Robotics Competition 2021-22

eYRC 2021-22: Agri Bot (AB)

Overview of rospy

Prerequisite: ROS Topics

In your theme, you will be required to do many of these things programatically, that is via programs or scripts. Advantages of using Python and rospy to do so are that the Python scripts do not need to be compiled by ROS build system and can be placed anywhere in the filesystem. Go through the ROS wiki and tutorials pages of rospy.

As an example we will be looking at how you can make the turtle in turtlesim travel in a circle and stop after one revolution, using rospy. For doing this we will be using the topic `/turtle1/cmd_vel` to give it appropriate angular and linear velocity, and we will be monitoring the `/turtle1/pose` topic to stop when one revolution is complete. To do so we will write simple publishers and subscribers in Python, to get started check out this tutorial on the ROS wiki.

We will first import the required Python libraries, that is rospy as well as sections of certain message libraries that we require.

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose
```

The very first line, called a shebang, is required to let rospy know where the version of python we want ot use is located. ROS uses the system Python by default.

We can initialize the ROS node as follows:

```
rospy.init_node('turtle_revolve', anonymous=True)
```

The topic `/turtle1/cmd_vel` publishes the message of type geometry_msgs/Twist. turtlesim interprets this in the units of m/s and rad/s for linear and angular velocities respectively. These velocities are in the turtle's own frame of reference, thus only the linear.x , linear.y and angular.z velocities are applicable in turtlesim as the turtle can only move in these directions. We initialize the publisher and the message it will publish as follows:

```
velocity_publisher = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
vel_msg = Twist()
```

After assigning the relevant velocities to `vel_msg`, we can publish it by doing the following at our desired rate.

```
velocity_publisher.publish(vel_msg)
```

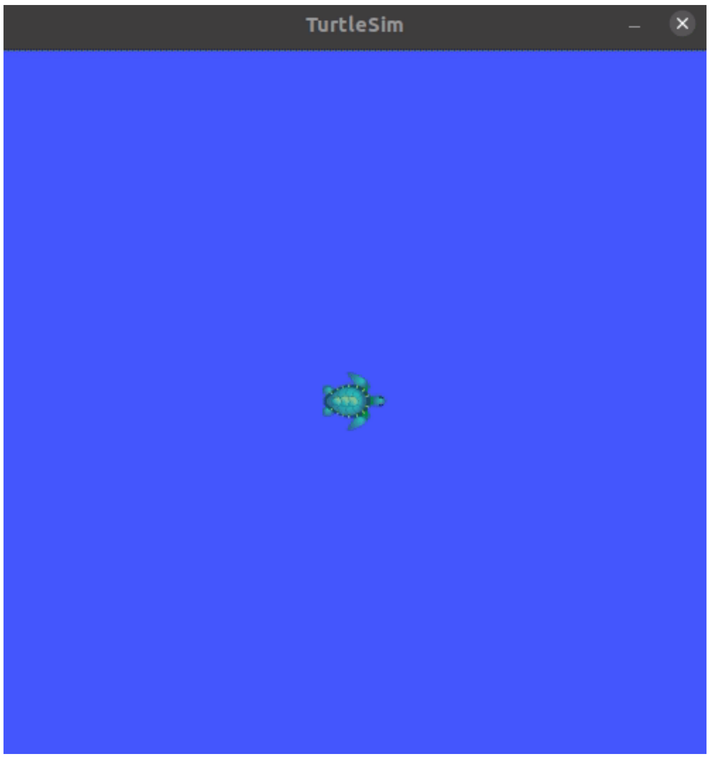
Meanwhile we need to subscribe to the turtle's pose messages.

```
    rospy.Subscriber("/turtle1/pose", Pose, pose_callback)
```

This will execute the pose_callback function whenever a message is received. An example pose_callback function:

```
def pose_callback(msg):
    print(msg.theta)
```

The final output should look like this, although you may play around with the velocities.


TurtleSim