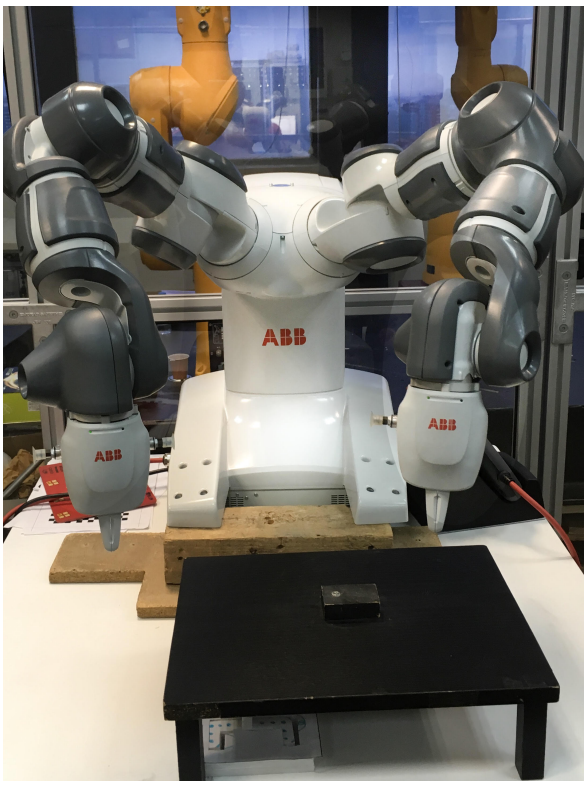
ROS_industrial Motion with Yumi

Jan Rosell & Júlia Marsal Perendreu

February 2017



Contents

1 Introduction

2 Tasks

The following information about installing and setting tasks and modules in Yumi is based on webpage [10].

2.1 Task Definition

The ROS Server code requires 4 tasks implemented in a rapid program.

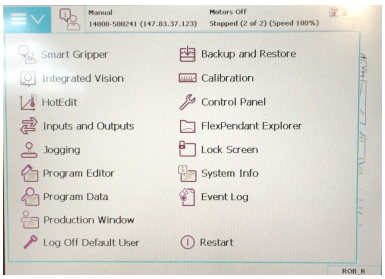
Yumi is composed of two arms working separately as two different robots. Two different motion tasks are allocated for each arm. In addition, Yumi needs a listener task *(motionServer)* in order to get incoming information from a client, and a request task *(stateServer)* to send information to an external client, getting, the robot state.

Some modules are loaded to specific tasks, and others are shared between tasks. Each modules and its specifications are explained in appendix A. Rapid modules codes are available in [8]

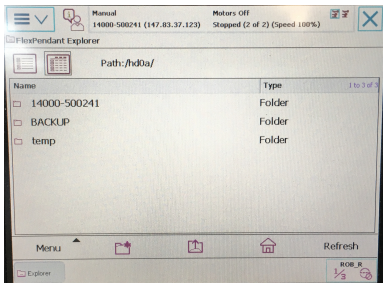
2.2 Installing tasks in yumi

2.2.1 Installing server code

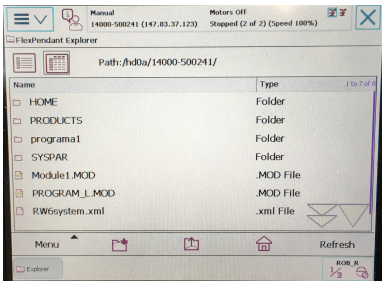
All files located in [8] should be copied to a flash drive. Then, plug it into the robot and under the system's HOME directory make a new directory named ROS (FlexPendant Explorer → HOME → ROS/*) and copy the files there. In case of doubt, look at the following pictures as represented in the figure.



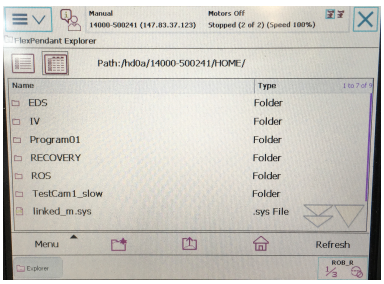
(a) Teachpendant Main Screen



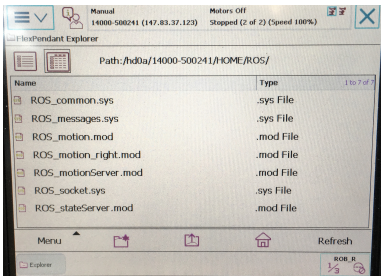
(b) Main Folder



(c) Yumi's Folder



(d) Home's Folder



(e) ROS folder with new rapid files

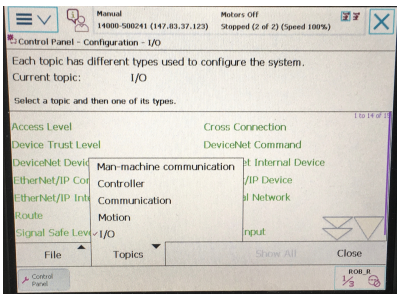
Figure 1: Steps to create a ROS directory

2.2.2 Creating tasks

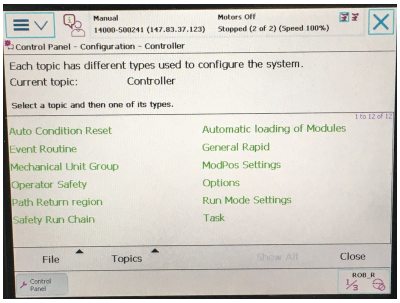
To create tasks browse to ABB Teach-pendant → Control Panel → Configuration → Topics → Controller → Task / New task. Table in figure 2 shows what to set in, and images in figure 3 represents the process of creating a new task.

NAME	TYPE	Trust Level	Entry	Motion Task	Use Mechanical Unit Group
ROS_StateServer	SEMISTATIC	NoSafety	main	NO	rob_l
ROS_StateServerRight	SEMISTATIC	NoSafety	main	NO	rob_r
ROS_MotionServer	SEMISTATIC	SyStop	main	NO	rob_l
ROS_MotionServerRight	SEMISTATIC	SyStop	main	NO	rob_r
T_ROB_L	NORMAL		main	YES	rob_l
T_ROB_R	NORMAL		main	YES	rob_r

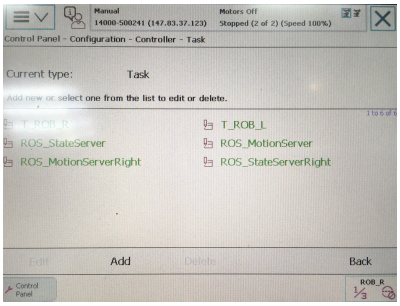
Figure 2: Table of yumi's Tasks and specifications



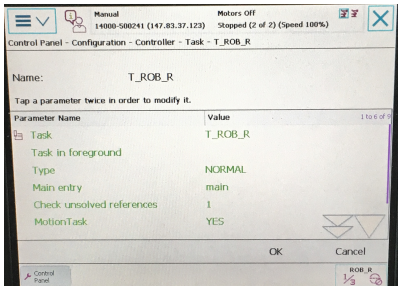
(a) Configuration → Topics screen



(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 Load Modules to Tasks

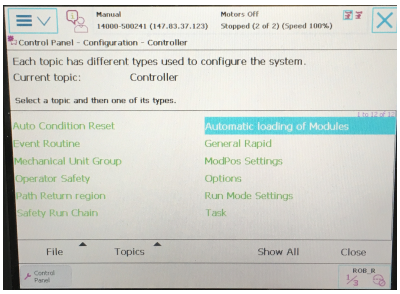
As it was explained in 2.1, some modules are loaded to specific tasks, and others are shared between tasks. For this reason, it will be referenced to one specific task or not. The process is explained in the following:

1. Browse to ABB → Control Panel → Configuration → Topics → Controller → Automatic Loading of Modules

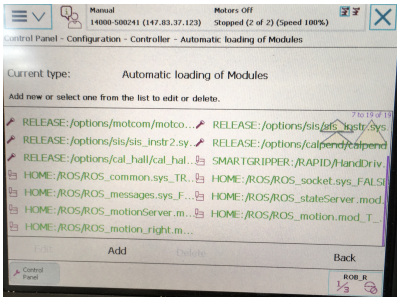
2. Add one entry for each server file as follows in figure 4.

File	Task	Installed	All Tasks	Hidden
HOME:/ROS/ROS_common.sys		NO	YES	NO
HOME:/ROS/ROS_socket.sys		NO	YES	NO
HOME:/ROS/ROS_messages.sys		NO	YES	NO
HOME:/ROS/ROS_stateServer.mod	ROS_StateServer	NO	NO	NO
HOME:/ROS/ROS_stateServer_right.mod	ROS_StateServer	NO	NO	NO
HOME:/ROS/ROS_motionServer.mod	ROS_MotionServer	NO	NO	NO
HOME:/ROS/ROS_motionServer_right.mod	ROS_MotionServer	NO	NO	NO
HOME:/ROS/ROS_motion.mod	T_ROB_L	NO	NO	NO
HOME:/ROS/ROS_motion_right.mod	T_ROB_R	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.

3 Moving the simulated robot:

3.1 Packages to be installed:

3.1.1 Yumi

Yumi's packages should be found in [7]. If you had not installed yumi packages make *git clone https://github.com/OrebroUniversity/yumi.git* in the catkin_ws/src location.

Also it is need Jose Agustin's folder. You should make a

Then make a *catkin_make -only-pkg-with-deps yumi* and a *catkin_make -only-pkg-with-deps yumiros*

Then, look for README.md file inside yumiros folder. There are inside the instructions of how to launch a gazebo file.

4 Moving the real robot:

4.1 Introduction

In the previous section it is explained how to fix ROS files inside the robot. But, how the communication between the robot and the computer is going to be established? The robot is going to communicate with the user's computer directly? Has it being set before? It is true that a socket communication has opened in the robot. But, in addition, we need to install new packages in the user's computer.

4.2 Packages to be installed:

4.2.1 ABB package

Clone git [2] into your catkin_ws/src repository.
~/catkin_ws/src$ git clone [2]

4.2.2 Industrial_robot_client

Clone git [3] into catkin_ws/src repository.
~/catkin_ws/src$ git clone [3]

4.2.3 Yumi

As it was explained in 3.1.1 yumi needs a launch file **yumi_support** to initialize the communication between the robot and user's computer.

<arg name="robot_ip" default="**robot_ip**"/>needs to be replace with *robot_ip:147.83.37.123* or *robotstudio* **robot_ip**.

4.2.4 Catkin_make

Catkin_make is a convenience tool for building code in a catkin workspace. It follows the standard layout of a catkin workspace. More information at [9].

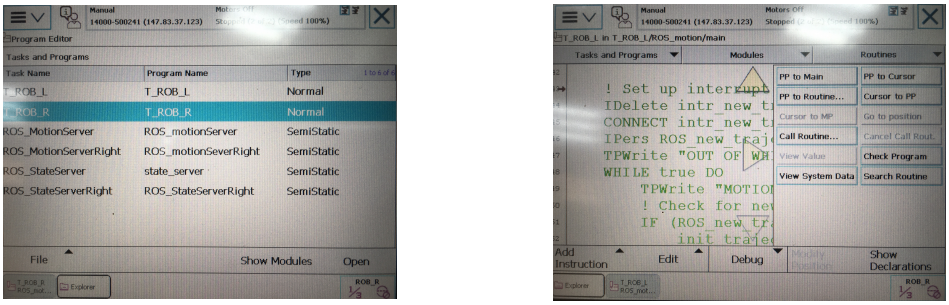
~/catkin_ws$ catkin_make in terminal.

4.3 How to move it:

1. Open the controller of the yumi real robot.

2. Set the Program Pointer to main in tasks T_ROB_L and T_ROB_R.

   Control Panel → Program Editor → double click in T_ROB_L or T_ROB_R (rapid program opening) → Debug → PP to Main. As is shown in figures 6.



(a) Control Panel → Program Editor (b) → T_ROB_L

Figure 6: Plots of setting the task PP to main.

Press play button in order to execute both tasks. If any trajectory has not sent before, the robot must be in a standby position without moving. If not, it should start to move.

3. Launch **robot_interface.launch** file which appears in *~/catkin_ws/src/yumi/yumi_support/launch.*

   ~/catkin_ws/src$ roslaunch yumi_support robot_interface.launch

   Teachpendant should inform that motionServer and stateServed are connected. As it is shown in figure 7.

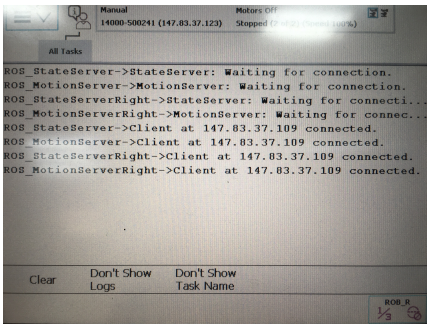


Figure 7: Connection has been established

4. Write in terminal **rostopic_list** to know which topics are available

~/catkin_ws/$ rostopic_list

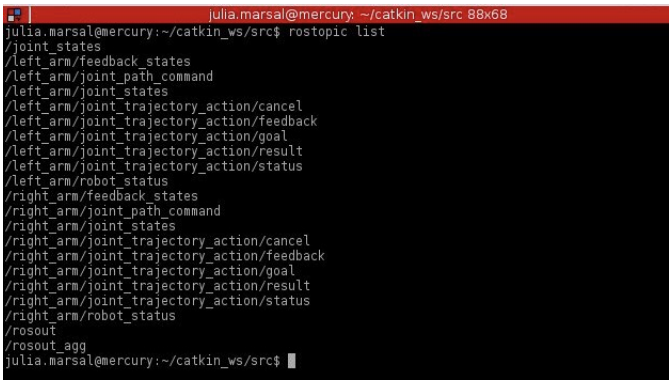


Figure 8: List of topics available

5. Write in terminal **rosservice_list** to know which services are available

~/catkin_ws/$ rosservice_list

4.4 Topics

4.4.1 What is a topic?

Topics are named buses over which nodes exchange messages. Topics have anonymous publish/subscriber semantics, which decouples the production of information from its consumption. In general, nodes are not aware of who they are communicating with. Instead, nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic.

More information about topics is available in [1]

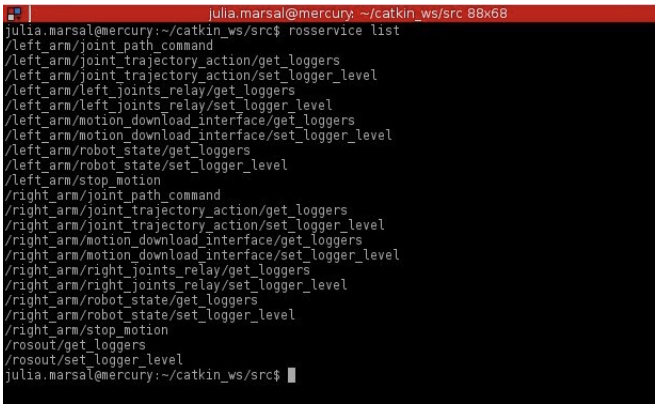


Figure 9: List of services available

4.4.2 Yumi's topic

As it was explained in 2.1 yumi is composed by two arms. For this reason, there are topics repeated for both arms. They are based on Industrial_Robot_Driver_Spec [4].

1. **State Feedback:** Used to provide feedback of joint positions, velocities, acceleration, etc.

   (a) **Feedback_states**

   - Provide feedback of current vs. desired joint position (and velocity/acceleration).
   - It is used by joint_trajectory_action to monitor in-progress motions.

   (b) **Joint_states**

   - Provide feedback of current joint position (and velocity/effort).
   - It is used by the robot_state_publisher node to broadcast kinematic transforms.

   Having established the communication between the robot and user's computer 4.3. Robot joint_states are shown by making an echo of the topic, <*rostopic echo /left_arm/joint_states* > As it is shown in figure 10. The same for the right arm robot <*rostopic echo /right_arm/ joint_states* >.

   (c) **Robot_status**

   - Provide current status of critical robot parameters.
   - Used by application code to monitor and react to different fault conditions.

Figure 10: Joint_state echo

2. **Motion Control:**

   This node implements methods to control the robot's movement.

   (a) Joint_path_command

   - Execute a pre-calculated joint trajectory on the robot
   - used by ROS's trajectory generators (e.g. joint_trajectory_action) to issue motion commands

   Having established the communication between the robot and user's computer 4.3. User's can make the robot move *<rostopic pub /left_arm/ joint_path_command trajectory_msgs/JointTrajectory >* → press tab two times, and refilling gaps → press enter as it shown in figure 11. The same for the right arm robot *<rostopic pub /right_arm/joint_path_command trajectory_msgs/JointTrajectory >*.

4.5 Services

4.5.1 What is a service?

It's a new way of sending/receiving information by Request / reply done via a Service, 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. Client libraries usually present this interaction to the programmer as if it were a remote procedure call.

More information about services are available in [6]

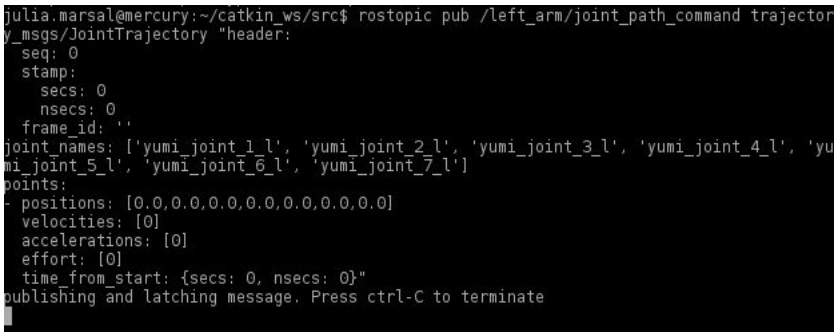


Figure 11: joint_path_command publishing

4.5.2 Yumi's services

1. **Motion Control:**

   This node implements methods to control the robot's movement.

   (a) Joint_path_command

   - Execute a pre-calculated joint trajectory on the robot
   - Used by ROS's trajectory generators (e.g. joint_trajectory_action) to issue motion commands
   - The service-implementation allows calling code to receive confirmation that a commanded trajectory has actually been received by a robot node.

   Having established the communication between the robot and user's computer 4.3. User's can make the robot move calling this topic, *<rosservice call /left_arm/joint_path_command >* → press tab two times, and refilling gaps → press enter. As it shown in figure 12. The same for the right arm robot *<rosservice call /right_arm/joint_path _command >*.

2. **Stop_motion:**

   - Stops current robot motion.
   - Resumes motion by sending a new motion command.

   Having established the communication between the robot and user's computer 4.3 and make it move. User's can make the robot stop *<rosservice call /left_arm/stop_motion >* → press tab two times, don't refill anything and press enter. As it shows in figure 13. The same for the right arm robot *<rosservice call /right_arm/stop_motion _command >*.

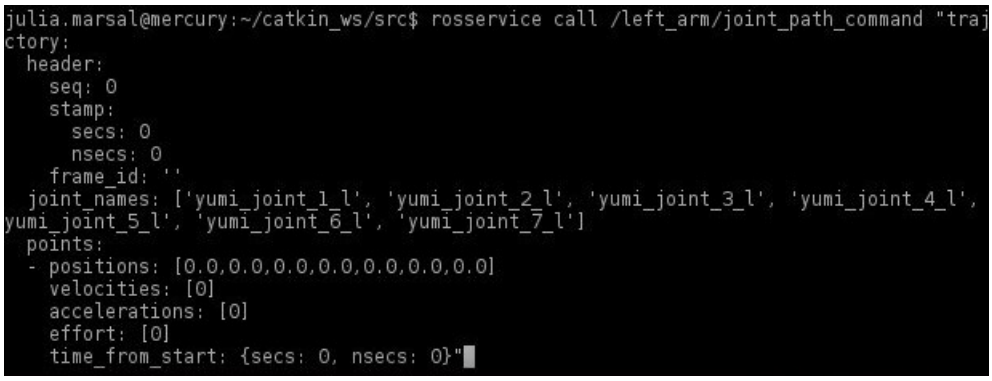


Figure 12: joint_path_command call

Figure 13: stop_motion call

4.6 Actions

The ActionClient and ActionServer communicate via a "ROS Action Protocol", which is built on top of ROS messages. The client and server then provide a simple API for users to request goals (on the client side) or to execute goals (on the server side) via function calls and callbacks. n order for the client and server to communicate, we need to define a few messages on which they communicate. This is with an action specification. This defines the Goal, Feedback, and Result messages with which clients and servers communicate:

- **Goal**

Goal To accomplish tasks using actions, we introduce the notion of a goal that can be sent to an ActionServer by an ActionClient. In the case of moving the base, the goal would be a PoseStamped message that contains information about where the robot should move to in the world. For controlling the tilting laser scanner, the goal would contain the scan parameters (min angle, max angle, speed, etc).

Feedback Feedback provides server implementers a way to tell an ActionClient about the incremental progress of a goal. For moving the base, this might be the robot's current pose along the path. For controlling the tilting laser scanner, this might be the time left until the scan completes.

Result A result is sent from the ActionServer to the ActionClient upon completion of the goal. This is different than feedback, since it is sent exactly once. This is extremely useful when the purpose of the action is

to provide some sort of information. For move base, the result isn't very important, but it might contain the final pose of the robot. For controlling the tilting laser scanner, the result might contain a point cloud generated from the requested scan.

A Rapid Modules:

A.1 File Overview

- **Modules Shared by all tasks:**
  - **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
- **Specific task modules:**
  - **ROS_stateServer.mod:** Broadcast joint position and state data
  - **ROS_motionServer.mod:** Receive robot motion commands
  - **ROS_motion.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.

The files can be found in [8]

RECORD: It's the same as a structure in cpp.

A.2 Modules Shared by all tasks:

A.2.1 ROS_common.sys

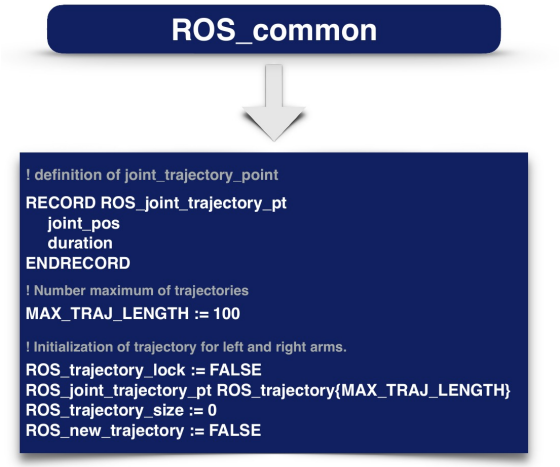


Figure 14: ROS_common variables

A.2.2 ROS_socket.sys

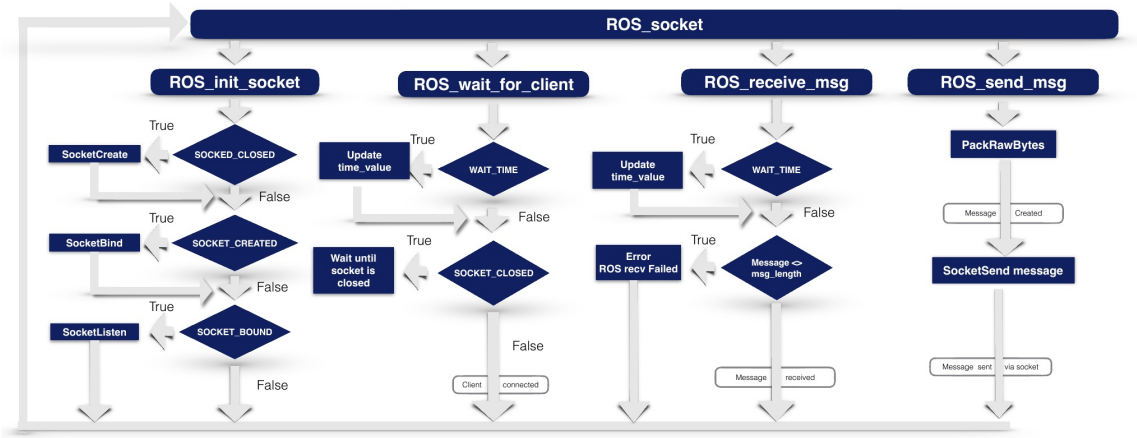


Figure 15: ROS_socket Flowchart

A.2.3 ROS_messages.sys

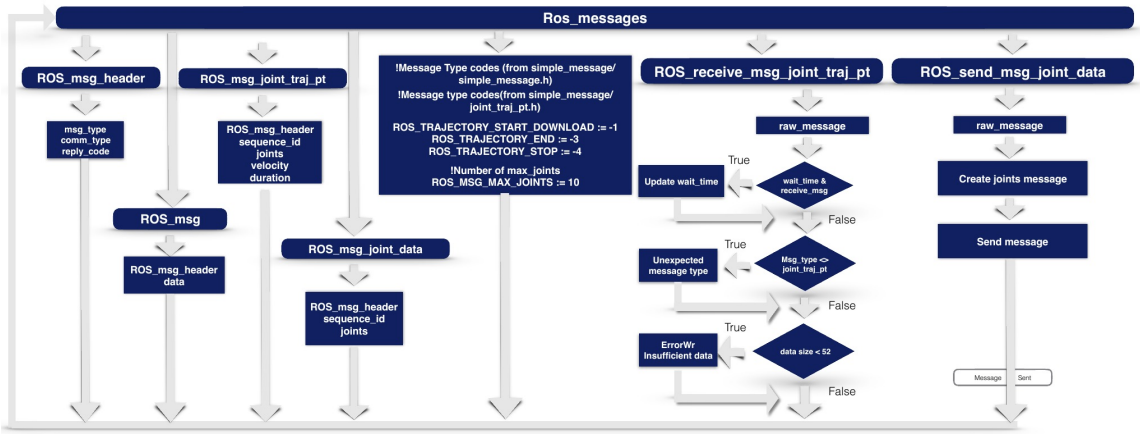


Figure 16: ROS_messages Flowchart

A.3 Specific task modules:

A.3.1 ROS_stateServer.mod

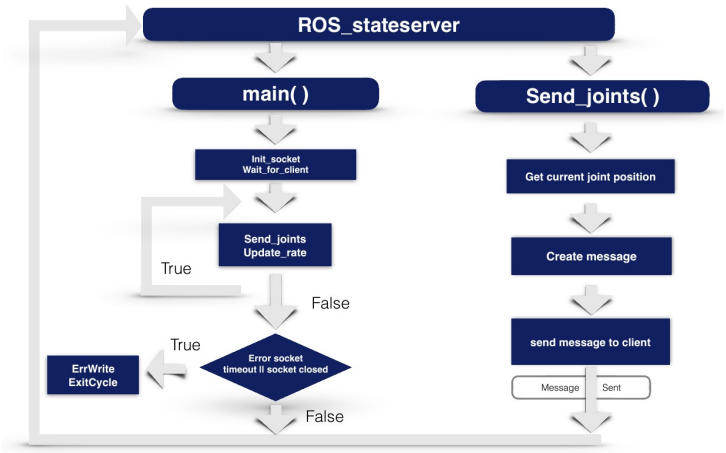


Figure 17: ROS_state Server Flowchart

A.3.2 ROS_motionServer.mod

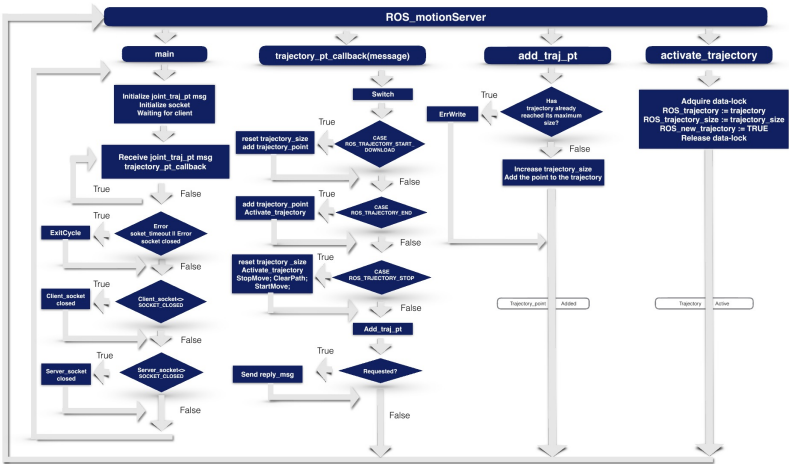


Figure 18: ROS_motionServer Flowchart

A.3.3 ROS_motion.mod — ROS_motion_right.mod

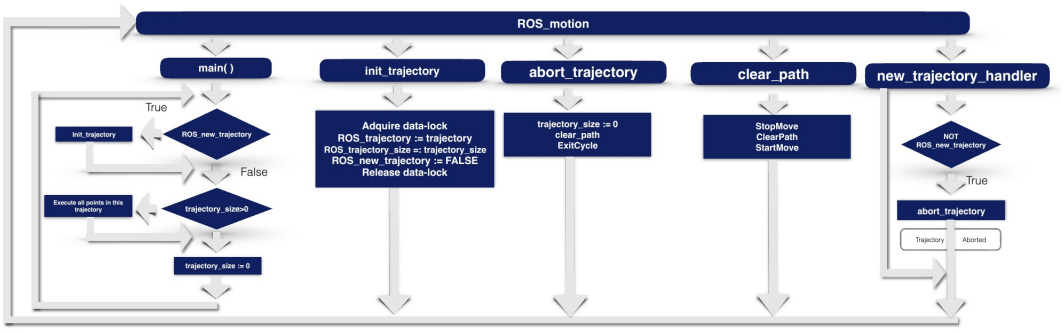


Figure 19: ROS_motion Flowchart

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 [5] 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] DariushForouher. Topics. http://wiki.ros.org/Topics.

[2] GvdHoorn. abb_driver. https://github.com/ros-industrial/abb.git.

[3] GvdHoorn. industrial_robot_client. githttps://github.com/ros-industrial/industrial_core.git.

[4] GvdHoorn. Industrial_robot_driver_spec. http://wiki.ros.org/Industrial/Industrial_Robot_Driver_Spec.

[5] ipacv. Multitasking. http://www.ipacv.ro/proiecte/robotstudio/textbooks/file/basic/multitasking.htm.

[6] KenConley. Services. http://wiki.ros.org/Services.

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

[8] Júlia Marsal Perendreu. Rapid server files. https://github.com/juliamp22/yumiros-rapid-files.git.

[9] rbaleksandar. catkin_make. http://wiki.ros.org/catkin/commands/catkin_make.

[10] Ros-Industrial. Installing abb ros server. http://wiki.ros.org/abb/Tutorials/InstallServer.