Robotics Club IITKanpur

Implementing Controllers

In case of any queries related to this tutorial, contact **Ashwin Shenai**

This tutorial will walk you through implementing a P controller for the Husky simulation.
For submissions, fill **this** form as you do the assignment.

- Edit the launch file previously written such that the keyboard twist node is removed.
- Download this world file. Put that file in the **/usr/share/gazebo-9/worlds** folder.
- Edit the world path in your launch file to point the file you just placed. Execute the launch file and confirm whether the world loads with just a single pillar. If it doesn't, check if you have placed the file and written its path properly.
- In the **husky_highlevel_controller** package, create a new node for the controller. In this node, create a publisher on the topic **/cmd_vel** to give commands to the robot. You will need to include necessary packages.
- In the same node, create a subscriber to the **min_dist** topic to get the laser scan data.
- In the callback of the above subscriber, write a simple **P controller** to drive the Husky into the pillar. Your input i.e process variable is the laser scan output, and the setpoint is zero. Generate the required velocity input inside the callback function, store it in a global variable and then publish it using your publisher. Remember to use ROS parameters for the controller gain. You can use the same param file previously used.
- Build your package and include the above node into the launch file. Refer previous assignments and slides if you get stuck.
- Set the initial gain params to 1. Execute the launch file and confirm that your controller works. Fix all the errors, if any.
- Launch the simulation again, and take a screen recording of the Husky running into the pillar. Make sure that your viewing angle is good enough.
- Experiment with different values for the controller gain, try to find the gain for which the Husky takes minimum time to hit the pillar.

- **Bonus:** Try adding D control to the controller, and experiment with the D gain.

We are always looking for feedback and suggestions.
Contact the creator of this tutorial if you wish to provide feedback on this tutorial.

Reference: ETH-RSL : Programming for Robotics

Abhay Varshney | Madhur Deep Jain | Neil Shirude
Robotics Club, IIT Kanpur
roboticsclubiitkanpur@gmail.com

Creating custom ROS messages

In case of any queries related to this tutorial, contact **Ashwin Shenai**

This tutorial will familiarize you with using your own custom ROS messages.
For submissions, fill **this** form as you do the assignment.

- In the controller you just wrote, we would also like to see the setpoint error and control input generated at each step in one message. For this, we will create a custom message.
- Create the **msg** folder in your package. In this folder, create a message named **Control** with the appropriate file type.
- In the message you just created, add a header - use **std_msgs/Header** and two **float32** fields named **error** and **input**.
- Add the message to the **CMakeLists.txt**, and verify whether your package builds. Remember to add **message_generation** and **message_runtime** as dependencies.
- Now, include the message in your controller node. Create a new publisher that publishes this message to the topic **/feedback**.
- Store the controller error and input in global variable. Now create a new message object and in the while loop, load the variables into the message. Also add a time stamp using **ros::Time::now()**. Publish the message at a rate of **30Hz.**
- Build your package and check whether the topic gets published.
- Launch your simulation and record a rosbag of the **/min_dist, /feedback** and **/cmd_vel** topics. The bag should be around **5 seconds.**

We are always looking for feedback and suggestions.
Contact the creator of this tutorial if you wish to provide feedback on this tutorial.

Reference: ETH-RSL : Programming for Robotics

Abhay Varshney | Madhur Deep Jain | Neil Shirude
Robotics Club, IIT Kanpur
roboticsclubiitkanpur@gmail.com

Using ROS services

In case of any queries related to this tutorial, contact **Ashwin Shenai**

This tutorial will familiarize you with creating your own service servers and clients.
For submissions, fill **this** form as you do the assignment.

- In the controller node add a service server that can start and stop the robot.
- Use the **std_srvs/SetBool** service type for the service to accept boolean input.
- In the service call function, write code to set a **publish** flag to true/false depending on the request. The velocity message should be published only if the flag has been set to true.
- Build your package after doing all the necessary steps and fix all errors, if any.
- Execute the launch file you wrote in the last section, and call your service from the terminal to start the robot. Also verify that the robot stops when the service is called.
- Record a **rosbag** of the topic **/cmd_vel, /min_dist** and **/odom**. Start the robot with your service once you have started the recording, then stop the robot after 5 seconds by calling the service again.
- Now in the same node add a service client for the service you created. Call the service using the client inside the while loop so that it automatically stops the robot if it is too close to the pillar. Specify the stopping distance using a ROS parameter.
- Build your package and test whether the above node works by publishing a velocity command from the terminal. The bot should stop automatically when it is too close to the obstacle. Take a screen recording that shows this

We are always looking for feedback and suggestions.
Contact the creator of this tutorial if you wish to provide feedback on this tutorial.

Reference: ETH-RSL : Programming for Robotics

Abhay Varshney | Madhur Deep Jain | Neil Shirude
Robotics Club, IIT Kanpur
roboticsclubiitkanpur@gmail.com