Introduction to Robot Operating System (ROS 1)

Dr. Essa Alghannam

1- **Provide the commands to initialize a ROS workspace, create a package named "turtle_pkg" within it, and build the workspace.**
2- **Provide the command to start the turtlesim node.**
3- **Give the command to start the teleop_key node.**
4- **Write the commands to print the type of /turtle1/cmd_vel and /turtle1/pose and also print their message fields.**
5- **Write the command to create another turtle on the same window at (7.0, 7.5, 0.7)**

1	mkdir -p ~/workspace_test/src cd ~/workspace_test/src catkin_make catkin_create_pkg turtle_pkg rospy std_msgs geometry_msgs cd .. catkin_make roscore
2	rosrun turtlesim turtlesim_node
3	rosrun turtlesim turtle_teleop_key
4	rostopic type /turtle1/cmd_vel rosmsg show geometry_msgs/Twist rostopic type /turtle1/pose rosmsg show turtlesim/Pose
5	rosservice call /spawn 7.0 7.5 0.7 "new_turtle"

6- **Python code to make the first turtle move from its default initial position (5.5, 5.5, 0) to the position of the second turtle.**

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

2   import rospy
3   from geometry_msgs.msg import Twist, Pose
4   from turtlesim.msg import Pose
5   import math

6   def move_turtle ():
7     rospy.init_node('turtle _mover', anonymous=True)
8     pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
9     rate = rospy.Rate(10) # 10Hz

10    # Start and target poses
11    start_pose = Pose() # start_pose = Pose(5.5 , 5.5 , 0.0 , 0.0 , 0.0)
12    start_pose.x = 5.5
13    start_pose.y = 5.5
14    start_pose.theta = 0.0 # Start orientation (radians)

15    target_pose = Pose()
16    target_pose.x = 7.0
17    target_pose.y = 7.5
18    target_pose.theta = 0.7   # Doesn't really matter for line following

19    current_pose = None

20    def pose_callback(pose_msg):
21      nonlocal current_pose
22      current_pose = pose_msg

23    rospy.Subscriber('/turtle1/pose', Pose, pose_callback)


24    # Calculate distance and angle to target
25    distance = math.sqrt((target_pose.x - start_pose.x)**2 + (target_pose.y - start_pose.y)**2)
26    angle = math.atan2(target_pose.y - start_pose.y, target_pose.x - start_pose.x)

27    # Create and publish Twist messages
28    twist = Twist()
```

```
29    while not rospy.is_shutdown():
30        if current_pose is not None:
31            # Calculate remaining distance
32            remaining_distance = math.sqrt((target_pose.x - current_pose.x)**2 + (target_pose.y - current_pose.y)**2
33
34            if remaining_distance > 0.1: # Tolerance for reaching the target
35                # Linear velocity towards the target (proportional to remaining distance)
                  twist.linear.x = 0.5 * remaining_distance
36
37                # Angular velocity to correct orientation (proportional to angle error)
38                angle_error = (angle - current_pose.theta + math.pi) % (2 * math.pi) - math.pi
                  twist.angular.z = 1.0 * angle_error
39
                  pub.publish(twist)
40
41            else:
42                twist.linear.x = 0.0
43                twist.angular.z = 0.0
44                pub.publish(twist)
45                rospy.loginfo("Reached the target")
                  break # Exit loop once the target is reached
46
              rate.sleep()
47
48    if __name__ == '__main__':
49        try:
50            move_turtle ()
51        except rospy.ROSInterruptException:
              pass
```

the `nonlocal` keyword is used to declare that a variable within a nested function (a function defined inside another function) should refer to a variable in the *enclosing* function's scope, rather than creating a new local variable. It's essential for modifying variables in the outer function's scope from within an inner function without using global variables.

Roslaunch

- `roslaunch` is a powerful command-line tool in the Robot Operating System (ROS) that allows you to easily start and manage multiple ROS nodes simultaneously. Instead of starting each node individually using commands like `rosrun`.
- `roslaunch` lets you specify all the nodes you need in a single XML file (a launch file).

A simple launch file (e.g., `my_launch.launch`) might look like this:

```
<launch>
  <node pkg="my_package" type="node1" name="node1"
output="screen"/>
  <node pkg="my_package" type="node2" name="node2"
output="screen"/>
  <param name="my_parameter" value="10"/>
</launch>
```

[Start a node named "node1" from the executable `node1` in the `my_package` package, sending the output to the screen.]

[Set a parameter named "my_parameter" to the value "10," which will be available to any nodes that subscribe to it.]

Example:

- Inside your package create a folder named “launch”.
- Inside the folder Create a file named `turtlesim.launch` (you can choose a different name, but the `.launch` extension is important). The suitable location within your ROS workspace should be for example: `mycatkin_ws/src/myturtlepackage/launch/turtlesim.launch`.
- Add the following content:

```
<launch>
 <!-- Start the turtlesim node -->
 <node pkg="turtlesim" type="turtlesim_node" name="turtlesim"/>
 <!-- Optionally, start the keyboard teleop node -->
 <node pkg="turtlesim" type="turtle_teleop_key" name="teleop_turtle">
  <param name="scale_linear" value="2.0"/> <!-- Adjust speed -->
  <param name="scale_angular" value="2.0"/> <!-- Adjust turning speed -->
 </node>
</launch>
```

Explanation:

1- `<launch>` and `</launch>`: These tags enclose the entire launch file.

2- `<node>`: This tag defines a ROS node to be launched.

a) `pkg`: Specifies the ROS package containing the node (e.g., `"turtlesim"`).

b) `type`: Specifies the executable file within the package (e.g., `"turtlesim_node"`).

c) `name`: Assigns a unique name to the node. This is crucial for identification and avoiding name conflicts.

3- `<param>` (optional): These tags are used to set parameters for the launched nodes. In this case, they adjust the speed of the turtle controlled by the keyboard.

Finally run it:

```
roslaunch myturtlepackage turtlesim.launch
```

In case of error, try:

```
chmod +x ~/mycatkin_ws/src/myturtlepackage/launch/turtlesim.launch
```

Another example:

Create a file named `my_publisher.py` in your ROS package's `src` directory. ~/mycatkin_ws/src/myturtlepackage/src/my_publisher.py
#!/usr/bin/env python3

```
import rospy
from std_msgs.msg import String
def talker():
  pub = rospy.Publisher('my_topic', String, queue_size=10) # Create a publisher
  rospy.init_node('my_publisher', anonymous=True) # Initialize the node
  rate = rospy.Rate(1) # 1hz
  while not rospy.is_shutdown():
    hello_str = "hello world %s" % rospy.get_time()
# rospy.get_time()returns the current time as a floating-point number #representing seconds since the ROS master started.
    rospy.loginfo(hello_str)
    pub.publish(hello_str) # Publish the message
    rate.sleep()

if __name__ == '__main__':
  try:
    talker()
  except rospy.ROSInterruptException:
    pass
```

~/mycatkin_ws/src/myturtlepackage/src/my_publisher.py

Edit CMAKELISTS.txt file

catkin_install_python(PROGRAMS src/ my_publisher.py

DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})

Then

cd ~/mycatkin_ws/src/myturtlepackage /src

chmod +x my_publisher.py

Modify the launch file:

`mycatkin_ws/src/myturtlepackage/launch/turtlesim.launch`.

```
<launch>
  <!-- Launch turtlesim for visualization -->
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
  <!-- Launch the Python publisher node -->
  <node pkg="myturtlepackage" type="my_publisher.py"
name="my_publisher" output="screen"/>
</launch>
```

* `output="screen"`: This option controls where the output of the node (standard output and standard error streams) is directed. `screen` means the

output (e.g., `rospy.loginfo` messages) will be printed to the terminal where you run `roslaunch`. Other options include `log` (which sends the output to ROS log files) or `none` (which suppresses output entirely).

Finally:

cd ~/mycatkin_ws

catkin_make

roslaunch myturtlepackage turtlesim.launch