ROS Made Easy
6: ROS Control

Aniruddh K Budhgavi
Enigma, IIIT-B

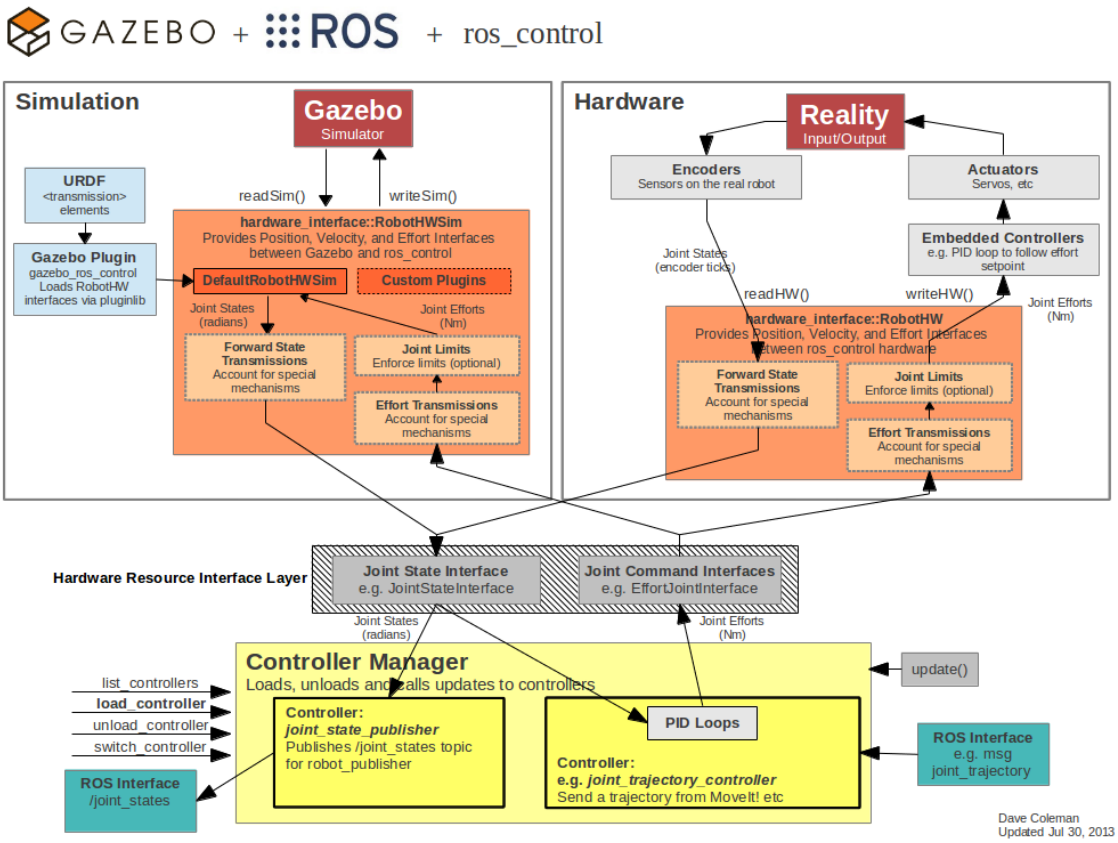
June 27, 2020

1 Notes

1. This tutorial was created for **ROS1 Melodic Morenia** on **Ubuntu 18.04 Bionic Beaver**, in **June 2020**. I expect them to become rapidly out of date. It is my hope that Team Enigma will continually maintain and update these tutorials.
2. This tutorial assumes that you are running Ubuntu, and have at least an elementary grasp of Python 2.7 and C/C++ .
3. All the code and the models for this tutorial are available at `https://github.com/aniruddhkb/enigmatutorials`.
4. The aim of this tutorial is to make you *functional* in ROS, not to make you a master. For that, look elsewhere.
5. Ensure that you have RViz, TF2, the robot_state_publisher package and the joint_state_publisher and joint_state_publisher_gui packages installed before proceeding further. You can install them using the relevant `apt-get` commands. See `http://wiki.ros.org/rviz/UserGuide`, this and this.
6. Ensure that you have **Meshlab** installed. Get it here.
7. Ensure that you have the **ros_control** and **ros_controllers** packages installed. Get them here.

2 Introduction

1. ROS provides an interface for controlling robot joints. These interfaces are fairly sophesticated – they provide a reasonable amount of feedback control. However, the documentation quality leaves something to be desired for beginners. Make yourself familiar with the contents of `http://wiki.ros.org/ros_control` and `http://gazebosim.org/tutorials/?tut=ros_control` before you begin this tutorial. Also ensure that you have the packages mentioned above.
2. Gazebo's tutorials page provides the following diagram to explain how the ROS interface and the Gazebo ROS control plugin work together.



Let's break it down.

(a) We have the Gazebo simulator, the URDF file and the Gazebo ROS control plugin.

(b) The Gazebo ROS control plugin sets up simulated hardware interfaces between ROS and Gazebo. These interfaces provide communication between Gazebo and the ROS controllers. They write and read from the sim, accept commands from the ROS interfaces and provide the feedback to the ROS interface. They also enforce joint limits, like the position, velocity and effort limits.

(c) The hardware interface set up by ROS is the hub of ROS control. *Theoretically*, it should be possible to use the same code to control the simulated robot and the real robot. Of course, there are some complications in this.

(d) The ROS controller manager handles the controllers. You can publish messages to this manager's topics to control the robot and you can get joint state information from this manager. This is also where the PID control loops are set up.

3 Enabling ROS control

3.1 Modifying the URDF

1. Take the URDF from the last tutorial and add the following lines between the `robot` tags.

```
<transmission name = "t1">
  <type>
    transmission_interface/SimpleTransmission
  </type>
  <joint name="j1">
    <hardwareInterface>
      hardware_interface/PositionJointInterface
    </hardwareInterface>
  </joint>
  <actuator name="a1">
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
<transmission name = "t2">
  <type>
    transmission_interface/SimpleTransmission
  </type>
  <joint name="j2">
    <hardwareInterface>
      hardware_interface/PositionJointInterface
    </hardwareInterface>
  </joint>
  <actuator name="a2">
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
<gazebo>
  <plugin name = "gazebo_ros_control"
          filename="libgazebo_ros_control.so"/>
</gazebo>
```

Line-by-line:

(a) The `transmission` tag is to set up the interfaces.

(b) There is only one `type` supported in Gazebo.

(c) The `hardwareInterface` corresponds to the intermediate layer which is used to communicate with the joints (both real and simulated). The full list of supported hardware interfaces is here. The important ones are:

- **Effort Joint Interface**: To pass effort commands to the joint.
- **Position Joint Interface**: To pass position commands to the joint.
- **Velocity Joint Interface**: To pass velocity commands to the joint.

All three interfaces mentioned above can accept effort, position and velocity commands from the controller manager. What changes is what commands they pass to the joints. Another important interface is

- **Joint State Interface**: To pass the data of the current joint positions, velocities and efforts to ROS in the form of arrays.

Note: The ROS and Gazebo tutorials erroneously mention that only Effort Joint Interface is supported in Gazebo. This information is out-of-date. I have successfully used both Position and Velocity interfaces in Gazebo, and will be showing you how to do the same in this and the next tutorial.

(d) The `actuator` tag is to specify things like gearbox ratios. Here, the gearbox ratio is one.

(e) The `plugin` tag is to launch the Gazebo ROS Control plugin.

3.2 The config file

Create `config/config.yaml`. The file:

```
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 60
j1_controller:
  type: position_controllers/JointPositionController
  joint: j1
j2_controller:
  type: position_controllers/JointPositionController
  joint: j2
gazebo_ros_control:
  pid_gains:
    j1:
      p: 150
      i: 0
      d: 50
    j2:
      p: 1
      i: 0
      d: 0.3
```

`position_controllers/JointPositionController` specifies that we wish to provide position commands to the joint, and we wish to give the controller position commands. The PID values are the same ones we calculated in the last tutorial.

3.3 The launch file

We're in the home stretch now. Create `launch/gazebo_ros_control.launch`:

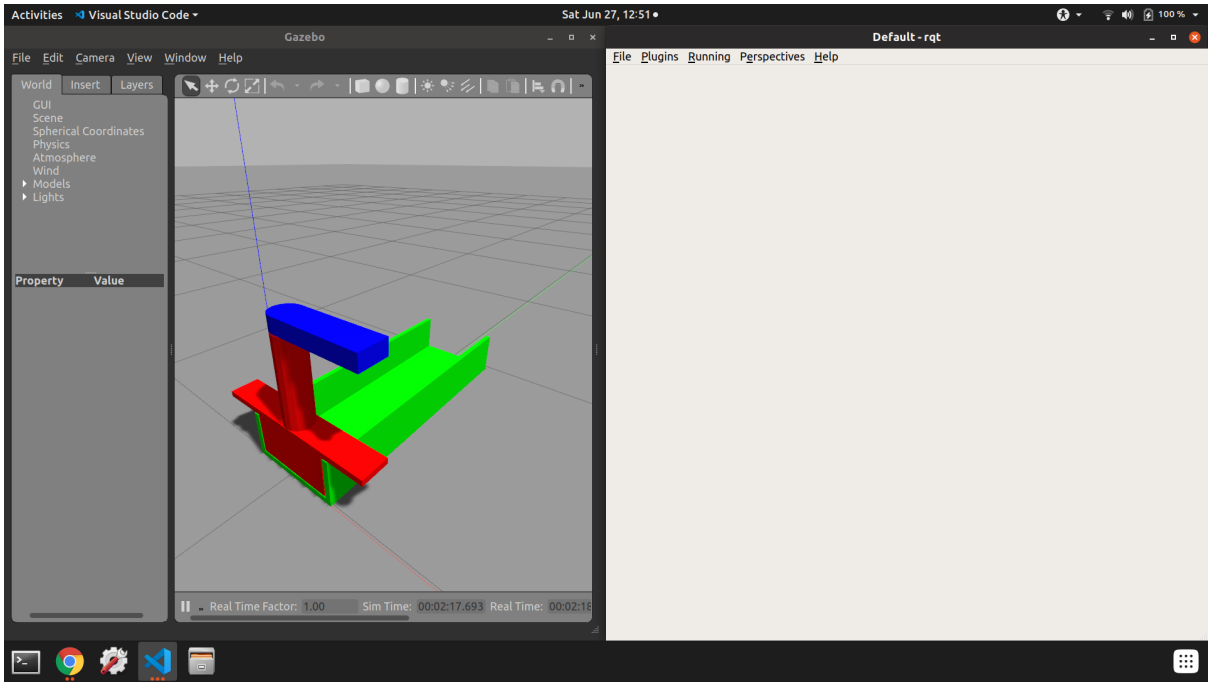
```
<?xml version="1.0"?>
<launch>
    <param name = "/robot_description"
           textfile="$(find turret_bot)/urdf/turret_bot.urdf"/>
    <rosparam command="load"
              file="$(find turret_bot)/config/config.yaml"/>
    <node name="robot_state_publisher"
          pkg="robot_state_publisher"
          type="robot_state_publisher"/>
    <include file = "$(find gazebo_ros)/launch/empty_world.launch"/>
    <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
        args=" -unpause -urdf -model turret_bot
            -param robot_description " respawn="false"/>
    <node name="controller_spawner"
          pkg = "controller_manager"
          type = "spawner" respawn = "false"
          args = "joint_state_controller j1_controller j2_controller" />
</launch>
```

The `rosparam load` command loads the configuration file into the parameter server. `controller_spawner` launches the controllers specified in the arguments.

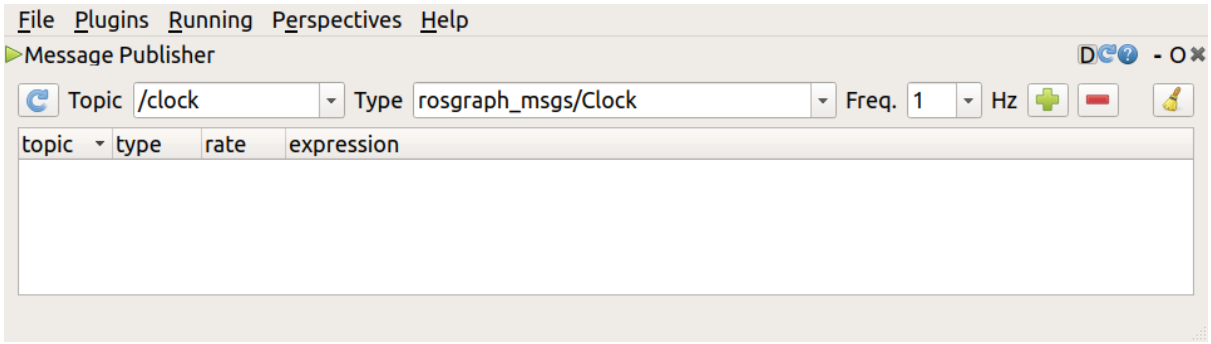
4 Controlling the robot

1. Launch the launch file. You should see the robot as before. **Pay close attention to the console output**. Anything like `P value not found` or `Controller not found` or `Hardware interface does not exist` should raise some alarm bells.

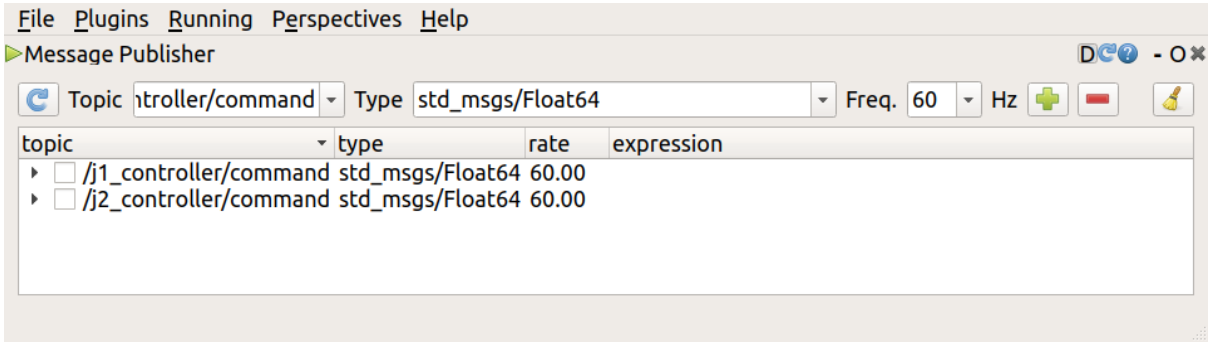
2. In a separate terminal, run `rosrun rqt_gui`. You should see:



3. Plugins – Topics – Message Publisher.



4. Add the following topics to the message publisher at 60 Hertz (use the + button).
   - `/j1_controller/command`
   - `/j2_controller/command`

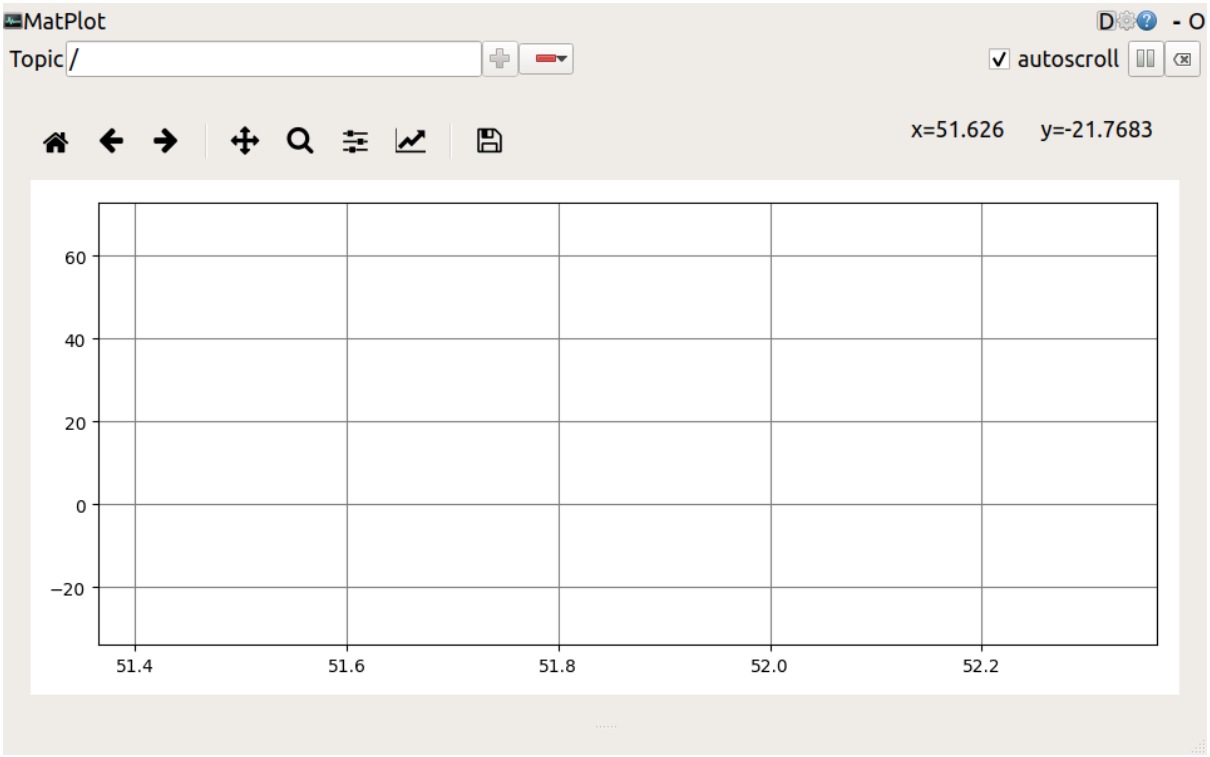


5. Click the small arrow beside the entry, change the data field from 0 to a sensible number, click the checkbox and see the robot move!

6. You can control the robot using the topics mentioned above. You can get joint state information from another topic. This is useful if you wish to write your own control algorithm or use a package like ROS MoveIt. **Note:** If the revolute joint is behaving strangely at +3.14 and -3.14, change the joint limits so that they are very clearly non-overlapping. The joint position controller gets confused if the joint limits are overlapping.

7. Now, let us tune the controller. Change the data field to $3.14\sin(\frac{i}{100})$. You should see the joint oscillate between the extreme positions.

8. In another terminal, run `rqt_plot`.

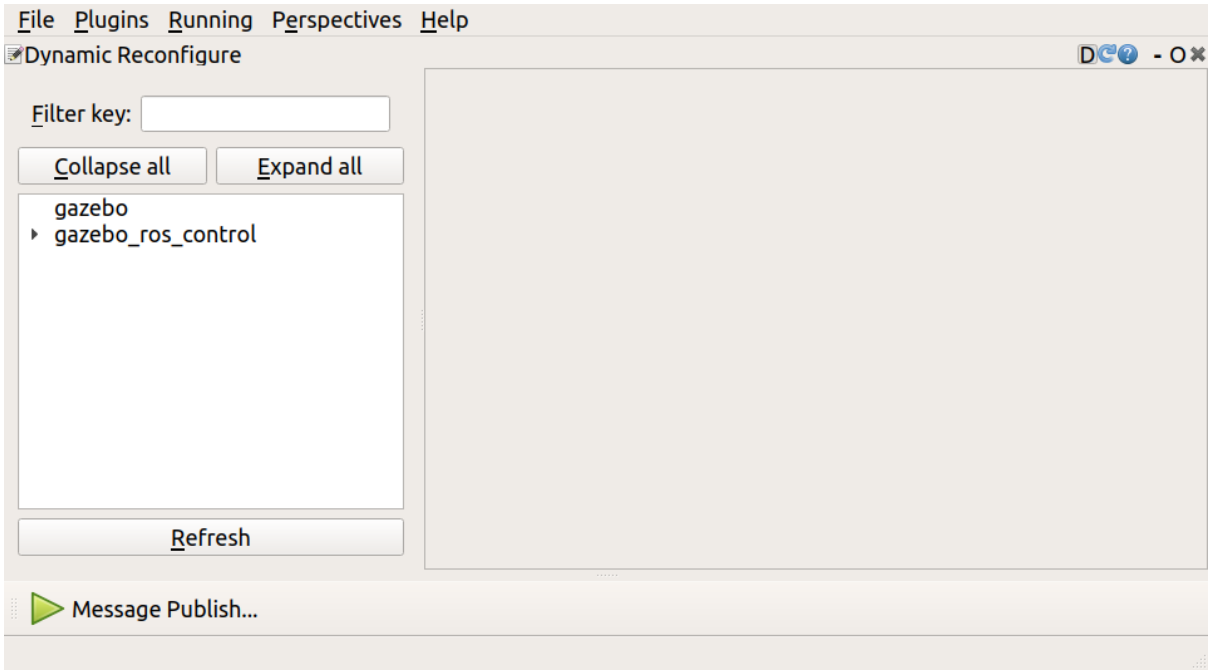
9. 

   Add the following topics:

   - `/j2_controller/command`
   - `/joint_states/velocity[1]`

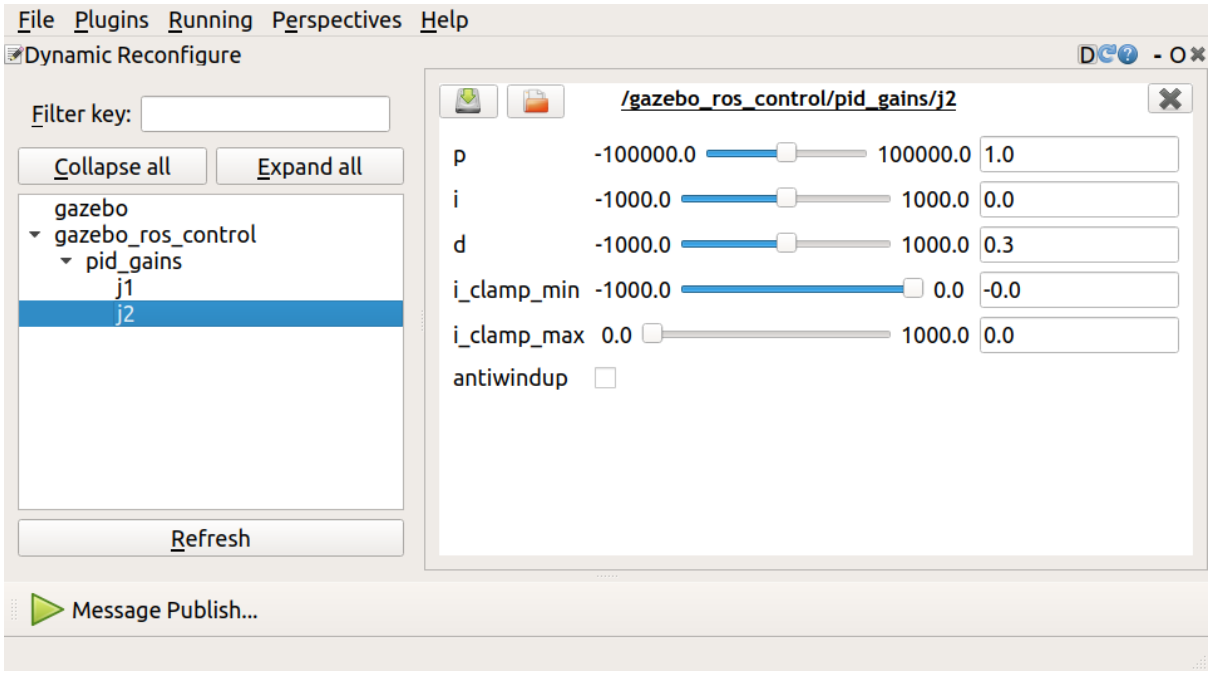
   You should see a moving plot.

10. To tune the PIDs to optimize the response, go back to `rqt_gui`, minimize the Message Publisher and add Plugins – Configuration – Dynamic Reconfigure.



Add `gazebo_ros_control/pid_gains/j2` to the display.

11.



You can tune the PIDs. Tune it such that the curves match up – but don't worry if you can't get it right. Manual PID tuning is a finicky thing.

5 Looking ahead

Congratulations! You should be able to simulate simple robots using Gazebo and ROS. Next, I recommend you learn how to integrate simulated sensors with the robot, and make something simple, like a line follower. See `http://gazebosim.org/tutorials?tut=ros_gzplugins` for this. (I may or may not release a tutorial for this, depending on demand).