ROS Assignment

Vasileios Papadopoulos

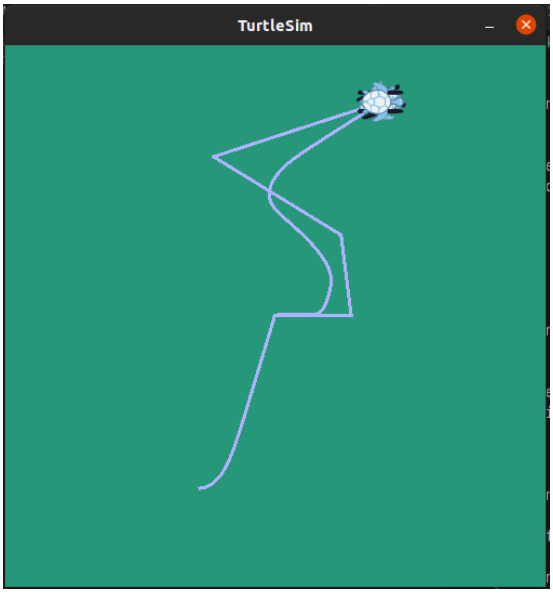
Binaries

ROS Noetic is installed in following directory: /opt/ros/noetic/bin

Launch turtle bot

A good practice when working with ROS is to launch the *roscore* underline system, which provides communication among ROS nodes. Though, in our case it is redundant since *roslaunch* launches it automatically if detects that has been stopped. We continue with turtle demo project.

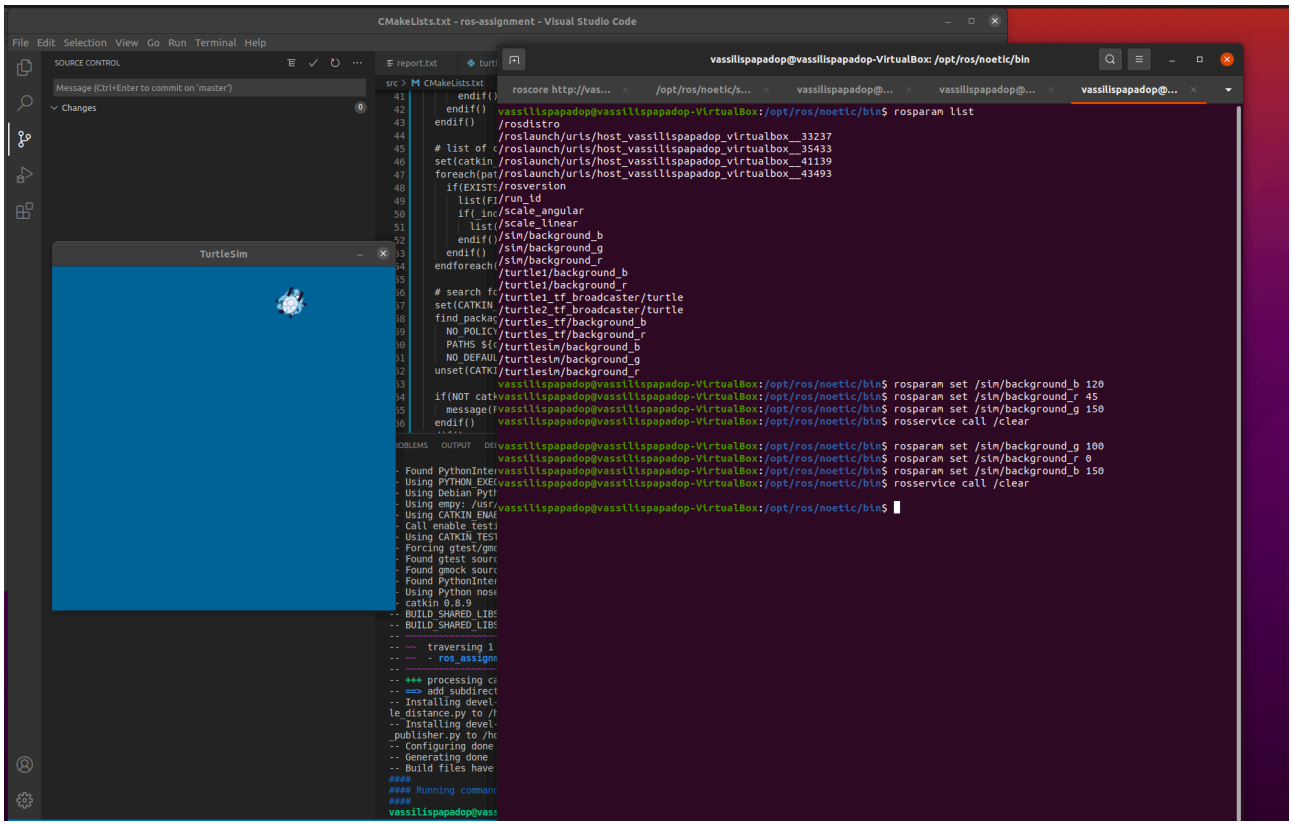
1. roscore
2. roslaunch turtle_tf turtle_tf_demo.launch



Change background

In order to change the background color of turtlesim we use the *rosparam* command. Particularly, we set RGB colors individually and arbitrarily and then we use *rosservice* command to apply the changes. In order find the available parameters, we first run the *rosparam list*

1. rosparam list
2. rosparam set /sim/background_r 0
3. rosparam set /sim/background_g 100
4. rosparam set /sim/background_b 150
5. rosservice call /clear #apply changes



Background changes to blue-ish, though we notice that previous line path has also been changed/ cleared.

Inspect tf tree

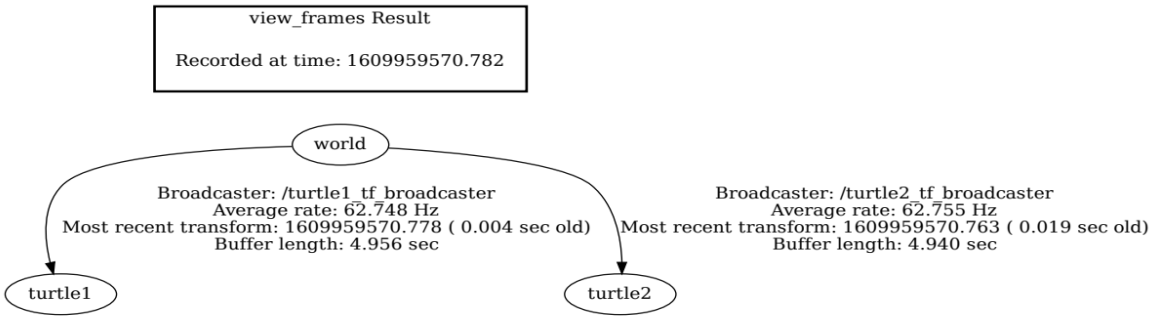
In order to run tf view_frames command I had to change line 89 in ***/opt/ros/noetic/lib/view_frames*** to avoid **TypeError: cannot use a string pattern on a bytes-like object**

line 89 replaced with :

decoded = vstr.decode('utf-8')
m = r.search(decoded)
$ lsb_release -a

Distributor ID: Ubuntu
Description: Ubuntu 20.04.1 LTS
Release: 20.04
Codename: focal

1. rosrun tf view_frames #OR
2. rosrun rqt_tf_tree rqt_tf_tree
3. evince frames.pdf #view pdf



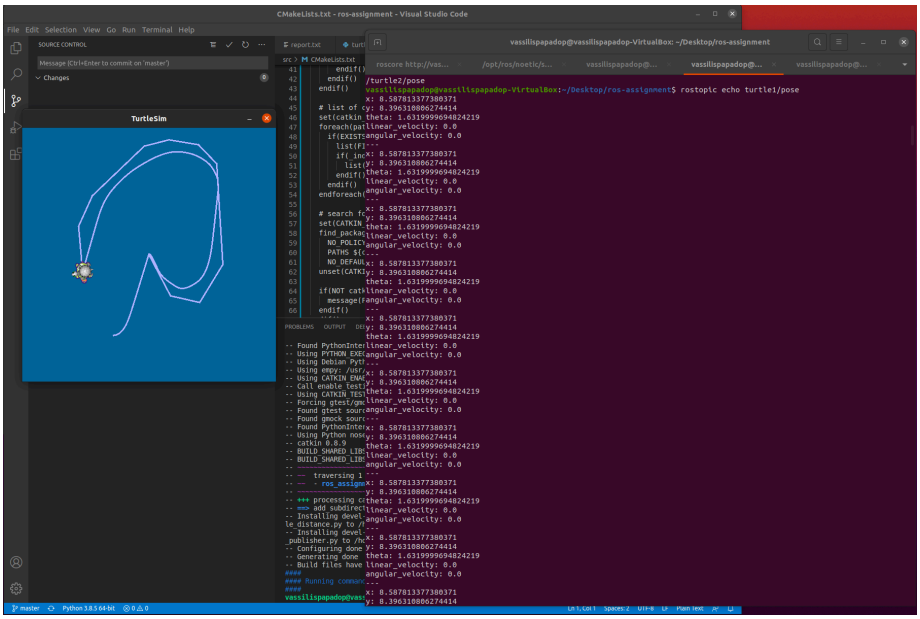
Display topic messages

rostopic command displays information of all active ROSTopics. It lists all subscribers/publishers, messages and more. With second command we subscribe to turtle1/pose topic as shown below.

1. rostopic list
2. rostopic echo turtle1/pose

Launch teleoperation and record moves

rosrun turtlesim turtle_teleop_key command allows us to navigate the turtle with keyboard arrows. In a separate terminal, we record and save a rosbag file which contains the navigation/path.



1. rosbag record -O subset /turtle1/cmd_vel /turtle1/pose
2. rosbag play subset.bag

Create catkin package

In order to create a package in ROS environment we install the build system *catkin,* which allows us to generate targets(executables, scripts, libraries) from source code. We navigate to desired folder and run *catkin_create_pkg.* This will generate *package.xml* and *CMakeLists.txt* files which contain information about the package. Then we build the package with *catkin_make* and add the work-space to ROS environment, . *~/desktop/ros-assignment/devel/setup.bash*

1. cd ~/desktop/ros-assignment/src
2. catkin_create_pkg ros_assignment std_msgs rospy roscpp
3. cd ~/desktop/ros-assignment
4. catkin_make
5. . ~/desktop/ros-assignment/devel/setup.bash
6. rospack depends1 ros_assignment

The last command lists the dependencies the *ros-assignment* package relies on.

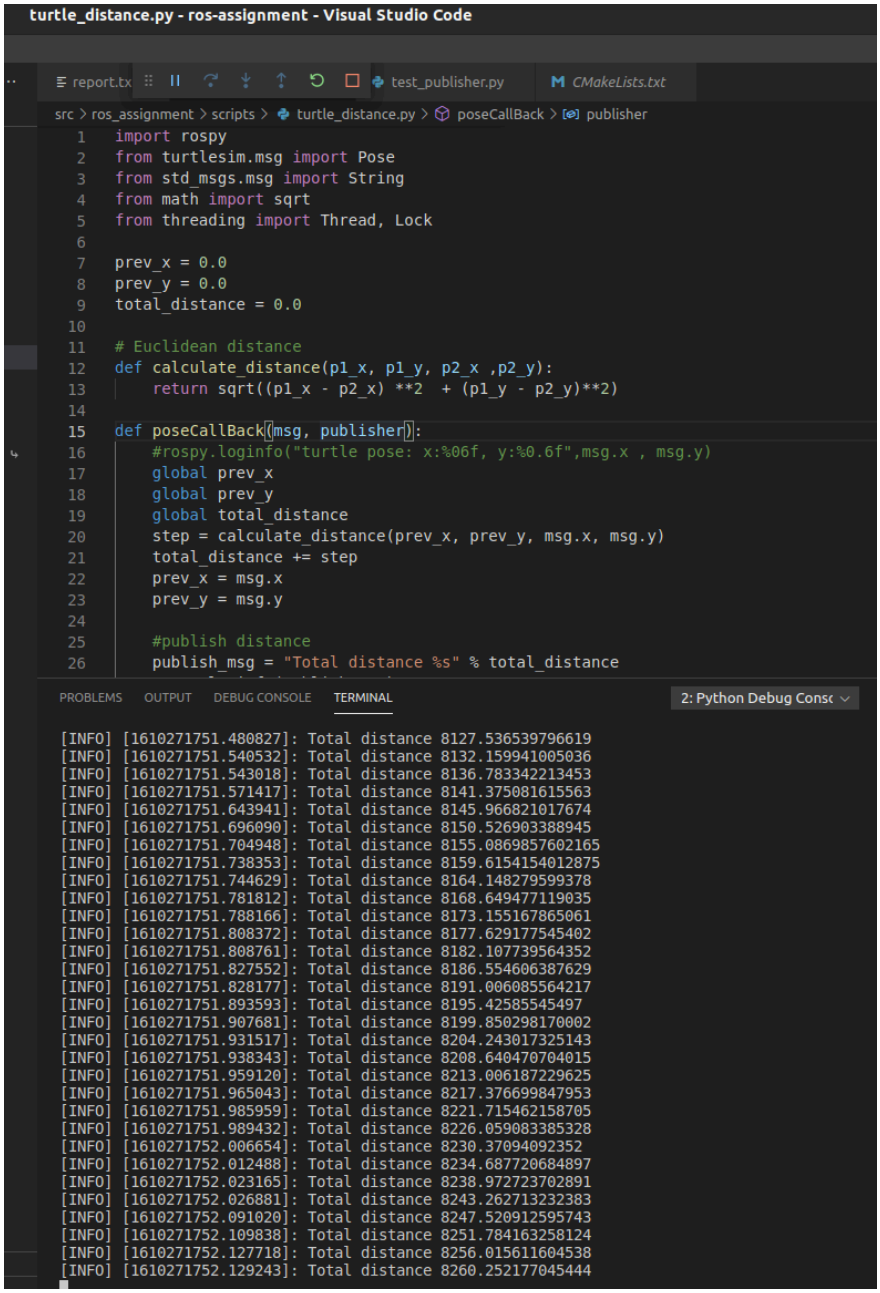
Calculate Distance(subscribe/publish)

Below is a simple python script that plays the role of both subscriber and publisher. Having the *ros-bag play subset.bag* command running in a separate terminal we launch the script. We give the *rospy* node the name “turtle_subscriber” and another *rospy.Publisher* instance is created with name “turtle_publisher”. *rospy.Subscriber* subscribes to */turtle1/pose* topic and a callback is set to be triggered for every new incoming message. From within the poseCallBack function we emit/publish the current travelled distance which is calculated as the Euclidean distance within *calculate_distance* method. poseCallBack method has the previously created *rospy.Publisher* instance as an argument. Finally, current travelled distance is published and logged.

cat ~/desktop/ros-assignment/src/ros-assignment/scripts/turtle_distance.py

```
import rospy
from turtlesim.msg import Pose
from std_msgs.msg import String
from math import sqrt
from threading import Thread, Lock

prev_x = 0.0
prev_y = 0.0
total_distance = 0.0
```

```
# Euclidean distance
def calculate_distance(p1_x, p1_y, p2_x ,p2_y):
    return sqrt((p1_x - p2_x) **2  + (p1_y - p2_y)**2)

def poseCallBack(msg, publisher):
    #rospy.loginfo("turtle pose: x:%06f, y:%0.6f",msg.x , msg.y)
    global prev_x
    global prev_y
    global total_distance
    step = calculate_distance(prev_x, prev_y, msg.x, msg.y)
    total_distance += step
    prev_x = msg.x
    prev_y = msg.y

    #publish distance
    publish_msg = "Total distance %s" % total_distance
    rospy.loginfo(publish_msg)
    publisher.publish(publish_msg)

def subscriber():
    rospy.init_node('turtle_subscriber', anonymous=True)

    publisher = rospy.Publisher('turtle_publisher', String, queue_size=10)

    rospy.Subscriber('/turtle1/pose', Pose, poseCallBack, publisher)

    rospy.spin()

    # print
    print('Total travelled distance', total_distance)

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



Test publisher

cat ~/desktop/ros-assignment/src/ros-assignment/scripts/test_published.py

```
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + 'received %s', data.data)

def listener():
    rospy.init_node('listener', anonymous=True)
```

```
    rospy.Subscriber('turtle_publisher', String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

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