Experiment number 2

Turtlesim

Objectives

1. To familiarize with simulation using ROS
2. To familiarize the topics and messages related to a mobile robot.
3. To implement basic path tracking.

Theory

A simple way to learn the basics of ROS is to use the turtlesim simulator that is part of the ROS installation. The simulation consists of a graphical window that shows a turtle-shaped robot. The background color for the turtle's world can be changed using the Parameter Server. The turtle can be moved around on the screen by ROS commands or using the keyboard. Turtlesim is a ROS package, and the basic concepts of package management .

Procedure

1. Open the terminal, change to ROS Directory and set up the environment

>> cd catkin_ws

>>source devel/setup.bash

2. Start a ROS master

>>roscore

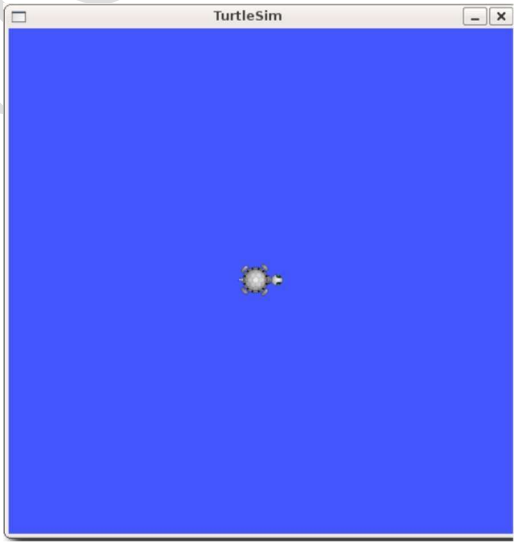
3. Open a new tab, from file and set up the environment

>> source devel/setup.bash

4. Type the following command

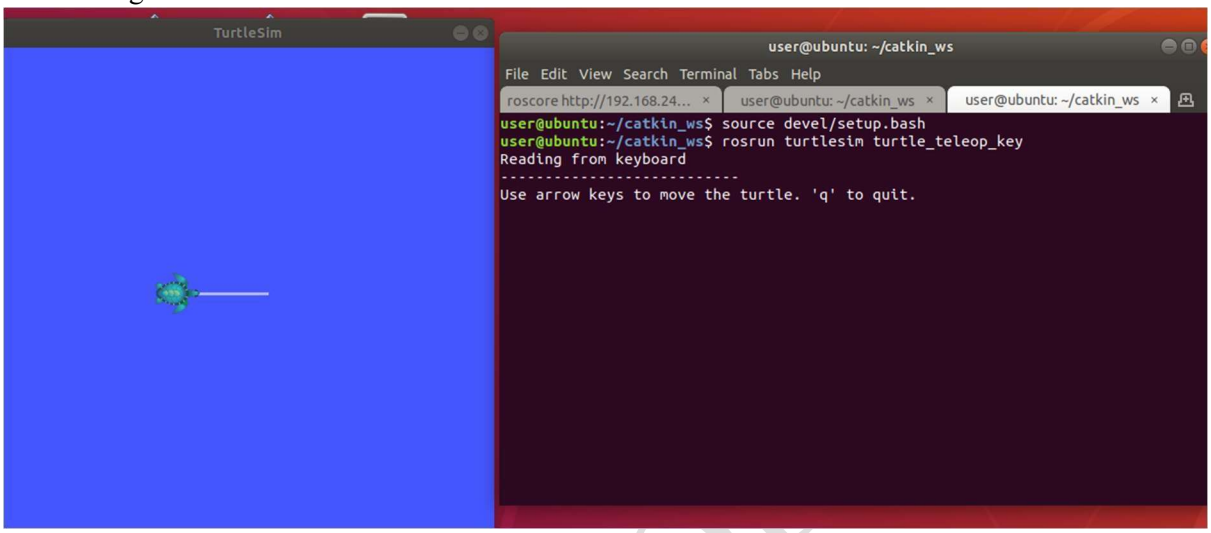
>> rosrun turtlesim turtlesim_node

A simulator will be open up now as below



Moving the turtle using keyboard keys:

5. Open a new tab and set up the environment
   >> source devel/setup.bash
   Then
   >> rosrun turtlesim turtle_teleop_key it
   will run as given below



6. Place your pointer in the terminal and use ‘up’, ‘down’, ‘left’ and ‘right’ using arrow keys.

Moving the Turtlesim using Python script

7. Stop the code running in step 5 using cntl+C
8. Stop the ROS master (first terminal with the last line ‘roscore’, and enter cntl+c)
9. Change the directort to src inside the catkin_ws
   >> cd ~/catkin_ws/src
10. Create a new package in the src using following command
    >>catkin_create_pkg turtle_tutorial rospy roscpp (General syntax is catkin_create_pkg <package_name> [depend1] [depend2] [depend3])
11. Run the following code
    >> cd ~/catkin_ws
    >>catkin_make
12. Goto the folder catkin_ws>>src>>turtle_tutorial and create a folder Scripts
13. Inside the above folder, create a python script with title ‘turtle_publish.py’ 14. Copy the code to the script and save
15.

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist

rospy.init_node('topic_publisher') pub =
rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=1) rate =
rospy.Rate(2)
move = Twist() # defining the way we can allocate the values
move.linear.x = 1 # allocating the values in x direction - linear
```

```
move.angular.z = 1 # allocating the values in z direction - angular while
not rospy.is_shutdown():
  pub.publish(move)
rate.sleep()
```

16. On the same folder , create a python script with title ‘turtle_subscribe.py’
17. Copy the code to the script and save

```
#!/usr/bin/env python

import rospy from geometry_msgs.msg
import Twist from turtlesim.msg
import Pose def
subscriberCallBack(data):
   rospy.loginfo(rospy.get_caller_id() + " I recieved --  %s", data.x) #prints on terminal

def listener():
   rospy.init_node('subscriberNode', anonymous=True)
rospy.Subscriber("turtle1/pose", Pose, subscriberCallBack)     rospy.spin()
# the python file does not exit

if __name__ == '__main__':
   listener()
```

18. Open a new roscore >>roscore
19. Enter a new terminal
    >>source devel/setup.bash
20. Run
    >>rosrun turtle_tutorial turtle_publish.py