Creating custom ROS packages

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

This tutorial will familiarize you with creating a ROS package from scratch.
For submissions, fill **this** form as you do the assignment.

- In this section, you will write a package from scratch that will ultimately be able to process data from a laser fixed on your Husky robot. First, navigate to the **src** folder of your workspace.
- Create a new package named **husky_highlevel_controller** with **roscpp** as the only dependency.
- Have a look at all the files and folders generated.

.

- We will be using the **LaserScan** message in our nodes. Find the ROS package which contains this message. Edit the **package.xml** and **CMakeLists.txt** files to include this dependency.
- Confirm that your edits are correct by building the package. Remember to **source** after your package has finished being built.
- If your package doesn't build, fix your mistakes by looking at the terminal output and referring to the slides. **You won't get any help here.**
- Now, create a **copy** of your **package folder. Zip** it and **upload** it to the form.

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 open-source packages

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

This tutorial will familiarize you with using an open-source ROS package.
For submissions, follow the instructions and fill **this** form as you do the assignment.

- In this section, you will control the robot in simulation using your keyboard. To do this, you will need the **teleop_twist_keyboard** package. Search for it on GitHub. Once you have found it, clone it into the src folder of your workspace.
- Build the package you have just cloned. Remember to source **devel/setup.bash**.
- Launch the Husky simulation in a suitable world. In a new terminal, run the node found in the **teleop_twist_keyboard** package using rosrun.
- Use your keyboard to confirm whether the Husky responds in simulation.
- Using the terminal commands, inspect all the nodes and topics.
- Instead of launching the teleop node in a different terminal, let us launch it with the simulation by creating a common launch file.
- In the same **teleop_twist_keyboard** package, create a new launch file in the appropriate directory. Name it like 'sim.launch' or something.
- In the launch file you have just created, **include** the Husky simulation launch file. Specify the world name as 'worlds/robocup14_spl_field.world'. Use the 'arg' tag as shown:

```
<include file="$(find husky_gazebo)/launch/husky_empty_world.launch">
        <arg name="world_name" value="worlds/robocup14_spl_field.world"/>
</include>
```

- After including the simulation launch file, now add the teleop node.
- Remember to check that all your tags are properly closed.
- Now, open up a new terminal and launch the file you have just written.
- Verify that the launch file works via keyboard.

- Once you have launched the file, open a new terminal and record a rosbag of the topics '/husky_velocity_controller/odom' and '/husky_velocity_controller/cmd_vel'. Increase linear velocity to at least 10, then start recording the bag. Drive the bot randomly for 10 seconds and then stop the recording.

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 publishers

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

This tutorial will familiarize you with creating a publisher and editing the CMakeLists file.
For submissions, follow the instructions wherever mentioned and fill **this** form .

- To test out whether the code you wrote in the previous section works properly, let's publish the variable **min_dist** to the terminal.
- In the same node create a publisher. We need to publish a **float**, so find the correct message and add its header. Also remember to include its package as a dependency. Build your package and source it at this point so that the required message gets added.
- Set the topic of the publisher to **/debug**. Set its queue size to **10.**
- Create an empty message object. Into the while loop that you have already created, load your variable data into the message object and add the command to publish it.
- To run the node, we will now need to edit the **CMakeLists.txt** file. Open it up, and add an executable linking to your node.
- Check if your package builds successfully. If it doesn't inspect the error messages and fix them. **You won't get any help from us here.** Remember to **source.**
- Verify that the correct values are published on **/debug**. To do this, launch your simulation and check that the published value is non-zero (but not inf.). If they aren't, you will have to verify the logic of your code to find the minimum value.
- Once you have corrected all mistakes and verified the output, record the output of **/debug** into a new rosbag. It should cover the duration of the given rosbag.

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 parameters

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

This tutorial will familiarize you with creating param files and adding params via launch files..
For submissions, fill **this** form as you do the assignment.

- Create a new parameter file that specifies the topic to **get laser data**, the topic to publish **min_dist**, and the queue size for the **subscriber and publisher**, and the **rate.** Place it in the appropriate folder.
- Implement the functionality of these parameters in the node you previously created. You can use either the **nh.Param()** or the **nh.getParam()** function. You will also need to replace the arguments of the subscriber and publisher.
- Open up the launch file you made in the first assignment. To the launch file, add your node. Inside this node, specify the path to your param file.
- Also, pass the argument **laser_enabled** as **true** in your launch file to the included **husky_empty_world.launch** file.
- Now in the param file, change the topic at which **min_dist** is published to **/min_dist**.
- Execute the launch file, and verify if your parameters are being loaded properly using the **rosparam get** command. If they are not, check whether you have used the right namespaces in your node and whether you have given the correct path in the launch file.

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