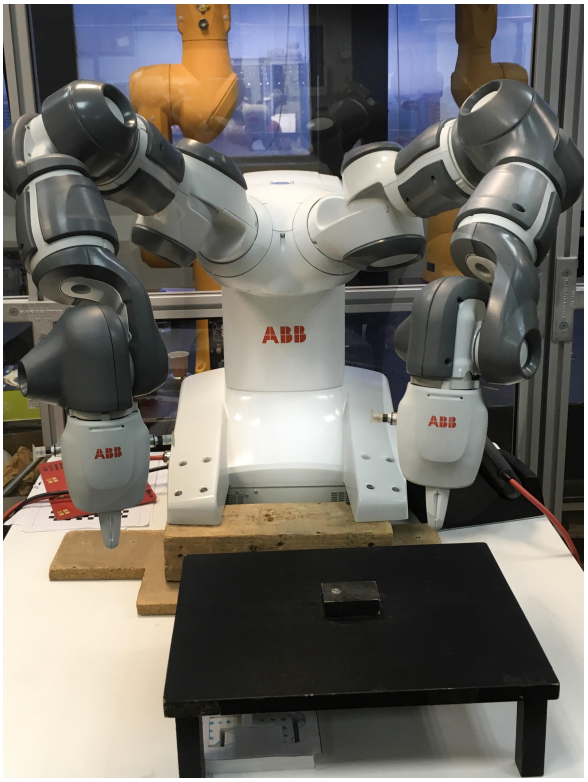
ROS_control Motion with Yumi

Jan Rosell & Júlia Marsal Perendreu

February 2017



Contents

1 Introduction

2 Tasks

The following information about installing and setting tasks and modules in Yumi with ROS_control is based on Orebro University packages, available in [7].

2.1 Task Definition

The ROS Server code requires 3 tasks implemented in a rapid program. Yumi is composed of two arms working separately as two different robots. For this reason, one different motion task are allocated for each arm *T_ROB_L* & *T_ROB_R*. Unlike before, we have both listener and client tasks in one *ROS_StateServer* task which gets the robot state information from the robot and shows in */joint_state topic* and sends motion configurations from *"/yumi/ joint_trajectory_pos_controller /command"* to the robot as it is going to be explained in the following. This package is based on [5] controller. In addition, grippers also contains a ROS Server. They are composed of a listener task *Gripper_motionServer* which gets incoming information from a client (if the gripper has to move or not) and a request task *Gripper_stateServer* to send information to an external client, sending, for instance, the gripper state. Also needs two motion task *Gripper_motion_left Gripper_motion_right* which makes the motion of each gripper. Some modules are loaded to specific tasks, and others are shared between tasks. Each modules and its specifications are explained in appendix A.

2.2 Installing tasks in yumi

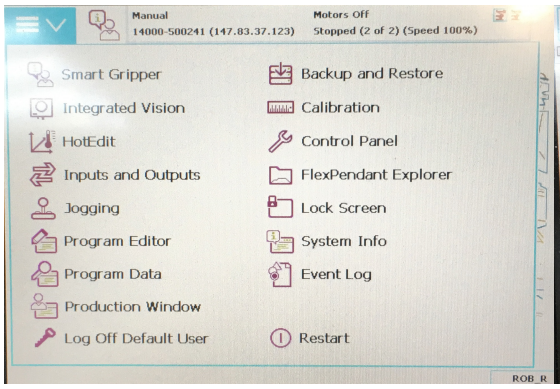
All code that Yumi need was created by OrebroUniversity [7], but it has adjusted to ETSEIB Yumi's robot.

2.2.1 Installing server code

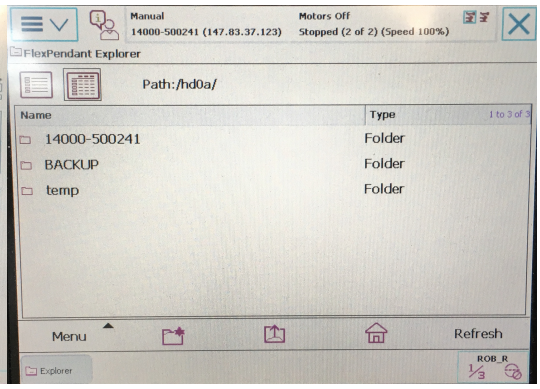
All files located in [6] should be copied to a flash drive. Then, plug it into the robot and under the system's HOME directory make a new directory named ROS. Inside of this one make another one named ROS_control (FlexPendant Explorer → HOME → ROS/ROS_control) and copy ROS_control files there. Do the same for GRASPING (FlexPendant Explorer → HOME → ROS/GRASP). In case of doubt, look at the following pictures as represented in the figure.

2.2.2 Creating tasks

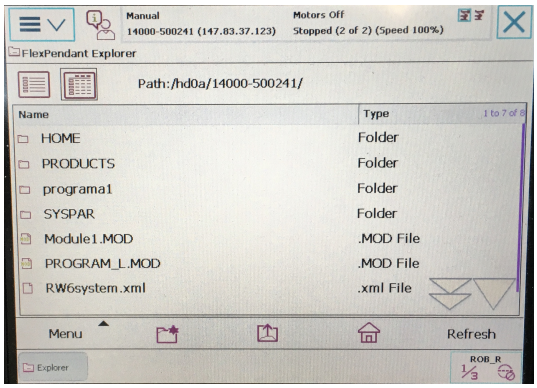
To create tasks browse to ABB Teach-pendant → Control Panel → Configuration → Topics → Controller → Task / New task. Table in figure 2 shows what to set in, and images in figure 3 represents the process of creating a new task.



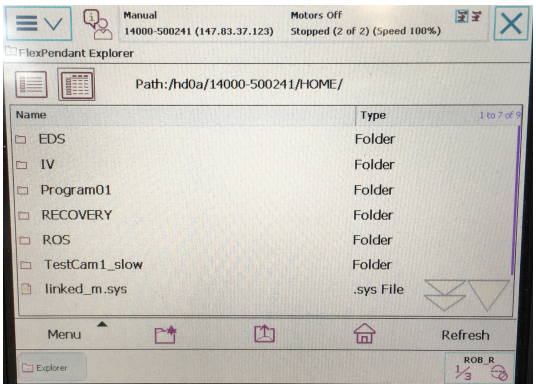
(a) Teachpendant Main Screen



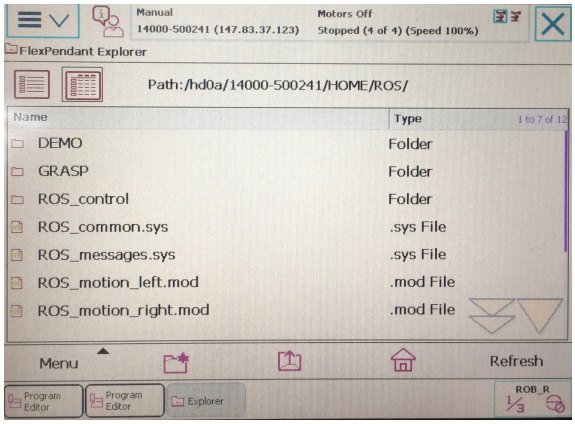
(b) Main Folder



(c) Yumi's Folder



(d) Home's Folder

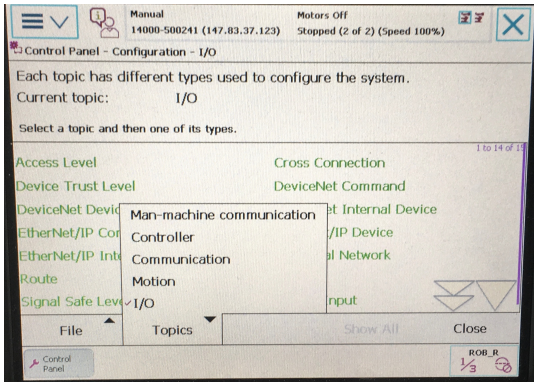


(e) ROS folder with new folders with each files

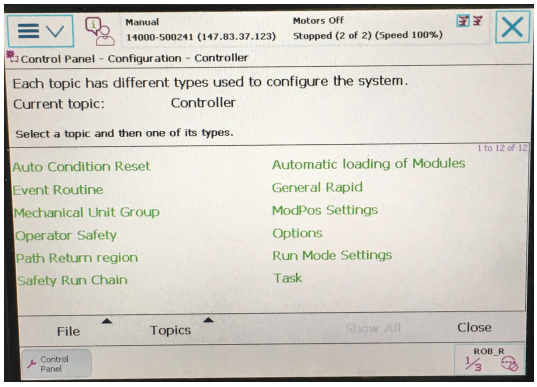
Figure 1: Steps to create a ROS directory

NAME	TYPE	Trust Level	Entry	Motion Task	Use Mechanical Unit Group
ROS_StateServer	SEMISTATIC	NoSafety	main	NO	rob_l
T_ROB_L	NORMAL		main	YES	rob_l
T_ROB_R	NORMAL		main	YES	rob_r
Gripper_stateServer	SEMISTATIC	NoSafety	main	NO	rob_l
Gripper_motionServer	SEMISTATIC	NoSafety	main	NO	rob_l
Gripper_motion_left	SEMISTATIC	SyStop	main	NO	rob_l
Gripper_motion_right	SEMISTATIC	SyStop	main	NO	rob_r

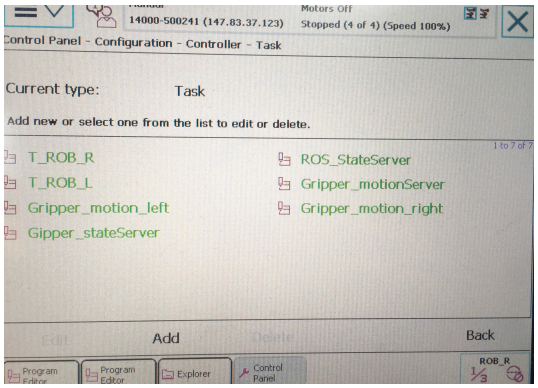
Figure 2: Table of yumi's Tasks and specifications



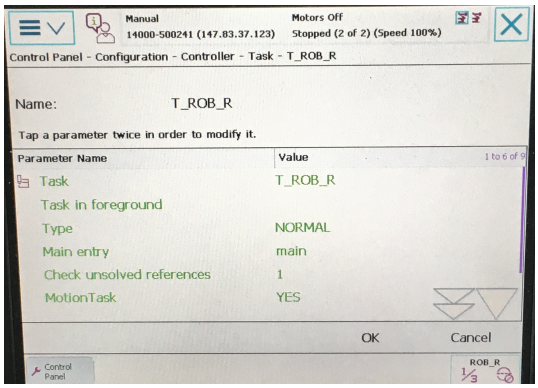
(a) Configuration → Topics screen



(b) Controller screen



(c) Controller → Task



(d) Controller → New Task

Figure 3: Steps to create different tasks in yumi

Parameters definition of tasks are available in appendix B

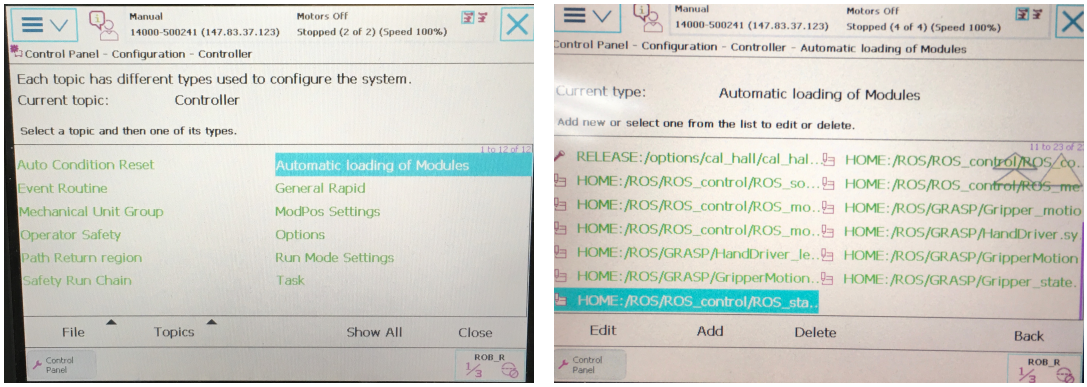
2.3 Modules to Tasks

As it was explained in 2.1, some modules are loaded to specific tasks *.mod*, and others are shared between tasks *.sys*. For this reason, we have to create a link between the task and which file needs, installed in 2.2.1. The process is going to be explained in the following:

1. Browse to ABB → Control Panel → Configuration → Topics → Controller → Automatic Loading of Modules 5
2. Add one entry for each server file as follows in figure 4.

File	Task	Installed	All Tasks	Hidden
HOME:/ROS/ROS_control/ROS_common.sys		NO	YES	NO
HOME:/ROS/ROS_control/ROS_socket.sys		NO	YES	NO
HOME:/ROS/ROS_control/ROS_messages.sys		NO	YES	NO
HOME:/ROS/GRASP/HandDriver.sys		NO	YES	NO
HOME:/ROS/GRASP/HandDriver_left.sys	Gripper_motion_left	NO	NO	NO
HOME:/ROS/ROS_control/ROS_motion_right.mod	T_ROB_R	NO	NO	NO
HOME:/ROS/ROS_control/ROS_motion_left.mod	T_ROB_L	NO	NO	NO
HOME:/ROS/ROS_control/ROS_stateServer.mod	ROS_StateServer	NO	NO	NO
HOME:/ROS/GRASP/Gripper_motionServer.mod	Gripper_motionServer	NO	NO	NO
HOME:/ROS/GRASP/GripperMotion_left.mod	Gripper_motion_left	NO	NO	NO
HOME:/ROS/GRASP/GripperMotion_right.mod	Gripper_motion_right	NO	NO	NO
HOME:/ROS/GRASP/Gripper_stateServer.mod	Gripper_stateServer	NO	NO	NO

Figure 4: Table of Tasks and modules needed in yumi



(a) Control Panel → Topics → Controller (b) → Automatic Loading of Modules

Figure 5: Plots of setting modules between tasks.

3. Restart the controller to set up the changes

2.4 Updating Software

To update robot-server files with new code versions, use the following procedure to ensure that changes are current applied:

1. Copy the new/updated files onto the robot controller, as before.
2. Restart the controller using a P-Start
   - ABB → Restart → Advanced → P-Start → OK
     NOTE: This will erase any existing modules that have been loaded to memory. This may cause compilation issues on restart. If this is a concern, try another method: Warm Start, manually reloading modules (may require setting SEMISTATIC tasks to NORMAL tasks), etc. After the controller reboots, the new changes should be active.

2.5 Installing linux packages

Yumi's packages should be found in [7].

1. If you do not have yumi's packages installed, make *git clone https://github.com/OrebroUniversity/yumi.git* in /catkin_ws/src/ as it shows in figure 6.

Figure 6: Installing linux packages

2. Then, compile all the files with in /catkin_ws making:
   - *catkin_make –only-pkg-with-deps yumi_description*
   - *catkin_make –only-pkg-with-deps yumi_launch*
   - *catkin_make –only-pkg-with-deps yumi_hw*

3 How to move the robot:

3.1 Introduction:

ROS_control gives us different ways to move the robot easily. When we tried to move in ROS_industrial mode it was a big difference between the simulated robot and the real robot. In this case, this difference has been avoided.

3.2 Controller types

There exists two kinds of control to move the robot. As it is explained in the following:

3.2.1 Position control

This control is based-on join control. It means sending the robot the position of each joint (in radians) in order to make the robot moving to a goal position.

3.2.2 Velocity control

Until now, this kind of control doesn't work as we would like to. So it appears a hole that it has to be solved from now on.

3.3 Moving the simulated robot

3.3.1 Introduction

Yumi description and urdf models are available in [4]. Gazebo simulator [1] allows you to move the simulated robot making the same movements as the real robot before trying with the real robot.

3.3.2 Moving the simulated robot with gazebo

In our case, we have the yumi's Gazebo model [4].

If you want to launch yumi's **position controller** write in catkin_ws location:

```
~/catkin_ws roslaunch yumi_launch yumi_gazebo_pos.launch
```

Otherwise, if you want to do a **velocity control** write in catkin_ws location:

```
~/catkin_ws roslaunch yumi_launch yumi_gazebo_vel.launch
```

In both cases yumi will appear in the zero position. As it shows in figure 7

3.4 Moving the real robot

If you desire is to move the real robot, there are also exists the same controllers:

If you want to launch yumi's **position controller** write in catkin_ws location:

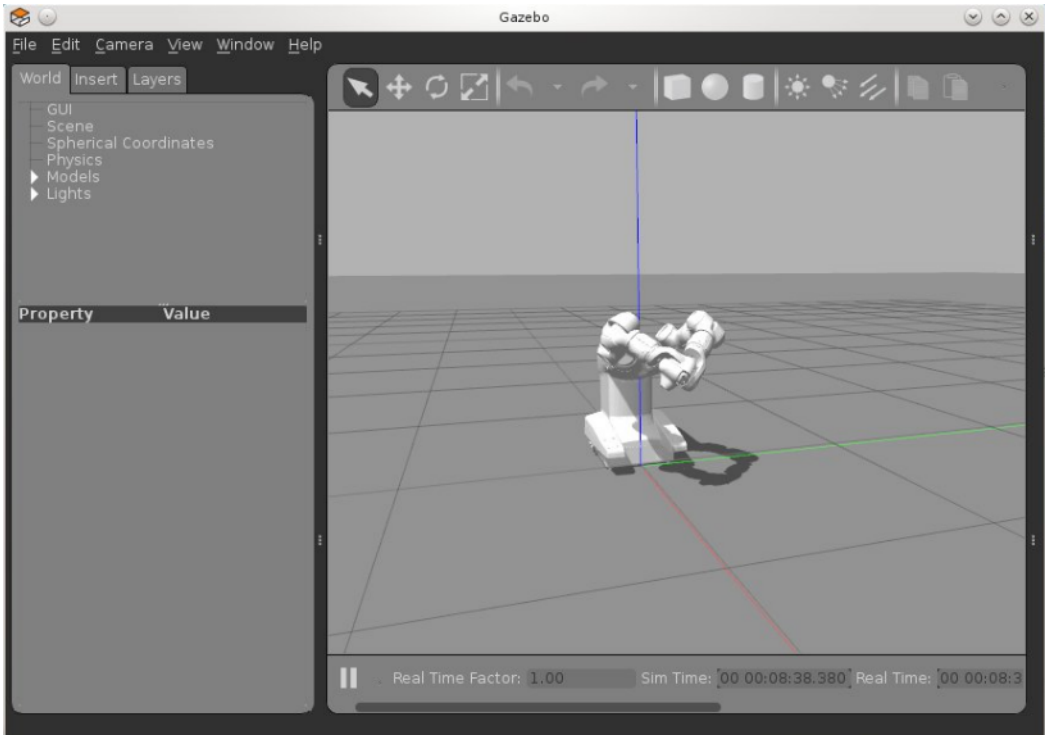


Figure 7: Yumi with gazebo

```
~/catkin_ws roslaunch yumi_launch yumi_pos_control.launch
```

Otherwise, if you want to do a **velocity control** write:

```
~/catkin\_ws roslaunch yumi_launch yumi_vel_control.launch
```

In teachpendant it will appear the information showed in figure 8 .

If one of this tasks doesn't work, restart the controller. With this steps: *Teachpendant menu* → *Restart*. Then, kill the previous launch file *yumi_gazebo_pos/vel control+C* and when the robot has been restarted, try to launch again the launch file and check if the connection is established.

Every time the connection between robot and the computer finished or do not work, restart the robot *menu* → *restart* and then try again launching the connection.

3.5 Topics, Services & Actions

ROS is a framework that contains a lots of tools in order to enable the connection between the robot and the computer. There are different types of messages that exchanged between the robot and the computer, which are going to be defined in the following:

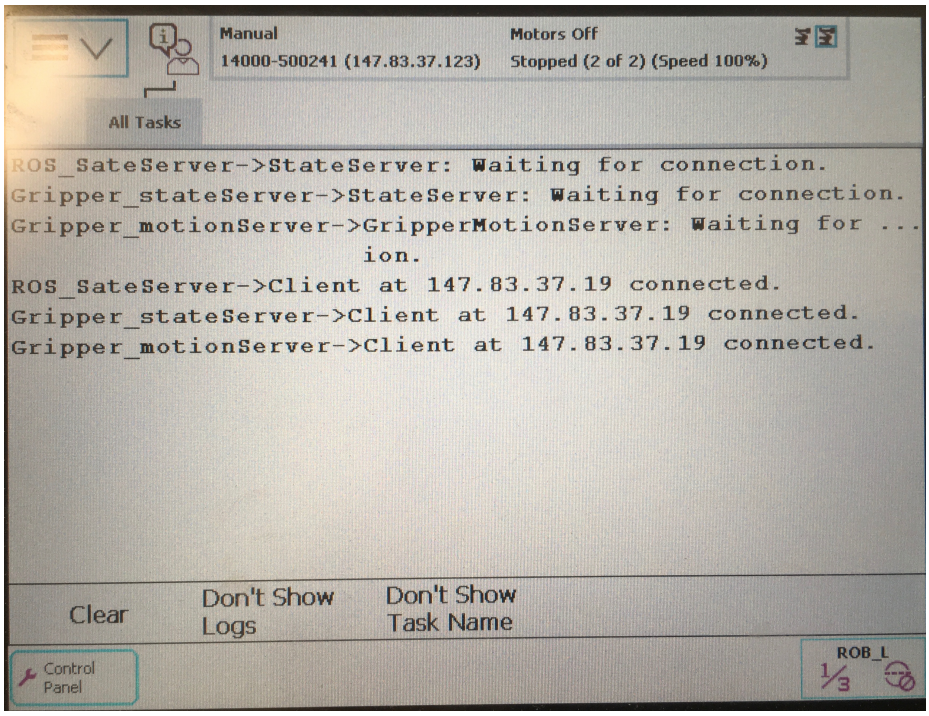


Figure 8: Connection established

3.5.1 Topics:

Topics[3] are based on a ROS message, exchanged between two different nodes. If you want to send information to a node, make a *publisher (send the message)*, however, if you want to get information from a node, you should make an subscriber *echo* of the node *(receiving a message)*.

If you want to know which topics does YUMI has after make the ros_launch explained in 3.3 and 3.4, write:

```
~/catkin_ws      rostopic list
```

In the terminal will be appear the following topics:

As it shows in figure 9, different type of topics are available. What they do and how to use it, is going to be explained in the following:

1. **/yumi/joint_states:** Will be the topic which are going to give us information about the current_state of the robot. Make *rostopic echo /yumi/joint_states* in the terminal and you will obtain the positions of each joint, as it shows in the figure 10.

2. **/yumi/joint_trajectory_pos_controller/status:** This topic gives the status information about the robot as it shows in figure 11. Velocity and

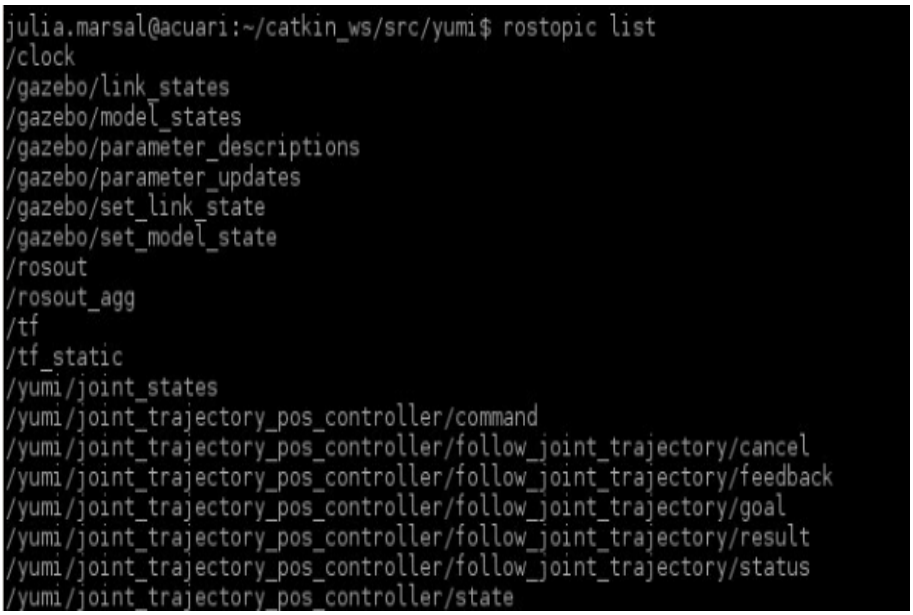


Figure 9: rostopic list information

acceleration values are not a valid value, fail to take full account of this values.

3. **/yumi/joint_trajectory_pos_controller/command:** Will be the topic able to publish to joint's of the robot. Make

   joint_trajectory_pos_controller/command trajectory_msgs/JointTrajectory →(tab)*rostopic pub /yumi/*

   joint_trajectory_pos_controller/command trajectory_msgs/JointTrajectory →(tab) in the terminal, you will be available to publish to this topic, as it shows in the figure 12. Yumi in gazebo will move as it shows in figure 13.

3.5.2 Actions:

Actions are a group of topics that give us the feedback, the status and the result of a topic. Also are able to send a goal and cancel it.

In our case we have the following actions:

Our action's name is **follow_joint_trajectory**.

- **Cancel**: if you want to cancel the movement, write *rostopic pub /yumi/joint_trajectory_pos_controller/follow_joint_trajectory/cancel actionlib_msgs/GoalID →(tab)* and press enter. You don't need to fill anything from the message.

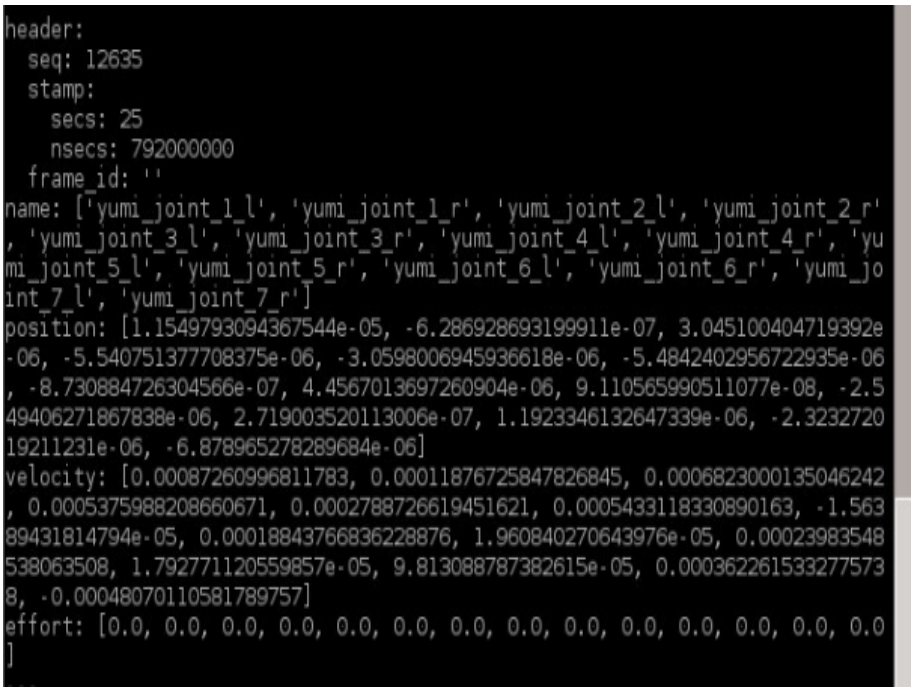


Figure 10: /yumi/joint_states information

- **Feedback**: if you want to obtain a feedback from yumi, write *rostopic echo /yumi/joint_trajectory_pos_controller/follow_joint_trajectory/feedback* and press enter. As it shows in figure 16.

- **Goal**: it makes the same as the topic */yumi/joint_trajectory_pos_controller/command*. In fact, it is based on this topic. It allows you to publish to the topic */yumi/joint_trajectory_pos_controller/follow_joint_trajectory/goal* in order to move yumi by a joint configuration.

  Write *rostopic pub /yumi/joint_trajectory_pos_controller/follow_joint_trajectory/goal (space) control_msgs/FollowJointTrajectoryActionGoal* and refill all the gaps as it shows in the figure 17

  In joint_names is defined the name of each join that will be defined in *positions* (in radians), velocity and acceleration needs to be filled, but the values would not influence to the robot, it computes its own velocity and acceleration.

- **Result:** It will return the result of the action. Make *rostopic echo /yumi/joint_trajectory_pos_controller/follow_joint_trajectory/result* as it shows in 18.

- **Status:** It will return the status of the actions. Make *rostopic echo*

Figure 11: states information

Figure 12: command

/yumi/joint_trajectory_pos_controller/follow_joint_trajectory/status. As it shows in figure 19.

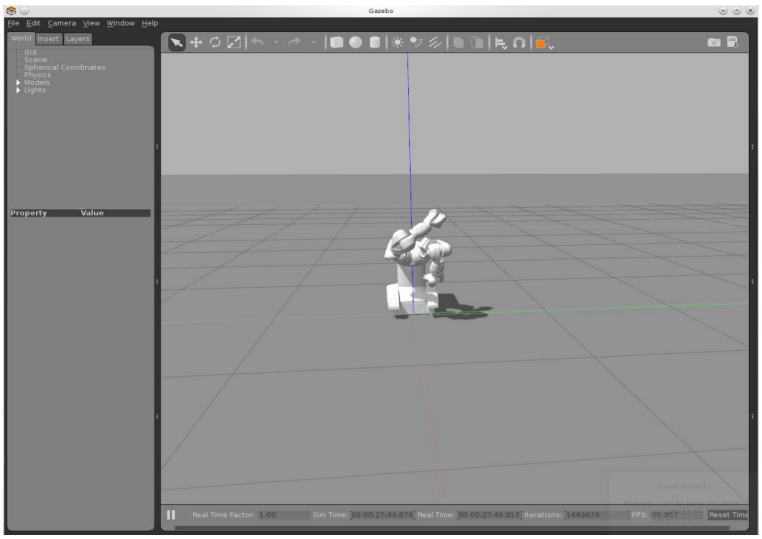


Figure 13: Yumi's in gazebo goal

Figure 14: actions available

3.5.3 Services:

ROS Services [2] use a Request / reply which is defined by a pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply.

The most important function using services is with grippers.A gripepr can be opened and closed with services.

Yumi's has different type of services, make

```
~/catkin_ws      rosservice list
```

in the terminal and it will appear all the services available in yumi as it shows in figure 20.

The most important ones are the following ones:

- **/yumi/yumi_gripper/do_grasp**

  It make the gripper close. To use it write in the terminal

  ```
  ~/catkin_ws  rosservice call /yumi/yumi_gripper/do_grasp
  (tab) "gripper_id: 0"
  ```

Figure 15: cancel topic

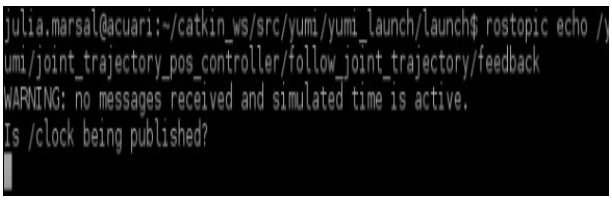


Figure 16: topic feedback

Refill *0* with an 1 or 2.

- 1: It is used for the left arm. If you use it left gripper will close.
- 2: It is used for the right arm. If you use it right gripper will close.

As it shows in figure 21

- **/yumi/yumi_gripper/release_grasp**

  It opens the gripper. To use it write in the terminal

  ```
  ~/catkin_ws rosservice call /yumi/yumi_gripper/release_grasp
  (tab) "gripper_id: 0"
  ```

  Refill *0* with an 1 or 2.

  - 1: It is used for the left arm. If you use it left gripper will open.
  - 2: It is used for the right arm. If you use it right gripper will open.

  As it shows in figure 22

4 Videos:

For more information watch this videos:

Ros_control with real yumi:
https://www.youtube.com/watch?v=bcHvZ7h3cnk (part1)
https://www.youtube.com/watch?v=vnWU_9IMqgk (part2)

Robotstudio with real yumi:
http://www.youtube.com/watch?v=W5lmYWYIpG8 (part1)
http://www.youtube.com/watch?v=Wvjdl4I7QE0 (part2)

Figure 17: topic goal

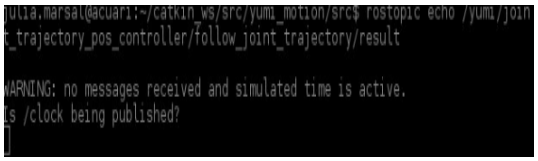


Figure 18: result action

Figure 19: status action

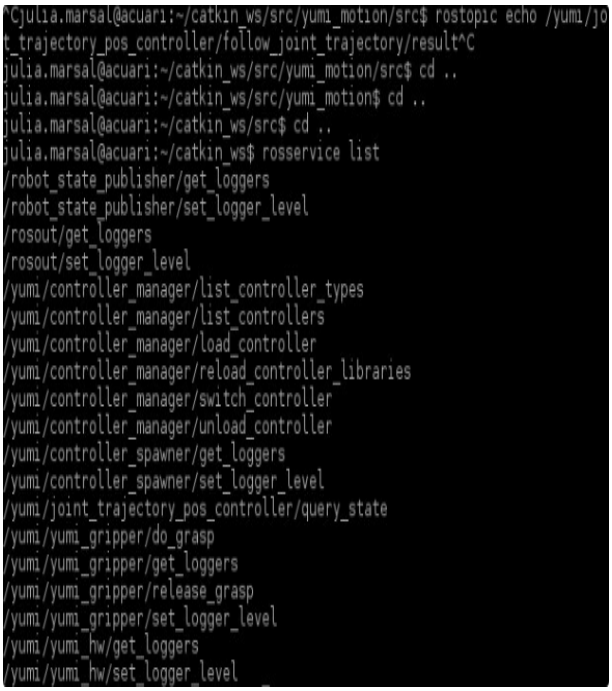


Figure 20: service list

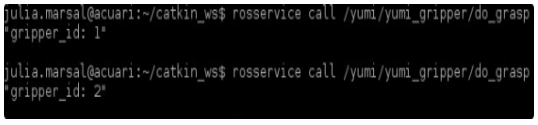


Figure 21: do grasp

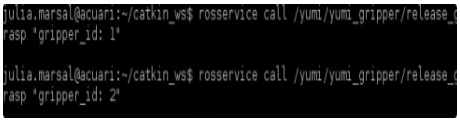


Figure 22: release grasp

A Rapid Modules:

A.1 File Overview

- **Modules Shared by all tasks:**
  - **ROS_CONTROL**:
    - **ROS_common.sys:** Global variables and data types shared by all files

    - **ROS_socket.sys:** Socket handling and simple_message implementation
    - **ROS_messages.sys:** Implementation of specific message types
  - **GRASP**:
    - **HandDriver.sys:** Global variables and data types shared by left gripper.
    - **HandDriver_left.sys:** Global variables and data types shared by right gripper .

- **Specific task modules:**
  - **ROS_CONTROL:**
    - **ROS_stateServer.mod:** Broadcast joint position, state data and motion commands to the robot.
    - **ROS_motion_left.mod:** Issues motion commands to the left arm of the robot.
    - **ROS_motion_right.mod:** Issues motion commands to the right arm of the robot.
  - **GRASP**:
    - **GripperMotion_left.mod:** Issues motion commands to the left gripper.
    - **GripperMotion_right.mod:** Issues motion commands to the right gripper of the robot.
    - **Gripper_motionServer.mod:** Receive gripper motion commands
    - **Gripper_stateServer.mod:** Broadcast joint position and state data of gripper.

  The files can be found in [7]

B Task Parameters definition:

Parameters definition

Tasks are defined by a different kind of parameters, which are going to be defined in the following.

- **TYPE**

  STATIC and SEMISTATIC tasks are started in the system startup sequence.

  - **STATIC**: when the robot stops, it will be restarted at the current position (where the Program Pointer was when the system was powered off).

- **SEMISTATIC:** it will be restarted at the beginning when power is turned on, and modules specified in the system parameters will be reloaded if the module file is newer that the loaded module.
- **NORMAL:** the task will not be started at startup, it will be started in a normal way, for example, from the FlexPendant.

- **Trust Level**

  TrustLevel handle the system behavior when a SEMISTATIC or STATIC task is stopped for some reason or not executable. For more information look at [**?**] web-side.

  - **SysFail**: This is the default behavior, all other NORMAL tasks will also stop, and the system is set to state SYS_FAIL. All jog and program start orders will be rejected. Only a new warm start reset the system. This should be used when the task has some security supervisors.
  - **SysHalt**: All *NORMAL* tasks will be stopped. The system is forced to "motors off". When taking up the system to "motors on" it is possible to jog the robot, but a new attempt to start the program will be rejected. A new warm start will reset the system.
  - **Systop(Ros_MotionServer)**: All *NORMAL* tasks will be stopped, but can be restarted. Jogging is also possible.
  - **NoSafety(ROS_StateServer)**: Only the actual task itself will stop.

- **Entry**

  The task accesses to main function when starts.

- **Motion Task** It will be True if the task it is a Motion Task (T_ROB_L) | (T_ROB_R) and False if not.

References

[1] Gazebo. Gazebo. http://gazebosim.org/.

[2] KenConley. Rosservice. http://wiki.ros.org/Services.

[3] Open. Topics. http://wiki.ros.org/Topics.

[4] OrebroUniveristy. Yumi_description. https://github.com/OrebroUniversity/yumi/tree/master/yumi_description.

[5] OrebroUniveristy. Yumi_hw. https://github.com/OrebroUniversity/yumi/tree/master/yumi_hw/rapid.

[6] OrebroUniveristy(adapted). yumi's files. https://github.com/juliamp22/Ros_control_files.git.

[7] OrebroUniversity. yumi. https://github.com/OrebroUniversity/yumi.git.