AMORO Lab - Part 1: Introduction

Damien SIX
damien.six@ls2n.fr

Released: August 18, 2021

1 Objectives of the lab

The purpose of this lab is to perform the simulation of two parallel robots using ROS2 and GAZEBO. For each robot, you will compute the kinematic and dynamic models and compare them to the simulator output. Then, you will develop a computed torque control for the simulator.

Two robots will be simulated: a five-bar mechanism and a biglide mechanism. For the five-bar, the models are already available in your course and a reminder is provided with the content of this lab. For the biglide, in addition to the simulation, you have to perform and report the computation of the several models.

All the reports for this lab have to be done using LaTeX. Templates are provided in the lab content. If you have no knowledge of what is LaTeX, please refer to the the official documentation, the documents provided and your lab supervisor.

There is no lab report on the first part of the lab (five-bar mechanism). However, your work on the second mechanism (biglide) have to be fully reported.

2 Gazebo and ROS2

The simulation for this lab is performed by Gazebo. ROS2 topics are used to communicate with the simulation, providing torque inputs and getting measures (joints, end effector) from the simulation.

For the purpose of this lab, ROS2 and communication with Gazebo is hidden via a Python class Robot. In the scripts you will write, you can interact with the robot using

- *robot.start_oscillate()* to perform some oscillations on the robot joint (usefull to test models without predefined trajectory). You can also stop oscillations using *robot.stop_oscillate()*.
- *robot.active_left_joint.position* to access the position of the left active joint of the robot. Similarly, you can access any joint (passive/active, left/right) and several data (position/velocity/acceleration/effort).
- You can also access the end effector data using *robot.end_effector.position_x* for the x coordinate of the end effector. You can also access the y coordinate and velocity/acceleration data.

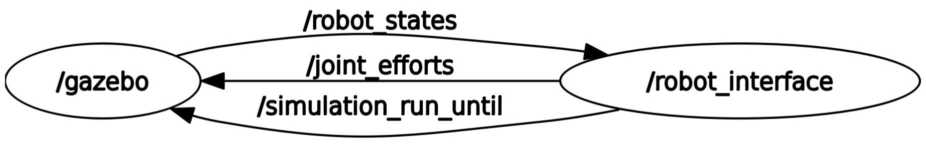

Figure 1: ROS2 nodes for the robot simulation

- You can set some joint efforts using *robot.apply_efforts(left_joint, right_joint)*. Indeed, the robot should not be in oscillate mode for this to work properly.
- *robot.get_time()* to get the current simulation time. Useful to plot curves.

The simulation is performed in a way that Gazebo is driven by the Robot class. Each time you send a torque input, the simulation will perform a 10 ms simulation then send you back the results. The normal way to control the simulation is

- Send a torque input using *robot.apply_efforts()* or *robot.continue_oscillations()* in oscillation mode.
- Wait for the results of Gazebo simulation. This is checked using *robot.data_updated()*.
- Do whatever you need to do.
- Send a new torque input.
- Loop as long as you need.

Scripts are already pre-filled to help you with this part.

3 Prepare for the lab

To prepare the lab, the following steps must be performed.

- Switch to ros2 by typing *ros2ws*.
- Create a *ros2* folder and a *src* sub-folder in it.
- Clone the lab folder into the *ros2/src* folder using the command
  $ git clone https://gitlab.univ-nantes.fr/six-d-1/lab_amoro
- Compile and install everything using the command *colbuild*.

- Test the correct installation using *ros2 launch lab_amoro gazebo.launch.py*
  If Gazebo is launched after typing this command, you are ready to go.

When you open a new terminal, you must always type the command *ros2ws* to work under ros2 (otherwise it is ros1 by default).