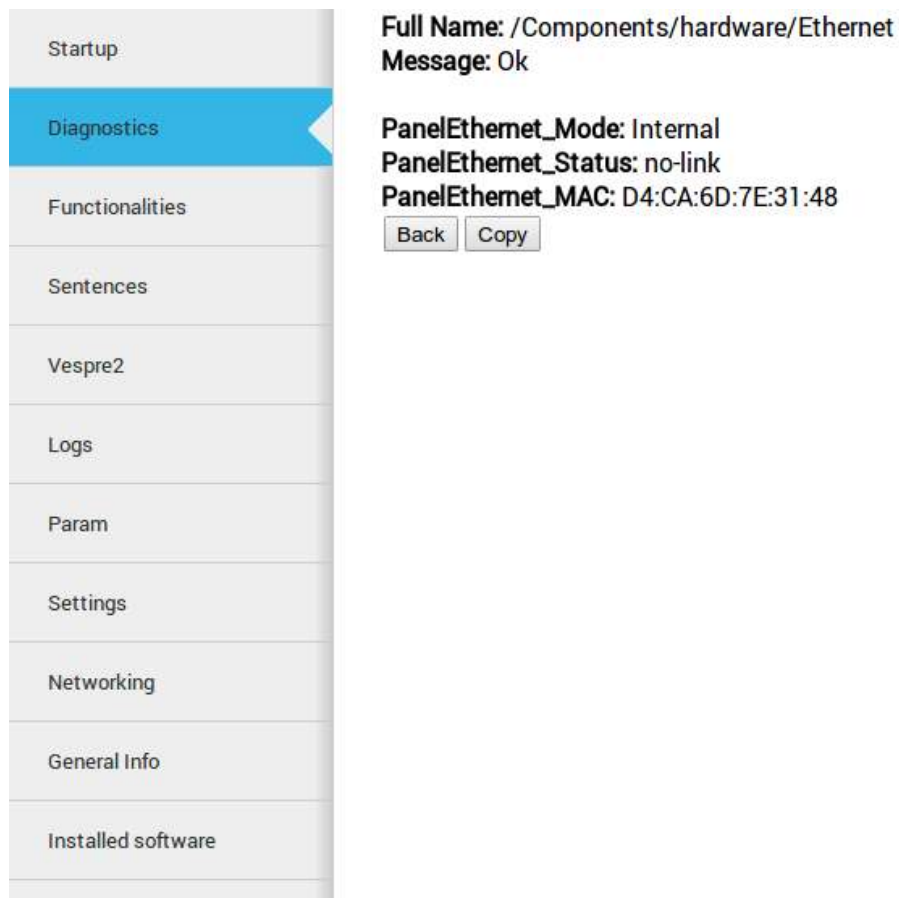
Full Name: /Components/hardware/Wifi
Message: Ok

WifiClient_Status: connected-to-ess
WifiClient_Bssid: 5C:50:15:23:D5:80
WifiClient_Signal: -48dBm
WifiClient_SignalToNoise: 69dB
WifiClient_IP: 192.168.1.237/24
WifiClient_MAC: D4:CA:6D:12:37:D9
  
```

At the bottom of the information block, there are two buttons: 'Back' and 'Copy'.

Figure 61: Networking Wifi diagnostics

And to have information about the state of the Ethernet click on *Components/hardware/Ethernet* (Figure 62).



The screenshot shows the WebCommander interface. On the left is a vertical sidebar with the following menu items: Startup, Diagnostics (highlighted in blue), Functionalities, Sentences, Vespre2, Logs, Param, Settings, Networking, General Info, and Installed software. The main content area on the right displays the following information:

- Full Name:** /Components/hardware/Ethernet
- Message:** Ok
- PanelEthernet_Mode:** Internal
- PanelEthernet_Status:** no-link
- PanelEthernet_MAC:** D4:CA:6D:7E:31:48

At the bottom of the main content area, there are two buttons: "Back" and "Copy".

Figure 62: Networking Ethernet diagnostics

21.3.8 Video Tab

Plugin Video

Description Displays the images from a ROS topic in the WebCommander

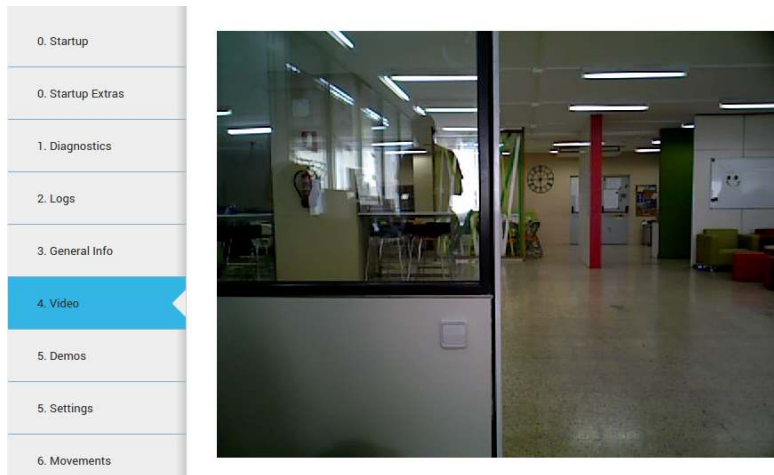


Figure 63: The Video Tab displays live video stream from the robot's camera

21.3.9 Movements Tab

Plugin Movements

Description Enables playing pre-recorded motions on REEM-C .

The movement tab that can be seen in figure 64 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.

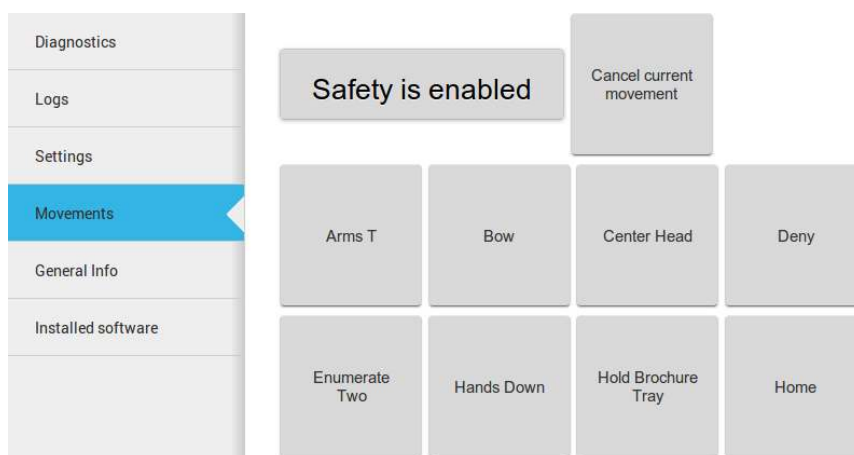


Figure 64: The Movement tab allows to send upper body motions to REEM-C

21.4 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.

21.4.1 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.

21.4.2 Startup Plugin Configuration

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

Parameters

startup_ids A list of strings that contains the startup groups handled the instance of the plugin. See section 30.1.3 on page 139.

21.4.3 Diagnostics Plugin Configuration

Description Displays the current status of REEM-C's hardware and software.

Parameters None required

21.4.4 Logs Plugin Configuration

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

Parameters None required

21.4.5 General Info Plugin Configuration

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

Parameters None required

21.4.6 Installed Software Plugin Configuration

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

Parameters None required

21.4.7 Settings Plugin Configuration

Description The settings tab allows to change the behaviour of REEM-C .

Parameters None required

21.4.8 Networking Plugin Configuration

Description This tab allows to change the network configuration.

Parameters None required

21.4.9 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: `/stereo/left/image/compressed`

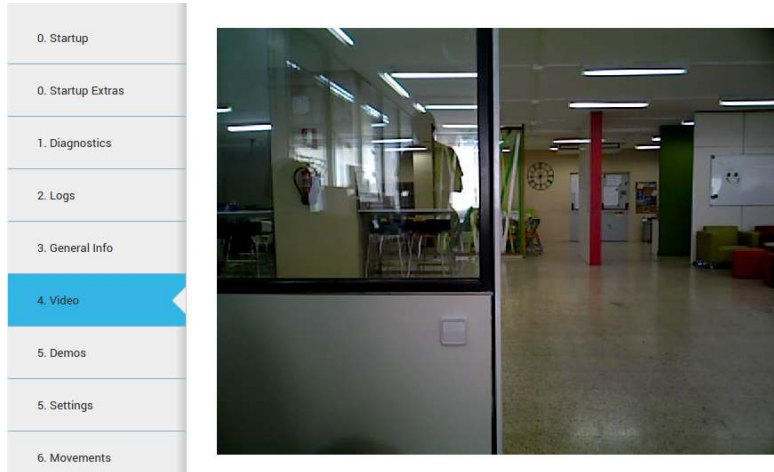


Figure 65: The Video Tab displays live video stream from the robot's camera

21.4.10 Movements Plugin Configuration

Description Enables playing pre-recorded motions on REEM-C .

Parameters None required

21.4.11 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. 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.

22 Camera publishing

22.1 RGBD camera

The following command will stream all the camera sensors and publish the appropriate ROS topics.

```
roslaunch realsense2_camera rs_camera.launch
```

23 Camera subscription

In this section different ways to get the images and the intrinsic parameters out of the ROS topics of the cameras of REEM-C are explained.

23.1 Command line

23.1.1 Image visualization

A simple way to visualize images of the REEM-C cameras is to use the ROS command for this purpose. First open a console to make the environment variable `ROS_MASTER_URI` point to REEM-C.

```
export ROS_MASTER_URI=http://reemc-6c:11311
```

RGB-D camera To visualize the RGB image

```
roslaunch image_view image_view image:=/rgbd/rgb/image_raw _image_transport:=compressed
```

and to visualize the depth image

```
roslaunch image_view image_view image:=/rgbd/rgb/image _image_transport:=compressedDepth
```

Another possibility is to visualize the image in Rviz. The 66 figure shows a Rviz visualization of the depth camera.

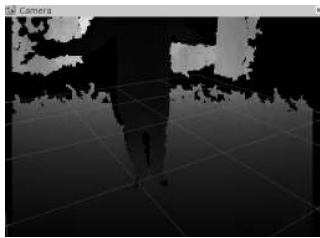


Figure 66: rviz depth camera widget

23.2 C++ API

23.2.1 ROS subscribers

The standard way to get images from a ROS camera is to use the `image_transport::Subscriber` documented in

http://www.ros.org/doc/api/image_transport/html/classimage__transport_1_1Subscriber.html

or the `image_transport::CameraSubscriber` documented in

http://www.ros.org/doc/api/image_transport/html/classimage__transport_1_1CameraSubscriber.html

which not only provides images but also the intrinsic parameters of the camera.

A tutorial of how to use these APIs can be found at

http://www.ros.org/wiki/image_transport/Tutorials/SubscribingToImages.

23.2.2 pal_camera_client package

This package provides a class that wraps all the ROS subscribers and callbacks required to get on demand images and camera intrinsic parameters from the corresponding ROS topics.

The documentation regarding the C++ API of the pal_camera_client package is in doxygen format and is accessible in [./doxygen/pal_camera_client/doc/html/index.html](http://doxygen/pal_camera_client/doc/html/index.html).

24 Sensors

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

24.1 Description of sensors

1. Feet force torque sensors
REEM-C has 6 axis force torque sensor on each ankle, before the foot sole. These sensors measure ground reaction forces and torques.
2. Laser sensors (optional)
There is one laser scanner on each foot. These sensors measure distances in a horizontal plane. They are valuable assets for navigation and mapping. Note that reflective or transparent surfaces can result in poor measurements.
3. Wrists force torque sensors
The robot features six axis force torque sensors mounted on each wrist, just before the hand. They can measure interaction forces and torques with the environment or with a human interacting with the end effectors.
4. Inertial measurement unit (IMU)
This sensor unit is mounted in the waist of REEM-C and provides acceleration, angular rates, magnetic field and attitude estimations.
5. Cameras
For documentation about camera specifications go to Section 3.1; for a description about how to access camera images go to Section 22.

The locations of the sensors are depicted in Figure 67.

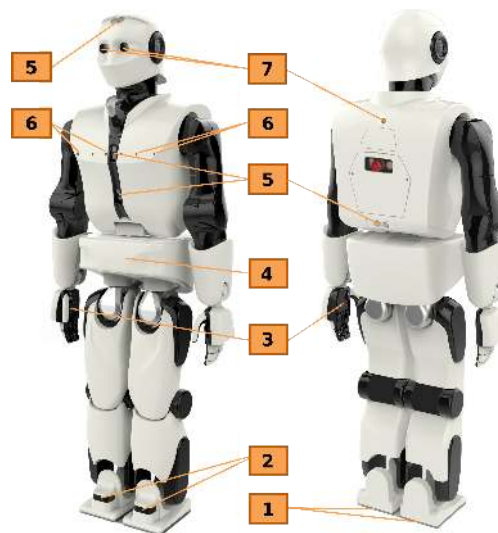


Figure 67: Sensors location

24.2 ROS API

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

24.2.1 Published topics

`/left_scan` ([sensor_msgs/LaserScan](#))

Laser scan data of the laser scanner on the left foot (optional).

`/right_scan` ([sensor_msgs/LaserScan](#))

Laser scan data of the laser scanner on the right foot (optional).

`/sonar_torso` ([sensor_msgs/Range](#))

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

`/left_ankle_ft` ([geometry_msgs/WrenchStamped](#))

Force torque sensor data from the left sole.

`/right_ankle_ft` ([geometry_msgs/WrenchStamped](#))

Force torque sensor data from the right sole.

`/left_wrist_ft` ([geometry_msgs/WrenchStamped](#))

Force torque sensor data from the left wrist.

`/right_wrist_ft` ([geometry_msgs/WrenchStamped](#))

Force torque sensor data from the right wrist.

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

Inertial data from inside the waist of REEM-C .

`/diagnostics` ([diagnostic_msgs/DiagnosticArray](#))

State of the hardware and software of REEM-C. There are different tools for visualization, [rqt_robot_monitor](#), [rqt_runtime_monitor](#) or the Diagnostic Tab of the WebCommander (see 21.4.3 on page 105). For details about the diagnostics system see <http://wiki.ros.org/diagnostics>.

`/joint_states` ([sensor_msgs/JointState](#))

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

25 Walking

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

25.1 Walking_controller

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

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

25.1.1 Action API

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

In Section 56.4 a tutorial for using the action interface with the walking algorithm is provided.

Action subscribed topics

`/walking_controller/footsteps_execution/goal` ([humanoid_nav_msgs/ExecFootstepsActionGoal](#))

A goal with the list of footsteps to be performed.

`/walking_controller/footsteps_execution/cancel` ([actionlib_msgs/GoalID](#))

A request to cancel a specific goal.

Action published topics

`/walking_controller/footsteps_execution/feedback`
([humanoid_nav_msgs/ExecFootstepsActionFeedback](#))

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

`/walking_controller/footsteps_execution/result` ([humanoid_nav_msgs/ExecFootstepsActionResult](#))

The result is the list of footsteps finally performed by REEM-C .

`/walking_controller/footsteps_execution/status` ([actionlib_msgs/GoalStatusArray](#))

Report the status of the action server.

25.1.2 Topic API

`/walking_controller/cmd_vel` ([geometry_msgs/Twist Message](#))

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

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

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

25.1.3 Published topics

`/walking_controller/walking_status` ([walking_msgs/WalkingStatus](#))

General status of the walking controller

25.1.4 Service API

`/walking_controller/walk_steps` ([walking_msgs/WalkSteps](#))

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

`/walking_controller/do_step` ([humanoid_nav_msgs/StepTargetService](#))

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

25.2 Walking parameters

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

There are three groups of parameters, as shown in Figure 68, that allow to operator to modify the walking behaviour though `rqt_reconfigure`: `biped_controller`, `zmp_solver` and `com_stabilizer` parameters.

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

```
roslaunch rqt_reconfigure rqt_reconfigure
```

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

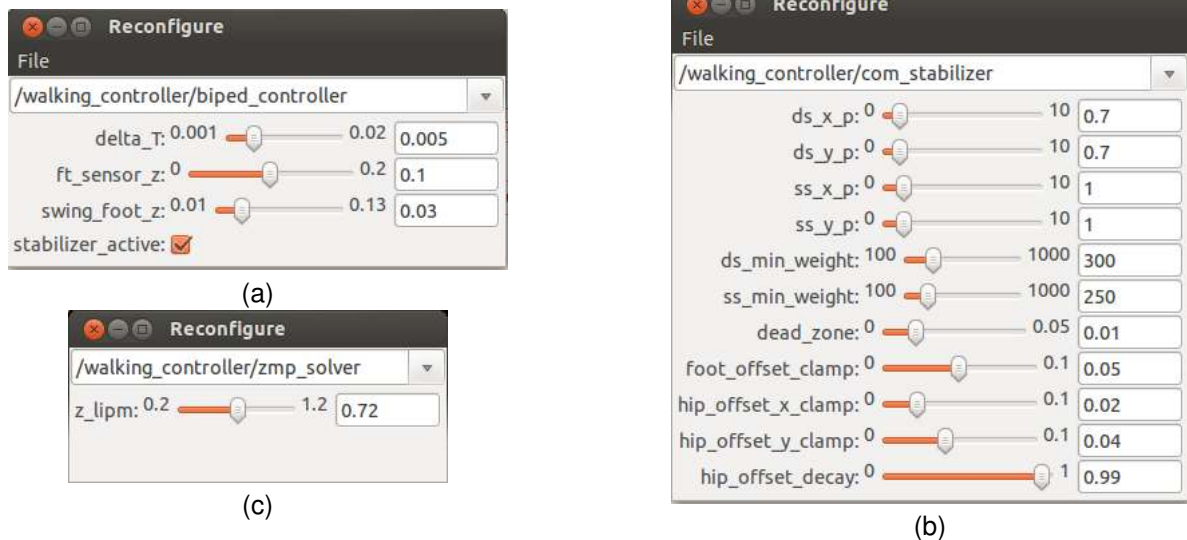


Figure 68: Walking controller parameters: biped_controller (a), com_stabilizer (b), zmp_solver (c).

Biped controller The general parameters for walking specify the height of the swing leg and the status of the stabilizer (enabled or disabled), as shown in Figure 68(a)

`~swing_foot_z` (double, default: 3.5)

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

`~stabilizer_active` (bool, default: true)

Enable/disable stabilizer.

COM Stabilizer This set of parameters allow to tune the walking controller stabilizer in order to add robustness to the walking gait. Figure 68(b)

`~ds_x_p` (double , default: 0.8)

Double support x-axis proportional gain.

`~ds_y_p` (double , default: 0.6)

Double support y-axis proportional gain.

`~ss_x_p` (double , default: 1.0)

Single support x-axis proportional gain.

`~ss_y_p` (double , default: 1.0)

Single support y-axis proportional gain.

`~ds_min_weight` (double , default: 300)

Double support weight threshold for enabling stabilizer.

`~ss_min_weight` (double , default: 250)

Single support weight threshold for enabling stabilizer.

`~dead_zone` (double , default: 0.01)

Dead zone for zmp error tracking.

`~foot_offset_clamp` (double , default: 0.05)

Max offset applied to swing foot by stabilizer.

`~hip_offset_x_clamp` (double , default: 0.02)

Max offset applied to hip x-coord by stabilizer.

`~hip_offset_y_clamp` (double , default: 0.04)

Max offset applied to hip y-coord by stabilizer

`~hip_offset_decay` (double , default: 0.99)

Decay value for hip offset.

Solver The walking controller uses the Linear Inverted Pendulum as a simplified model of the robot's dynamics. The only parameter of the model that can be modified by the user is the height of the pendulum (Figure 68(c)).

`~z_lipm` (double , default: 0.7)

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

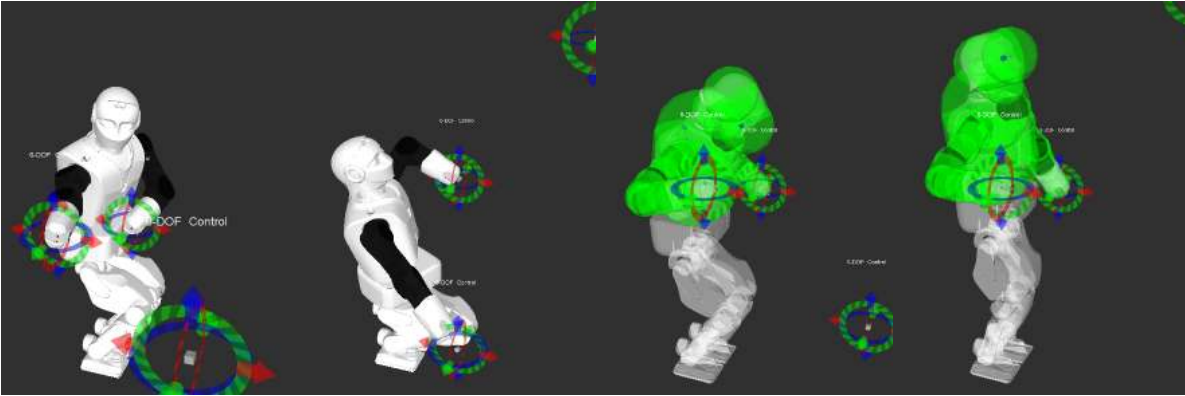


Figure 69:

26 Whole Body Control

In this Section we describe PAL Robotics whole body control framework.

26.1 Overview

The framework is implemented as a `ros_control` plugin that takes control of all the joints of the robot with the exception of the hands. The controller can load a predefined Stack of Tasks. A Stack of Tasks is a strict hierarchization of a list of objectives. This approach to humanoid robot control is very useful as objectives like balancing can always be enforced at a higher priority to the other objectives like manipulation.

The Stacks are configured once at run time during the loading phase of the controller, and thus currently **there is no support for modifying the stack at runtime**. Every new Stack should have its associated configuration file and launch file to run in the robot. In the following sections we will describe how to create and use them.

While the framework supports kinematic and dynamic formulations **only the kinematic formulations are supported**. The state vector is the joint configuration of the robot and the generated command is a joint velocity vector that is integrated in time and sent as a command to the robot.

To account for modeling uncertainties or perturbations the force torque sensors can be used to stabilize the robot using admittance control laws, for example tracking the actual center of pressure with the center of mass in order to avoid generating a tipping moment. All the launch files that use the force torque sensors to stabilize the robot have `ft_imu` in their name.

26.2 Task

All the tasks that are supported inherit from the `TaskAbstract` base class, and they must implement the following functions:

`setUpTask` This function receives a pointer to the Stack to retrieve information of the problem and the robot, such as the formulation type, number of degrees of freedom and the robot model. This function is called from a non Real Time thread and all the allocations should be done inside of it. It is the responsibility of the user to call this function after creating the new task.

`update` This function is called periodically at every control cycle by the solver. It receives the actual joint configuration from the controller.

To help the user configure a Stack there are specializations of tasks called MetaTasks. A MetaTask is an instantiation of a Task with a particular configuration, for example the GoTo task, which allows to control the six degrees of freedom, has a MetaTaskPosition to control only the cartesian position. MetaTasks don't need to be `setUp` since they themselves call the `setUp` function of the Task they inherit from.

There is a special task called `GenericMetatask` that is used to create new Tasks that are a combination of existing Tasks. It is useful to run different tasks at the same level in the hierarchy, the resultant objective is a weighted average of the objectives of the individual Tasks that compose it.

All the tasks that receive a reference goal, can get the desired goals through **interactive_markers** or **topics**. **Interactive markers are not real time safe**, only use them for testing purposes in simulation.

Damping is very important to avoid numerical instabilities, those tasks that might become singular (kinematic or algorithmic singularities) must have an added regularization term. The amount of regularization for every task can be set using the `GenericMetatask` method. The `setDamping` method should be called in the `GenericMetatask` if it contains a task that can become singular.

26.2.1 Center of mass

Relates joint velocities to the center of mass velocity of the robot. It is used to restrict the ground projection of the center of mass to stay in the middle of the ankles of the robot.

26.2.2 GoTo

Relates joint velocity to the cartesian velocity of a link. Given the actual position of a link and its desired one, a velocity target is created with a proportional controller to reach the desired target.

26.2.3 Fixed Constraint

This task has the same form as the previous one, but the target is zero, meaning that the link should have zero velocity and thus stay fixed.

26.2.4 Gaze

This task maps the velocity of a 3d point to its 2d image velocity. Computing the error between the 2d projection of a 3d point and the origin of the camera frame we can make the robot look directly at the desired point with a target proportional to the error.

26.2.5 Self collision

Given a simplification of the robot, where every link is fitted with a minimum volume capsule, we can avoid self collision by adding an inequality to the relative velocity between the capsules of two links. The relative velocity has to be sufficiently small to avoid penetration.

26.2.6 Joint Limit Task

It can avoid joint limits by restricting the maximum allowed velocity proportional to how far we are from the limits.

26.3 Stack

Every stack should have a *Yaml* configuration file and *launch file* for convenience to launch it. The stack class must implement the `setupStack` method and inherit from the base class `StackConfigurationKinematic`

26.3.1 Parameters in Yaml file

`~type` (string, default: "wbc/WholeBodyControlKinematicController")

The class type of the whole body controller (only the kinematic controller is supported).

`~floating_base` (string, default: "true")

Indicated to add six more degrees of freedom to the robot to model the floating base.

`~formulation` (string, default: "velocity")

The formulation of the problem, only velocity is supported.

`~stack_configuration`

Name of the class that describes the stack that needs to be loaded. The stack must be registered as a plugin

`~solver` (double, default: "QpReductionuEqualitiesQuadprogHeapAllocation")

Name of the hierarchical quadratic solver, only QpReductionuEqualitiesQuadprogHeapAllocation is supported".

`~robot_model_chains`

Vector specifying the sub chains that will be used instead of the full robot model. Its useful to avoid loading the joints of the hands in the whole body controller

`~default_configuration`

Vector that specifies the initial configuration of the joints. The unspecified joints will be initialized to zero

26.3.2 Launch file

The required components that should be in the launch file are show in the following example:

```

1 <launch>
2 <!-- Upload capsule collision operations & description -->
3 <include file="$(find capsule_collision)/launch/reemc_capsule_description.launch" />
4 <!-- Configure wbc -->
5 <rosparam command="load" file="$(find reemc_wbc)/config/reemc_wbc.yaml" />
6 <!-- Floating base transform publisher -->
7 <node name="floating_base_publisher"
8   pkg="pal_wbc_controller" type="floating_base_publisher" />
9 <!-- Spawn controller -->
10 <node name="wbc_spawner"
11   pkg="controller_manager" type="spawner" output="screen"
12   args="whole_body_kinematic_controller" />
13
14 </launch>

```

Listing 12: Example launchfile to launch a stack of tasks.

27 Joint Trajectory Controller

27.1 Overview

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

27.1.1 Trajectory representation

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

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

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

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

27.1.2 Hardware interface type

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

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

27.1.3 Other features

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

27.2 Sending trajectories

27.2.1 Available interfaces

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

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

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

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

27.2.2 Preemption policy

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

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

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

27.2.3 Trajectory replacement

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

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

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

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

The following examples describe this behavior in detail.

27.2.4 Trajectory replacement example: basics

This example, illustrated in Figure 70a shows a joint which is in hold position mode (flat grey line labeled `pos hold`). A new trajectory (shown in red) arrives at the current time (`now`), which contains three waypoints and a start time in the future (`traj start`). The time at which waypoints should be reached (`time_from_start` member of the `trajectory_msgs/JointTrajectoryPoint` message) is relative to the trajectory start time.

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

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

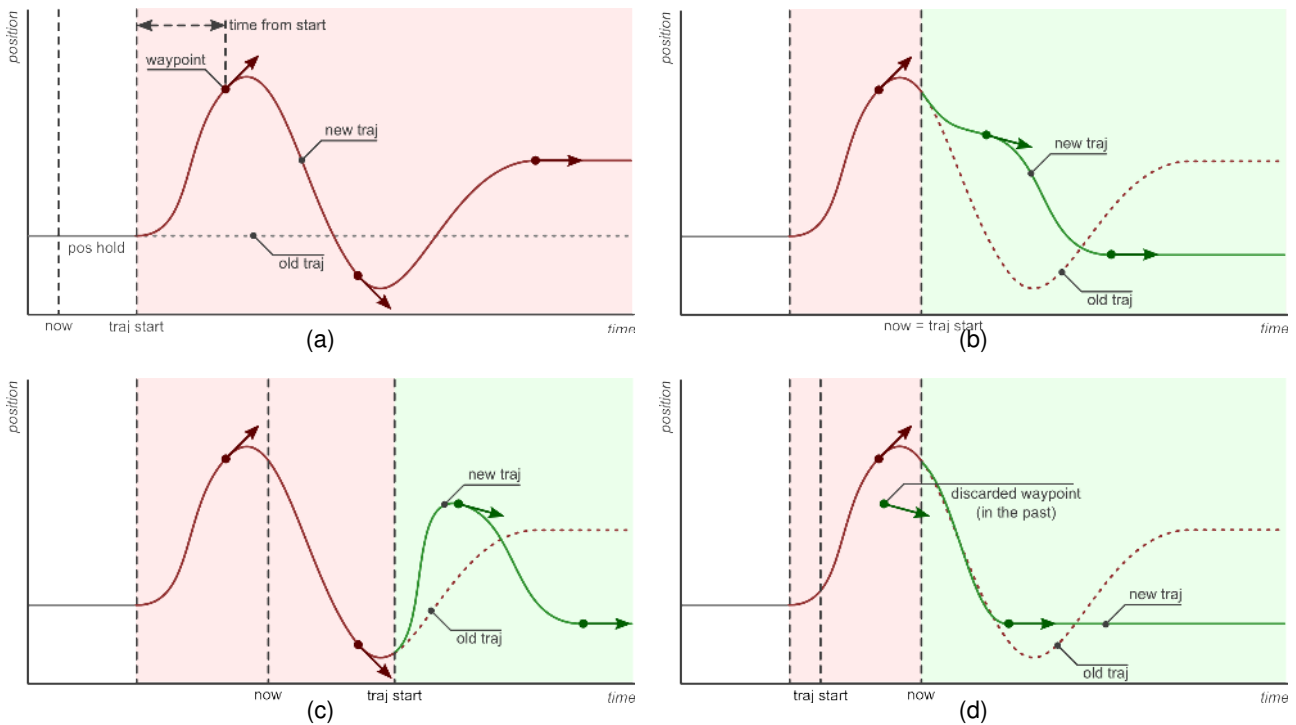


Figure 70: Trajectory replacement. Details of sending a new trajectory (a). Effects of trajectory start time set to: now (b), in the future (c), and in the past (d).

27.2.5 Trajectory replacement example: effects of trajectory start time

This example describes the effect of sending the same trajectory to the controller with different start times. The scenario is that of a controller executing the trajectory from the previous example (shown in red), and receiving a new command (shown in green) with a trajectory start time set to either zero (start now, Figure 70b), a future time (Figure 70c), or a time in the past (Figure 70d).

Of special interest is Figure 70d, where the new trajectory start time and first waypoint are in the past (before `now`). In this case, the first waypoint is discarded and only the second one is realized.

27.3 ROS API

27.3.1 Action interface

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

27.3.2 Subscribed topics

command (`trajectory_msgs/JointTrajectory`)

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

27.3.3 Published topics

state (`control_msgs/JointTrajectoryControllerState`)

Current controller state.

27.3.4 Services

`query_state` ([control_msgs/QueryTrajectoryState](#))

Query controller state at any future time.

27.3.5 Parameters

`joints` (`string[]`)

The list of joints to control.

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

If the timestamp of the goal trajectory point is t , then following the trajectory succeeds if it reaches the goal within $t \pm \text{goal_time}$, and aborts otherwise.

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

Velocity to be considered approximately equal to zero.

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

Position tolerance for a particular joint to reach the goal. When the joint is within `goal_position` \pm `goal_tolerance`, then the trajectory can succeed.

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

Position tolerance for a particular joint throughout the trajectory. If the joint position ever falls outside `trajectory_position` \pm `tolerance`, then the trajectory aborts.

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

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

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

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

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

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

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

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

28 ros_control

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

28.1 Introduction to ros_control

`ros_control` is a consistent interface used to access the actuators and sensors of a robot. It presents the RAW data that comes from hardware components to the various controllers in an organised and intuitive way. It also has the function of being a resource handler for all sensors and actuators, allowing different controllers to access the same device simultaneously using custom policies. Controllers are implemented as plugins that can be loaded and unloaded dynamically at runtime. Since `ros_control` is an abstraction layer for hardware, it allows the operator to write controllers that are robot agnostic, and can be reused in different types of robots with different physical configurations. This is one of the features that makes it possible to write controllers that are 100% compatible between the `gazebo` simulator and the real robot. Figure 71 shows the architecture of the system.

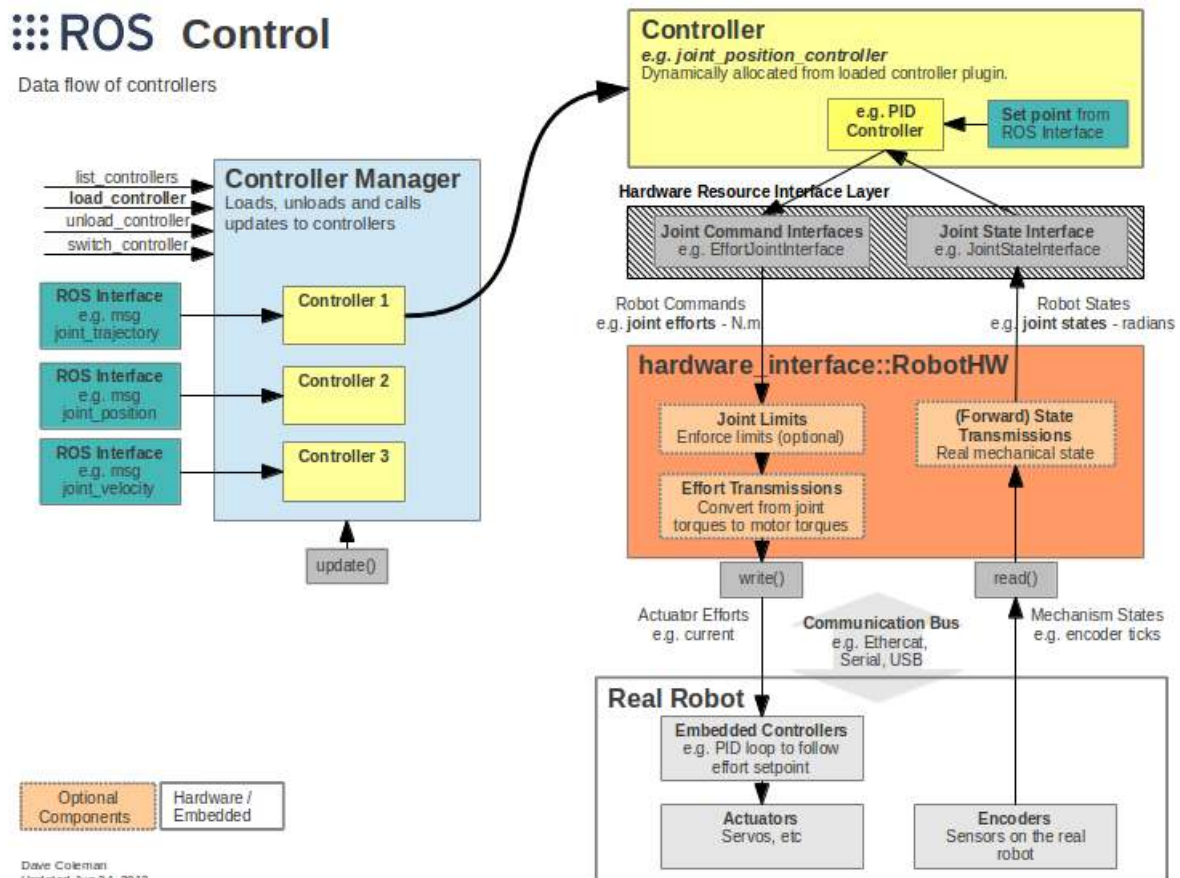


Figure 71: Data flow of controllers

The main hub of `ros_control` is the `controller_manager`. The controller manager exposes its interface through ROS services. It's responsible for managing all the controllers and resources of REEM-C, and triggering events. For every robot, a robot hardware interface must be created. This interface abstracts the real

robot's hardware to the system by exposing standard interfaces to the custom hardware. We provide two robot hardware interfaces, one for the real REEM-C robot and one for the simulated robot.

A controller can have states on two levels based on

1. Presence

- (a) loaded: All controller-specific configurations are loaded in the parameter server and `ros_control` has created an instance of the controller that is in initialised state.
- (b) not loaded: `ros_control` has knowledge of the controller type, but no instance of the controller was created.

2. Activity

- (a) running: The update function of the controller is called at every control cycle.
- (b) stopped: The controller is initialized but is not performing any action.

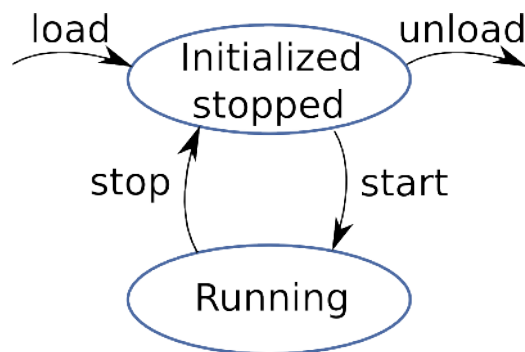


Figure 72: Controller states

28.1.1 How to load and unload controllers

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

28.1.2 How to start and stop controllers

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

28.2 How to create a custom controller

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

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

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

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

REQUIREMENTS TO CREATE A CONTROLLER

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

```

1 <class name="ros_controllers_tutorials/HelloController"
2 type=" ros_controllers_tutorials :: HelloController"
3   base_class_type="controller_interface::ControllerBase">
4   <description>
5     The HelloController does nothing useful.
6     It's only a demonstrational controller that prints a hello message.
7   </description>
8 </class>

```

Listing 13: The HelloController is a skeleton controller with a single `ROS_INFO` in the update function.

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

```

1 hello_controller :
2   type: " ros_controllers_tutorials / HelloController"

```

Listing 14: The HelloController is a skeleton controller with a single `ROS_INFO` in the update function.

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

28.2.1 Hello controller

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

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

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

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

Source code:

```

1 #include <controller_interface / controller .h>
2 #include <hardware_interface/joint_state_interface .h>
3 #include <pluginlib /class_list_macros.h>
4
5 namespace ros_controllers_tutorials{
6
7 class HelloController
8   : public controller_interface :: Controller<hardware_interface::JointStateInterface>
9   {
10  public:
11    bool init (hardware_interface::JointStateInterface* hw, ros::NodeHandle &n)
12    {
13      return true;
14    }
15    void update(const ros::Time& time, const ros::Duration& period)
16    {
17      // WARNING! This will not be realtime-safe
18      ROS_INFO_NAMED("hello_controller", "Hello ros_control!");
19    }
20    void starting (const ros::Time& time) { }
21    void stopping(const ros::Time& time) { }
22  private:
23  };
24  PLUGINLIB_DECLARE_CLASS(ros_controllers_tutorials,
25                          HelloController,
26                          ros_controllers_tutorials :: HelloController,
27                          controller_interface :: ControllerBase);
28 }

```

Listing 15: The HelloController is a skeleton controller with a single ROS_INFO in the update function.

28.2.2 Moving a position controlled joint

Source code:

```

1 #include <controller_interface / controller .h>
2 #include <hardware_interface/joint_command_interface.h>
3 #include <pluginlib /class_list_macros.h>
4 namespace ros_controllers_tutorials{
5   class JointController : public
6     controller_interface :: Controller<hardware_interface::PositionJointInterface >
7   {
8   public:
9     bool init (hardware_interface::PositionJointInterface * hw, ros::NodeHandle &n)
10    {
11      cont = 0;
12      controlled_joint_name_ = "head_2_joint";
13      ROS_INFO_STREAM("LOADING JOINT CONTROLLER ");
14      // Get a joint handle
15      try
16      {
17        joint_ = hw->getHandle(controlled_joint_name_);
18      // throws on failure
19        ROS_INFO_STREAM("Found joint " << controlled_joint_name_);
20      }
21      catch (...) {
22        ROS_ERROR_STREAM("Could not find joint " << controlled_joint_name_);
23        return false;
24      }
25      return true;
26    }
27    void update(const ros::Time& time, const ros::Duration& period)
28    {
29      // Move the head using a sine wave
30      double joint_comand = 0.2*sin(cont/1000.0);
31      joint_ .setCommand(joint_comand);
32      cont = cont + 1;
33    }

```

```

34 void starting(const ros::Time& time) { }
35 void stopping(const ros::Time& time) { }
36 private:
37 hardware_interface::JointHandle joint_;
38 std::string controlled_joint_name_;
39 double cont;
40 };
41 PLUGINLIB_DECLARE_CLASS(ros_controllers_tutorials,
42 JointController,
43 ros_controllers_tutorials::JointController,
44 controller_interface::ControllerBase);
45 }

```

Listing 16: The JointController is a controller that moves a single joint in position control

28.2.3 Inertial measurement unit (IMU)

```

1 #include <controller_interface/controller.h>
2 #include <hardware_interface/joint_state_interface.h>
3 #include <pluginlib/class_list_macros.h>
4 #include <realtime_tools/realtime_buffer.h>
5 #include <realtime_tools/realtime_publisher.h>
6 #include <boost/assign.hpp>
7 #include <hardware_interface/imu_sensor_interface.h>
8 #include <sensor_msgs/Imu.h>
9
10 namespace ros_controllers_tutorials{
11 class ImuController :
12 public controller_interface::Controller<hardware_interface::ImuSensorInterface> {
13 public:
14 bool init(hardware_interface::ImuSensorInterface* hw, ros::NodeHandle &n) {
15 // get all imu sensor names
16 const std::vector<std::string>& sensor_names = hw->getNames();
17 for (unsigned i=0; i<sensor_names.size(); i++)
18 ROS_INFO("Got sensor %s", sensor_names[i].c_str());
19 for (unsigned i=0; i<sensor_names.size(); i++){
20 // sensor handle
21 sensor_ = hw->getHandle(sensor_names[i]);
22 }
23 return true;
24 }
25 void update(const ros::Time& time, const ros::Duration& period)
26 {
27 using namespace hardware_interface;
28 sensor_msgs::Imu value;
29 // Orientation
30 if (sensor_.getOrientation())
31 {
32 value.orientation.x = sensor_.getOrientation()[0];
33 value.orientation.y = sensor_.getOrientation()[1];
34 value.orientation.z = sensor_.getOrientation()[2];
35 value.orientation.w = sensor_.getOrientation()[3];
36 }
37 ROS_INFO_STREAM("IMU orientation x: "<<value.orientation.x<<" y:
38 "<<value.orientation.y<<" z: "<<value.orientation.z<<"
39 w: "<<value.orientation.w);
40 }
41 void starting(const ros::Time& time) { }
42 void stopping(const ros::Time& time) { }
43 private:
44 hardware_interface::ImuSensorHandle sensor_;
45 };
46 PLUGINLIB_DECLARE_CLASS(ros_controllers_tutorials,
47 ImuController,
48 ros_controllers_tutorials::ImuController,
49 controller_interface::ControllerBase);
50 }

```

Listing 17: The ImuController is a controller that subscribes and print the value of the IMU.

28.2.4 Force torque sensors

```

1 #include <controller_interface/controller.h>
2 #include <hardware_interface/joint_state_interface.h>
3 #include <pluginlib/class_list_macros.h>
4 #include <geometry_msgs/Wrench.h>
5 #include <realtime_tools/realtime_buffer.h>
6 #include <realtime_tools/realtime_publisher.h>
7 #include <boost/assign.hpp>
8 #include <hardware_interface/force_torque_sensor_interface.h>
9
10 namespace ros_controllers_tutorials{
11
12 class ForceTorqueController :
13 public controller_interface::Controller<hardware_interface::JointStateInterface>
14 {
15 public:
16 bool init (hardware_interface::ForceTorqueSensorInterface* hw, ros::NodeHandle &n)
17 {
18 // get force torque sensors names package.
19 const std::vector<std::string>& sensor_names = hw->getNames();
20 for (unsigned i=0; i<sensor_names.size(); i++)
21 ROS_INFO("Got sensor %s", sensor_names[i].c_str());
22 for (unsigned i=0; i<sensor_names.size(); i++){
23 // sensor handle
24 sensors_.push_back(hw->getHandle(sensor_names[i]));
25 }
26 return true;
27 }
28 void update(const ros::Time& time, const ros::Duration& period)
29 {
30 geometry_msgs::Wrench ft[2];
31 for (unsigned int s = 0; s<2; ++s){
32 ft[s].force.x = sensors_[s].getForce()[0];
33 ft[s].force.y = sensors_[s].getForce()[1];
34 ft[s].force.z = sensors_[s].getForce()[2];
35 ft[s].torque.x = sensors_[s].getTorque()[0];
36 ft[s].torque.y = sensors_[s].getTorque()[1];
37 ft[s].torque.z = sensors_[s].getTorque()[2];
38 }
39 ROS_INFO_STREAM("Force sensor left: fx = "<<
40 ft[0].force.x<<" fy: "<<
41 ft[0].force.y<<" fz: "<<
42 ft[0].force.z<<" tx: "<<
43 ft[0].torque.x<<" ty: "<<
44 ft[0].torque.y<<" tz: "<<
45 ft[0].torque.z);
46 ROS_INFO_STREAM("Force sensor right: fx = "<<
47 ft[1].force.x<<" fy: "<<
48 ft[1].force.y<<" fz: "<<
49 ft[1].force.z<<" tx: "<<
50 ft[1].torque.x<<" ty: "<<
51 ft[1].torque.y<<" tz: "<<
52 ft[1].torque.z);
53 }
54 void starting(const ros::Time& time) { }
55 void stopping(const ros::Time& time) { }
56 private:
57 std::vector<hardware_interface::ForceTorqueSensorHandle> sensors_;
58 };
59 PLUGINLIB_DECLARE_CLASS(ros_controllers_tutorials,
60 ForceTorqueController,
61 ros_controllers_tutorials::ForceTorqueController,
62 controller_interface::ControllerBase);
63 }

```

Listing 18: The ForceTorqueController is a controller that subscribes to the FT sensors and prints its values

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

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

Source code:

```

1 namespace ros_controllers_tutorials{
2 class CombinedResourceController : public controller_interface::ControllerBase
3 { public:
4   bool initRequest(hardware_interface::RobotHW* robot_hw,
5                   ros::NodeHandle&      root_nh,
6                   ros::NodeHandle &    controller_nh,
7                   std::set<std::string>& claimed_resources)
8   {
9     // Check if construction finished cleanly
10    if (state_ != CONSTRUCTED)
11    {
12      ROS_ERROR("Cannot initialize this controller because it
13                failed to be constructed");
14      return false;
15    }
16    // Get a pointer to the joint position control interface
17    PositionJointInterface* pos_iface = robot_hw->get<PositionJointInterface>();
18    if (!pos_iface)
19    {
20      ROS_ERROR("This controller requires a hardware interface of type '%s'."
21                " Make sure this is registered in the
22                hardware_interface::RobotHW class.",
23                getHardwareInterfaceType().c_str());
24      return false;
25    }
26    // Get a pointer to the force-torque sensor interface
27    ForceTorqueSensorInterface* ft_iface =
28      robot_hw->get<ForceTorqueSensorInterface>();
29    if (!ft_iface)
30    {
31      ROS_ERROR("This controller requires a hardware interface
32                of type '%s'." " Make sure this is registered in
33                the hardware_interface::RobotHW class.",
34                internal::demangledTypeName<ForceTorqueSensorInterface>().c_str());
35      return false;
36    }
37    // Get a pointer to the IMU sensor interface
38    ImuSensorInterface* imu_iface = robot_hw->get<ImuSensorInterface>();
39    if (!imu_iface)
40    {
41      ROS_ERROR("This controller requires a hardware interface of type '%s'."
42                " Make sure this is registered in the
43                hardware_interface::RobotHW class.",
44                internal::demangledTypeName<ImuSensorInterface>().c_str());
45      return false;
46    }
47    // Return which resources are claimed by this controller
48    pos_iface->clearClaims();
49    if (!init (pos_iface,
50              ft_iface,
51              imu_iface,
52              root_nh,
53              controller_nh))
54    {
55      ROS_ERROR("Failed to initialize the controller");
56      std::cerr << "FAILED LOADING WALKING" << std::endl;
57      return false;
58    }
59    claimed_resources = pos_iface->getClaims();
60    pos_iface->clearClaims();
61    // success
62    state_ = INITIALIZED;

```



```

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

```

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

Listing 19: The CombinedResourceController is a controller that subscribes to different sensors and prints its values.

Joint name	Torque constant	Reducer ratio
arm_right_1_joint	0.1358	100.0
arm_right_2_joint	0.0869	100.0
arm_right_3_joint	0.0869	80.0
arm_right_4_joint	0.0869	80.0
arm_left_1_joint	0.1358	100.0
arm_left_2_joint	0.0869	100.0
arm_left_3_joint	0.0869	80.0
arm_left_4_joint	0.0869	80.0
leg_right_1_joint	0.098	133.6
leg_right_2_joint	0.106	160
leg_right_3_joint	0.18	64
leg_right_4_joint	0.134	100.0
leg_right_5_joint	0.106	96
leg_right_6_joint	0.18	160.0
leg_left_1_joint	0.098	133.6
leg_left_2_joint	0.106	160
leg_left_3_joint	0.18	64
leg_left_4_joint	0.134	100
leg_left_5_joint	0.106	96
leg_left_6_joint	0.18	160.0
torso_1_joint	0.1358	100.0
torso_2_joint	0.1358	100.0

Table 17: REEM-C motor parameters

28.3 Controlling REEM-C in effort

`ros_control` supports writing controllers that command the robot in position control mode or effort control mode. Once a controller is switched on, `ros_control` will change automatically the control mode of the joints that is claiming to the mode needed by the controller.

Effort control implies that we are actually sending current commands to the motors, in order to map them to desired torques a transformation has to be done to change from desired torque to desired motor current. The following tables contain the reducer ratio of every motor and its torque constant already multiplied by the reducer. The mapping from torque to current is the inverse of the torque constant parameter.

REEM-C supports effort control in all its joints with the exception of the head, wrist and hand joints.

This package `force_control_reemc` has an example controller to do gravity compensation with REEM-C. Two different controller configurations are available, one for compensating gravity in all the joints of the robot and one that only does gravity compensation on the upper body.

Full body gravity compensation (*ATTENTION! This controller does not implement contact dynamics and thus the robot will collapse due to its own weight. The robot must be hanging in the air before turning on this controller*)

```
roslaunch force_control_reemc full_body_gravity_compensation.launch
```

Upper body gravity compensation

```
roslaunch force_control_reemc upper_body_gravity_compensation.launch
```

This two controllers have a duplicated launch file with the underscore `_simulation` where there is no torque conversion from desired torque to desired current, be very careful not to use this controllers in the real robot since they would send the wrong untransformed torque commands to robot and can potentially break it.

To switch the joints back to position mode a controller that requires a joint position interface has to be turned on, for example `default_controllers`.

A reference implementation for an effort controller can be found in the package `reemc_effort_control`

29 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.

29.1 Introduction

When REEM-C 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/erbium/`, the other is a fixed location at `/home/pal/deployed_ws`, which is where the `deploy` tool installs to. This location precedes the rest of the software installation, making it possible to overlay previously installed packages.

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

29.2 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. reemc-6c)

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 reemc-6c -u root -p pal_tts -c="-DCMAKE_BUILD_TYPE=Release"
```

29.3 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 overlaid, 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.

29.4 Deploy tips

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

```
alias deploy="roslaunch pal_deploy deploy.py"
```
- You can omit the `--user pal` as it is the default argument
- You may deploy a single specific package instead of the entire workspace:

```
deploy -p hello_world reemc-6c
```
- You can deploy multiple specific packages instead of the entire workspace:

```
deploy -p "hello_world other_local_package more_packages" reemc-6c
```
- 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.

29.5 Use-case example

29.5.1 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/erbium/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 73.

```
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 73: Hello world CMakeLists.txt

Edit the `src/hello_world_node.cpp` file with the contents in figure 74.

```
// 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 74: Hello world C++ source code

Build the workspace

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

The expected output is shown in figure 75.

```
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 75: Build output of hello world package

Deploy the package to the robot:

```
cd ~/example_ws
roslaunch pal_deploy deploy.py --user pal reemc-6c
```

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 76.

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

Profile:                pal_deploy
Extending:              [env] /home/pal/example_ws/devel:/opt/pal/dubnium:/opt/ros/indigo
Workspace:              /home/pal/example_ws
Source Space:          [exists] /home/pal/example_ws/src
Build Space:           [exists] /home/pal/example_ws/build_pal_deploy
Devel Space:          [missing] /home/pal/example_ws/devel_pal_deploy
Install Space:        [missing] /home/pal/example_ws/install_pal_deploy
DESTDIR:               None

-----
Isolate Develspaces:   False
Install Packages:     True
Isolate Installs:     False
-----
Additional CMake Args: -DCATKIN_BUILD_BINARY_PACKAGE=0 -DCMAKE_CXX_FLAGS_DEBUG=-g -O0 -DCMAKE_C_FLAGS_DEBUG=-g -O0 -DCATKIN_ENABLE_TESTING=OFF
LD_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@ttago-0c:/home/pal/deployed_ws
Do it? (Y/n):
```

Figure 76: Deployment of hello world package

Press `Y` so that the package files are installed on the robot computer. Figure 77 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 'ttago-0c (192.168.1.33)' can't be established.
RSA key fingerprint is 4c:0e:17:4c:39:48:f7:af:51:87:62:7d:b0:0b:fb:be.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/pal/.ssh/known_hosts).
pal@ttago-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 77: Installation of the hello world package to the robot

Then connect to the robot:

```
ssh pal@reemc-6c
```

And run the new node as follows:

```
roslaunch hello_world hello_world_node
```

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

29.5.2 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://reemc-6c: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 `reemc_ws` that contains the sources of `ros_controllers_tutorials`, open a new terminal and execute the following commands:

```
cd reemc_ws
catkin_make #-j5 #optional
source devel/setup.bash # to get this workspace into the development environment
roslaunch pal_deploy deploy.py --package ros_controllers_tutorials reemc-6c
```

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://reemc-6c: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/GravityCompensationJointSpacePD', 'walking_force_control/GravityCompensationPD', '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 78: List of controller types

29.5.3 Modifying an installed package

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

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

```
cd reemc_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
roslaunch pal_deploy deploy.py --package joint_state_controller reemc-6c
```

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

30 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.

30.1 Introduction

REEM-C 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.

30.1.1 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 105. 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: []
```

30.1.2 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
```

30.1.3 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.

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

30.2 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 in 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 in 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 in 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 in YAML files for the application start configuration files). Returns the path of the log file of the specified app.

30.3 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.

30.4 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.

30.4.1 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
```

30.4.2 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.

31 Motion planning with *MoveIt!*

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

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

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

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

1. Start a simulation with default controllers.

To launch a simulation in one terminal, execute:

```
roslaunch reemc_gazebo reemc_gazebo.launch
```

And in another terminal, launch the default controllers.

```
roslaunch reemc_controller_configuration_gazebo default_controllers.launch
```

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

```
roslaunch reemc_moveit_config moveit_rviz.launch
```

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

1. The GUI is RViz, with a custom plugin for executing and visualizing motion planning.
2. *MoveIt!* uses planning groups to generate solutions for different kinematic chains. Figure 79 shows that by selecting different planning groups, the GUI shows only the relevant “flying end-effectors”.

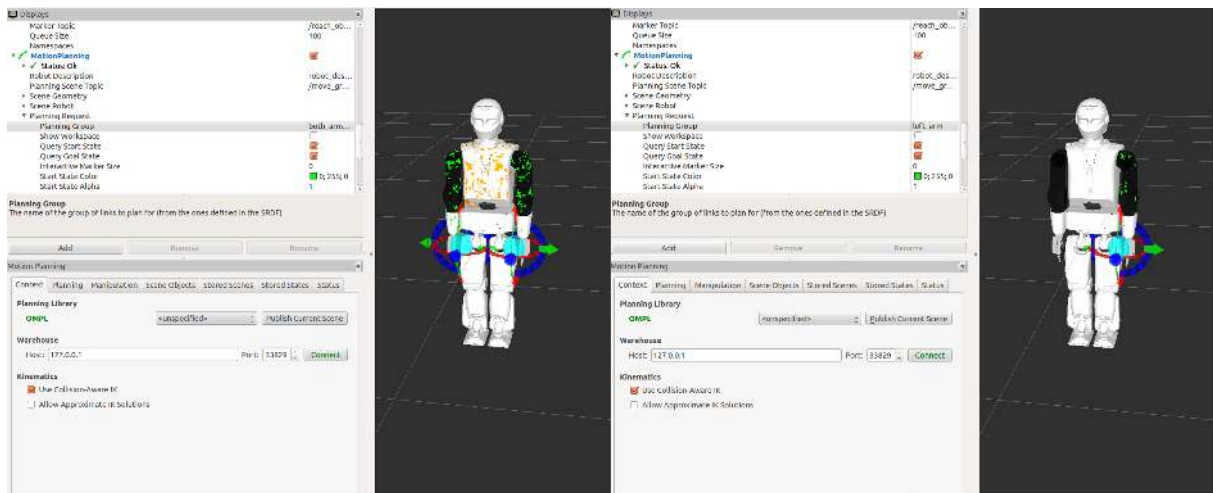


Figure 79: *MoveIt!* graphical interface in RViz. REEM-C with the planning group *both_arms* (left) and with the planning group *left_arm* (right).

1. In order to use motion planning, a Start State and Goal State has to be known, to do this with the *MoveIt!* GUI navigate to the tab called “Planning” in the display called MotionPlanning. On this tab, further nested tabs provide the functionality required to update Start State and Goal State depicted in Figure 80.

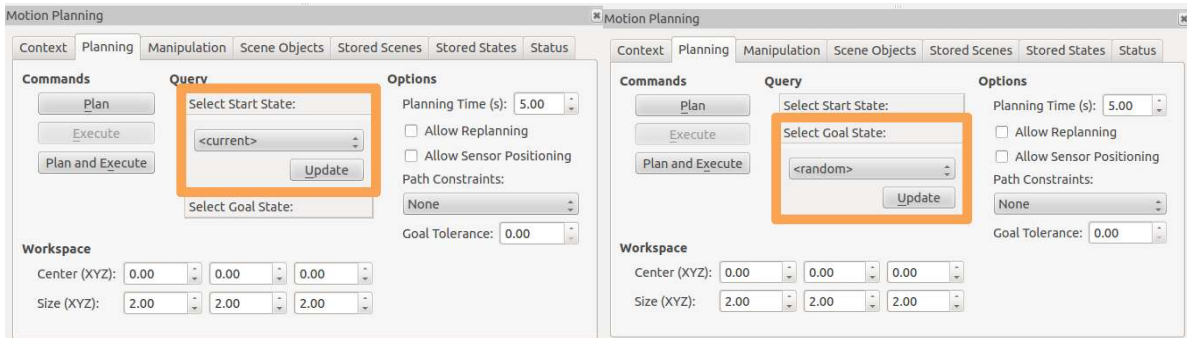


Figure 80: Tab to set start state (left) and tab to set goal state (right).

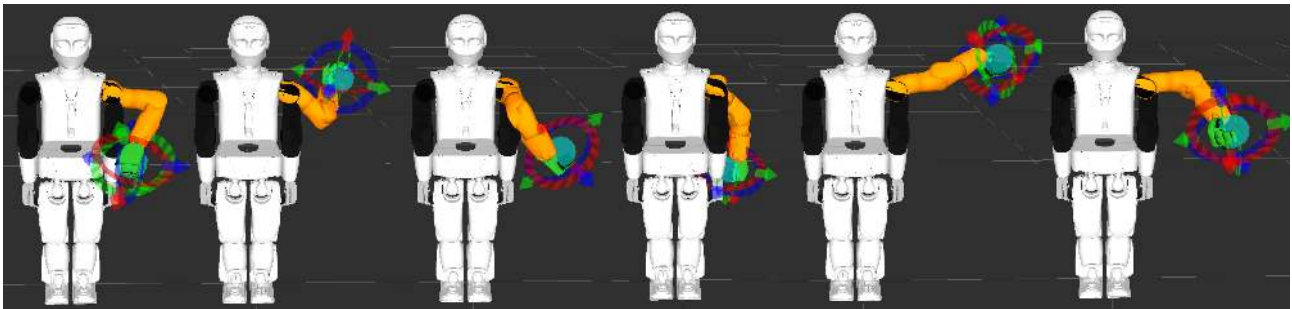


Figure 81: Random poses generated with the GUI

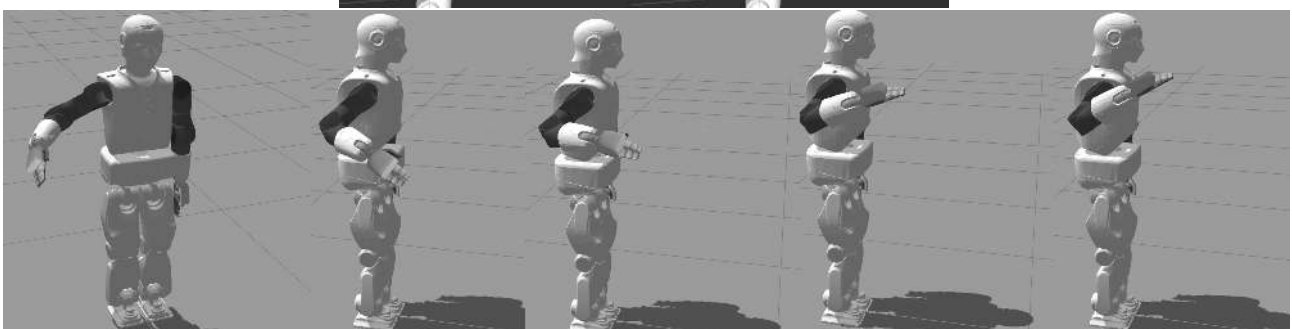
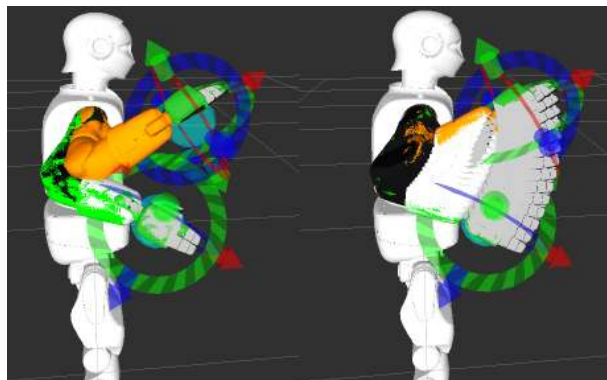


Figure 82: Sequence of plan and execute states

2. By clicking the “Update” button of the goal state, new poses can be randomized. Figure 81 shows a few randomly generated poses.
3. The sequence of images in Figure 82 shows the result of clicking the “Update” goal pose with random in RViz, then clicking “Plan and Execute” in the simulator. RViz will visualize the plan before executing.

4. The GUI also allows the operator to define goal poses using the “flying end-effector” method. As shown in Figure 81, the 6DOF pose of the end-effector can be defined using the visual marker attached to it. The red, green and blue arrows define the translation of x, y and z coordinates respectively; the colored rings define the rotation around these axes, following the same logic.

31.2 End test

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

32 SLAM map building

32.1 Overview

This section covers how to build a metric map of an indoor environment using the readings provided by the 2D laser scanners in the robot feet. The robot should be moved around with keyboard/joystick while the map is built. The map obtained is an Occupancy Grid Map (OGM) that you can use to localize and navigate the robot autonomously in the environment.

The problem of creating a map of the environment is known as SLAM (Simultaneous Localization and Mapping) because the mapping and localization problems have to be solved simultaneously, since there is no map to localize and no localization system to easily it. This section covers how to run `gmapping` in simulation and on the real robot. This is a node that implements the FastSLAM algorithm.

32.2 Start SLAM in simulation

We can run the simulator, rviz and all the nodes required for mapping with a single launch file:

```
roslaunch reemc_2dnav_gazebo reemc_mapping.launch rviz:=true
```

Apart from launching the navigation mapping state, we ask for rviz to be opened with a layout already adapted for the mapping topics, which is basically the map in the topic `/map`. The SLAM algorithm starts mapping automatically, so we just have to move the robot with the keyboard/joystick. After you move the robot a little around the environment, you will see a map as the one shown in the sequence of snapshots of Figure 83.

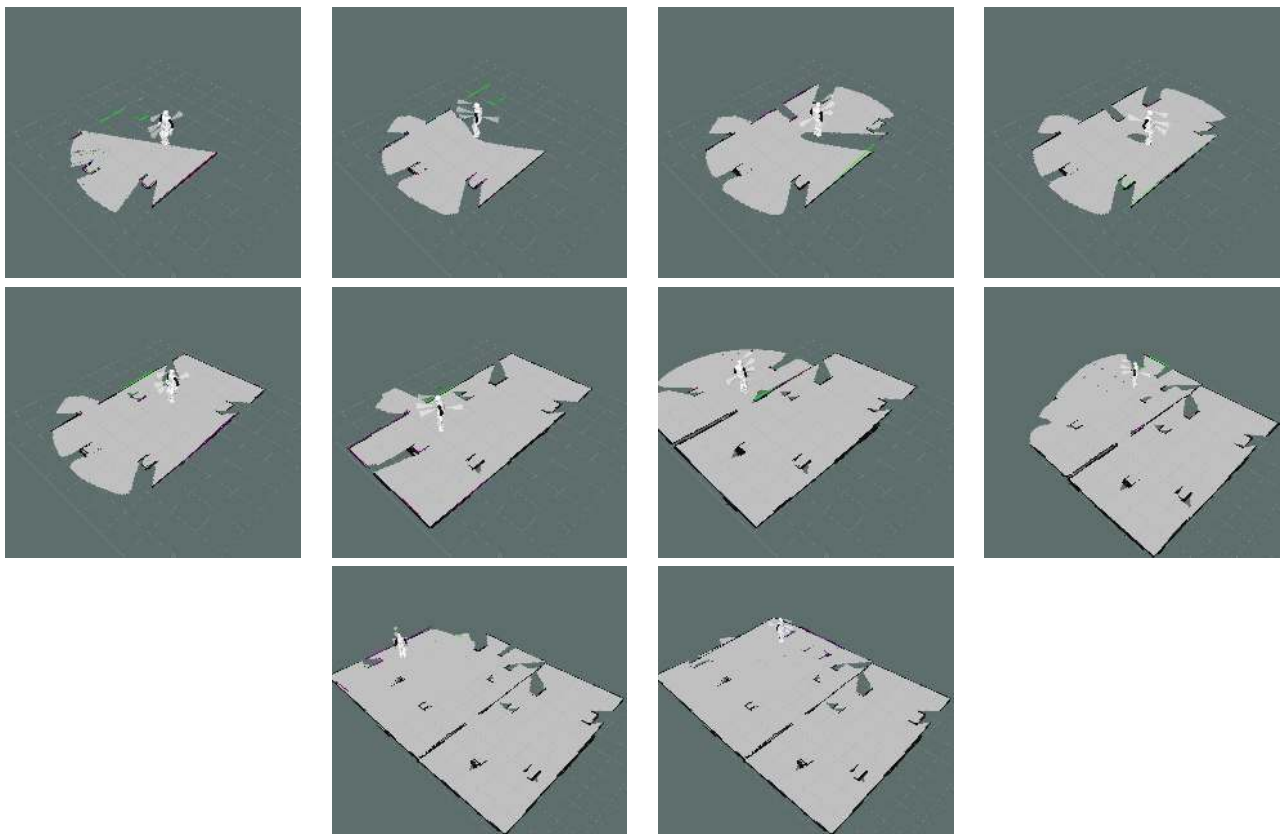


Figure 83: Partial maps built by the SLAM algorithm, starting (top left) and moving until a successful map is built (bottom right)

At the end you will have a map that models the environment, which in this case is the Gazebo world shown in Figure 84.



Figure 84: Small office world simulated in Gazebo

32.3 Save the map

Once we are satisfied with the map, we can save it manually in the current directory by executing:

```
roslaunch map_server map_saver
```

This command saves two files (as shown in Figure 85): a *map.pgm* image file and a *map.yaml* configuration file. Both will be needed later to use the map for localization and path planning.

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

Figure 85: Output of saving the map created

The *map.pgm* file will contain a graphic representation of the map built. You can edit it with *gimp* if there is some noise.

REEM-C's maps are usually saved in a new folder inside the `reemc_maps` package. To save your map there in a folder named `new_map`, use:

```
roscd reemc_maps/configurations
mkdir new_map
roslaunch map_server map_saver
```

32.4 Stop mapping

In order to stop mapping we must stop the SLAM node:

```
roslaunch map_server map_saver
```

At this point we might launch the localization and path planning nodes in order to do autonomous navigation. However, it is generally easier to stop the simulation pressing `Ctrl+C` and start it again in localization state, as will be explained in Section 33. An even better solution is provided by means of the Navigation State Machine, which is explained in Section 36.

32.5 Start SLAM on the robot

The procedure for the real robot is almost the same, we just do not need the simulator, and the rviz visualization must be run separately in the development machine outside the robot. Before you start the SLAM algorithm you need the walking controller running, so you can move it with the joystick to map the environment. Log into the robot and start the SLAM algorithm doing:

```
ssh pal@reemc-6m
roslaunch reemc_2dnav navigation.launch state:=mapping
```

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

Additionally, in order to visualize the map in rviz from your development machine, do the following:

```
export ROS_MASTER_URI=http://reemc-6c:11311
roslaunch rviz rviz -d `rospack find reemc_2dnav`/config/rviz/navigation.rviz
```

32.5.1 Filter the crane during mapping or navigation

In case you are moving the robot with a moving crane like the one in Figure 86, you might have to filter the wheels of such a crane.

Note that it is important that the crane legs are not in the same plane as the laser, because the field of view would be occluded. In order to filter out the crane's wheels the robot has a scan range filter which can be configured before running the navigation in the following file:

```
rosed reemc_2dnav lasermux.yaml
```

These are the parameters concerning the scan range filter:

```
range_min: 1.0
range_max: 10.0
```

Just set the `range_min` to something like one or half a meter.

33 Localization in a known map

33.1 Overview

This section covers how to localize the robot in a known map, which has been probably built using a SLAM algorithm. The robot uses the Occupancy Grid Map (OGM) to find out the robot's pose (position and orientation) using odometry, and laser readings.

In this section is shown how to run `amcl` in the simulation and on the real robot. This is a node that implements an Adaptive Monte Carlo Localization algorithm.

33.2 Start localization in simulation

We can run the simulator, rviz and all the nodes required for localization with a single launch file:

```
roslaunch reemc_2dnav_gazebo reemc_navigation.launch rviz:=true
```

Apart from launching the navigation localization state, we ask for rviz to be opened with a layout already adapted for the localization and path planning topics, which for localization is basically the `/particle_cloud` that shows the robot pose hypotheses. At the beginning you should set the initial robot pose estimate (2D Pose Estimate button) close to the actual one (use Gazebo to find it out), as shown in Figure 87.

Then you can start moving the robot around in the environment. You will see that the robot pose hypotheses convergence to almost a single hypothesis if the localization works well, as in Figure 88.



Figure 86: Robot with the moving crane

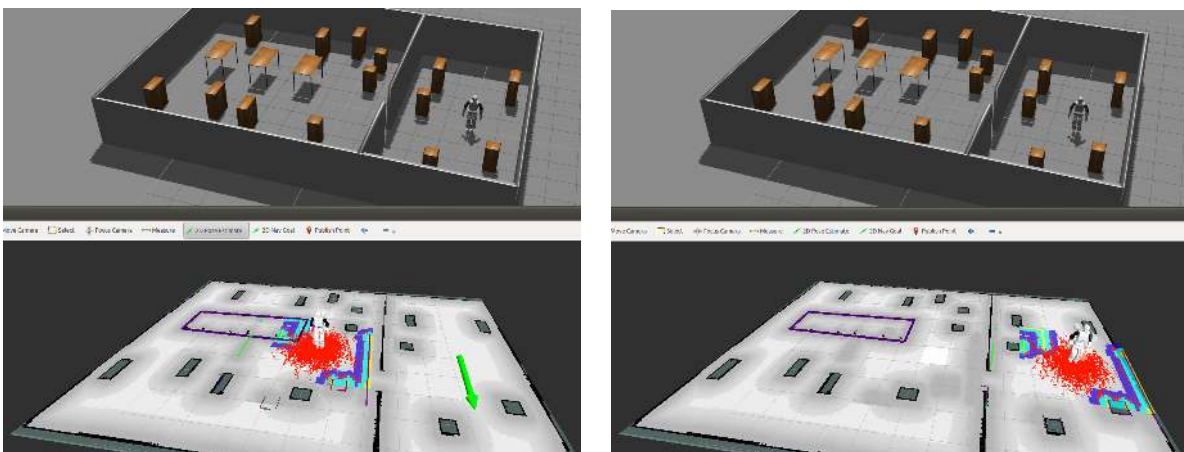


Figure 87: Setting initial pose in rviz

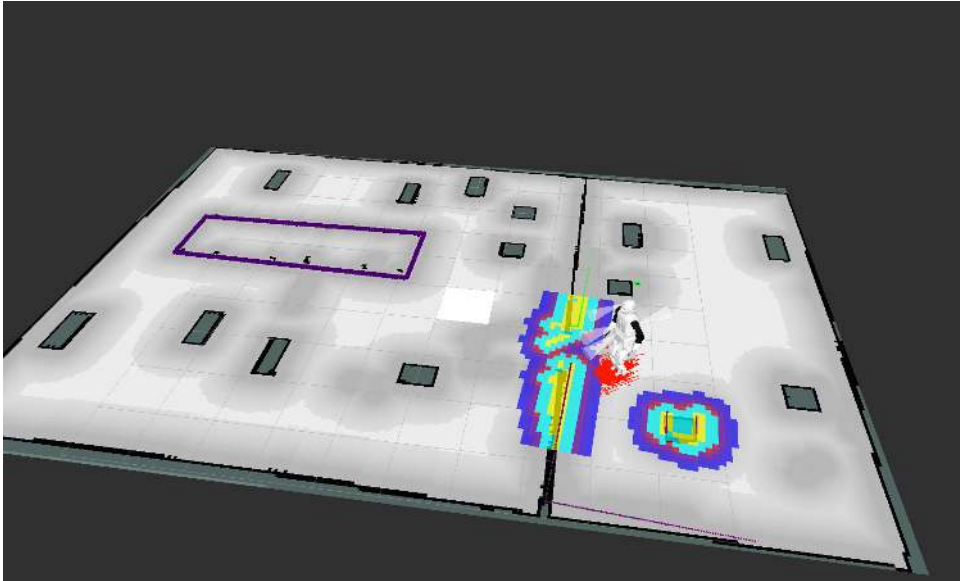


Figure 88: Robot pose after moving around, as estimated by the localization algorithm

33.3 Start localization on the robot

The procedure for the real robot is almost the same, we just do not need the simulator, and rviz visualization must be run separately in the development machine outside the robot. Before you start the localization algorithm you need the walking controller running, so you can move it with the joystick around the map/environment. Log into the robot and start the localization algorithm doing:

```
ssh pal@reemc-6m
roslaunch reemc_2dnav navigation.launch state:=localization
```

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

Additionally, in order to visualize the map in rviz from your development machine, do the following:

```
export ROS_MASTER_URI=http://reemc-6c:11311
roslaunch rviz rviz -d `rospack find reemc_2dnav`/config/rviz/navigation.rviz
```


34 Control source switch

34.1 Overview

This section covers how to switch the control source or, in other words, the source of the velocity commands the robot executes. Since the robot can walk controlled remotely with a joystick or autonomously using a path planning algorithm, we need a mechanism to choose between them, so only one is active at a time. In this section the mechanism is explained, as well as how to add more control sources and the priority scheme they follow.

34.2 Change between Manual and Autonomous control sources

By default, when the robot starts up, the control source is the joystick (manual), i.e. it has the priority. When we send a goal to the robot, using the autonomous navigation explained in Section 35, the robot is sent velocity commands from another source. Since now there are two different control sources, we need something to

prioritize them. Use the  button in the joystick to toggle the priority between joystick and autonomous navigation.

If you have `rviz` running with the following layout:

```
export ROS_MASTER_URI=http://reemc-6c:11311
roslaunch rviz rviz -d `rospack find reemc_2dnav`/config/rviz/navigation.rviz
```

You should see a text and an arrow on top of the robot head (see Figure 89). The text shows the current control source, being `Manual` or `Autonomous`, for the joystick and the autonomous navigation, respectively. The arrow shows the velocity that has been commanded to the robot by the current control source.

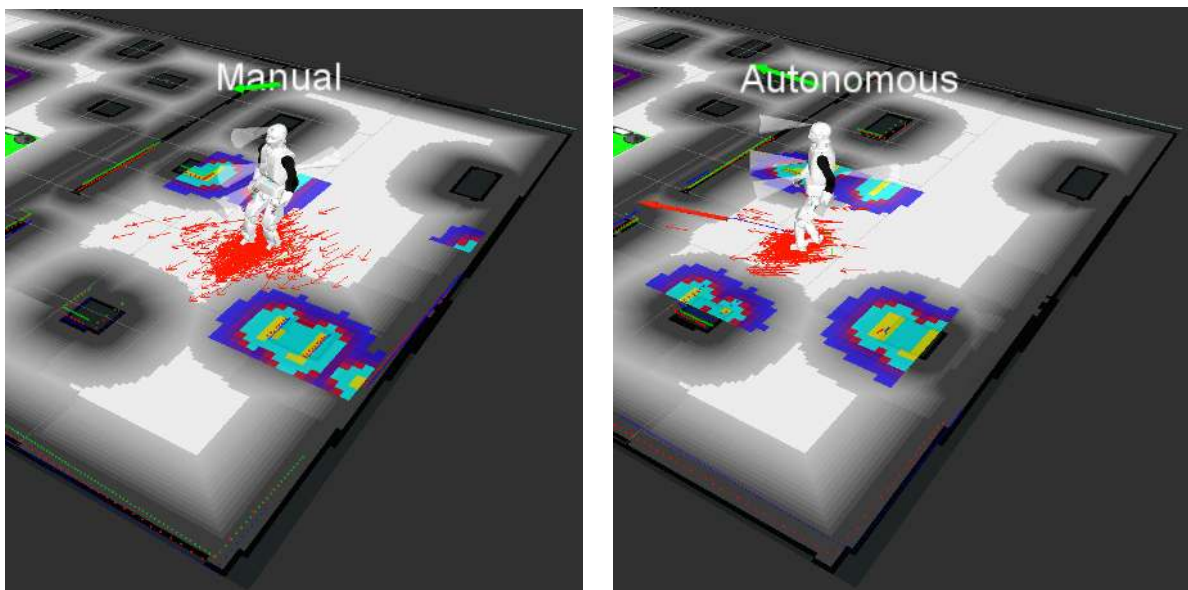


Figure 89: Current control source (text) and velocity command (arrow), showing joystick/manual (left) and autonomous navigation (right)

34.3 Change the control source priorities

The control sources are multiplexed using a priority-based scheme which uses `Twist` topics and `Bool` locks. The configuration of the input topics is located in:

```
roscd reemc_bringup twist_mux_topics.yaml
```

This specifies the control sources, which are the joystick and the autonomous navigation. The configuration provided also includes the keyboard and a tablet as examples:

```

topics:
-
  name      : navigation
  topic     : nav_vel
  timeout   : 0.5
  priority  : 10
-
  name      : joystick
  topic     : joy_vel
  timeout   : 0.5
  priority  : 100
-
  name      : keyboard
  topic     : key_vel
  timeout   : 0.5
  priority  : 90
-
  name      : tablet
  topic     : tab_vel
  timeout   : 0.5
  priority  : 100

```

The topics with higher `priority` are the control sources that are prioritized when more than one is sending commands. The `timeout` is used to detect when the control sources has dead, so we can stop the robot; for example, in case the joystick fails.

The locks allow to stop the robot in some cases, as the `joy_priority` mentioned above. When the mapping algorithm is closing a loop, the robot cannot be moved (the joystick commands are ignored), otherwise the map will be invalid. This is provided with the following configuration:

```

locks:
-
  name      : pause
  topic     : pause_navigation
  timeout   : 0.0
  priority  : 100
-
  name      : loop_closure
  topic     : stop_closing_loop
  timeout   : 0.0
  priority  : 200
-
  name      : joystick
  topic     : joy_priority
  timeout   : 0.0
  priority  : 100

```

where the `timeout = 0.0` means that the `twist_mux` does not check for timeouts for these lock topics.

35 Path Planning in a known map

35.1 Overview

This section covers how to send a goal and make the robot go there autonomously. A path planning algorithm finds the path within the map so the robot can reach the desired goal. Additionally, and obstacle avoidance algorithms uses the robot's sensors (lasers and sonars) to detect the obstacles around the robot and avoid

them, which might imply re-planning of the initial path. This allows to avoid moving obstacles as well as persons or elements that have been moved since the creation of the original map, so the robot can move autonomously and safely without colliding with them.

35.2 Start path planning in simulation

We can run the simulator, rviz and all the nodes required for path planning with a single launch file:

```
roslaunch reemc_2dnav_gazebo reemc_navigation.launch rviz:=true
```

Apart from launching the navigation localization state, we ask for rviz to be opened with a layout already adapted for the localization and path planning topics, which includes global, local plans and CostMaps, provided by the `/move_base` node.

35.2.1 Send a goal to the robot

Set the initial pose of the robot as explained in Section 33, and send a goal (2D Nav Goal button) in rviz to any place on the map (in free space, otherwise there will not be any path to reach). Figure 90 shows how to send a goal (top) as well as the path found and the CostMap around the robot with the obstacles detected by the sensors (bottom).

The robot will start walking, updating the path and the CostMap as it moves; you can try moving objects in Gazebo and see how it avoids them.

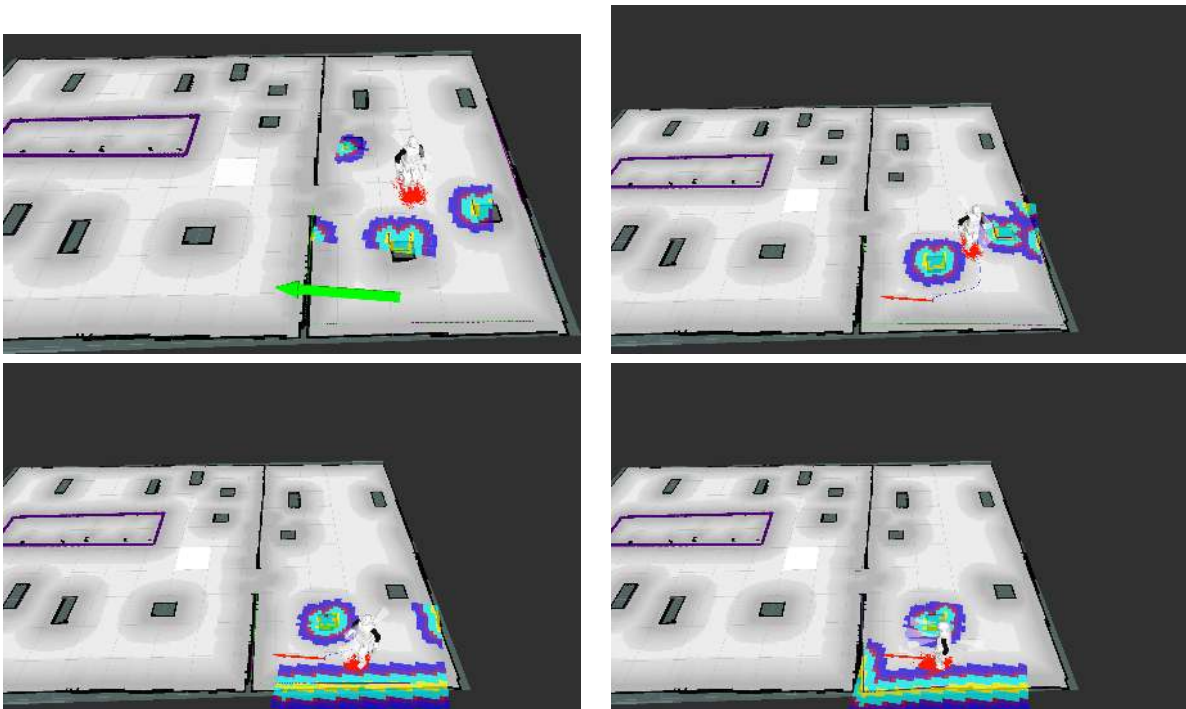


Figure 90: Path planning to a given goal (top left), and walking until it is reached (bottom right)

35.3 Start path planning on the robot

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

path planning algorithms, you need the walking controller to be running. Log into the robot and start the localization algorithm doing:

```
ssh pal@reemc-6m
roslaunch reemc_2dnav navigation.launch state:=localization
```

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

Additionally, in order to visualize the map in rviz from your development machine, do the following:

```
export ROS_MASTER_URI=http://reemc-6c:11311
roslaunch rviz rviz -d `rospack find reemc_2dnav`/config/rviz/navigation.rviz
```

35.4 Virtual Obstacle Avoidance

Apart from the obstacles that the robot can see with its sensors (lasers and sonars), we can include virtual obstacles on the map, using the map editor tool explained in Section 37. It is useful to set forbidden areas or to avoid the robot entering places where there are objects that cannot be detected by the sensors. The virtual obstacles are shown as a purple rectangle in Figure 90.

You can see the map with the virtual obstacles by changing the map topic in rviz to `/vo_map`. You could also add another map and see it overlaid on top of the `/map` topic, or visualize the point cloud `/vo_cloud` that publishes the virtual obstacles. When you send a goal, you will see that the global planner discards the goal if it lies inside the virtual obstacle zone. Similarly, during navigation, both the global and local planner avoid any virtual obstacle, which appears in the CostMap.

35.5 Changing the map

A list of all existing maps can be obtained with:

```
rosls reemc_maps/configurations
```

Then it is possible to change the active map using:

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

Also it is integrated in the navigation state machine and the map editor tool, which are explained in Section 36 and 37, respectively. Together, they provide a cleaner way to change from localization to mapping state and to change the current map, as well as adding elements to the map, like virtual obstacles.

36 Navigation State Machine

36.1 Overview

This section covers how to change from localization to mapping state and vice versa. It is done by means of a state machine that controls the current state and the nodes that should be running for each of them. It means that when we change from one state to the other, several nodes are stopped and others are started. Here is shown how to change from one state to the other.

36.2 Start the State Machine in simulation

In simulation we just have to run the navigation launch file in the desired state. By default it starts in localization state, so just run this and you will have Gazebo and rviz, as we did in Section 33.

```
export PAL_HOST=reemc
roslaunch reemc_2dnav_gazebo reemc_navigation.launch rviz:=true
```


36.3 Change from localization to mapping

The navigation state machine provides a service to change from one state to another. In order to change from localization (LOC) to mapping (MAP) we do:

```
rosservice call /pal_navigation_sm MAP
```

We can check the current state with:

```
rostopic echo /pal_navigation_sm/state
```

After having changed to mapping you should see the following:

```
status:  
  data: LOC  
msg:  
  data: ''  
---  
status:  
  data: CHANGING  
msg:  
  data: OK  
---  
status:  
  data: MAP  
msg:  
  data: OK  
---
```

Additionally, you will see in rviz that the previous map disappeared and a new one started being built, as shown in Figure 91.

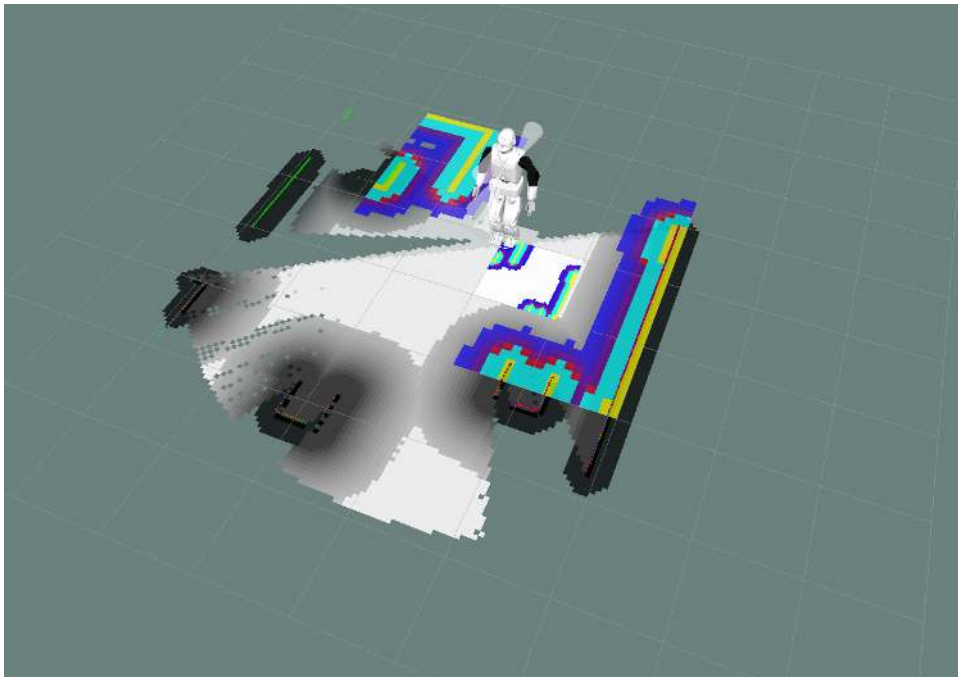


Figure 91: State changed to mapping

36.4 Change from mapping to localization

Similarly, we change to the mapping state doing:

```
rosservice call /pal_navigation_sm LOC
```

The map that was built during mapping will be saved in:

```
rosls reemc_maps/configurations
```

Additionally, we will see it in rviz, so we can navigate the robot autonomously within it, just sending goals as shown in Figure 92.

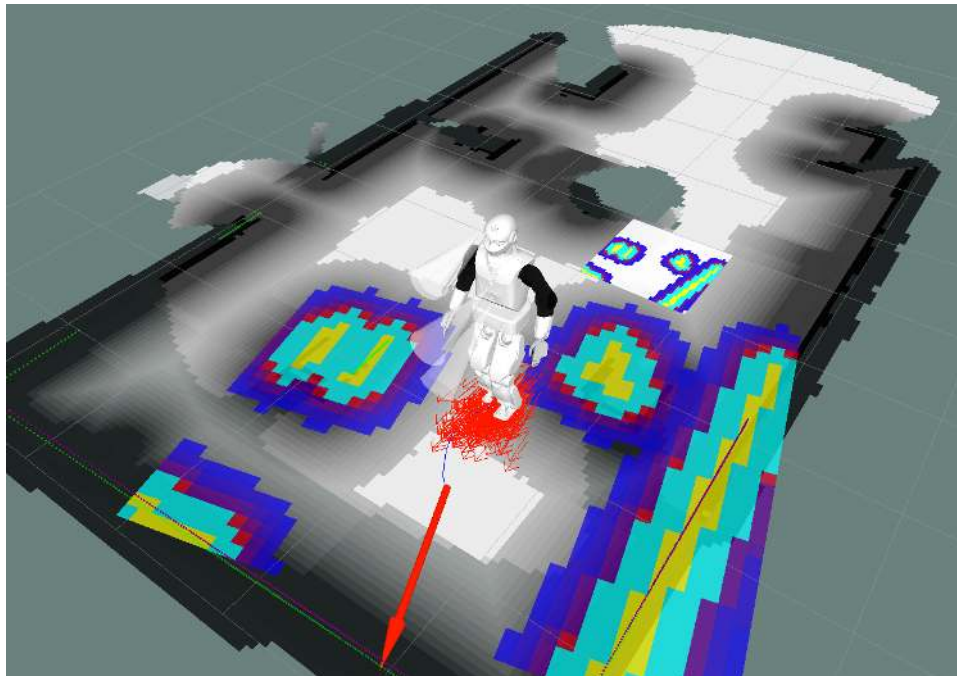


Figure 92: State changed to localization

36.5 Start the State Machine in the robot

The procedure in the real robot is almost the same, we just do not need the simulator, and the rviz visualization must be run separately in the development machine outside the robot. Before you start the navigation state machine you need the walking controller running. Log into the robot and start the navigation state machine in localization state doing:

```
ssh pal@reemc-6m
roslaunch reemc_2dnav navigation.launch state:=localization
```

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

Additionally, in order to visualize the map in rviz from your development machine, do the following:

```
export ROS_MASTER_URI=http://reemc-6c:11311
roslaunch rviz rviz -d `rospack find reemc_2dnav`/config/rviz/navigation.rviz
```

37 Map Editor

37.1 Overview

The Map Editor is a tool that allows a user to manage the navigation maps created by PAL Robotics robots.

It allows to create and remove Points of Interest (POIs), Zones of Interest (ZOIs) and Virtual Obstacles (VOs) as well as choosing an image that will be overlaid on top of the navigation map in order to display to the customers. This functionality can be use to select a better looking image such as the building plans to be displayed instead of the navigation map.

37.2 Glossary

POI Point of Interest, a defined location on the map where the robot can be sent to. In the Speech Editor it is possible to add text and upper body movements that will be performed when the robot reaches the POI.

VO Virtual obstacle, a zone limits of which the robot will not cross when it navigates either autonomously or remotely controlled. Can be used to prevent the robot from getting into certain spaces, but also to prevent the robot from leaving them if the robot is placed inside.

ZOI Zone of Interest, a zone of special interest. The robot will publish the name of the area the robot just entered in the ROS topic `/current_zone_of_interest`. This can be used by other application to alter the robot behaviour according to the current area.

37.3 Start

In a computer with the `pal-software-apps` debian package installed, click on the Dash Home icon on the top left side of the screen and write in the search bar "Pal Map Editor" and when it is displayed click on the icon that appears as in Figure 93. Enter the hostname of the control computer of the robot in the terminal that is displayed and the map editor will be opened.



Figure 93: Map Editor launch icon

You will see the GUI shown in Figure 94. The main area will show the Occupancy Grid Map (OGM) obtained by the robot along with the POIs, ZOIs and VOs, and the building map aligned with it. In the right panel we have several options for the alignment, while in the bottom panel we have four buttons to save/load the map to/from the local disk and download/upload it from/to the robot; in REEM-C only the zoom scroll bar at the bottom is available, the rest elements will not appear.



Figure 94: GUI Map Editor

37.4 Download OGM from the robot

Once we have finished mapping an area with the robot, you must go to localization mode. When the robot is in localization mode, the current OGM is published and you can download it from the robot to our computer running the map editor. For that, just press the `Download Map` button and you will see the OGM as in Figure 95.



Figure 95: Download OGM map from the robot

37.5 Load and Align Building Plan

If we have the plan of the building just mapped with the robot, we can load it in the application by pressing the `Load Nice Map Image` button. As you can see in Figure 96, the building plan (also called nice map) is not aligned with the OGM.



Figure 96: Load nice map

In order to align the nice map we have two options. On one hand, we can do it manually with the `Rotation`, `Movement` and `Scale` panels. On the other hand, we can use the `Automatic Aligning` button to give three points in the OGM with its three corresponding points on the map, and align them automatically (see Figure 97). Later we can also tune the result manually. Additionally, we can control the transparency of the nice map with scrollbar of the `Alpha Channel` panel, and the zoom with the one of the `Zoom` panel.

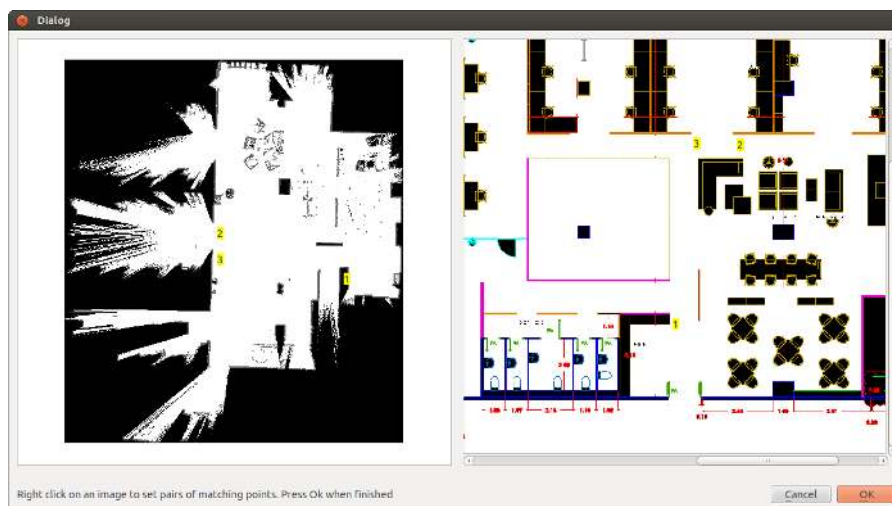


Figure 97: Nice map aligning

The result should be as the one shown in Figure 98.



Figure 98: Human-friendly map aligned

37.6 Add and remove virtual elements to the map

We have three types of virtual elements on the map:

1. Points of Interest (POIs): Specify a location where the robot may navigate autonomously.
2. Zones of Interest (ZOIs): Specify a zone of interest.
3. Virtual Obstacles (VOs): Specify a zone where the robot will not enter when it navigates either autonomously or remotely controlled.

37.6.1 Points of Interest

Right click on the map and you will see the pop up menu of Figure 99a. Then, give a name to the POI in the text field in the modal window that will appear, and you will see something like the POI on Figure 99b.

37.6.2 Zones of Interest

Right click on the map and then select `Start drawing Zone of Interest` in the pop up menu. This will start a green polygon. You will just see a point representing the first vertex. To add more vertices just right click again in the desired location. Once you are finished, just right click around the polygon and select `Finish Zone of Interest` (see Figure 100).

37.6.3 Virtual Obstacles

Virtual obstacles are defined as the ZOIs; we just have to select `Start drawing Virtual Obstacle` in the pop up menu. The VOs appear as red polygons (see Figure 100 for instance).

37.6.4 Remove elements

All the elements can be removed by right clicking on them. A pop up menu will appear and then you can remove it, as in the example of Figure 100 for the `software` POI.

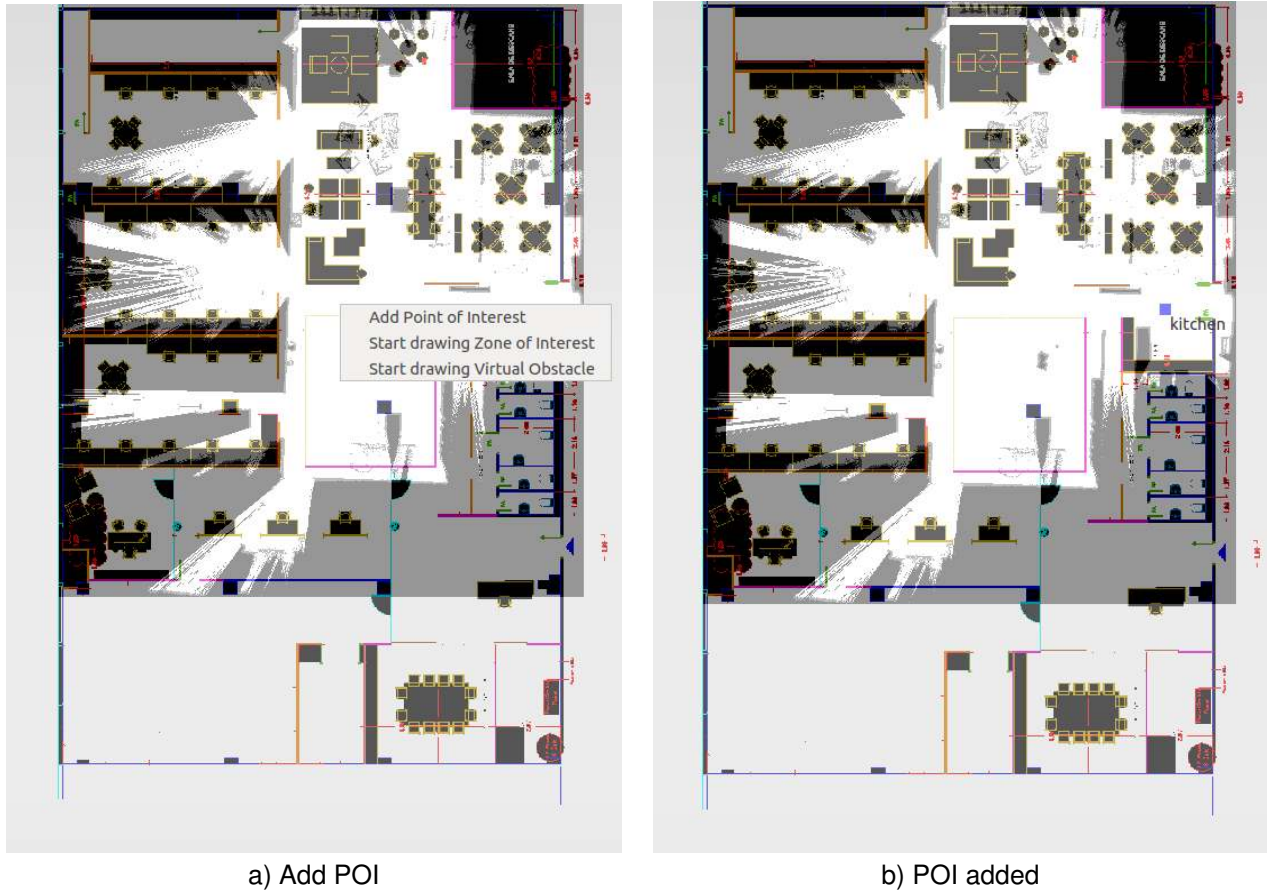


Figure 99: Adding POI

37.7 Upload Map to the robot

Once we have aligned the human-friendly map with the OGM and created all the virtual elements (POIs, ZOIs or VO) desired, we can upload the map to the robot by clicking on the `Upload Map` button of the bottom panel.

37.8 Save and Load the map to the disk

The map editor also has an option to save the map edited to the disk of the local computer. Just press in the `Save to disk` button of the bottom panel and select/create the folder where you want to save the map. This will save the OGM, the human-friendly map with its transformation with respect to the OGM, and all the virtual elements.

Later, you can open the map editor and load this map from the disk by clicking on the `Load from disk` button and selecting the folder that contains the map previously saved on the disk. Then, we can edit the map if required and upload it to the robot. This way we have a tool to manage the maps acquired by the robot, edit them, and send the desired one to the robot at any time.



Figure 100: Adding and removing ZOI

REEM C

Examples

PALO
ROBOTICS 

38 Overview

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

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

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

38.1 REEM-C's ROS environment

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

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

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

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

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

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

38.2 Source code

The test programs and configuration files covered in this document are distributed across multiple tutorial ROS packages, and follow the naming convention `reemc*_tutorials`, (e.g. `reemc_basic_tutorials`). The exact set of available tutorial packages depends on the functionalities that have been shipped with REEM-C.

To browse the contents of any of these packages from the command line, use the `roscd` command. For example:

```
roscd reemc_basic_tutorials
```

In the following section, the `reemc_basic_tutorials` package will be used as an example, but the same considerations are valid for other tutorial packages.

The `reemc_basic_tutorials` package has a folder structure that adheres to the usual [ROS package convention](#), a `src` folder containing source code, a `launch` folder containing ROS launch files (which are helpful for launching nontrivial setups), a `config` folder containing configuration files (e.g. parameter values and `rviz` configurations), and a `scripts` folder containing executable scripts.

39 Manually building the tutorials (optional)

The installed software suite already includes a pre-compiled version of the tutorial packages that can be used out-of-the-box. However, it is also possible to inspect their source code and manually build them in an overlaid ROS workspace. To do so, open a command-line terminal and follow these steps:

1. Open a new terminal, and make sure REEM-C's ROS environment has been setup, as explained in section 38.1.

2. Create a workspace directory

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

3. And download the tutorials source code:

- (a) Generic tutorials

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

- (b) Walking tutorials

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

- (c) Text-To-Speech tutorials

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

4. Build the tutorial packages:

```
cd ~/pal_ws
catkin_make
```

5. At this point, the tutorials are built but are not yet part of the ROS environment. For example, running:

```
rospack find reemc_basic_tutorials
```

should output:

```
/opt/pal/erbium/share/reemc_basic_tutorials
```

which is the system install, not the one in the overlay workspace. It is therefore required to:

6. Add the current workspace to the active ROS workspace:

```
source ~/pal_ws/devel/setup.bash
```

7. Check that the overlay packages are being picked up. For example:

```
rospack find reemc_basic_tutorials
```

which should output:

```
/home/pal/pal_ws/src/reemc_basic_tutorials
```

Note that it differs from the output of point 5.

8. If you wish to develop new ROS packages of your own, you can follow the above instructions, adding your source packages to `~/pal_ws/src` instead of (or alongside) REEM-C's tutorials.

40 Setup

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

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

40.1 In a simulated environment

40.1.1 Start a simulated REEM-C

1. Open a terminal and launch a Gazebo simulation of REEM-C in an empty world (Figure 101).
Select *Robot Model* and then *base_link* as a *Fixed Frame*.



Figure 101: Simulated robot

40.1.2 Connect a terminal to the robot

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

40.2 On the real robot

40.2.1 Start a real REEM-C

Please refer to section 12 – Getting started, for details on how to start REEM-C.

40.2.2 Connect a terminal to the robot

Open a new terminal and execute the following:

```
export ROS_MASTER_URI=http://reemc-6c:11311
```

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

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

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

41 Device State Notifications

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

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

```
ssh pal@reemc-6c
pgrep -f pal_webcommander_node
exit
```

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

41.1 Accessing the device state notification website

1. Open a web browser⁴, and connect to the `WebCommander` website following the instructions in the `Software Development Environment` section of the manual. In most setups, you must type:

```
http://reemc-6c:8080
```

41.2 Diagnostics Tab

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

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

```
ssh pal@reemc-6c
```

Once logged in, execute:

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

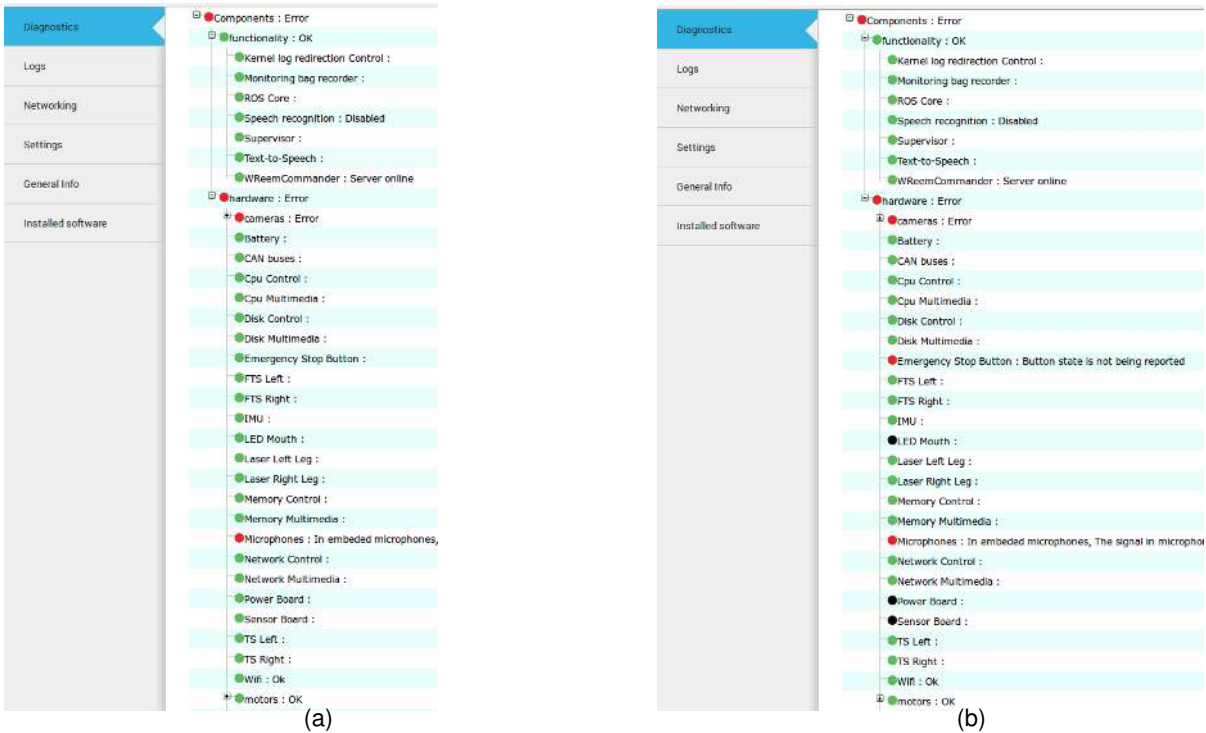


Figure 102: Diagnostics. Before stopping sensor board application (a). After stopping sensor board application (b).

```
sb04r03ProdStop.sh
```

After stopping the application, we can see that in the hardware diagnostics, the Power Board and Sensor Board went black, meaning that we stopped receiving diagnostic information from the applications that can be seen in Figure 102(b).

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

From the same terminal used before:

```
sb04r03ProdStart.sh
```

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

Finally, log off REEM-C's control computer.

```
exit
```

41.3 Logs Tab

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

42 Joints initialization

Description This test covers how to initialize the hand joints, which are the only joints in the robot not equipped with absolute encoders. All other robot joints do *not* require an initialization routine. This test applies only to real REEM-C deployments, not to simulated ones.

Requisites A running REEM-C, as described in section 40.

42.1 Usage

The Hey5 hands are equipped with an automatic joint initialization process. This process is triggered every time the hand is powered, and requires no software support to take place. Please refer to the Hey5 hand user manual for details of its joint initialization process.

42.2 Example

This test will trigger hand initialization from an appropriate robot configuration.

1. Open a terminal connected to REEM-C, and start joint trajectory controllers for the full robot.

```
roslaunch reemc_controller_configuration joint_trajectory_controllers.launch
```

2. Open another terminal connected to REEM-C, and bring the robot to the `interact` configuration, shown in Figure 103, from which an emergency stop will not cause the arm to drop, and from which hand initialization can be performed.

```
axcli /play_motion "motion_name: interact"
```

For a more detailed description on how to execute gestures, or coordinated robot motions, please refer to section 51.

3. Turn off the robot .
4. Turn on the robot.
5. Observe how the hand initialization process takes place. Once complete, the hand is ready to be operated again.

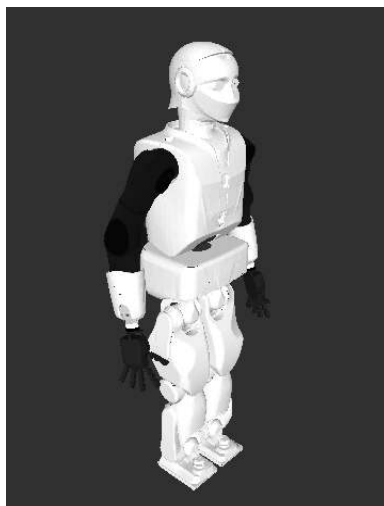


Figure 103: Robot arm in the `interact` configuration.

42.3 End test

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

43 Rviz Basics

Description Rviz is a useful tool for visualizing the geometric and kinematic model of REEM-C, as well as the robot's view of the world. As shown in Figure 104, rviz consists of a main 3D rendering area which is surrounded by a number of dock widgets. The 3D rendering area allows for the usual scene navigation operations, such as translation, rotation and zoom. We will now focus mainly on the left dock widget containing *Displays*. Refer to the official documentation for an in-depth description of [rviz](#).

Requisites A running REEM-C, as described in section 40.

43.1 Setup

Open a terminal connected to REEM-C, and execute the following command

```
roslaunch reemc_basic_tutorials rviz.launch
```

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

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

```
roslaunch reemc_gazebo reemc_gazebo.launch
```

and Rviz from another terminal:

```
roslaunch rviz rviz
```

RobotModel Displays a rendering of REEM-C links (Figure 104). It allows –among other things– to toggle between visualization and collision geometries, to set the model transparency, to show the axis of the body links' reference frames and the trail followed during motions.

TF Displays the `tf` transform tree. Each transform in the tree represents a frame that might correspond to robot link origins, sensor locations, or any other feature that can have a coordinate system associated to it. Figure 105 displays part of the current TF tree, in particular the origin of some of the robot's links. Note that we have set the RobotModel display transparency to 0.5 to better visualize the frames contained inside REEM-C .

Sensors: Camera, LaserScan, Range These displays gather the robot's view of the world through its sensors (Figure 106), namely:

- Two head-mounted *cameras*.
- Four *sonars*: two in the chest, one in the back and one in the head

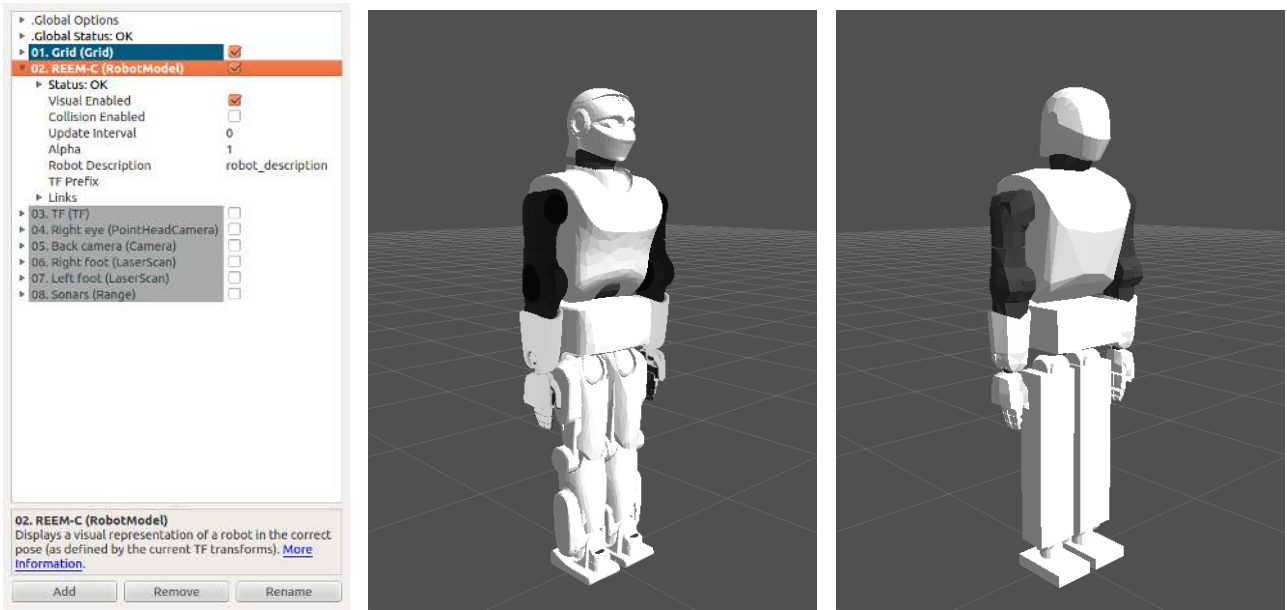


Figure 104: RobotModel display (left), visualization geometry (center) and collision geometry (right).

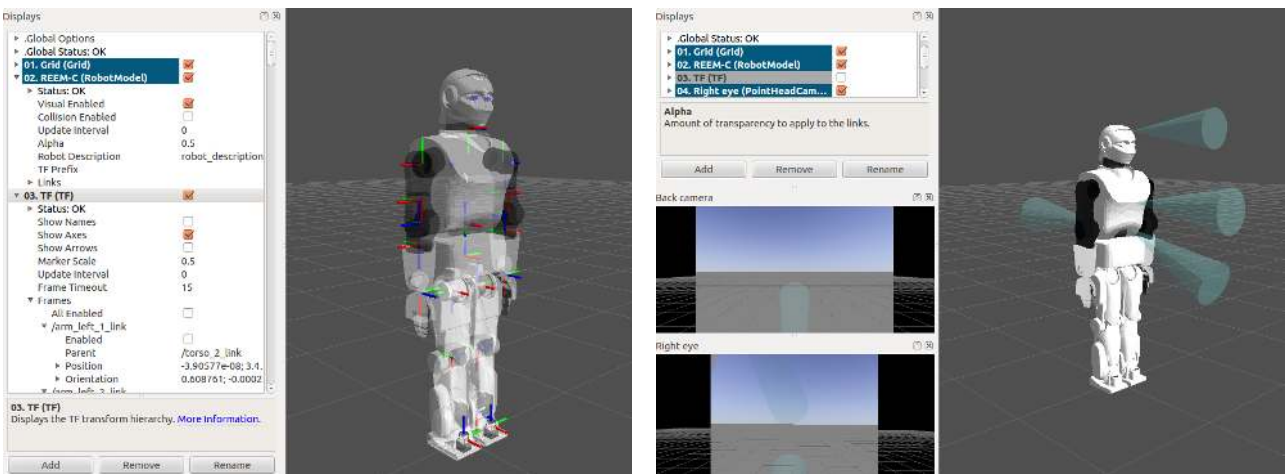


Figure 105: TF display with semitransparent REEM-C.

Figure 106: Sensor displays: Cameras, LaserScan and Range (sonar).

44 Rqt Basics

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

This test presents an overview of an example *perspective* used to graphically interact with the robot's system. It will be used during the tests in sections 49 and 50.

Requisites A running REEM-C, as described in section 40.

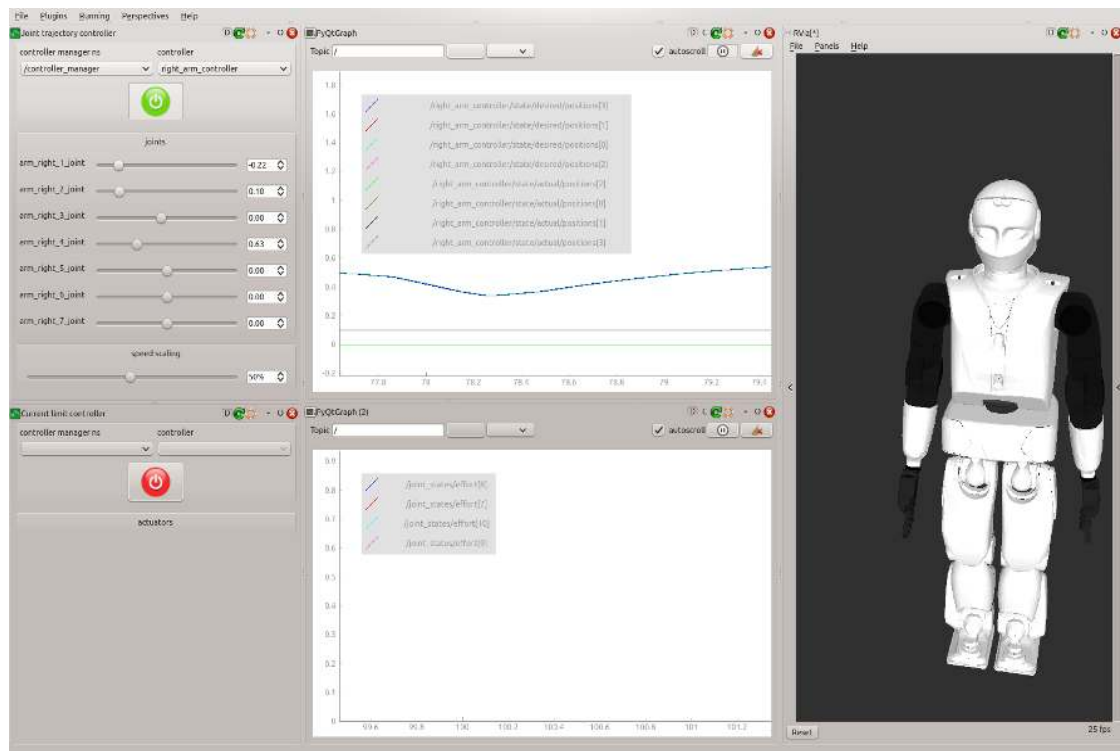


Figure 107: Rqt GUI displaying controls for commanding the robot joints (left) plots displaying the time evolution of selected joint positions and motor currents (center), and a 3D rendering of the robot system (right).

44.1 Setup

44.1.1 Default Rviz configuration

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

1. Open a new terminal and set the default Rviz configuration

```
cp `rospack find reemc_basic_tutorials`/config/reemc_minimal.rviz ~/.rviz/default.rviz
```

44.1.2 Launch the GUI

1. Open a terminal connected to REEM-C, and start joint trajectory controllers for the full robot.

```
roslaunch reemc_controller_configuration joint_trajectory_controllers.launch
```

2. Open another terminal connected to REEM-C, and launch an instance of Rqt loaded with the example *perspective* to graphically interact with the robot system.

```
roslaunch reemc_motion_tutorials reemc_gui.launch
```

3. A window similar in appearance to Figure 107 should be displayed.

44.1.3 GUI details

The user interface from Figure 107 has three main components:

3D model A virtual reconstruction of the robot system is shown on the right-most part of Figure 107. This visualization is implemented by an [Rviz](#) plugin.

Online data plots The time evolution of selected signals is shown in the center of Figure 107. The top plot displays the actual and desired position of the right arm shoulder and elbow joints, while the bottom plot displays the electrical current being applied to them. This visualization is implemented by an [rqt_plot](#) plugin.

Control commands Two kinds of plugins, shown on the left-most part of Figure 107, allow the user to send commands to running controllers:

- **Joint positions:** Allows the user to set the desired position of individual joints. This visualization is implemented by the `rqt_joint_trajectory_controller` plugin. Its operation is detailed in section 49.
- **Actuator current limits:** Allows the user to set the maximum current individual actuators can apply. This visualization is implemented by the `rqt_current_limit_controller` plugin. Its operation is detailed in section 50.

44.2 End test

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

45 Modifying the simulation world

Description This test covers how to spawn REEM-C in simulated worlds different from the one previously showed. It also describes how to modify a simulation world by adding new models.

Requisites A succesful build of the `reemc_tutorials` package, as per the Setup section of the Overview.

45.1 Setup

1. Open a terminal and launch a Gazebo simulation of REEM-C in an indoor world (Figure 108, left).

```
roslaunch reemc_gazebo reemc_gazebo.launch world:=reemc_indoor
```

45.2 Modifying the simulation world

1. **Add a geometric primitive:** Click on the cylinder icon of the top toolbar (depicted below) and place it near the REEM-C .



2. **Add a simulation model:** On the side panel, activate the insert tab, click on the `reemc_bookshelf` entry (depicted below) and place it near the REEM-C .



The simulation world with the cylinder and the bookshelf is shown in Figure 108, right.

3. **Remove an object from the simulation:** right-click on the object, and select `Delete` on the displayed menu.

A complete description of the Gazebo simulator user interface can be found in the [Gazebo user manual](#).

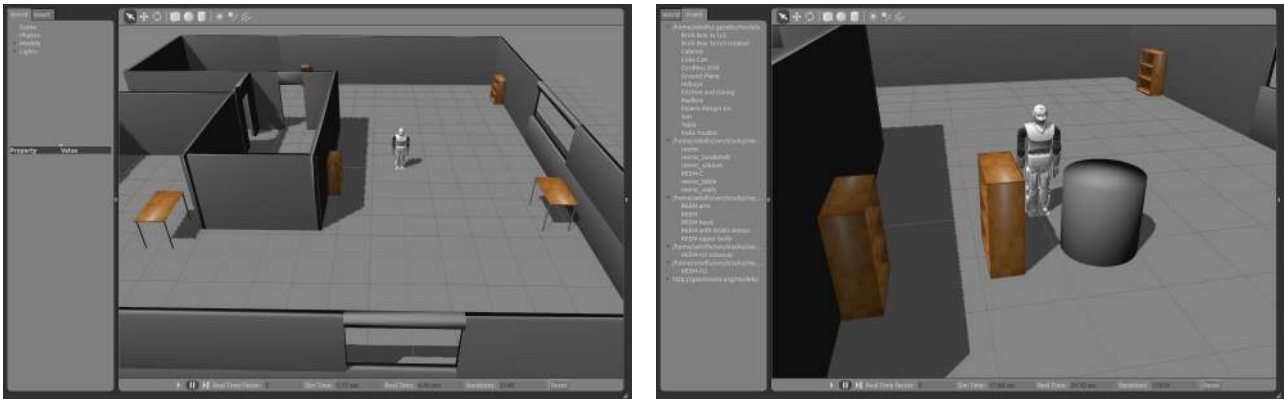


Figure 108: Gazebo simulation of REEM-C in an indoor world (left). The simulation after adding a cylinder and a bookshelf (right).

45.3 Sensor readings

1. In another terminal, launch the `rviz` visualizer (Figure 106, right).

```
roslaunch reemc_tutorials rviz.launch
```

2. Activate the sensor displays (cameras, laser scanners, sonars) to see the robot's view of the world through its sensors. In Figure 109 it can be seen how the feet-mounted laser scanners detect the bookshelf and the cylinder added above (red point cloud), and how the lower torso sonar is detecting the cylinder (notice the smaller acoustic cone size).

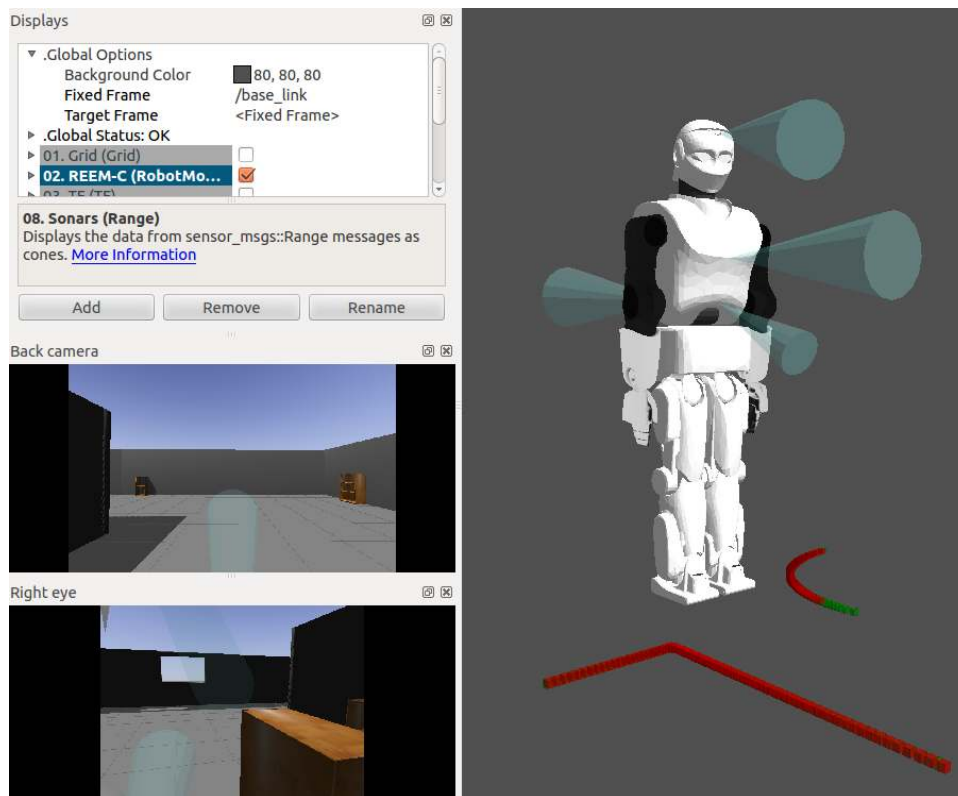


Figure 109: REEM-C sensors view of the world through rviz.

46 Inspect the kinematic workspace of REEM-C

Description This test covers how to inspect the kinematic workspace of REEM-C by means of a slider-based user interface for moving individual joints. Unlike other tests, which use a dynamic simulation of REEM-C (ie. subject to gravity, contacts and other forces), this test uses a pure kinematic simulation. Since REEM-C is a biped robot, there exist many body configurations that are unstable from a balance point of view and lead to a fall. This test allows to inspect all reachable joint configurations while disregarding whether they are unstable or not. It does not apply to real REEM-C deployments.

Requisites A successful build of the `reemc_basic_tutorials` package, as described in section 39.

46.1 Setup

As stated in the description, this test is only for simulation.

1. Open a terminal and launch the *kinematic* simulation of REEM-C (Figure 110, left).

```
roslaunch reemc_basic_tutorials reemc_kinematic_sim.launch
```

Two windows should be visible:

- A set of sliders, each corresponding to one REEM-C joint⁵ (Figure 110, left).
- A 3D rendering of the kinematic simulation showing REEM-C in the default pose with all joints set to zero (Figure 110, center).

⁵The sliders window does not render well on small screens, given the large number of joints.

46.2 Moving joints with the sliders interface

1. Use the sliders to explore the valid motion range of the joints of interest. A particular configuration is shown in Figure 110, right.
2. To revert to the initial configuration (Figure 110, center), press the *Center* button located at the bottom of the sliders window (Figure 110, left).

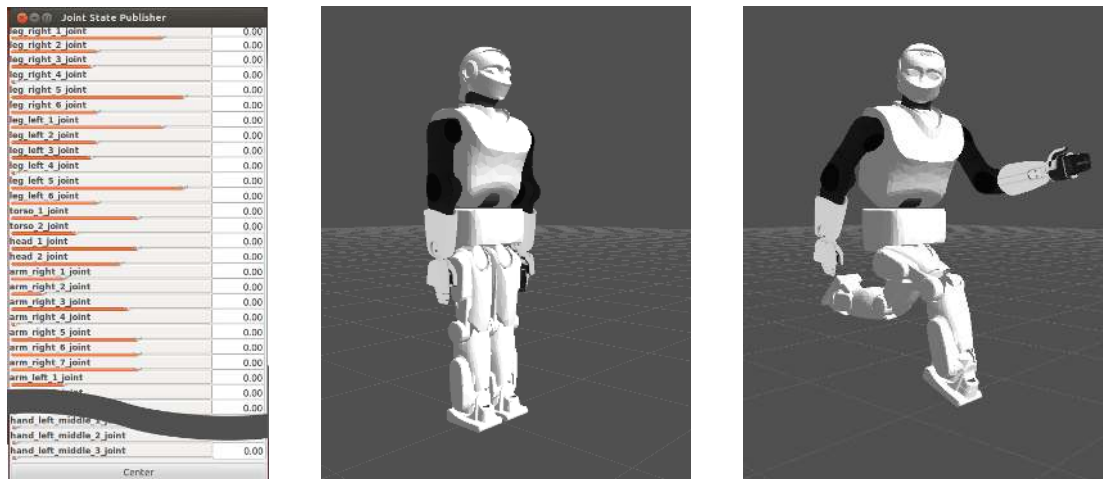


Figure 110: Slider interface for each joint (left, not all joints are shown for brevity). Two joint configurations: the default (center) and a custom one (right).

46.3 End test

Press `Ctrl-C` in the terminals used for this acceptance test.

47 Read sensors

Description The procedure described in this section tells how to read the sensors of REEM-C.

Requisites A running REEM-C, as described in section 40.

47.1 Setup

Open a terminal connected to REEM-C. Do not close the terminal as the examples must be run here.

47.2 Test the sensors

47.2.1 Test

1. Force torque sensors
 - (a) Execute the following command to monitor the values of the force torque sensor on the left ankle:


```
roslaunch reemc_basic_tutorials ankle_left_ft_plot.launch
```

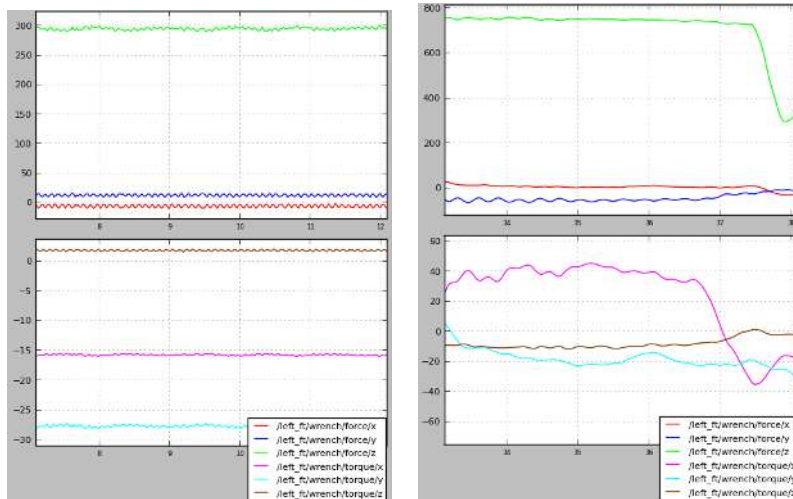



Figure 111: Visualization of force torque sensors with the robot in straight standing position(left) and with the feet pressed by hand(right).

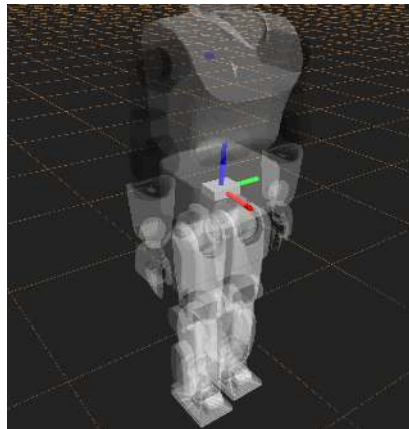


Figure 112: IMU sensor visualization in rviz.

- (b) To test the sensor apply small pressure on the left foot of the robot pushing it against the ground. See that the 'z' value in the 'force' section will increase as shown in Figure 111.
- (c) The right foot can be tested using the same process but executing:


```
roslaunch reemc_basic_tutorials ankle_right_ft_plot.launch
```
- (d) The left wrist force torque sensor can be tested using:


```
roslaunch reemc_basic_tutorials wrist_left_ft_plot.launch
```
- (e) The right wrist force torque sensor can be tested using:


```
roslaunch reemc_basic_tutorials wrist_right_ft_plot.launch
```

2. Inertial measurement unit

- (a) Start rviz to visualize the sensors by executing the following:


```
roslaunch reemc_basic_tutorials rviz.launch config:=imu
```

 The visualization of the IMU can be seen in the waist of REEM-C as depicted in Figure 112.
- (b) Apply a small push to the waist of REEM-C and watch how the displayed frames change.

3. Laser sensors

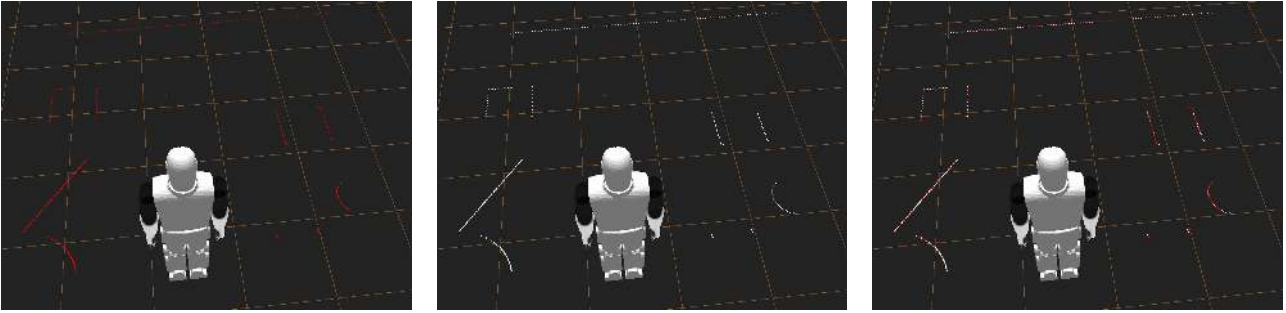


Figure 113: Laser sensors visualization. Left laser only (left), right laser only (middle), both lasers (right).

- (a) Start rviz to visualize the sensors by executing the following:

```
roslaunch reemc_basic_tutorials rviz.launch config:=lasers
```

The expected visualization can be seen in Figure 113.

- (b) Make sure that the laser readings are correct by putting objects on the floor in front of the robot or having a person walk by.

4. Ultrasound (sonar) sensors

- (a) Start rviz to visualize the sensors by executing the following command:

```
roslaunch reemc_basic_tutorials rviz.launch config:=sonars
```

The expected visualization is shown in Figure 114.

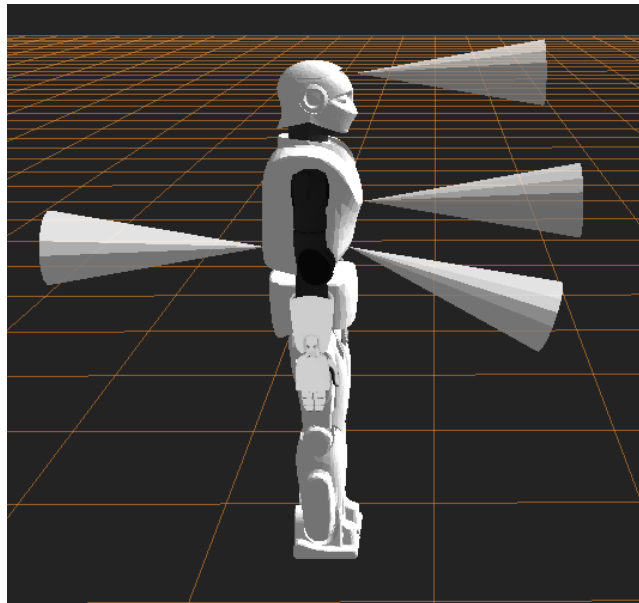


Figure 114: Sonar sensors visualization

- (b) Verify that each sensor reacts when an object is moved back and forth in front of them.

47.2.2 End of test

Close the current test by pressing `CTRL+C` in the terminal.

48 Moving individual joints – command line

Description This test covers how to move individual joints of REEM-C to a desired goal configuration using the command line. Generated single-joint trajectories are calculated by means of spline interpolation with zero-velocity endpoint constraints. Note that no motion planning is performed in this test, hence there are no collision avoidance or balance maintaining guarantees.

Requisites A running REEM-C robot, as described in section 40.

48.1 Usage

The `move_joint` command has the following syntax:

```
roslaunch reemc_controller_configuration joint_trajectory_controllers.launch
rosrun play_motion move_joint joint_name goal_position [duration]
```

- `joint_name` is the name of the joint to move.
- `goal_position` is the desired joint position (in radians).
- `duration` is the optional motion duration (in seconds). When unspecified, a reasonable duration will be automatically computed.

48.2 Example

48.2.1 Setup

1. Open a terminal connected to REEM-C, and start joint trajectory controllers for the full robot.

```
roslaunch reemc_controller_configuration joint_trajectory_controllers.launch
```

2. The expected command-line output should end by displaying the list of running controllers:

```
...
[INFO] [WallTime: 1.0] Started controllers: left_leg_controller,
right_leg_controller, torso_controller, head_controller, left_arm_controller,
right_arm_controller, left_hand_controller, right_hand_controller
```

48.2.2 Run the example

1. Open a terminal connected to REEM-C, and run `move_joint` to make REEM-C turn its head 30 degrees (0.524 radians) towards its left using the default duration:

```
rosrun play_motion move_joint head_1_joint 0.524
```

2. The expected goal configuration is shown in Figure 115, left, and the expected command-line output should look similar to the following:

```
Moving joint head_1_joint to position 0.524
```

3. Now make REEM-C look forward (joint at zero), this time with a custom duration of two seconds:

```
rosrun play_motion move_joint head_1_joint 0.0 2.0
```

4. The expected goal configuration is shown in Figure 115, right, and the expected command-line output should look similar to the following:

```
Moving joint head_1_joint to position 0.0 in 2.0s
```

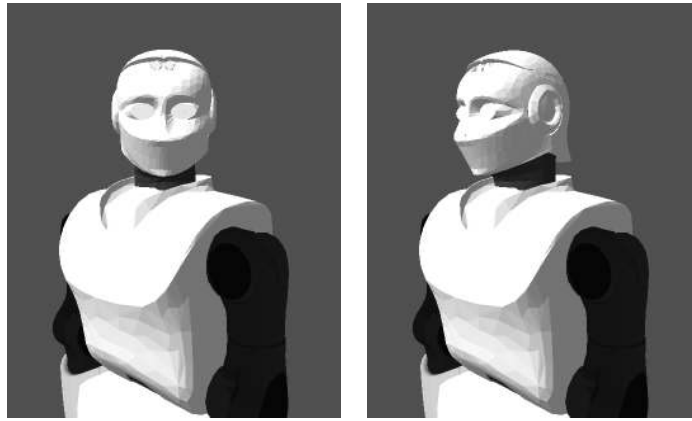


Figure 115: Moving a single joint: turn head to the left (left), and to the front (right).

48.3 A note on safe execution

`move_joint` is a tool intended to test low-level joint access. As such, the only safety checks that are enforced are **joint position and velocity limits**. Constraints involving higher-level functionality like **collision avoidance and maintaining balance are not enforced**. To make sure that a given motion will safely execute, consider the following recommendations:

- The kinematic workspace of REEM-C (ie. joint ranges, possible self-collisions) can be interactively inspected following the instructions of section 46.
- Test the motion in simulation.
- When moving individual leg joints, it is advisable to lift REEM-C from the ground with the crane, as it might lose its balance.

48.4 Testing all joints

There are scripts in the `reemc_motion_tutorials/scripts` folder whose purpose is to exercise all robot joints in a safe sequence that does not cause self-collisions. Make sure that the full body joint trajectory controllers are running (section 48.2.1) before executing these scripts.

48.4.1 Upper body

- For testing the joints in the **upper body** (ie. head, arms, hands and torso), REEM-C should be **secured to the crane** and **standing with its feet on the ground**. Then run:

```
rosrun reemc_motion_tutorials test_upper_body_joints
```

- The script will prompt for confirmation that REEM-C meets these requisites. Answer **yes** and press enter:

```
Is the robot secured to the crane and standing with its feet on the ground? [yes/no]
```

- In case of an affirmative answer, the test will begin and you should see the following output as REEM-C moves (cropped with '...' for brevity):



Figure 116: Exemplary robot configuration before running the lower body joints test.

```
Going to home pose...
Done!
Testing head joints...
Moving joint head_1_joint to position 0.5
...
Testing left arm joints...
...
...
Upper body joints test finished successfully in 3m 8.17s.
```

48.4.2 Lower body

- As opposed to the upper body test, for testing the joints in the **lower body** (ie. legs), REEM-C should be **secured to the crane** and **lifted so its feet do not touch the ground**. It should be oriented with respect to the crane as shown in Figure 116. Then run⁶:

```
roslaunch reemc_motion_tutorials test_lower_body_joints
```

- The script will prompt for confirmation that REEM-C meets these requisites. Answer **yes** and press enter:

```
Is the robot secured to the crane and lifted so its feet do not touch the ground?
[yes/no]
```

- In case of an affirmative answer, the test will begin and you should see the following output as REEM-C moves (cropped with '...' for brevity):

```
Going to home pose...
Done!
Testing left leg joints...
Moving joint leg_left_4_joint to position 1.5708
...
Testing right leg joints...
...
...
Lower body joints test finished successfully in 1m 22.28s.
```

⁶The lower body joints test cannot be executed in simulation, as the robot would fall.

48.5 End test

Press `Ctrl-C` in the terminal used to start the controllers.

49 Moving individual joints – graphical user interface

Description This test covers how to move REEM-C's individual joints to a desired goal configuration using a graphical user interface. This test is similar to the one presented in section 49, the only difference being the mechanism used to send motion commands.

Requisites A running REEM-C, as described in section 40.

49.1 Setup

49.1.1 Launch the GUI

1. Launch the Rqt GUI and joint trajectory controllers, as described in section 44.1.2 and shown in Figure 107.
2. On the top-left of the GUI is the panel for controlling joint positions, named `Joint trajectory controller`, detailed in Figure 117. Activate the right hand controller by selecting `/controller_manager` and `right_hand_controller` on the panel's combo boxes, as shown below.



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

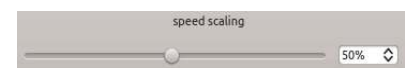
3. By default, the controller panel is in **monitor mode**, as indicated by the red status button shown below. In monitor mode the joint sliders display the current joint positions but do not accept new commands. By clicking on the status button, the controller panel enters **control mode**, and the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



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



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



6. As the operator sends commands to the joints, their time evolution can be monitored both in the data plots and the 3D rendering of the robot, as depicted in Figure 107.



Figure 117: Detail of the joint control panel in monitor (left) and control (right) modes.

Note In control mode, joint commands sent using other mechanisms (such as those presented in sections 48, 51 or 52) will be aborted. Make sure to switch back to monitor mode before sending commands through another mechanism.

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

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

49.2 End test

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

50 Setting current limits – graphical user interface

Description This test covers how to set the maximum electrical current individual actuators REEM-C can apply, using a graphical user interface. This test is similar to the one presented in section 49, but commands actuator current limits instead of joint positions. This test applies only to real REEM-C deployments, not to simulated ones.

Requisites A running REEM-C, as described in section 40.

50.1 Setup

50.1.1 Launch the GUI

1. Launch the Rqt GUI, as described in section 44.1.2 and shown in Figure 107.
 - (a) On the bottom-left of the GUI is the panel for controlling actuator current limits, named `Current limit controller`, detailed in Figure 118. Activate the hand controller by selecting `/controller_manager` and `right_hand_current_limit_controller` on the panel's combo boxes, as shown below.



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

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



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



- Actuator currents being applied to the actuators are shown in the data plot of Figure 107 (center, bottom). Plot values are not normalized, and represent actual current values.

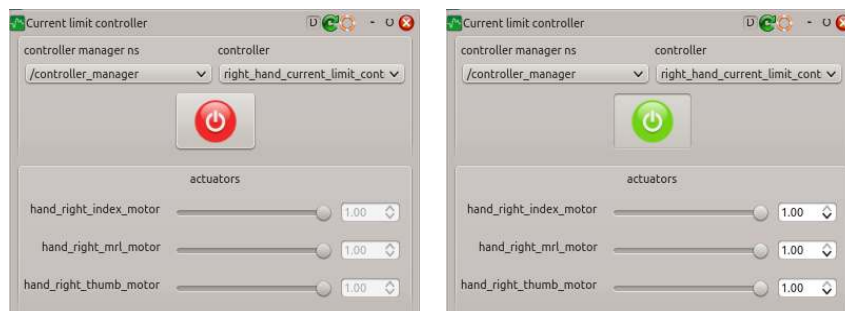


Figure 118: Detail of the joint control panel in monitor (left) and control (right) modes.

50.2 End test

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

s

51 Perform simple upper-body motions.

Description This test covers how to move the upper-body joints of REEM-C to perform a set of pre-defined motions. Generated trajectories are calculated by means of spline interpolation with zero-velocity endpoint constraints. Collision-free motion planning is performed between the actual robot state and the initial state of the motion (the approach), but is not performed for the pre-defined part of the motion. The provided list of motions has been tested to be collision-free.

Requisites A running REEM-C robot, as described in section 40, with its feet touching the ground.

51.1 Setup

1. Open a terminal connected to REEM-C, and launch the default robot controllers.

```
roslaunch reemc_controller_configuration default_controllers.launch
```

Do not close this terminal as the joint trajectory controllers must be running during this acceptance test.

51.2 Querying and executing motions from the command line

1. Open a terminal connected to REEM-C.
2. The `play_motion` package is used to execute motions. Available motions are loaded as ROS parameters into the `/play_motion/motions` namespace, and their names can be queried from the command line:

```
rosparam list /play_motion/motions | grep joints
```

3. The details of a particular motion like its joints and trajectory waypoints can also be introspected. For instance, the `wave` motion is queried as:

```
rosparam get /play_motion/motions/wave -p
```

which outputs (contents clipped for brevity)

```
joints: [torso_1_joint, torso_2_joint, head_1_joint, ...
...hand_right_index_joint, hand_right_mrl_joint]
meta:
  description: Waves with right hand and nodes.
  name: Wave
  usage: greet
points:
- positions: [0.0, 0.0, 0.0, 0.0, -0.4, 0.25, -0.1, 0.6109,
0.0, 0.0, 0.0, -0.4, 0.25, -0.1, 0.6109, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
  time_from_start: 0.0
...
- positions: [0.0, 0.0, 0.0, 0.0, -0.4, 0.25, -0.1, 0.6109,
0.0, 0.0, 0.0, -0.4, 0.25, -0.1, 0.6109, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
  time_from_start: 8.3
```

1. To trigger the execution of a particular motion from the command line, such as `wave`, execute the following command⁷

```
axcli /play_motion "motion_name: 'wave'"
```

Pressing `Ctrl-C` while a motion is active will immediately stop its execution.

2. To trigger the execution of a particular motion **skipping the motion planning phase**, execute the following command (for the `arms_t` motion)

```
axcli /play_motion "motion_name: 'arms_t', skip_planning: true"
```

Safety note: Motions should always be executed with motion planning enabled (the default). The only circumstances in which motion planning is skipped is when the planner fails to obtain a valid motion plan. If you decide to execute a motion skipping the planning phase, make sure that it can be safely executed without the robot coming in collision with itself or the environment. Testing motions in simulation is one way to perform this validation.

If you need to move the robot away from a configuration it cannot plan from, try the above `arms_t` motion, which avoids most self-collision scenarios. Be prepared to trigger an e-stop whenever the robot unexpectedly collides against itself or an obstacle.

51.3 Triggering motions with the joystick

1. Open a terminal connected to REEM-C.
2. Execute the `wave` motion and monitor its execution. For this, press the green button while holding the upper right button.



Figure 119: Upper right button (RB) and trigger (RT)

3. The robot will then perform a waving motion.
4. Execute other example motions





Motion	Joystick buttons	Description
Wave	RB + 	REEM-C greets you waving its right arm
Shake hand	RB + 	REEM-C extends its right hand to shake yours
Bow	RB + 	REEM-C bows respectfully
Open arms	RB + 	REEM-C open its arms

Table 18: Motion joystick triggers.

⁷`axcli` is a command-line tool for sending goals to an action server. Run `axcli -h` for advanced usage options.

Figure 120: Waving motion

The configuration files defining these joystick triggers can be found on the following file:

```
ls `rospack find reemc_bringup`/config/joy_teleop.yaml
```

51.4 Example motion sequence

1. To demonstrate the robot performing a selection of the available motions, stand back from the robot so it can use its entire workspace and run the following command

```
roslaunch reemc_motion_tutorials gesture_sequence.launch
```

which outputs (text clipped for brevity)

```
[INFO] Waiting for valid clock...
[INFO] Clock is valid!
[INFO] Executing motion 'open_both_hands'...
[INFO] Executing motion 'interact'...
[INFO] Executing motion 'wave'...
...
[INFO] Executing motion 'thumbs_up_right_arm'...
[INFO] Executing motion 'open_both_hands'...
[INFO] Executing motion 'interact'...
```

51.5 A note on unexpected hand contacts

When the hand is moving in conditions where unexpected environment interactions might occur (contacts, collisions), there are two recommended strategies:

Fist configuration A closed hand configuration (Fig. 121), has a small footprint in a plane parallel to the palm surface, and lends itself to *avoiding* contact, especially with objects that could become tangled between the fingers.

- To bring the right hand to a fist configuration, execute the following command

```
axcli /play_motion "motion_name: 'fist_right_hand'"
```



Figure 121: Hand in fist configuration.

Fully open configuration An open hand configuration has a large footprint in the plane parallel to the palm surface. For each of the Hey5 hand actuators, finger motion takes place in the positive range of motion, while the negative range of motion unwinds and slackens the tendons, causing no finger motion but allowing a more compliant interaction with the environment⁸. Using the negative range of motion of an actuator allows to control the amount of compliance and hyperextension its associated joints can achieve. To illustrate this behavior, Fig 122 compares the response of a disturbance applied to the index finger with actuators set to zero (taut) and fully unwound (slack) configurations. Fig 123 shows additional disturbances applied to a hand with slack tendons.

- To bring the right hand⁹ to a fully open configuration where the tendons are *taut*, run

```
axcli /play_motion "motion_name: 'open_right_hand'"
```

- To bring the right hand to a fully open configuration where the tendons are *slack*, run

```
axcli /play_motion "motion_name: 'open_right_hand_slack'"
```

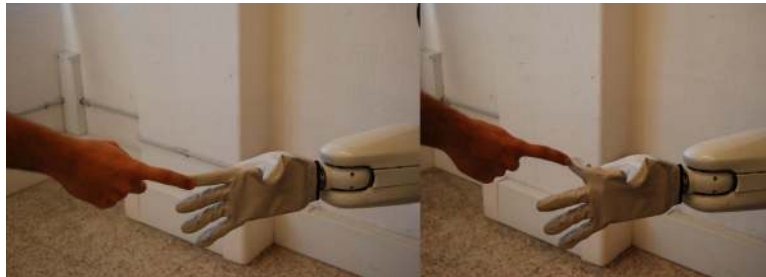


Figure 122: External disturbance applied to index finger with taut (left) and slack tendon (right).



Figure 123: External disturbances applied to hand with slack finger tendons. From left to right: Open hand, index hyperextension, middle and ring hyperextension, abduction disturbance.

Because of the way the middle, ring and little fingers are coupled together through a single tendon, it is possible to reject strong disturbances affecting one or two of the fingers even when the driving tendon is taut. Making this particular tendon slack is relevant for rejecting disturbances that affect the three fingers *simultaneously*.

⁸Refer to the Hey5 hand user manual for more details.

⁹Similar motions exist for the left hand as well.

51.6 Note on ABORTED hand goals

Motions that include the hands may return an *ABORTED* status as they may not reach the motor position defined in the tolerance of a motion, this is expected as it is a consequence of defining a maximum current on the hand for safety reasons (to lengthen the life of the tendons and to not apply excessive pressure on grasped items). Keeping the final position of a hand motion far enough from its joint limit will make this behaviour stop for motions that have an aesthetical goal.

51.7 End test

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

52 Grasping objects

Description This test describes the usage of a controller for performing object grasping, and presents a set of exemplary graspable objects REEM-C can hold. An interactive demonstration with actual hardware is required to fully validate it.

Requisites A running REEM-C, as described in section 40.

Source code The source code for the controller used this test can be found in the `simple_grasping_action` package, while the source code of the examples belong to `reemc_motion_tutorials`.

52.1 Usage

The `simple_grasping_action` package implements a very simple grasping controller that closes the fingers until either a desired actuator position or effort is reached, such as when touching an object. Setting different values for the desired effort allows to control the firmness of the grasp.

An instance of the `simple_grasping_action` sits on top of a `joint_trajectory_controller`, and exposes the same `FollowJointTrajectory` action interface. Goals consist of single-waypoint trajectories with a position goal and an optional effort threshold. Joints move at constant velocity, so the position goal does not require specifying velocity or acceleration boundary constraints.

Typically the position goal consists of a closed hand configuration, while the effort goal is expressed in actuator current units, with values belonging to the range specified in the Hey5 hand user manual. Note that although smaller effort goal values correspond to softer grasps, they cannot be too small, otherwise the fingers will stall prematurely.

The script in `reemc_motion_tutorials/scripts/simple_grasp` leverages the `simple_grasping_action` to implement a straightforward grasping strategy with all fingers, and has the following syntax:

```
roslaunch reemc_motion_tutorials simple_grasp [max_curr_fraction]
```

- `max_curr_fraction` is an optional parameter for specifying the effort goal. It represents the fraction of the maximum actuator current. Values belong to the $[0, 1]$ interval, where 0 means zero current, and 1 means the maximum actuator current. For grasping, it is typical to specify values in the $[0.8, 1]$ range.

52.2 Graspable objects

A set of six graspable objects have been chosen to validate the hand design (Fig. 124):

1. Soda can
2. Tennis ball
3. 1kg dumbbell
4. Cellphone
5. Pliers
6. Spray cleaner

The *soda can* and *tennis ball* were chosen because they represent simple cylindrical and spherical objects with a size typical of household objects. The *1kg dumbbell* allows to validate the payload capabilities of the hand/arm. The *cellphone* was chosen because of its flat shape. Finally, the *pliers* and *spray cleaner* represent more complex objects, which are articulate and have an irregular form. Fig. 124 depicts the objects and their characteristic dimensions.

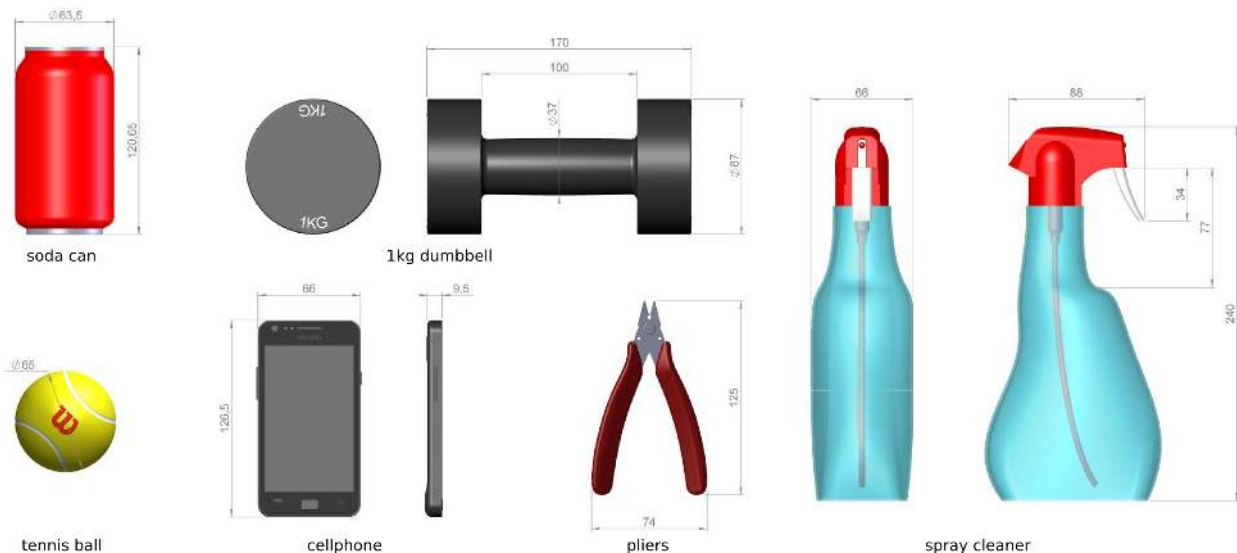


Figure 124: Graspable objects and their estimated characteristic dimensions.

52.3 Example

1. Open two terminals connected to REEM-C.
2. On the first terminal, launch the graphical user interface presented in section 49.
3. On the second terminal, bring the arm and hand to a position suitable for holding objects in a side-grasp, as shown in Fig. 125, left.

```
axcli /play_motion "motion_name: 'interact'"
axcli /play_motion "motion_name: 'side_grasp_right'"
```

Note that the hand is in an open configuration with the thumb opposing the other fingers, as shown in Fig. 125, top right.

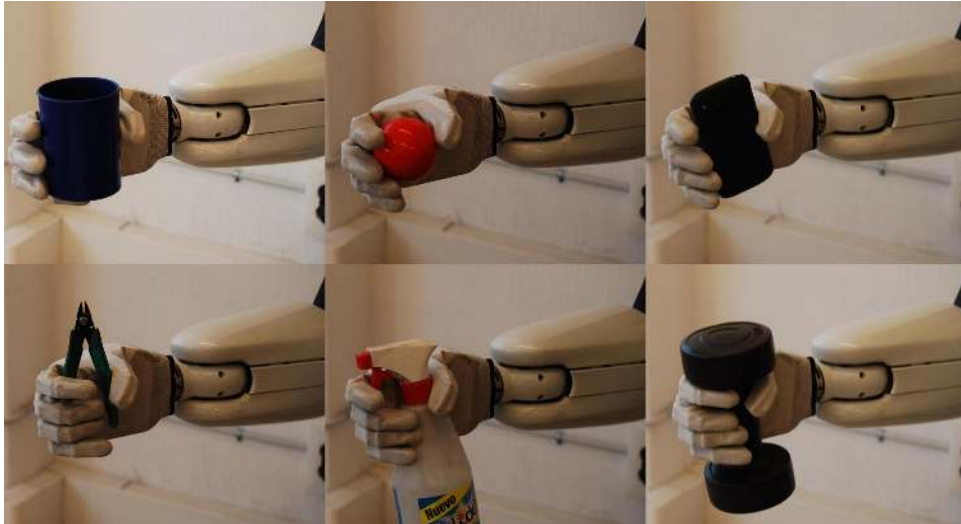


Figure 126: Hand grasping the different objects from Fig. 124 with a side-grasp strategy.

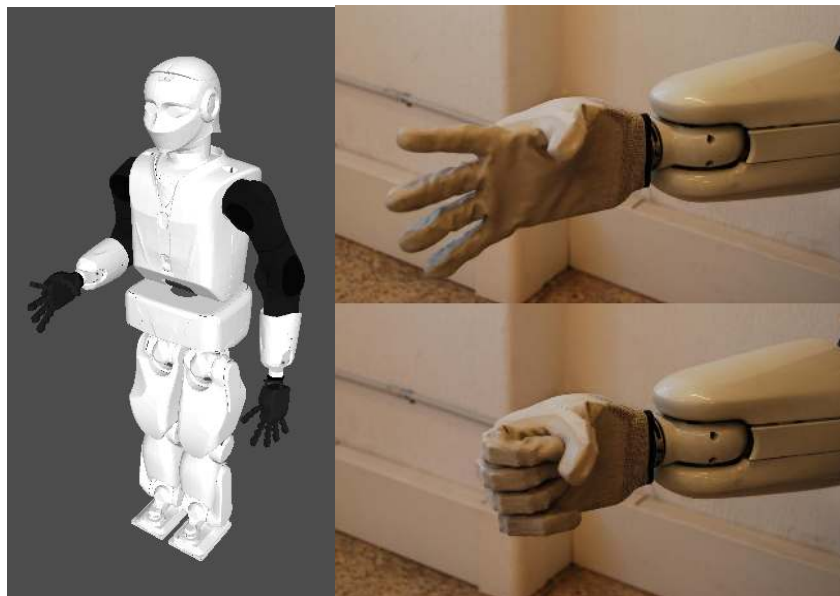


Figure 125: Arm in a side-grasping configuration (left). Gloved hand in pre-grasp configuration (top right). Gloved hand in position goal of grasping motion (bottom right).

- Place an object in front of the hand palm for grasping, and use the `simple_grasping_action` to perform a grasping motion with an effort goal of 80% the maximum actuator currents.

```
roslaunch reemc_motion_tutorials simple_grasp 0.8
```

Fig. 126 displays the hand grasping the different objects from Fig. 124. If no object is placed for grasping, the hand will close to a configuration similar to the one shown in Fig. 125, bottom right.

- Release the grasp¹⁰ by bringing the hand back to the pre-grasp configuration of Fig. 125, top right.

```
axcli /play_motion "motion_name: 'pregrasp_right_hand'"
```

Note that this motion is not executed by the `simple_grasping_action`.

¹⁰If the grasp is released and the object is not supported (e.g., by a person or surface), it will fall. Take the necessary precautions to prevent damaging the object and its surrounding environment, including people.

6. For grasping and releasing new objects, repeat steps 4 and 5 as many times as needed. Heavier objects like the dumbbell require an effort goal higher than 80%.
7. To validate different arm configurations for grasping, activate the `right_arm_controller` in the graphical user interface launched in step 2 and bring the arm to the new desired configuration. Fig. 127 shows the dumbbell being grasped at different hand orientations.
8. Bring REEM-C to the interact configuration.

```
axcli /play_motion "motion_name: 'open_right_hand'"
axcli /play_motion "motion_name: 'interact'"
```



Figure 127: Dumbbell being grasped at different hand orientations.

52.4 End test

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

53 Control REEM-C with keyboard or joystick input

Description This test explains how to control the walking of REEM-C with the keyboard or a joystick, either in a Gazebo simulated indoor environment or on the real robot.

Requisites A running REEM-C robot, as described in section 40, and a successful build of the `reem_basic_tutorials` package, as per the Setup section of the Overview.

53.1 Setup

1. Open a terminal connected to REEM-C, and launch the walking controller.

```
roslaunch reemc_controller_configuration default_controllers.launch
```

2. To use the keyboard to control REEM-C, launch a `keyboard_teleop` application.

```
roslaunch reemc_basic_tutorials keyboard_teleop.launch
```

The user should see in the terminal an interface similar to Figure 128. Following the on-screen instructions allows the user to send control commands to the walking controller. Remember that in order to activate those key, the focus of the Linux system should be on the window where you executed the previous command.

3. To use a USB joystick (as the one shown in Figure 129) to control REEM-C, just plug one. Moving the analog sticks while keeping pressed the upper left button (LB) the user to make REEM-C step forward and to turn (using only the left hand side stick). To make REEM-C step sideways, move the right analog stick to the left or to the right while keeping the upper left button (LB) pressed. REEM-C will stop walking automatically after the deadman switch (LB button) has been released, or after not receiving any signal from the joystick for more than one second.



Figure 128: USB joystick controller

NOTE: Make sure that the small switch at the top of the controller is on the 'D' position, and not the 'X'. The 'X' position may produce strange behaviors and lead to unsafe operation of REEM-C.



Figure 129: USB joystick controller

53.2 End test

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

54 Leg inverse kinematics library

Description This example shows how to use analytic inverse kinematics library for REEM-C legs.

54.1 Example code

The code used for this example is available in `pal_walking_tutorials/src` folder, in the `ik_example.cpp` file.

```
#include <walking_controller/ik/pair_ik_leg_analytic_kajita.h>
#include <walking_controller/reemc_definitions.h>

int main(int argc, char **argv) {

    /// Instantiating an object IK for legs
    /// Rototranslation matrix from base_link to first joint, femur and tibia length are provided
    LegsPairIKAnalyticKajita legs_kinematics(pal::center_to_left_matrix,
                                             pal::center_to_right_matrix,
                                             pal::FEMUR_LENGTH,
                                             pal::TIBIA_LENGTH);

    /// Left foot pose
    eVector3 leftFootPos(0.0, pal::HIP_SPACING, 0.0);
    eVector3 leftFootRPY(0.0, 0.0, 0.0);

    /// Right foot pose
    eVector3 rightFootPos(0.0, -pal::HIP_SPACING, 0.0);
    eVector3 rightFootRPY(0.0, 0.0, 0.0);

    /// CoM pose
    eVector3 hipPos(0.0, 0.0, pal::TIBIA_LENGTH + pal::FEMUR_LENGTH + pal::FOOT_HEIGHT - 0.001);
    eVector3 hipRPY(0.0, 0.0, 0.0);

    /// Setting IK with CoM and feet poses
    legs_kinematics.setBaseFrameCoord(createMatrix(hipRPY, hipPos));
    legs_kinematics.setFootCoord(pal::LEFT, createMatrix(leftFootRPY, leftFootPos), pal::ankle_to_foot_center[pal::LEFT]);
    legs_kinematics.setFootCoord(pal::RIGHT, createMatrix(rightFootRPY, rightFootPos), pal::ankle_to_foot_center[pal::RIGHT]);

    std::vector<double> leftAngles(6,0.0);
    std::vector<double> rightAngles(6,0.0);

    /// Compute inverse kinematics for left leg joints
    legs_kinematics.ik_analytic(pal::LEFT, &leftAngles[0]);

    /// Compute inverse kinematics for right leg joints
    legs_kinematics.ik_analytic(pal::RIGHT, &rightAngles[0]);

    return 0;
}
```

54.2 Build and execute the example

In the following we assume that `pal_walking_tutorials` package is in `~/pal_ws/src`.

To build the tutorials execute:

```
cd ~/pal_ws
catkin_make
```

Once successfully built, to execute the example run the following commands:

```
source ~/pal_ws/devel/setup.bash
roslaunch pal_walking_tutorials walk_ik_example
```

The execution of previous commands in a terminal should produce the this output:

```
CoM      pos (0, 0, 0.709)      rpy (0, 0, 0)
Left foot pos (0, 0.0725, 0)  rpy (0, 0, 0)
Right foot pos (0, -0.0725, 0) rpy (0, 0, 0)
Left leg joints (-0, 0, -0.057743, 0.115486, -0.057743, 0)
Right leg joints (-0, 0, -0.057743, 0.115486, -0.057743, 0)
```

The first three lines of the output represent the CoM and feet cartesian poses used in the example code, while the last two lines represent the angles for leg joints found by the inverse kinematic algorithm.

55 Walking safety library

Description This example shows how to use the walking safety library to avoid leg self collisions when operating REEM-C .

55.1 Example code

The code used for this example is available in `pal_walking_tutorials/src` folder, in the `safety_example.cpp` file.

```
#include <ros/ros.h>
#include <walking_controller/safety.h>
#include <sensor_msgs/JointState.h>

int main(int argc, char **argv) {

ros::init(argc, argv, "safety_example");
ros::NodeHandle nh;

sensor_msgs::JointState lower_body_joint_states;

// Setup joint states message for leg joints
lower_body_joint_states.name.resize(12);
lower_body_joint_states.position.resize(12,0.0);
lower_body_joint_states.velocity.resize(12, 0.0);
lower_body_joint_states.effort.resize(12), 0.0;

lower_body_joint_states.name[0] = "leg_left_1_joint";
lower_body_joint_states.name[1] = "leg_left_2_joint";
lower_body_joint_states.name[2] = "leg_left_3_joint";
lower_body_joint_states.name[3] = "leg_left_4_joint";
lower_body_joint_states.name[4] = "leg_left_5_joint";
lower_body_joint_states.name[5] = "leg_left_6_joint";
lower_body_joint_states.name[6] = "leg_right_1_joint";
lower_body_joint_states.name[7] = "leg_right_2_joint";
lower_body_joint_states.name[8] = "leg_right_3_joint";
lower_body_joint_states.name[9] = "leg_right_4_joint";
lower_body_joint_states.name[10] = "leg_right_5_joint";
lower_body_joint_states.name[11] = "leg_right_6_joint";

// Control loop duration (10 ms) only used for velocity estimation
ros::Duration dT(0.01);

// Creation of BipedSafety object
```

```

pal :: BipedSafety biped_safety(&nh, &lower_body_joint_states, dT);

/// init time
ros :: Time time(0.0);

/// Check for safety of the starting configuration ( all joints at zero)
bool is_safe = biped_safety.is_safe(lower_body_joint_states.position, time);

if (is_safe)
  ROS_INFO_STREAM("First configuration is safe");
else
  ROS_INFO_STREAM("First configuration is not safe");

/// Setting a joints configuration that would cause a self collision
lower_body_joint_states.position[0] = 0.0;
lower_body_joint_states.position[1] = -0.25;
lower_body_joint_states.position[2] = 0.0;
lower_body_joint_states.position[3] = 0.0;
lower_body_joint_states.position[4] = 0.0;
lower_body_joint_states.position[5] = 0.0;
lower_body_joint_states.position[6] = 0.0;
lower_body_joint_states.position[7] = 0.25;
lower_body_joint_states.position[8] = 0.0;
lower_body_joint_states.position[9] = 0.0;
lower_body_joint_states.position[10] = 0.0;
lower_body_joint_states.position[11] = 0.0;

time+=dT;
/// Check for collision
is_safe = biped_safety.is_safe(lower_body_joint_states.position, time);

if (is_safe)
  ROS_INFO_STREAM("Second configuration is safe");
else
  ROS_INFO_STREAM("Second configuration is not safe");

/// Setting a joint position outside joint limit
lower_body_joint_states.position[0] = 0.0;
lower_body_joint_states.position[1] = -0.3;
lower_body_joint_states.position[2] = 0.0;
lower_body_joint_states.position[3] = 0.0;
lower_body_joint_states.position[4] = 0.0;
lower_body_joint_states.position[5] = 0.0;
lower_body_joint_states.position[6] = 0.0;
lower_body_joint_states.position[7] = 0.0;
lower_body_joint_states.position[8] = 0.0;
lower_body_joint_states.position[9] = 0.0;
lower_body_joint_states.position[10] = 0.0;
lower_body_joint_states.position[11] = 0.0;

time+=dT;
/// Check for safety (one joint will violate its position limit)
is_safe = biped_safety.is_safe(lower_body_joint_states.position, time);

if (is_safe)
  ROS_INFO_STREAM("Third configuration is safe");
else
  ROS_INFO_STREAM("Third configuration is not safe");

/// Configuration that would violate speed limit for 4th left leg joint
lower_body_joint_states.position[0] = 0.0;
lower_body_joint_states.position[1] = 0.0;
lower_body_joint_states.position[2] = 0.0;
lower_body_joint_states.position[3] = 1.2;
lower_body_joint_states.position[4] = 0.0;
lower_body_joint_states.position[5] = 0.0;
lower_body_joint_states.position[6] = 0.0;
lower_body_joint_states.position[7] = 0.0;
lower_body_joint_states.position[8] = 0.0;
lower_body_joint_states.position[9] = 0.0;

```

```

lower_body_joint_states.position[10] = 0.0;
lower_body_joint_states.position[11] = 0.0;

    /// Check for speed limit violations
is_safe = biped_safety.velocity_limits_respected(lower_body_joint_states.position);
if (is_safe)
    ROS_INFO_STREAM("Fourth configuration is safe");
else
    ROS_INFO_STREAM("Fourth configuration is not safe");
return 0;
}

```

55.2 Build and execute the example

In the following we assume that `pal_walking_tutorials` package has been successfully built and sourced as explained in Subsection 54.2.

Robot description should be loaded in the param server, executing from a terminal:

```
roslaunch reemc_description upload_reemc.launch
```

To run the example, execute from the terminal in which `pal_walking_tutorials` workspace has been sourced:

```
roslaunch pal_walking_tutorials walk_safety_example
```

The execution of previous commands in a terminal should produce the following output:

```

[ INFO] [0.0]: First configuration is safe
[ERROR] [0.0]: Collision detected between :leg_left_3_link and leg_right_3_link
[ERROR] [0.0]: 0.00693518 0.0330352 -0.238716
[ INFO] [0.0]: Second configuration is not safe
[ERROR] [0.0]: joint: leg_left_2_joint violates minimum angle,
             current is: -0.3 and minimum is: -0.261999
[ INFO] [0.0]: Third configuration is not safe
[ERROR] [0.0229189]: leg_left_2_joint speed is 30, max is 2.27 diff is 27.73
[ INFO] [0.0]: Fourth configuration is not safe

```

The first three lines of the output represent the CoM and feet cartesian poses used in the example code, while the last two lines represent the angles for leg joints found by the inverse kinematic algorithm.

56 Walking examples

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

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

Requisites A running REEM-C robot, as described in section 40.

56.1 Setup

56.1.1 Notes when running on the real robot

If running this test on real hardware, REEM-C must be turned on hanging on the crane with the feet not touching the ground before it starts walking.

56.1.2 Starting the walking controller

The following commands in the manual are independent of the real robot or simulation.

In order to start the `walking_controller`, the parameters have to be loaded in the parameter server and the controller has to be started. REEM-C must be started in a upright position with the knees extended as shown in Figure 130 (a), once the `walking_controller` starts it will place REEM-C in the walking configuration, with the knees bent and the arms in walking position depicted in Figure 130 (b). Once the controller has started, REEM-C can be lowered from the crane making the feet touch the ground and with enough slackness on the safety ropes so they don't pull the robot while walking.

```
roslaunch reemc_controller_configuration default_controllers.launch
```

The `default_controllers` launch file will start `joint_trajectory_controllers` for the upper body, and the `walking_controller` for the lower body. This is the controllers configuration that will be used most frequently in the robot, hence its name. A stabilizer algorithm is included in the walking controller.

Don't execute anything in this terminal and leave it open during all the walking process, open a new terminal for each test and only close this terminal when you want to stop the `walking_controller` and the `joint_trajectory_controllers`.

(a)

(b)

Figure 130: Walking configuration. Start position (a). Walking controller started (b).

56.2 Walking using topic interface

Description This acceptance test will show how to send velocity commands to REEM-C using velocity references for the center of mass. REEM-C will generate the steps automatically in order to try to track this reference velocity. The test will command REEM-C to go in the front, back, sideways direction and turn.

Source code The source code for this test can be found in

`pal_walking_tutorials/src/walk_client_example_topic.cpp`.

56.2.1 Setup

1. Open a terminal connected to REEM-C, and launch the `walking_client_topic` application as follows

```
roslaunch pal_walking_tutorials walk_client_example_topic
```

56.3 Walking using a service interface

56.3.1 Setup

Description This acceptance test will show how to send a parametrized list of steps to REEM-C . The parameters are the step length, step time and number of steps. REEM-C will add the appropriate steps to its step list and execute them. The test will add 5 steps forward and 5 steps backwards.

Source code The source code for this test can be found in

`pal_walking_tutorials/src/walk_client_example_service.cpp`.

56.3.2 Setup

1. Open a terminal connected to REEM-C, and launch the `walking_client_service` application as follows

```
roslaunch pal_walking_tutorials walk_client_example_service
```

56.4 Walking using an action interface

Description This acceptance test will show how to send commands to REEM-C specifying directly the foot steps. REEM-C will place its feet on the corresponding desired feet positions. This action server should be used with caution since is up to the user to specify a valid sequence of steps. This action service is meant as an interface to a step planner that outputs a valid step sequence given a goal. The action server has feedback on the status of the sent command. This test will generate a step list that resembles the previous test, it commands five steps forward and reports feedback on the steps that have been executed.

Source code The source code for this test can be found in

`pal_walking_tutorials/src/walk_client_example_action.cpp`.

56.4.1 Setup

1. Open a terminal connected to REEM-C, and launch the `walking_client_action` application as follows

```
roslaunch pal_walking_tutorials walk_client_example_action
```

56.5 End test

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

56.6 Whole Body Control

Description This examples show how to use PAL Robotics whole body control framework.

56.6.1 REEM-C with interactive markers

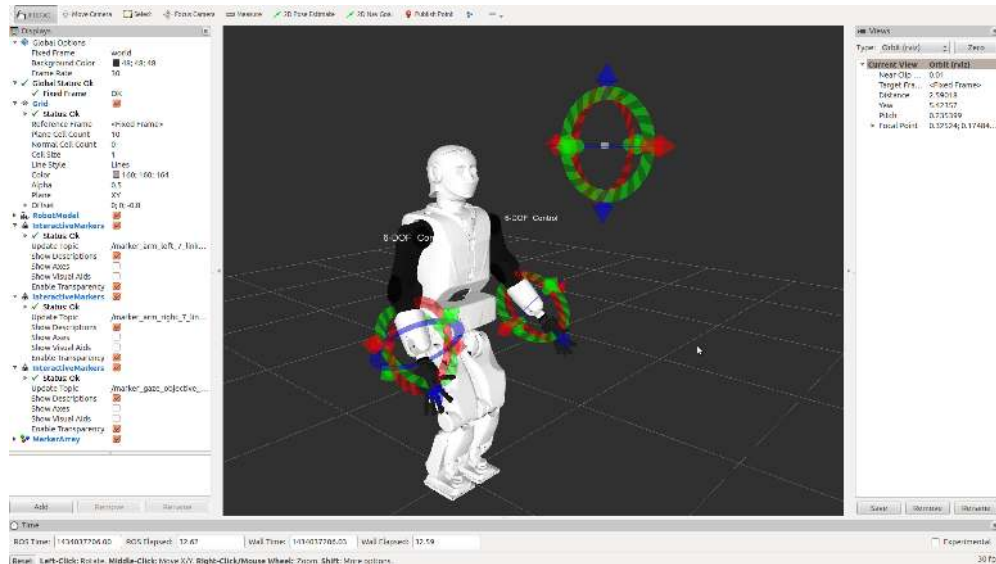


Figure 131: Visualization with interactive markers to control the robot

The following launch file launches a simple interactive demo to show the potential of the framework:

```
roslaunch reemc_wbc_tutorials reemc_wbc_standalone.launch
```

56.6.2 Dynamic simulation in gazebo

To reproduce the above example using a dynamic simulation launch the simulator:

```
roslaunch reemc_gazebo reemc_empty_world.launch
```

Once the simulation has started launch the controller in a separate terminal:

```
roslaunch reemc_wbc_tutorials reemc_wbc_ft_imu.launch
```

Launch in another terminal `Rviz` to command the robot:

```
roslaunch reemc_wbc_tutorials rviz.launch
```

56.6.3 REEM-C dancing

56.6.4 Dancing in dynamic simulation

The following example moves the robot by updating the references to the tasks through topics. This stack constrains the base orientation of the robot in order to give it stability and avoid torque limits. First launch the simulation of the robot:


```
roslaunch reemc_gazebo reemc_empty_world.launch
```

Launch the controller in a separate terminal:

```
roslaunch reemc_wbc_tutorials reemc_wbc_humanoids_ft_imu.launch
```

Launch the visualizer in a separate terminal :

```
roslaunch reemc_wbc_tutorials rviz.launch
```

Launch in a separate terminal the example node that generates the comands:

```
roslaunch reemc_wbc_tutorials wbc_reemc_hardware_test
```

56.6.5 Dancing with real robot

To reproduce the results in the robot, the same commands above have to executed except that instead of the simulation all the terminals must have the `ROS_MASTER_URI` pointing to the robot. **Be sure to leave enough slack of the wire that holds the robot**, since the robot will move up and down.

Set the `ROS_MASTER_URI` pointing to the robot:

```
export ROS_MASTER_URI=http://reemc-6c:11311
```

Launch the controller in a separate terminal:

```
roslaunch reemc_wbc_tutorials reemc_wbc_humanoids_ft_imu.launch
```

Launch the visualizer in a separate terminal :

```
roslaunch reemc_wbc_tutorials rviz.launch
```

Launch in a separate terminal the example node that generates the commands:

```
roslaunch reemc_wbc_tutorials wbc_reemc_hardware_test
```

REEM C

Customer Service

PALO
ROBOTICS 

57 Customer service

57.1 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 132 shows the initial page of the site.

New accounts will be created on request by PAL Robotics.

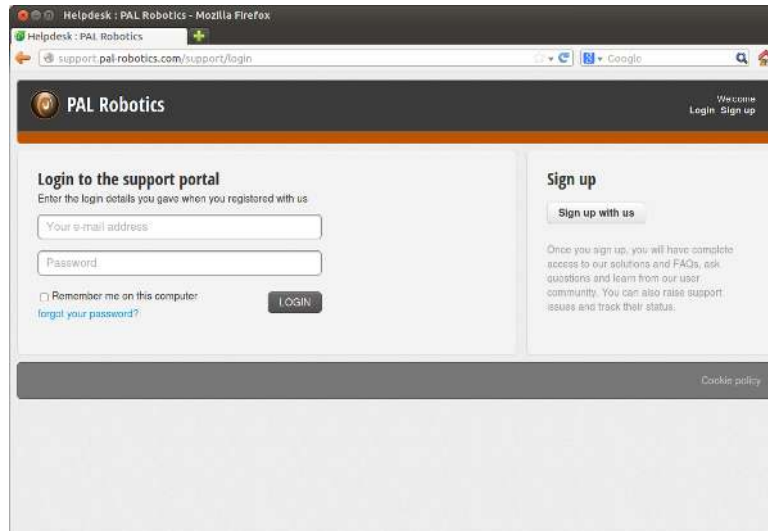


Figure 132: PAL Robotics support website

Once the customer has entered the system (Figure 133), 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.

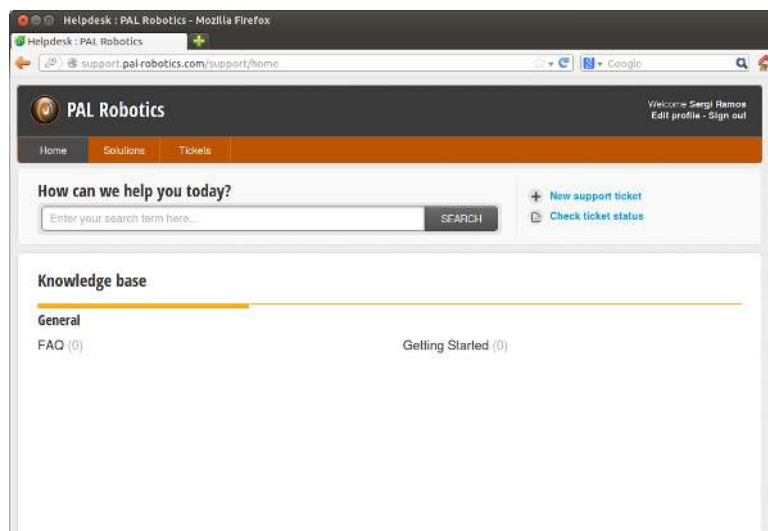


Figure 133: Helpdesk

Figure 134 shows the ticket creation webpage.

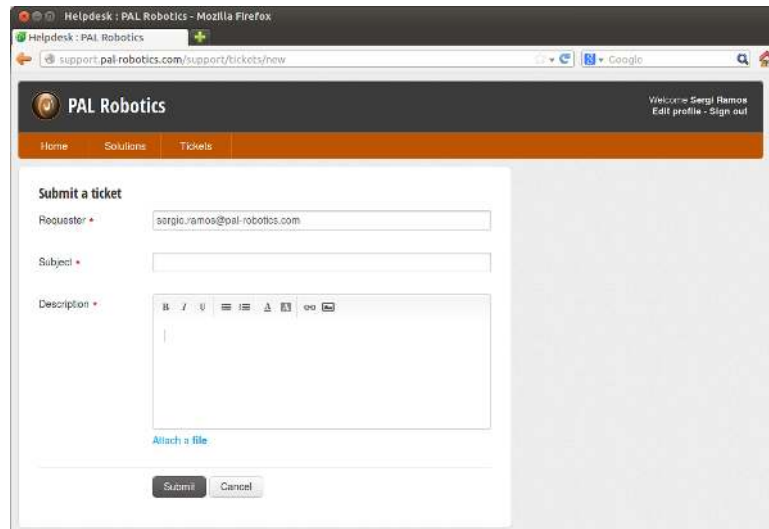


Figure 134: Ticket creation

57.2 Remote support

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

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

```
root@reemc-6c:~# remoteAssistanceConnect ipAddress port
```

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

REEM-C



PAL

ROBOTICS

PAL
ROBOTICS



PAL ROBOTICS S.L.

Pujades 77-79, 4o 4a
08005 Barcelona, Spain

Tel.: +34 934 145 347
Fax: +34 932 091 109

info@pal-robotics.com
www.pal-robotics.com

