Sadaan Tahir - 25100224

EE 3002 L1 (Junior Design Studio - Robotics)

Spring 2025 - LAB 2

SOLUTION

Task 1: Creating a Launch File for Complex Number Publisher and Subscriber [15 MARKS]

The launch file **complex_number.launch**:

```
<launch>
    <!-- These are the arguments for the real and the imaginary parts of
the complex number, default complex number is 1+i -->
    <arg name="real" default="1.0" />
    <arg name="imaginary" default="1.0" />

    <!-- Now, we are going to launch the publisher node, or in other words
launching publisher.py -->
    <node pkg="lab1" type="publisher.py" name="complex_publisher"
output="screen">
        <param name="real" value="$(arg real)" />
        <param name="imaginary" value="$(arg imaginary)" />
    </node>

    <!-- Lastly, we launch the subscriber node to essentially listen to the
publisher -->
    <node pkg="lab1" type="subscriber.py" name="complex_subscriber"
output="screen" />
</launch>
```

The updated publisher node (**publisher.py**):

```
#!/usr/bin/env python3
import rospy
from lab1.msg import ComplexNumber

def publish_complex_number():
    pub = rospy.Publisher('/complex_numbers', ComplexNumber, queue_size=10)
    rospy.init_node('complex_number_publisher', anonymous=True)
```

```
    # The only thing that changes in this new updated publisher.py file is
that now we take user input for the real and imaginary parts of the cn
    real = rospy.get_param('~real', 1.0)
    imaginary = rospy.get_param('~imaginary', 1.0)

    rate = rospy.Rate(1)  # 1 Hz

    while not rospy.is_shutdown():
        msg = ComplexNumber()
        msg.real = real
        msg.imaginary = imaginary
        rospy.loginfo(f"Publishing: real={msg.real},
imaginary={msg.imaginary}")
        pub.publish(msg)
        rate.sleep()

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

Task 2: Working with the Turtlesim package and Rosbags [20 MARKS]

2.1 Go to Goal and Straight Line Tutorial using Turtlesim:

Just follow the steps in the tutorials.

2.2 Square and Circle Movement of Turtles:

The python script **move_circle.py**:

```
#!/usr/bin/env python3
import rospy
from geometry_msgs.msg import Twist

def move_circle():
    rospy.init_node('move_circle', anonymous=True)
    velocity_publisher = rospy.Publisher('/turtle2/cmd_vel', Twist,
queue_size=10)
```

```
    vel_msg = Twist()

    # Setting the circular movement speed and turn rate
    speed = 2
    turn_rate = 1.0  # Constant angular velocity for circular motion

    # Moving in a circle
    vel_msg.linear.x = speed
    vel_msg.angular.z = turn_rate
    while not rospy.is_shutdown():
        velocity_publisher.publish(vel_msg)

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

The python script **move_square.py**:

```
#!/usr/bin/env python3
import rospy
from geometry_msgs.msg import Twist

def move_square():
    rospy.init_node('move_square', anonymous=True)
    velocity_publisher = rospy.Publisher('/turtle1/cmd_vel', Twist,
queue_size=10)
    vel_msg = Twist()

    # Setting the square movement speed and the turn rate
    speed = 2
    turn_rate = 1.57  # 90 degrees in radians

    # Looping to move the turtle in a square
    for _ in range(10):
        # Moving forward
        vel_msg.linear.x = speed
        vel_msg.angular.z = 0
        velocity_publisher.publish(vel_msg)
        rospy.sleep(2)

        # Turning 90 degrees
```

```
        vel_msg.linear.x = 0
        vel_msg.angular.z = turn_rate
        velocity_publisher.publish(vel_msg)
        rospy.sleep(1)

    # Stopping after square movement completes
    vel_msg.linear.x = 0
    vel_msg.angular.z = 0
    velocity_publisher.publish(vel_msg)

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

The launch file **circle_plus_square.launch**:

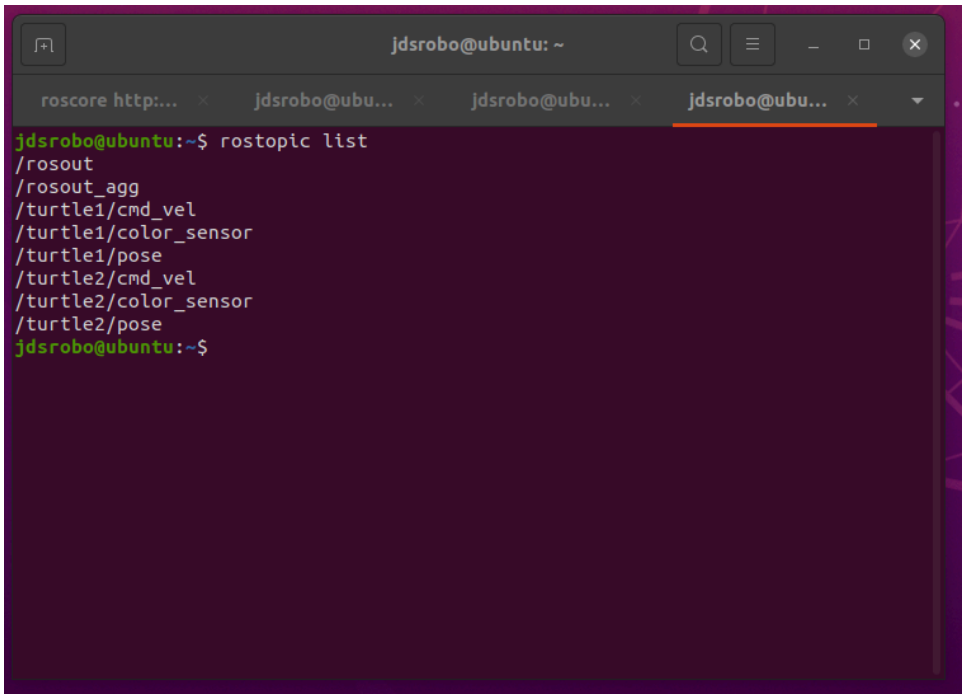
```
<launch>
    <!-- Launching the turtlesim environment -->
    <node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node"
output="screen"/>

    <!-- Executing the move_square script for Turtle 1 -->
    <node name="move_square" pkg="turtlesim_cleaner" type="move_square.py"
output="screen" />

    <!-- Executing the move_circle script for Turtle 2 -->
    <node name="move_circle" pkg="turtlesim_cleaner" type="move_circle.py"
output="screen" />
</launch>
```

2.3 Recording a Rosbag:

Just follow the steps in the tutorials.

Task 3: Turtle Chasing Behavior (baraf paani) [15 MARKS]

Once you have launched everything, if you do rostopic list in a new terminal, you should see these ROS topics:



The python script, **chase.py**:

```
#!/usr/bin/env python3

import rospy
from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
from math import atan2, sqrt, pi

pose1 = Pose()
pose2 = Pose()

def pose_callback1(msg):
    global pose1
    pose1 = msg

def pose_callback2(msg):
    global pose2
    pose2 = msg
```

```
def chase_turtle1():
    rospy.init_node('chase_turtle')
    rospy.Subscriber('/turtle1/pose', Pose, pose_callback1)
    rospy.Subscriber('/turtle2/pose', Pose, pose_callback2)

    pub = rospy.Publisher('/turtle2/cmd_vel', Twist, queue_size=1)
    twist = Twist()

    rate = rospy.Rate(10)

    while not rospy.is_shutdown():
        if pose1.x == 0 and pose1.y == 0:
            continue

        dx = pose1.x - pose2.x
        dy = pose1.y - pose2.y
        distance = sqrt(dx**2 + dy**2)

        # If distance is too small, stop Turtle 2, the extra marks part
        if distance < 0.1:
            twist.linear.x = 0
            twist.angular.z = 0
        else:
            angle_to_target = atan2(dy, dx)
            angle_diff = angle_to_target - pose2.theta

            if angle_diff > pi:
                angle_diff -= 2 * pi
            elif angle_diff < -pi:
                angle_diff += 2 * pi

            twist.linear.x = 1 * distance  # Proportional speed based on
distance
            twist.angular.z = 4 * angle_diff  # Proportional turning based
on angle

        pub.publish(twist)
        rate.sleep()

if __name__ == '__main__':
    try:
        chase_turtle1()
    except rospy.ROSInterruptException:
```

```
        pass
```

The launch file **chase_turtle.launch**:

```
<?xml version="1.0"?>
<launch>
    <!-- Launching the turtlesim environment -->
    <node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node"
output="screen"/>

    <!-- Launching the turtle teleop keyboard controller for Turtle 1 -->
    <node name="turtle_teleop_key" pkg="turtlesim" type="turtle_teleop_key"
output="screen"/>

    <!-- Launching the chase.py script to control Turtle 2 -->
    <node name="turtle_chase" pkg="turtlesim_cleaner" type="chase.py"
output="screen"/>
</launch>
```