Lecture 2 | ROS Hands On

Husky Control, Data Visualization, and Object Following

Husky Control

Any robot spawned can be controlled programmatically (as well as using GUI if supported).

In our case, we will be using the Husky bot, which can be controlled with GUI as well, but we will explore how to control it programmatically.

The robot's rostopics are of importance here. The only way to interact with a bot, whether to control it or to interface it for info, is through its different rostopics to which it either subscribes to (can be used for control) or it publishes (can be used for collecting data).

First and foremost, in a terminal, start the master node roscore

```
roscore
```

Let's take a closer look at how to control Husky.

In a new terminal, spawn the robot.

```
roslaunch husky_gazebo husky_empty_world.launch laser_enabled:=True realsense_enabled:=True
```

Tip: Always keep a terminal in handy for listing rostopics and identifying required rostopics and their rosmsgs.

In a new terminal, list all the rostopics on roscore right now.

```
rostopic list
```

From this list, the one of importance to us as of now is the `/husky_velocity_controller/cmd_vel` topic.

Traditionally, most rostopics which can be used to control the velocity of the bot have the name `*/cmd_vel` (* indicates 0 or more characters; regex ftw 🙂). If in other applications/projects you don't observe this to be the case, then look for the rosmsg which the rostopic uses for being queried.

```
rostopic info /husky_velocity_controller/cmd_vel
```

In the output, the "Type" of message tells you the exact rosmsg which the rostopic uses, and for controlling a bot's velocity, the rosmsg `geometry_msgs/Twist` is the one.

In order to know the exact parameters required for publishing to the rostopic:

```
rosmsg info geometry_msgs/Twist
```

Based on this format, the publish command can be written. Again, for publishing, we need to use the `rostopic` command. Go through its various arguments to get a better understanding of what it does. For now, let's just get a feel of it:

```
rostopic pub /husky_velocity_controller/cmd_vel geometry_msgs/Twist "linear:
        x: 0.5
        y: 0.0
        z: 0.0
angular:
        x: 0.0
        y: 0.0
        z: 0.0" -r 10
```

This should make the Husky bot move in the x-direction at 0.5 m/s velocity. Changing the other parameters can make it traverse different trajectories, and this enables movement in multiple directions.

Data Visualization

For this part, you need to have rviz set up. Especially for the Husky bot, it is recommended that you install and start a preconfigured rviz instance. It provides a very complete and beautiful visualization of where the bot is in the environment and what it is currently sensing.

```
sudo apt-get update
sudo apt-get install ros-melodic-husky-desktop
sudo apt-get install ros-melodic-husky-simulator
```

That being said, it is not necessary to install them. It is possible to work with rviz off-the-shelf as well, and it is much better to get used to it because not all applications/projects are released with custom rviz plugins.

If you go through the robot's model or even through its rostopics, you will notice that the bot has a few sensors on-board. In our setup, it has a laser scanner (LiDAR), an RGB-D camera (Intel RealSense), an IMU, and wheel encoders for odometry. All of this data can be subscribed to, either in code to process data from those topics, or by directly visualizing it on rviz, or even to debug on a terminal. We can view it on the terminal as follows:

```
rostopic echo <rostopic-name>
```

Like above, going through `rostopic`'s help section will give you a better understanding of how it can be used.

To visualize directly on rviz, open a new terminal, and either directly enter `rviz` and manually set all the rostopics you would like to view (will show this through screenshare), or just open the preconfigured plugin which was mentioned above.

```
roslaunch husky_viz view_robot.launch
```

This opens an rviz window fully customized for Husky. Now, just for the sake of good visualizations, drag and drop a few shapes and assets if you can. If not, just go to `Insert -> /path/to/.gazebo/models` and select a cool world from it.

Now that you have your bot in this world, turn on your velocity publisher code and observe how the robot moves (odom) and scans the environment and creates a pointcloud (which comes from the RGB-D camera; check rostopic name) of whatever it senses. Apart from this, at the bottom of the `Displays` tab, you can `Add` more topics you want to visualize, like `/realsense/color/image_raw` for example.

Hands-On Application

Applications in this are virtually limitless. You could run a full end-to-end SLAM pipeline, could simulate object detection and collision detection, navigation in an environment, and probably many more.

We will implement a very basic linear object follower on a toy sequence which we will simulate on Gazebo, and visualize everything on RViz as well.

Firstly, launch Husky in Gazebo:

```
roslaunch husky_gazebo husky_empty_world.launch
```

We could implement this as a part of a ROS package, but this example simply shows that there is no need to create a whole package with a launch file and other formalities, rather all of the confusion can be omitted. That being said, creating a package with proper files in place is the right way to go about with it.

```
import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
import numpy as np

def callback(msg):
    distance = np.min(np.array(msg.ranges))
    move_cmd = Twist()
    if distance > 1:
        move_cmd.linear.x = distance # P-controller, if you are aware
    else:
        move_cmd.linear.x = 0
    velPub.publish(move_cmd)

rospy.init_node("scans")
laserScanSub = rospy.Subscriber("/scan", LaserScan, callback)
velPub = rospy.Publisher("/cmd_vel", Twist, queue_size=1)
rospy.spin()
```

1. Create a node

    ```
    rospy.init_node("scans")
    ```

2. Initialize a subscriber to `/scan`

    ```
    laserScanSub = rospy.Subscriber("/scan", LaserScan, callback)
    ```

    Need to specify the rostopic, the rosmsg, and initialize a callback function (a function passed as an argument and triggered when required).

3. Setup the velocity publisher by publishing to `/cmd_vel`

    ```
    velPub = rospy.Publisher("/cmd_vel", Twist, queue_size=1)
    ```

    Need to specify the rostopic, rosmsg, and queue_size (basically a buffer) of messages required for the query.

4. Start the node by running:

    ```
    rospy.spin()
    ```

    It simply keeps your node from exiting until the node has been shutdown.

5. The callback function uses the current message from the subscriber to extract info and perform actions based on it.

    ```
    def callback(msg):
        distance = np.min(np.array(msg.ranges))
        move_cmd = Twist()
        if distance > 1:
            move_cmd.linear.x = distance
        else:
            move_cmd.linear.x = 0
        velPub.publish(move_cmd)
    ```

    `msg.ranges` gives an array of the ranges of points detected by the LiDAR.

    Using the message type of `geometry_msgs/Twist`, we can publish velocity to the bot using our velocity publisher above.

    Look up the different message formats which could be used to get an idea of its parameters

```
rosmsg info geometry_msgs/Twist
rosmsg info sensor_msgs/LaserScan
```

In a new terminal, run this script and you will notice changes in Gazebo.

To visualize this in RViz,

```
roslaunch husky_viz view_robot.launch
```

Using the GUI, access any topic you wish to visualize on RViz. It could be images, odometry, LiDAR pointclouds, stereo-reconstructed pointclouds, etc.

▼ Exercises (Optional)

1. Implement the above pipeline for obstacles not necessarily along the X-axis
2. Implement an object follower where the object can move freely as well anywhere in the X-Y plane (uses CV concepts; take lite for now xD)
3. Implement a freely moving autonomous bot which detects obstacles and turns away from them (easy, just stop the bot at some distance from an obstacle, rotate it randomly, proceed to do the same thing)