ROBT403
Robotics II: Control, Modeling and Learning with Laboratory labs

Instructor:

Zhanat KAPPASSOV, PhD

Assistant Professor, Robotics Dept

Assistants:

Nurlan Kabdyshev, MS student

Amina Asrepova, Coordinator

Course logistics

- 25% of the total course grade
- Files to submit:

  report (PDF),

  screen recordings (short) or a link to the video,

  software uploaded to Moodle or on github (when needed)

Attendance is compulsory

- Write names and surnames on the blank list
- Lab report will not be accepted if you are absent (zoom is possible sometimes).

Telegram for conversations and help

- https://t.me/+fJLK-A66uQMwNGNi

Bootable memory stick

- Howto use BOOTABLE flash drive is on Moodle
- Bootable_flash_Linux_ROS.pdf

How to create bootable USB flash drive ...
accatech.com

Git for uploading your code→ OneDrive, Gmail, etc (Sanzhar git push trick)

- Git tutorial (on Moodle)
- Git tutorial on web (https://www.atlassian.com/git/tutorials):
  - https://www.atlassian.com/git/tutorials/install-git
  - https://www.atlassian.com/git/tutorials/setting-up-a-repository
  - https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-init

Previous Lab I: ROS Basics

0) Publisher and Subscriber with the Topic name of your name that sends the digits of your NU id numbers one by one in the loop at 1 Hz rate and 50Hz rate

1) Two nodes (Publisher and Subscriber) talk via ROS Network
2) Callback function to see on a terminal turtlesim's pose and velocity
3) Publisher to set the turtlesim's velocity
4) Services Request/Reply Interaction in ROS (Remote Procedure Call)
5) Turtlebot move to draw a square and a triangle

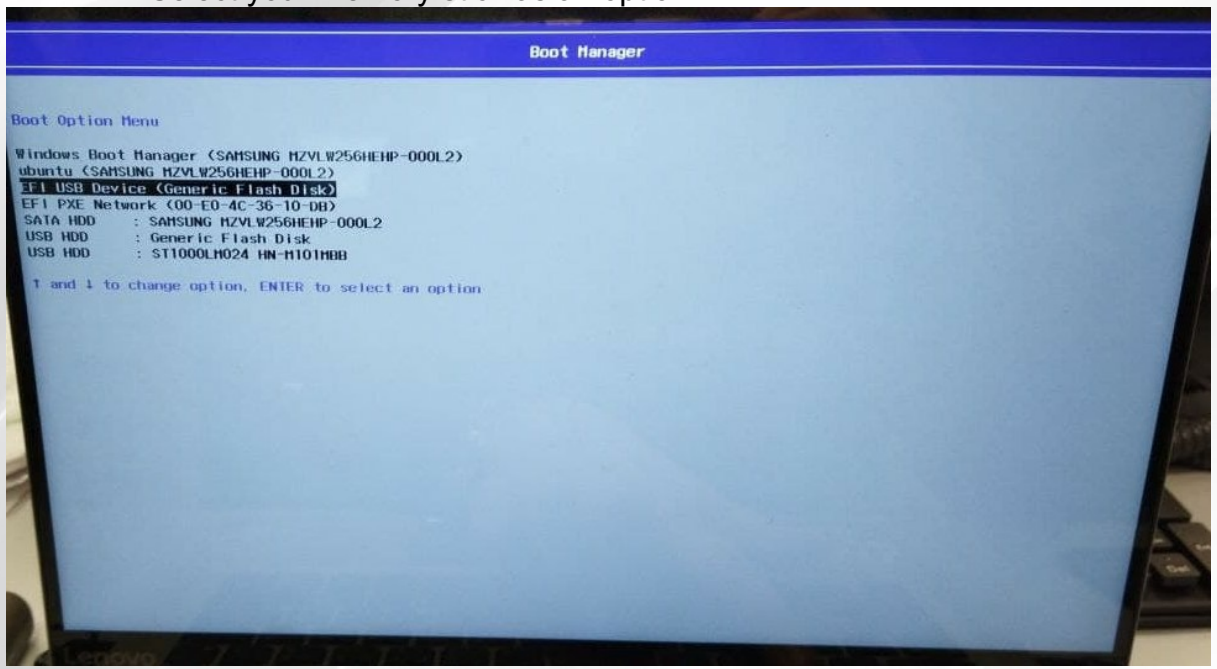
Todays lab - Lab 2

- Compile packages (from Git repasitories) that were developed by the ROS community
- Howto start our laboratory robot (5 DoF Planar Manipulator)
- Single joint control
- PID of a joint controller: response to a step function (visualized using tool called “rqt”)
- Multiple joints control to make the manipulator to move like a snake

LAB II: Basics of Robot Manipulator Control in Joint Space

Prepare your PC:
Start your OS (Linux) From Bootable Stick

- Restart your computer
- Boot from your memory stick
    1. Push F12 button
    2. Select your memory stick as an option



Prepare your PC:
Source the ROS libraries

- The bootable memory stick has preinstalled Robot Operating System (ROS). We need to read and execute the content of the file (generally set of commands) that has all preinstalled ROS commands. It means, that every time we turn on a machine from the bootable flash, we need to do it. For this purpose. Open a terminal and type*

source /opt/ros/melodic/setup.bash

*In any new opened terminal, we need to run

source

Prepare your PC:
Create your ROS worskspace

The next steps are for working with the Planar Robot. The libraries are on GitHub. Please follow the tutorials for using the GIT first to understand the Git commands.

- In the opened terminal (in which your have done source) type:

  mkdir -p ~/catkin_ws/src

- Build your code:

catkin_make

Prepare your PC:
Download packages to control the robot into your ROS worskspace

- **Prerequisites:** The project has been implemented on Ubuntu 20.04 machine with ROS Noetic. To be able to run the project ROS must be installed on your local machine. Also, the following libraries are required:
- Gazebo
- MoveIt. If moveit libraries are missing, install them:

sudo apt install ros-noetic-moveit

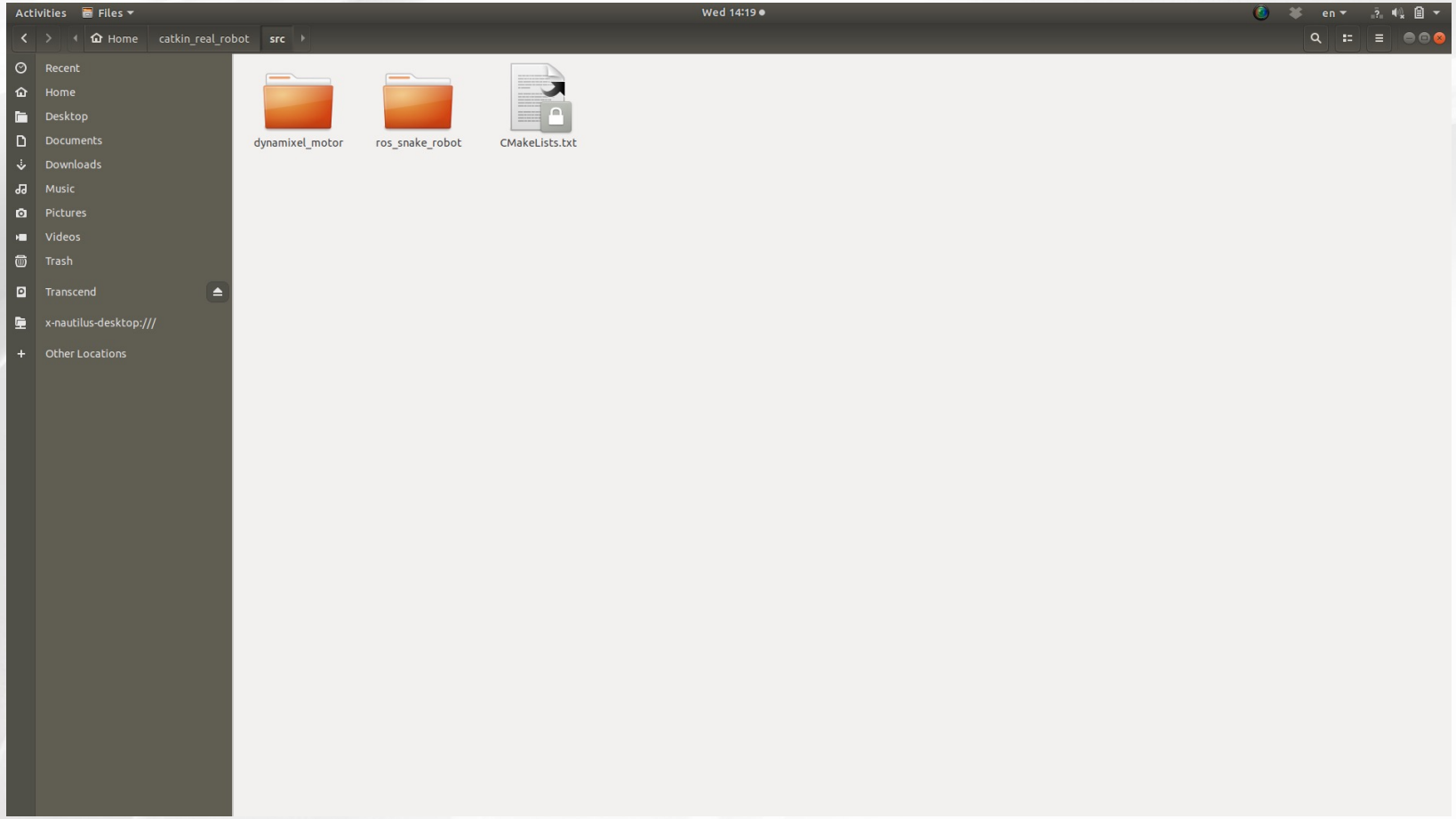
- Download effort controller:

sudo apt-get install ros-YOUR_ROS_DISTRIBUTION-ros-control

Prepare your PC:
Download packages to control the robot into your ROS worskspace

- Clone needed files from Github into your src:
- git clone https://github.com/arebgun/dynamixel_motor.git
- Build your code:
- catkin_make
- Source your workspace:
- source devel/setup.bash
-
- Clone needed files from Github into your src:
- git clone https://github.com/KNurlanZ/snake-noetic.git
- Build your code:
- catkin_make
- Source your workspace:
- source devel/setup.bash

Inside your workspace in src, you will see two packages



Run the simulator

- $ roslaunch gazebo_robot gazebo.launch

In a new terminal publish to the last joint a new angle /robot/joint5/command or so

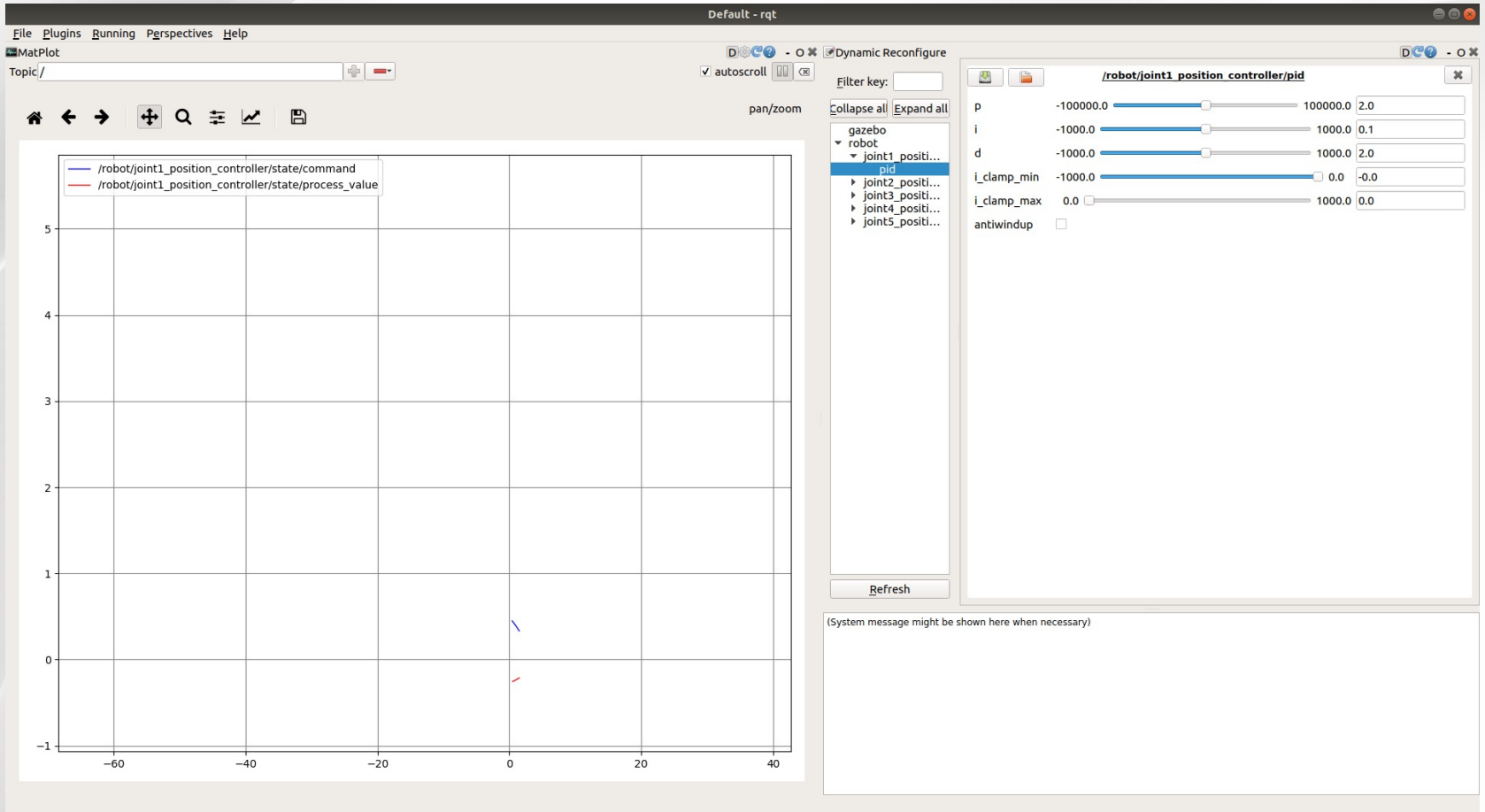
Do not run this now:

- **$ roslaunch gazebo_robot moveit_gazebo.launch**
- **$ roslaunch moveit_arm moveit_planning_execution.launch**

Visualize topics
rqt

- In a new (sourced) terminal, in order to see any Plot type

rqt



Robot in Simulator

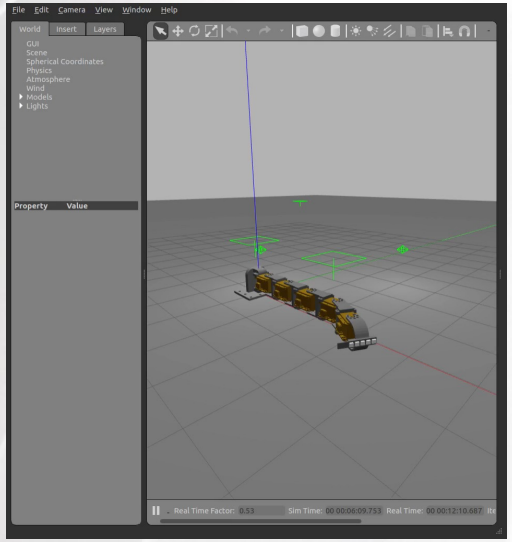
- Run Gazebo simulator

  roslaunch gazebo_robot moveit_gazebo.launch

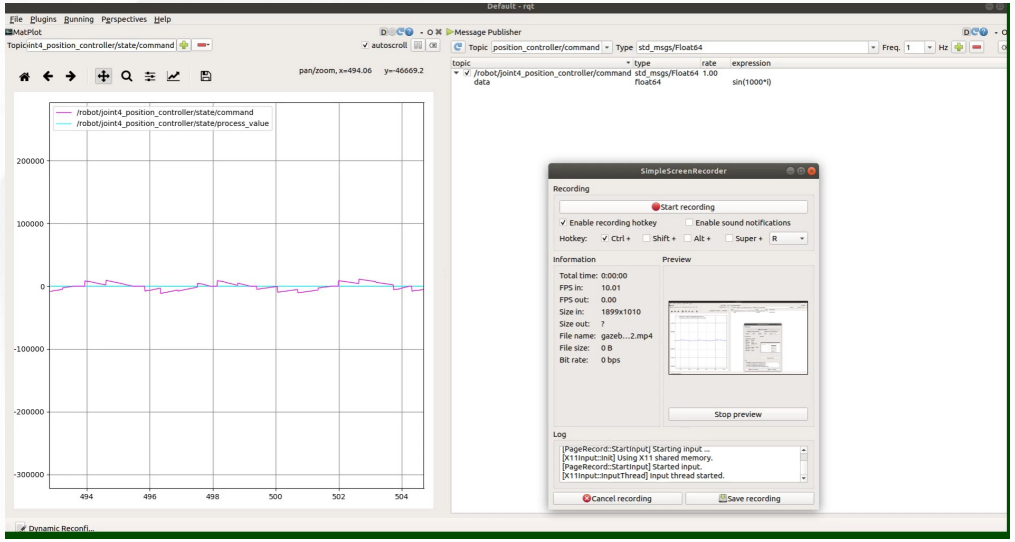
- Run controllers to visualize the robot

  roslaunch moveit_arm moveit_planning_execution.launch

Gazebo



rqt



Close all ROS nodes

- In every opened terminal do following:

Press buttons Ctrl and C simultaneously (CTRL+C).

Wait all terminals to stop the processes.

Run Real Robot

- Connect the usb adapter of the robot and then provide the access to the USB connection:

sudo chmod -R 777 /dev/ttyUSB0

- Prepare configuration files to run the real robot instead of simulation:
- In the folder moveit_arm/config/

Modify the file hand_tutorial_moveit_controller_manager.launch.xml

By changing

<rosparam file="$(find moveit_arm)/config/controllers.yaml"/>

- to

<rosparam file="$(find moveit_arm)/config/controllers_**real**.yaml"/>

- Save changes

Run the robot

- Connect the robot to a power source providing 12V.

You have already connected USB2Dynamixel adapter and gave the root access to the access to the USB connection.

- Run the packages that establish the connection with the motors

roslaunch my_dynamixel_tutorial controller_manager.launch

- The output "5 motors found" Then, start the controllers of the Planar Manipulator:

roslaunch my_dynamixel_tutorial start_moveit_arm_controllers.launch

After that corresponding topics will appear:

/motortom2m - first joint

/joint2 - second joint

/joint4 - third

/joint joint6 - fourth joint

/end - fifth joint

Move the joint

- From the terminal move the /end joint to 0
- From the terminal move the /end joint to 1

What do you think in which units this values are given?

Visualize the robot

- RVIZ is the tool that can visualize the state of the robot. In a new terminal type:

roslaunch moveit_arm moveit_planning_execution.launch

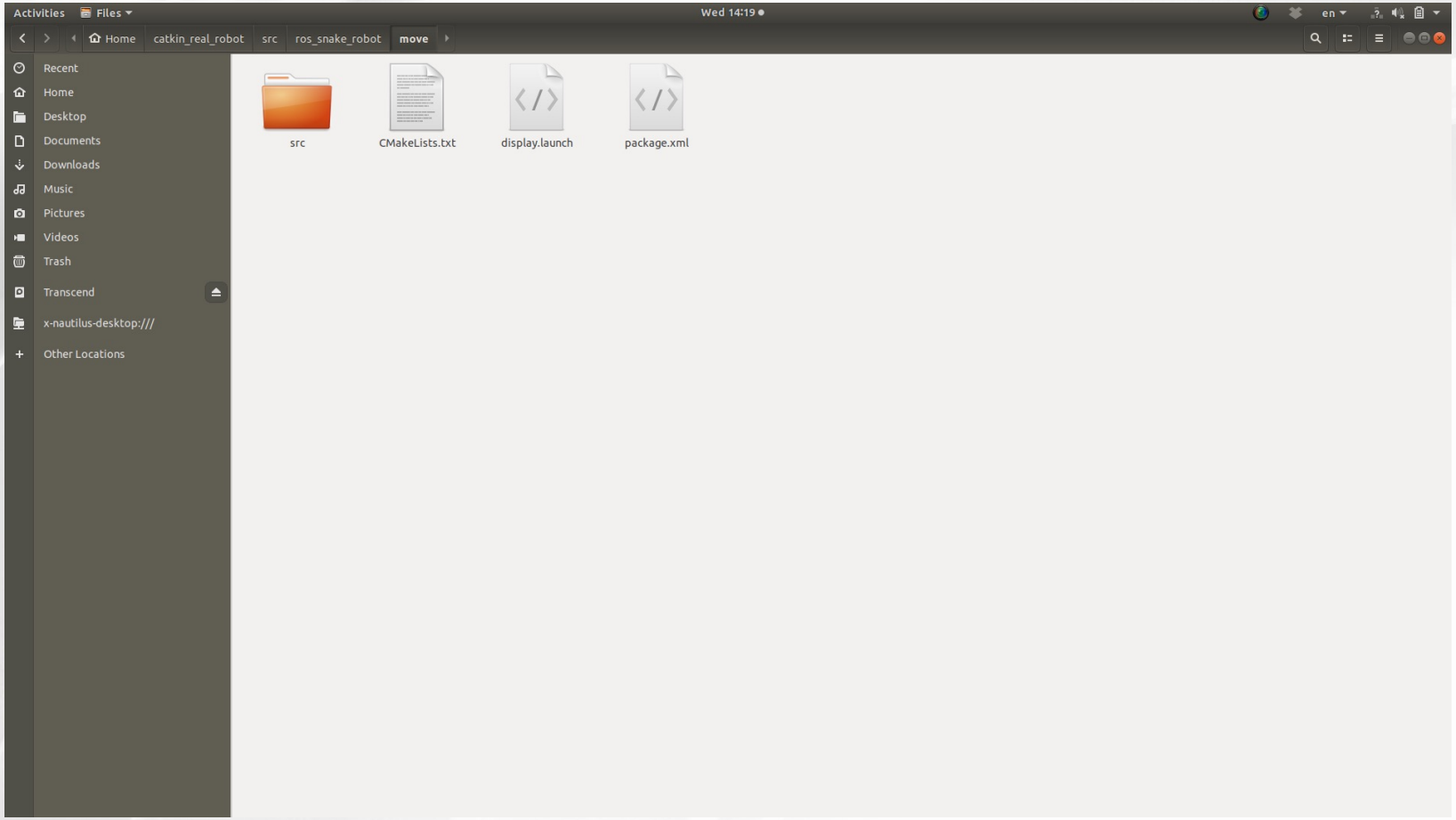
- From the terminal move the /end joint to 0

See the robot in RVIZ and Reality (or simulator if you are on Gazebo)

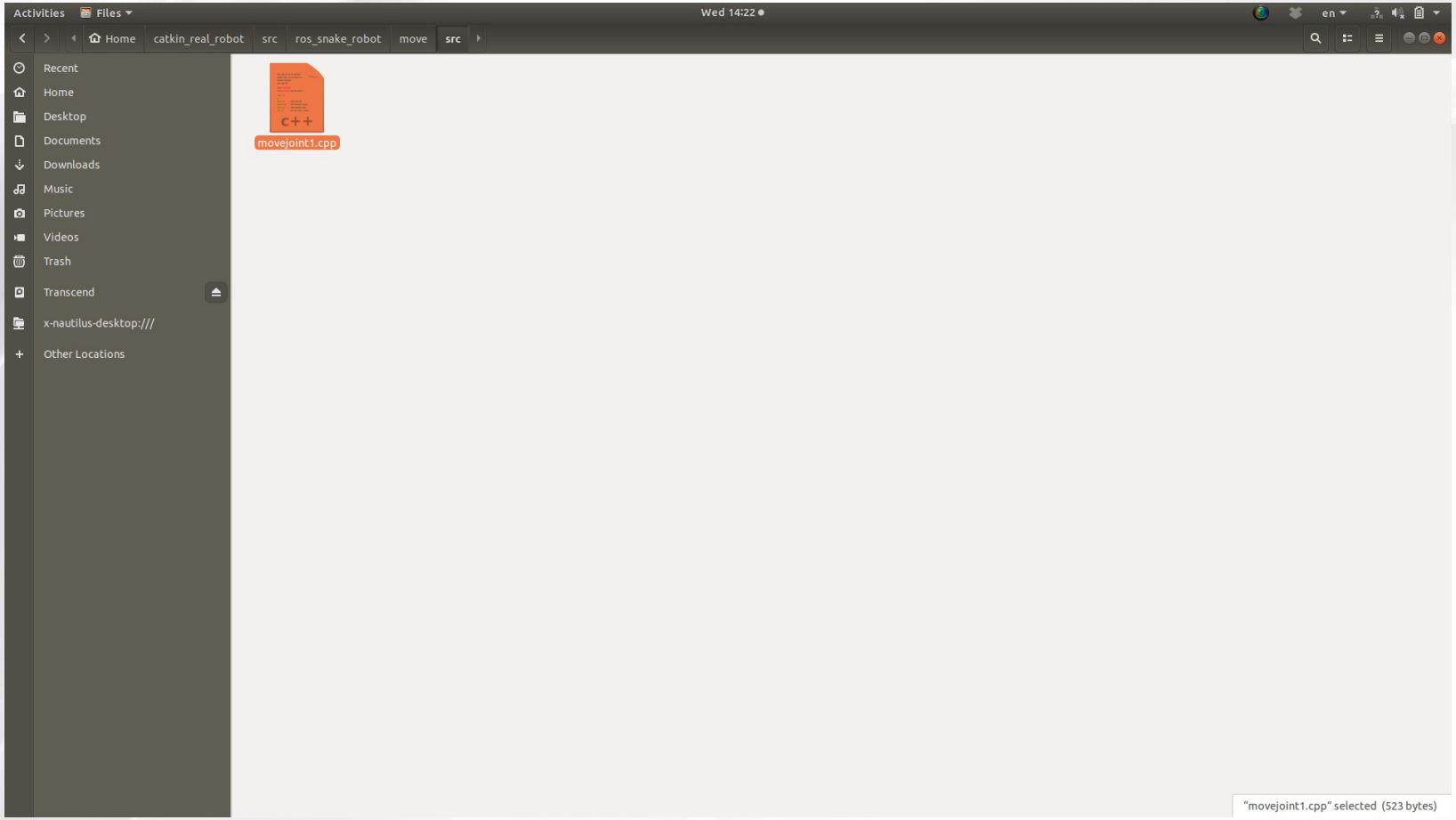
- From the terminal move the /end joint to 1

See the robot in RVIZ and Reality (or simulator if you are on Gazebo)

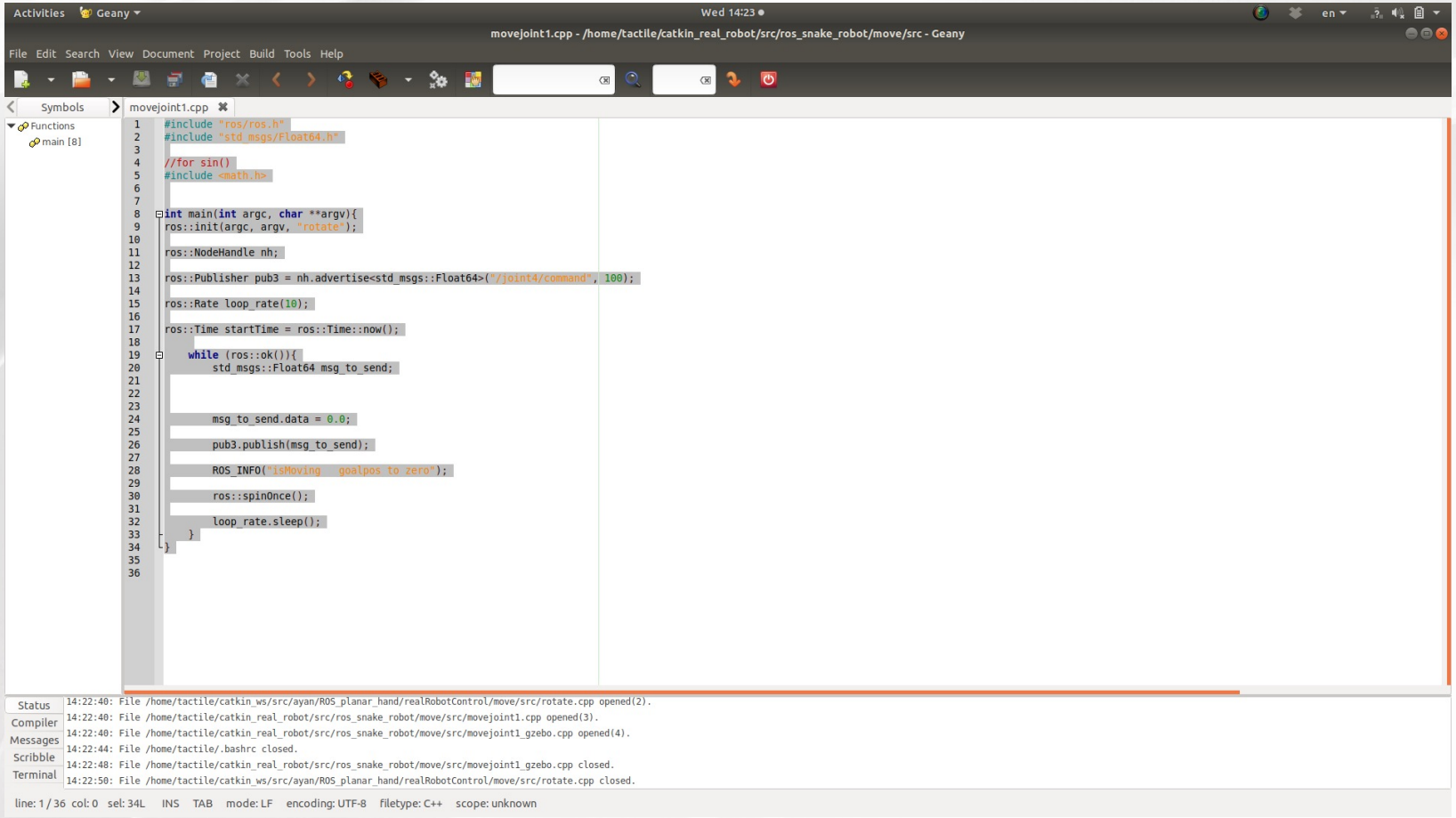
Control robot from a Node:
Create a new package to move a joint



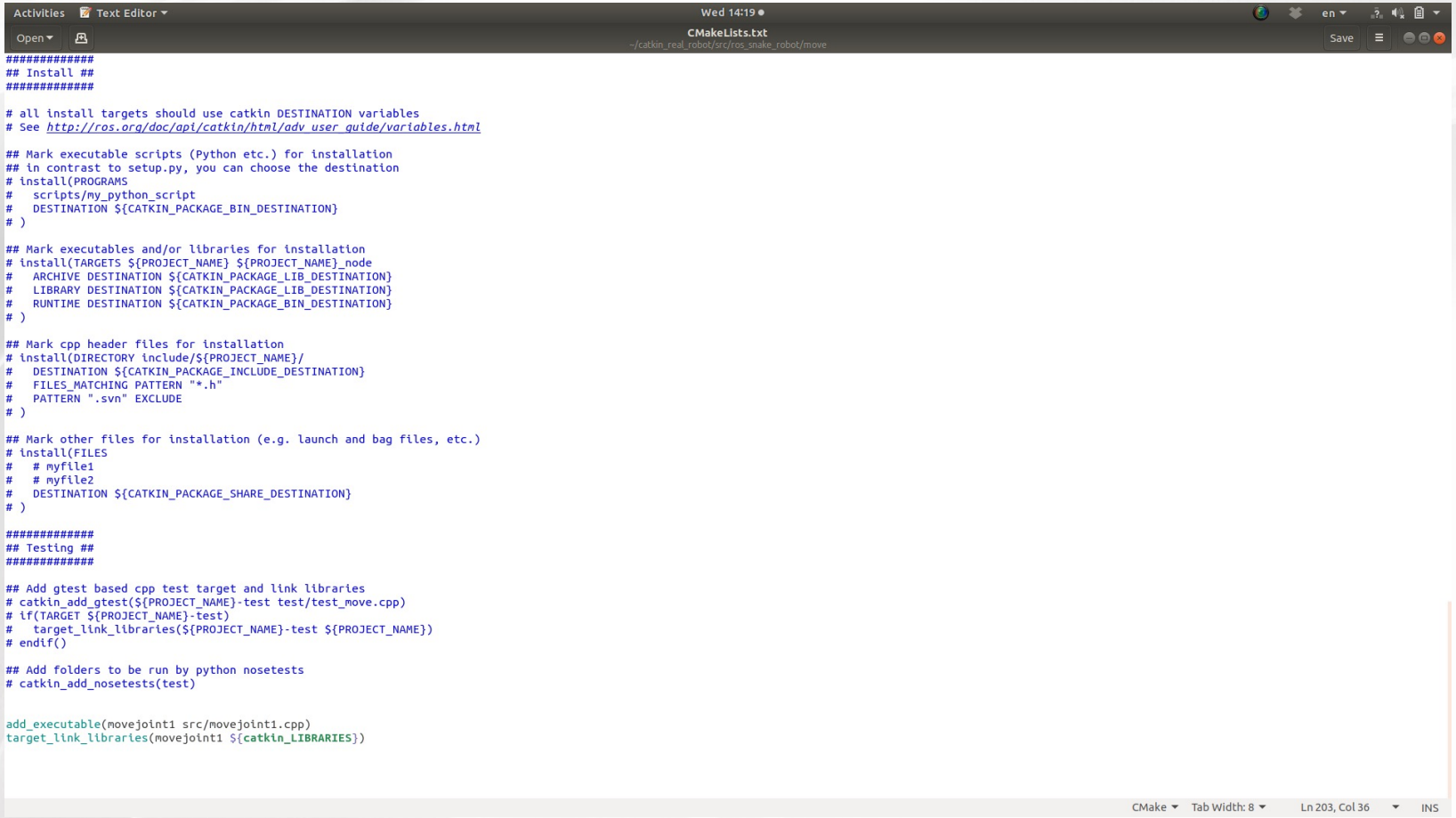
Control robot from a Node: Create a cpp or python file



Control robot from a Node: cpp file

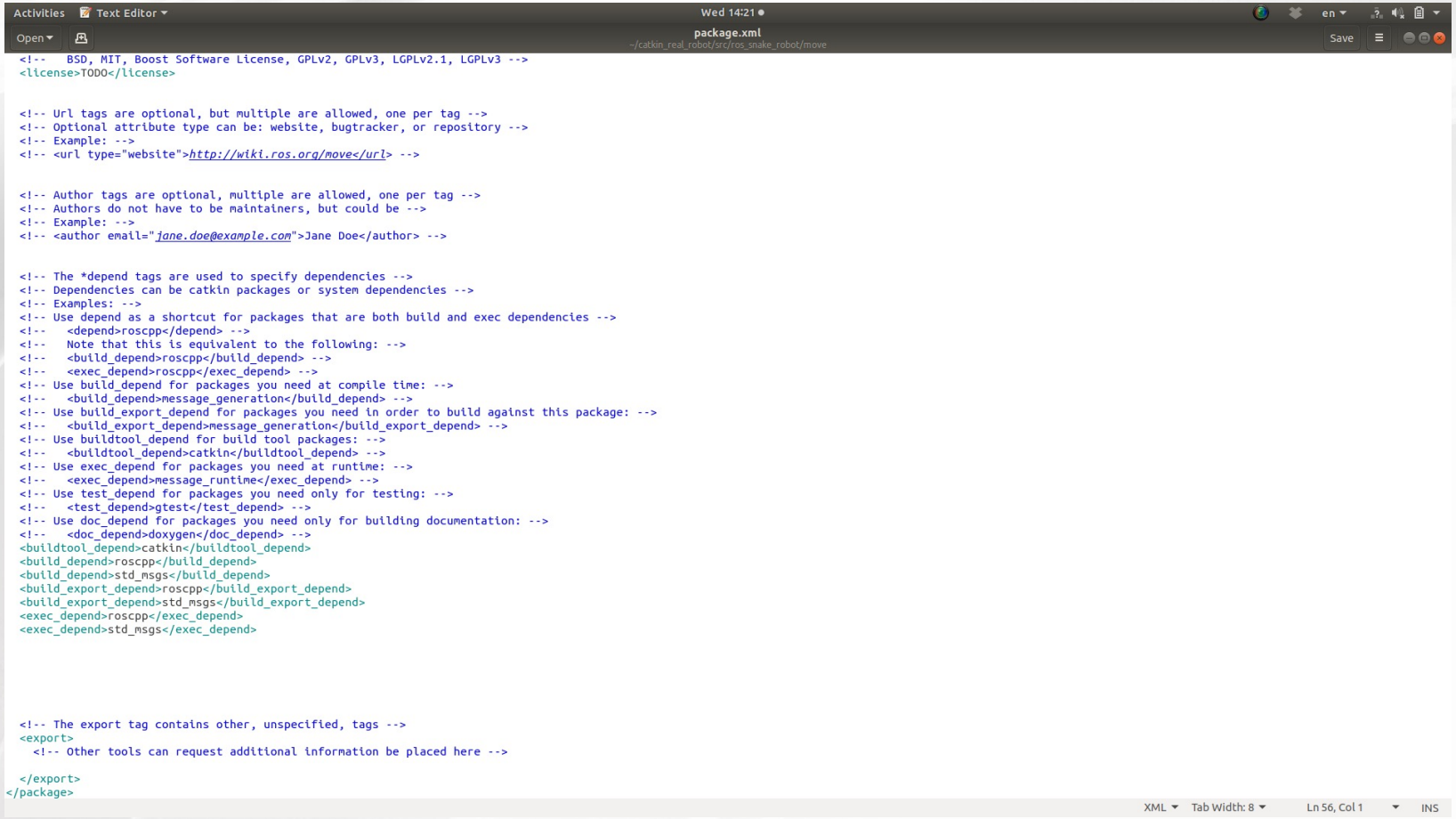


Control robot from a Node:
In cmake add it as executable




```
#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# install(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables and/or libraries for installation
# install(TARGETS ${PROJECT_NAME} ${PROJECT_NAME}_node
#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

#############
## Testing ##
#############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_move.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)


add_executable(movejoint1 src/movejoint1.cpp)
target_link_libraries(movejoint1 ${catkin_LIBRARIES})
```

Control robot from a Node: Package xml



Tasks:

1) Create a rosnode that will “listen” for std_msgs/Float64 type data and “publish” this data to the joint of the planar robot [1]. The node should send the command to move if the any new incoming value is higher than the previous one.
2) Get the step response (see example fig.1) of (you can create a node that will send a square-wave function):
the joint at the base of the robot
the joint at the end-effector of the robot

***Run “rqt” to Plot**

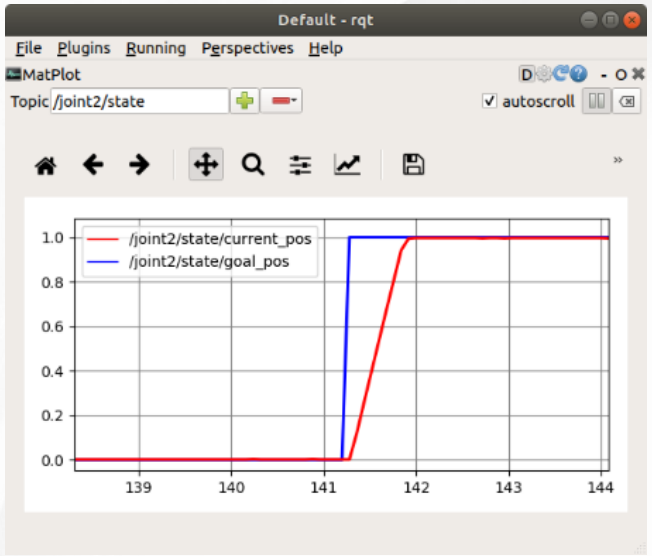


Fig.1 – example of the step response plot: blue- desired (command value), read the processing value (real value).

Tasks:

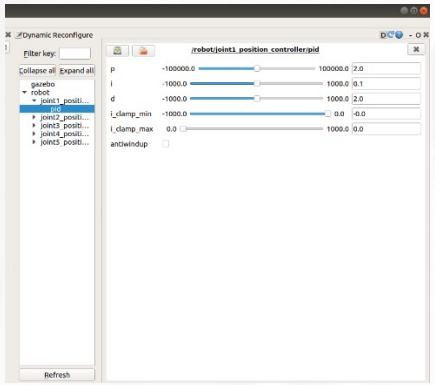
3) Get the sine-wave response of (you can create a node that will send a sine-wave function):

the joint at the base of the robot

the joint at the end-effector of the robot

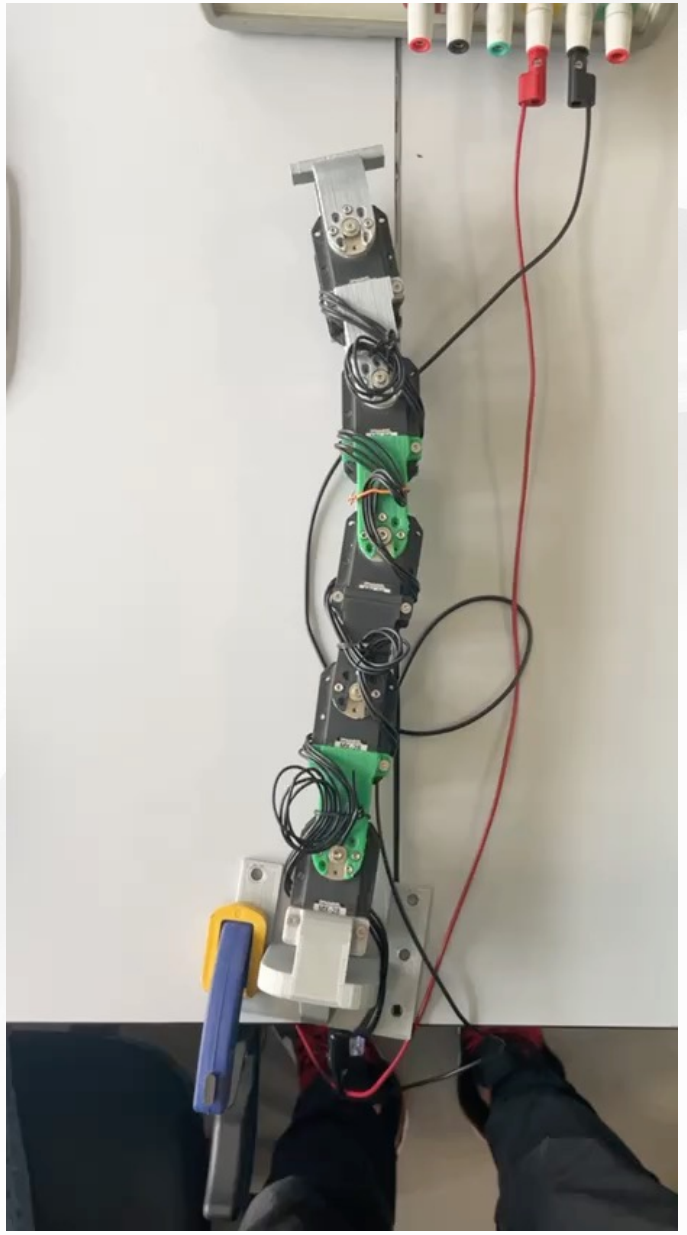
4) Decrease the Proportional gain of PID in the joints if possible and repeat 2 and 3.

Use rqt (type *rqt* in terminator, open tab Plugins → Configuration → Dynamic Reconfigure*)*. From the available configs find robot→jointNUMBER→PID→P



Tasks:

5) Make the robot move all joints like a snake



Debugging:

- 1) If you are using ROS Melodic:
- *sudo apt-get install ros-melodic-gazebo-ros-pkgs ros-melodic-gazebo-ros-control*
- *sudo apt-get --only-upgrade install ros-**
- 2) If you are using ROS Kinetic:
- *sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control*
- *3) if you are using ROS Noetic*
- *Rename “xacro.py” to “xacro”*

Thank you for your attention!

Okay, now we understood how we can use Nodes, Topics, Services. It is a building blocks of all ROS projects. In further labs we will explore RQT, RViz, Gazebo and other ROS tools that would be helpful for your future projects