Section 1 Individual section

Personal Assignment 1

Question 1:

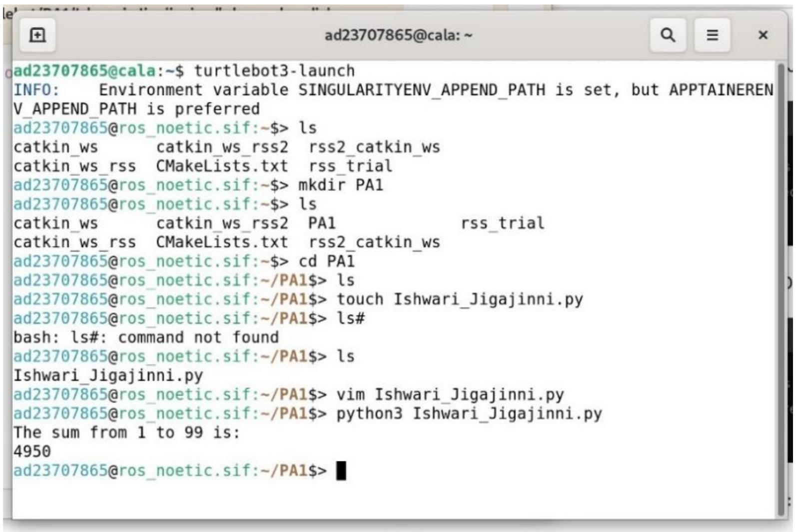
```
Could you use those (some of those) commands, to write and execute a '
    yourfirstname_lastname.py' file, to calculate the sum of 0 to 99?
```

Answer:

The above python with the name 'Ishwari_Jigajinni.py' will calculate the sum of 0 to 99 and print it to the terminal. 'Python3 Ishwari_Jigajinni.py' is used to run the program. The result 4950 was printed in the terminal.

Question 2.

```
Detail what commands you used and for what purpose?
```



Answer:

The commands used were.

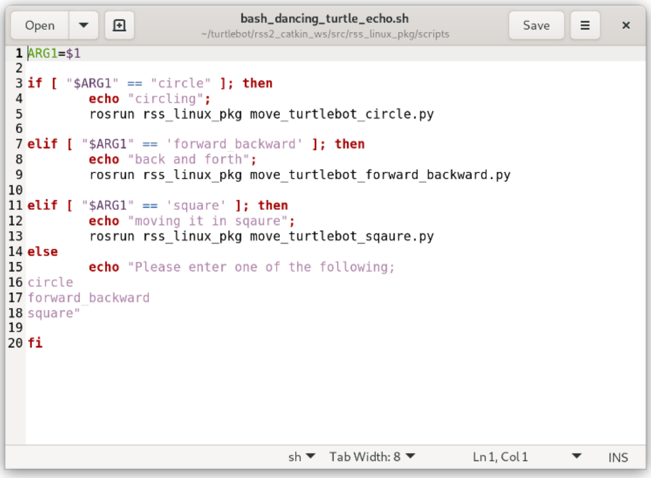
1. 'turtlebot3-launch': is used to enter the ROS.
2. 'ls': is used to list the number of files in the current directory.
3. 'mkdir PA1': is used to create a directory (PA1).
4. 'cd PA1: is used to enter the directory PA1.
5. 'cd ..': is used to go back.
6. 'touch Ishwari_Jigajinni: is used to create the file.
7. 'vim Ishwari_Jigajinni: is used to edit the file.

8. ‘chmod +x Ishwari_Jigajinni: is used to assign the permissions.
9. ‘python3 Ishwari_Jigajinni.py’: is used to run the python file (Ishwari_Jigajinni).

Personal Assignment 2

```
After fully understanding files in
1. '~/catkin_ws_rss/src/rss_linux_pkg/scripts'
2. '~/catkin_ws_rss/src/rss_linux_pkg/src'

Could you please write a '.py' file and then modify

3. bash bash_dancing_turtle_echo.sh circle or forward_backward

to make the robot move in a square?
```

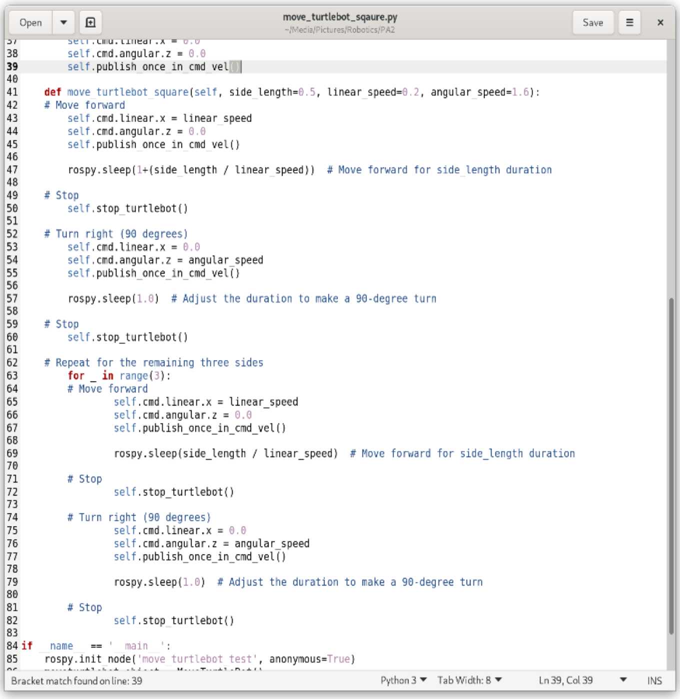
Answer:



```
bash_dancing_turtle_echo.sh
~/turtlebot/rss2_catkin_ws/src/rss_linux_pkg/scripts

ARG1=$1

if [ "$ARG1" == "circle" ]; then
        echo "circling";
        rosrun rss_linux_pkg move_turtlebot_circle.py

elif [ "$ARG1" == 'forward backward' ]; then
        echo "back and forth";
        rosrun rss_linux_pkg move_turtlebot_forward_backward.py

elif [ "$ARG1" == 'square' ]; then
        echo "moving it in sqaure";
        rosrun rss_linux_pkg move_turtlebot_sqaure.py
else
        echo "Please enter one of the following;
circle
forward backward
square"

fi
```

The script‘bash_dancing_turtle_echo.sh’ provided in week 2 is edited as shown on the left. If the input given is square it calls ‘move_turtlebot_square.py’ from rss_linux_pkg.

The small snippet of the ‘move_turtlebot_square.py’



```
move_turtlebot_sqaure.py
~/Media/Pictures/Robotics/PA2

        self.cmd.angular.z = 0.0
        self.publish_once_in_cmd_vel()

    def move_turtlebot_square(self, side_length=0.5, linear_speed=0.2, angular_speed=1.6):
    # Move forward
        self.cmd.linear.x = linear_speed
        self.cmd.angular.z = 0.0
        self.publish_once_in_cmd_vel()

        rospy.sleep(1+(side_length / linear_speed))  # Move forward for side_length duration

    # Stop
        self.stop_turtlebot()

    # Turn right (90 degrees)
        self.cmd.linear.x = 0.0
        self.cmd.angular.z = angular_speed
        self.publish_once_in_cmd_vel()

        rospy.sleep(1.0)  # Adjust the duration to make a 90-degree turn

    # Stop
        self.stop_turtlebot()

    # Repeat for the remaining three sides
        for _ in range(3):
        # Move forward
                self.cmd.linear.x = linear_speed
                self.cmd.angular.z = 0.0
                self.publish_once_in_cmd_vel()

                rospy.sleep(side_length / linear_speed)  # Move forward for side_length duration

        # Stop
                self.stop_turtlebot()

        # Turn right (90 degrees)
                self.cmd.linear.x = 0.0
                self.cmd.angular.z = angular_speed
                self.publish_once_in_cmd_vel()

                rospy.sleep(1.0)  # Adjust the duration to make a 90-degree turn

        # Stop
                self.stop_turtlebot()

if __name__ == '__main__':
    rospy.init_node('move_turtlebot_test', anonymous=True)
```

The robot is designed to travel in a square with 0.5 units for each side. It moves forward at a speed of 0.2 units per second and turns a

The workflow is:

1. The robot begins by traveling 0.2 units per second forward in the x direction.
Using the formula,
time = side_length / linear_velocity.
The robot moves in linear direction for 2.5 seconds
2. After moving forward, the robot stops and then rotates 90 degrees. Since the angular velocity is 1.6 radians per second, it takes 1 second to make the turn.
3. The robot repeats this process for the remaining three sides of the square.

When observed in the Gazebo simulation environment, the robot moves in a square path. However, the shape isn't a perfect square due to inertia or environmental factors.

Personal Assignment 3

Question 1.

```
1. Rewrite the 'pengwang_publisher.py' such that the robot goes along a straight
   line (e.g. x direction or y direction, etc.), and name it as '
   pengwang_publisher_line.py'.
```

Answer:

Publisher: A publisher in ROS is a node that sends out data or messages to a specific channel (topic). It shares information like sensor readings or commands with other nodes.

Subscriber: A subscriber in ROS is a node that receives data or messages from a specific channel (topic). It listens for updates and processes the received information.

Topic: A topic in ROS is a named bus over which nodes exchange messages. It acts as a communication channel connecting publishers and subscribers.

The ish_publisher.py is edited to ish_publisher_line.py, so that it goes along a straight line. The program above creates a publisher that publishes messages of type 'Twist' to cmd_vel topic.

The Ish_publisher_line sets the linear velocity to 0.5 and angular velocity to 0.

So, the robot only has linear motion and no angular motion.

This node continuously publishes the velocity commands indicating robot is moving in a straight line.

ish_publisher_line.py
~/turtlebot/rss2_catkin_ws/src/rss_pu...

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

import rospy
from geometry_msgs.msg import Twist

rospy.init_node('ish_publisher_line')
pub=rospy.Publisher('/cmd_vel',Twist,queue_size=1)
rate=rospy.Rate(1)
move=Twist()
move.linear.x=0.5
move.angular.z=0.0


stop=Twist()
stop.linear.x=0.0

while not rospy.is_shutdown():
    pub.publish(move)
    rate.sleep()

pub.publish(stop)
```

Question 2

```
2. Write a lauch file for 'pengwang_publisher_line.py'.
```

Answer: Launch files in ROS are used to start multiple nodes and set parameters efficiently, simplifying the initialization of complex systems. Here only one node "ish_publisher_line" is initialised from rss_pubsub_pkg.

pub_line.launch
~/turtlebot/rss2_catkin_ws/src/rss_pubsub_pkg/launch

```xml
<launch>
<node name="ish_publisher_line" pkg="rss_pubsub_pkg" type= "ish_publisher_line.py"
output="screen"/>
</launch>
```

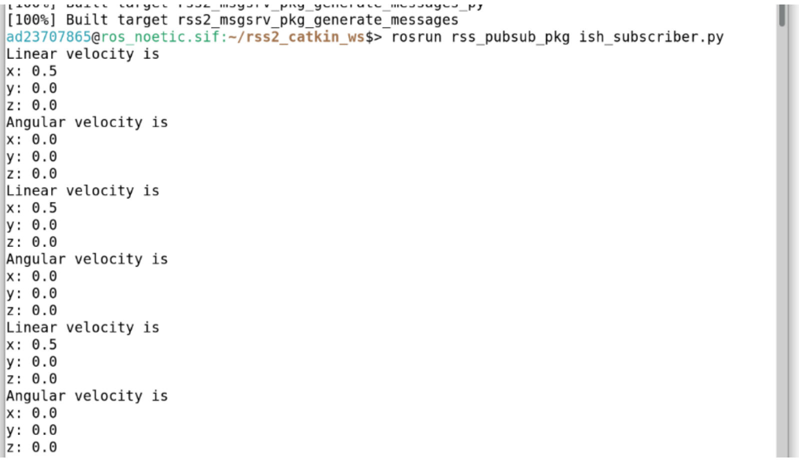
The launch file for Ish_publisher_line is written as above. We can launch it using the command
‘roslaunch rss_pubsub_pkg Ish_publisher_line.launch’

Question 3

```
3. Use the ‘pengwang_subscriber.py’ to listen to the topic published by ‘
   pengwang_publisher_line.py’, to show the robot is indeed going along a straight
    line.
```

Answer:

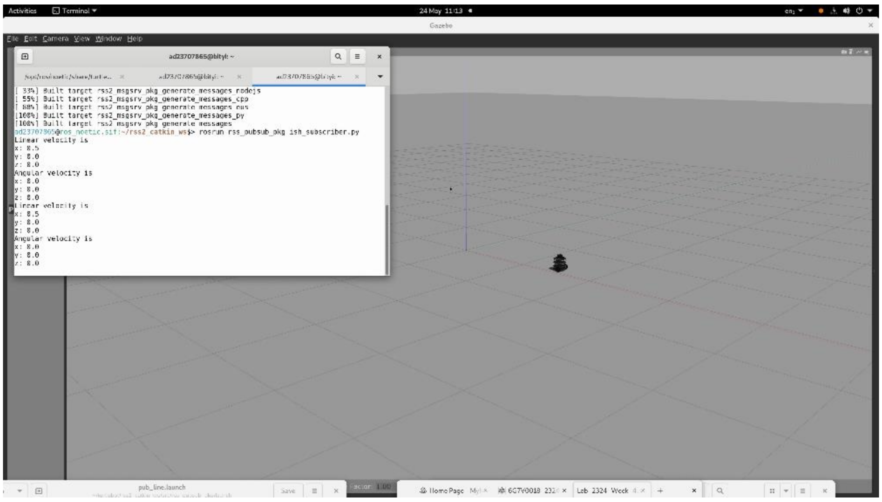
The same subscriber can be used for both Ish_publisher.py and Ish_publisher_line.py. The subscriber script ish_subscriber.py listens to the /cmd_vel topic and prints the received commands as seen below. The linear velocity is consistently 0.5 and angular velocity is 0.0. This indicates that there is a straight-line motion.



```
[100%] Built target rss2_msgsrv_pkg_generate_messages
ad23707865@ros_noetic.sif:~/rss2_catkin_ws$> rosrun rss_pubsub_pkg ish_subscriber.py
Linear velocity is
x: 0.5
y: 0.0
z: 0.0
Angular velocity is
x: 0.0
y: 0.0
z: 0.0
Linear velocity is
x: 0.5
y: 0.0
z: 0.0
Angular velocity is
x: 0.0
y: 0.0
z: 0.0
Linear velocity is
x: 0.5
y: 0.0
z: 0.0
Angular velocity is
x: 0.0
y: 0.0
z: 0.0
```

Question 4

```
4. Launch the turtelbot3 empty environment.
```



After launching Ish_publisher_line.py, Ish_subscriber.py, and turtlebot3_empty_world is three separate terminals. We can see that the robot correctly moves in a straight line. In the subscriber terminal, we see the messages indicating linear velocity of 0.5 and angular velocity 0.0 confirming straight line indication.

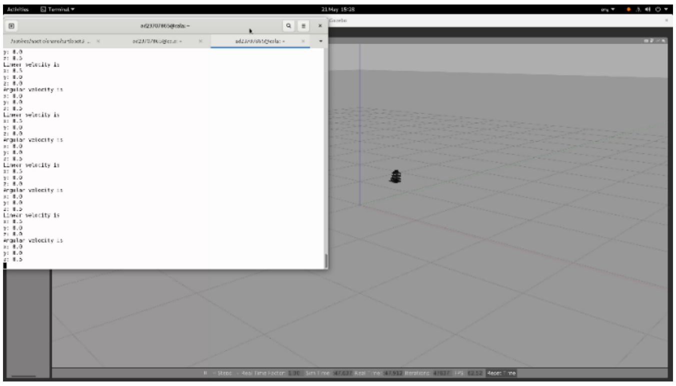
In the Gazebo simulation we can visually confirm that it indeed goes in a straight line

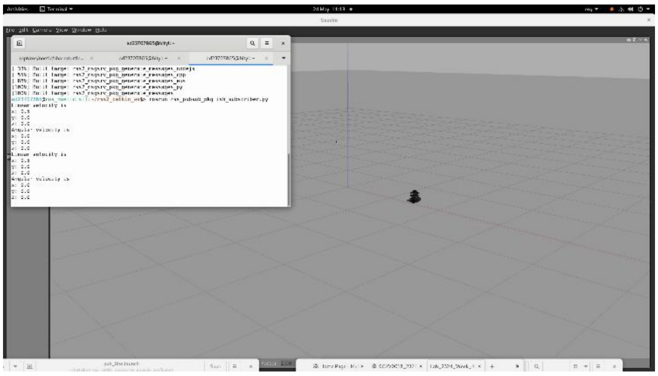
The turtlebot keeps moving in a straight line for infinite duration.

Question 5:

```
5. Observe and describe what happens when you launch 'pengwang_publisher.py' and '
   pengwang_publisher_line.py', respectively.
```

Answer:





In the Ish_publisher.py, the turtlebot3_model in the gazebo moves in a circular motion, because it has both angular velocity and linear velocity of 0.5.

Whereas in Ish_publisher_line.py, the turtlebot3_model moves in a straight line for infinte time duration.

Question 6

```
6. Can you stop the robot from running? (Hint: There are two ways!)
```

Answer:

The robot runs for an unlimited time, this is because we don't call the stop function till the rospy is shutdown. So to stop the robot from running we have to shut down publisher_line file by pressing Ctrl+C or

Personal Assignment 4

Questions 1 and 2

```
1. Write a 'pengwang_pubsub.py' file, where you subscribe to '/odom' and publish
   to '/cmd_vel'.
2. In the file, define a distance the robot needs to run. After the robot reach
   the distance, stop.
```

Answer

This Python script employs ROS (Robot Operating System) for robot motion control. It subscribes to odometry data to monitor distance traveled, issues velocity commands to `/cmd_vel`, and halts the robot once it travels a designated distance. The linear velocity is set to 0.5 in the x-direction, while velocities in other directions remain zero. The robot continues moving unstil it covers 5 units.

23707865

ish_pubsub.py

~/turtlebot/rss2_catkin_ws/src/rss_pubsub_...

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

import rospy
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
import math

class PengwangPubSub:
    def __init__(self):
        rospy.init_node('ish_pubsub')

        # Publisher for the /cmd_vel topic
        self.pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)

        # Subscriber for the /odom topic
        self.sub = rospy.Subscriber('/odom', Odometry, self.odom_callback)

        # Define the distance the robot needs to run
        self.distance_to_run = 5.0  # meters
        self.current_distance = 0.0

        # Variables to store previous position
        self.prev_x = None
        self.prev_y = None

    def odom_callback(self, msg):
        # Get the current position from the odometry message
        x = msg.pose.pose.position.x
        y = msg.pose.pose.position.y

        # Calculate the distance traveled from the previous position
```

Python 3 Tab Width: 8 Ln 59, Col 61 INS

ish_pubsub.py

~/turtlebot/rss2_catkin_ws/src/rss_pubsub_pkg/sc...

Calculate the distance traveled from the previous position

```python
        if self.prev_x is not None and self.prev_y is not None:
            distance_increment = math.sqrt((x - self.prev_x)**2 + (y - self.prev_y)**2)
            self.current_distance += distance_increment

        # Update previous position
        self.prev_x = x
        self.prev_y = y

        # Display the current distance traveled
        rospy.loginfo(f"Current Distance: {self.current_distance:.2f} meters")

    def run(self):
        rate = rospy.Rate(10)  # Set the rate to 10 Hz
        move = Twist()
        move.linear.x = 0.5
        move.linear.y = 0.0
        move.linear.z = 0.0
          # Set a constant linear velocity in x direction

        while not rospy.is_shutdown() and self.current_distance < self.distance_to_run:
            self.pub.publish(move)
            rate.sleep()

        # Stop the robot
        move.linear.x = 0.0
        self.pub.publish(move)

        # Display a message indicating the robot has stopped
        rospy.loginfo("Target distance reached. Stopping the robot.")

if __name__ == '__main__':
    try:
        pubsub = PengwangPubSub()
        pubsub.run()
    except rospy.ROSInterruptException:
        pass
```

Python 3 Tab Width: 8 Ln 59, Col 61 INS

Question 3

```
3. Write a launch file to start it.
```

Answer

ish_pubsub.launch
~/turtlebot/rss2_catkin_ws/src/rss_pubsub_pkg/launch

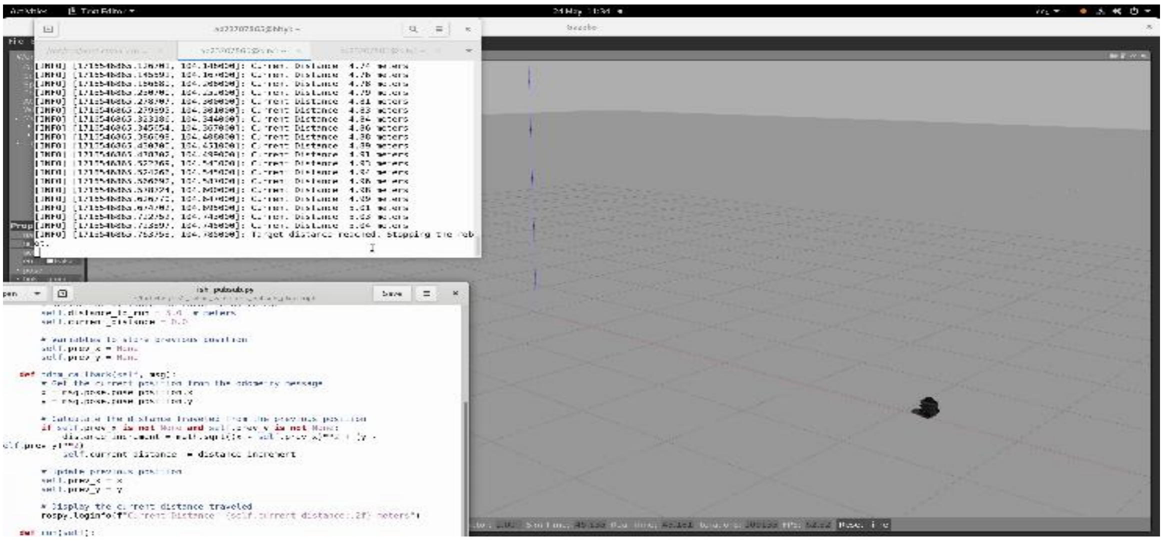
```
<launch>
<include file="$(find turtlebot3_gazebo)/launch/turtlebot3_empty_world.launch"/>
<node name="ish_pub" pkg="rss_pubsub_pkg" type= "ish_pubsub.py" output="screen"/>
</launch>
```

The launch file is written as above. When you run this the launch file launches gazebo and then creates a node called ish_pub which runs the ish_pubsub.py python script.

Question 4

```
4. Launch the turtelbot3 empty environment to observe if it works.
```

Answer



When turtlebot3_empty_world is launched in Gazebo, the robot is supposed to move in a straight line for 5 units with a linear velocity of 0.5. then it stops.
But in the simulation, we see that the robot moves more than 5 units and then stops. This is due to a robot scale error.

23707865

PERSONAL ASSIGNMENT 5

Question 1

- Create launch files to start the publisher and subscriber in section 5.2 in Part I.

Answer

msg_sub.launch
~/turtlebot/rss2_catkin_ws/src/rss2_msgsrv_pkg/launch

```
<launch>
<node name="ish_sub" pkg="rss2_msgsrv_pkg" type="msg_sub.py" output="screen"/>
</launch>
```

msg_pub.launch
~/turtlebot/rss2_catkin_ws/src/rss2_msgsrv_pkg/launch

```
<launch>
<node name="ish_pub" pkg="rss2_msgsrv_pkg" type="msg_pub.py" output="screen"/>
</launch>
```

The launch files for msg_pub.py and msg_sub.py are created as shown above.

Question 2

- Create launch files to start the server and client in sections 4.7 and 4.8 in Part II.

Answer

ish_srv_client.launch
~/turtlebot/rss2_catkin_ws/src/rss2_msgsrv_pkg/launch

```
<launch>
<node name="ish_client" pkg="rss2_msgsrv_pkg" type="ish_srv_client.py"
output="screen"/>
</launch>
```

ish_srv_server.launch
~/turtlebot/rss2_catkin_ws/src/rss2_msgsrv_pkg/launch

```
<launch>
<node name="ish_server" pkg="rss2_msgsrv_pkg" type="ish_srv_server.py"
output="screen"/>
</launch>
```

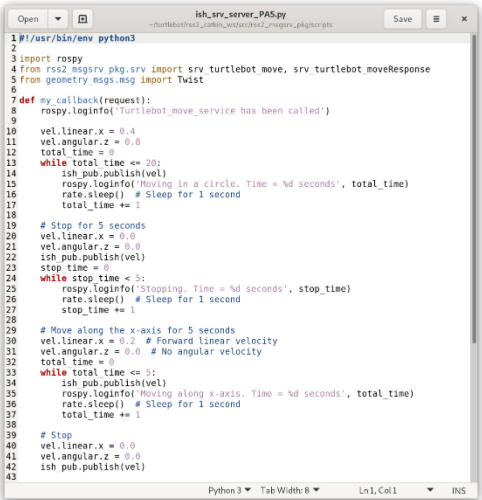
The launch files for server and client are created as above.

Question 3, 4, 5, 6

- Modify (or you can do it from scratch) the codes in sections 4.7 and 4.8 in Part II, such that the robot moves for 30 seconds.
  1. The first 20 seconds the robot moves in a circle
  2. Then stops for 5 seconds.
  3. Then moves along x-axis for 5 seconds
  4. Stop

Answer

The server file in 4.7 is changed as below.



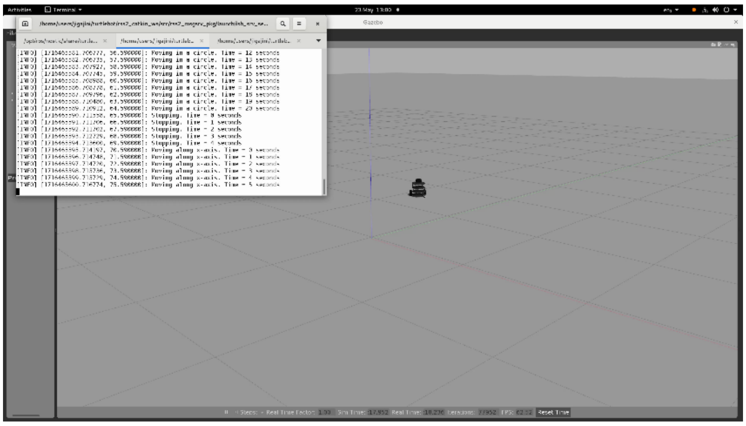
ish_srv_server_PA5.py
~/turtlebot/rss2_catkin_ws/src/rss2_msgsrv_pkg/scripts

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

import rospy
from rss2_msgsrv_pkg.srv import srv_turtlebot_move, srv_turtlebot_moveResponse
from geometry_msgs.msg import Twist

def my_callback(request):
    rospy.loginfo('Turtlebot_move_service has been called')

    vel.linear.x = 0.4
    vel.angular.z = 0.8
    total_time = 0
    while total_time <= 20:
        ish_pub.publish(vel)
        rospy.loginfo('Moving in a circle. Time = %d seconds', total_time)
        rate.sleep()  # Sleep for 1 second
        total_time += 1

    # Stop for 5 seconds
    vel.linear.x = 0.0
    vel.angular.z = 0.0
    ish_pub.publish(vel)
    stop_time = 0
    while stop_time < 5:
        rospy.loginfo('Stopping. Time = %d seconds', stop_time)
        rate.sleep()  # Sleep for 1 second
        stop_time += 1

    # Move along the x-axis for 5 seconds
    vel.linear.x = 0.2  # Forward linear velocity
    vel.angular.z = 0.0  # No angular velocity
    total_time = 0
    while total_time <= 5:
        ish_pub.publish(vel)
        rospy.loginfo('Moving along x-axis. Time = %d seconds', total_time)
        rate.sleep()  # Sleep for 1 second
        total_time += 1

    # Stop
    vel.linear.x = 0.0
    vel.angular.z = 0.0
    ish_pub.publish(vel)

```

Python 3 Tab Width: 8 Ln 1, Col 1 INS

```python
    # Stop
    vel.linear.x = 0.0
    vel.angular.z = 0.0
    ish_pub.publish(vel)

    return srv_turtlebot_moveResponse(True)

rospy.init_node('turtlebot_move_server')

# This is the service called '/turtlebot_move_service'
ish_service = rospy.Service('/turtlebot_move_service', srv_turtlebot_move,
  my_callback)

ish_pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
vel = Twist()

# Make sure counting second by second
rate = rospy.Rate(1)  # 1 Hz rate for 1 second intervals

rospy.loginfo('Service /turtlebot_move_service is ready!')

# Maintain the service open
rospy.spin()

```

To manage service requests, the script establishes a callback function called "my_callback" and initialises a ROS service called "turtlebot_move_service`. It adjusts the TurtleBot's velocity to move in a circle for 20 seconds after receiving a request. The robot is then stopped for five seconds and then moves for five seconds in the x-axis direction. Using a `Twist` message, velocity commands are published to `/cmd_vel}. A rate of one hertz (Hz) maintains time synchronization, so that operations take place at one-second intervals.



When the gazebo is launched we see the robot circles for time 20 seconds and then it stops for 5 seconds and then it starts moving in straight line for 5 seconds then stops completely.

Even if the time synchronisation is set at 1 Hz, the actual time, and the time according to the robots does not tally with each other. For robots, the time moves little faster.

Personal Assignment 6

Question 1

Given the following service message type

```
float64 sideLength
int32 repetitions
---
bool success
```

- Create a ‘turtlebot_move_square.srv’ message, and put it in the right place.

Answer

```
float64 sideLength
int32 repetitions
---
bool success
```

The service file turtle_move_square.srv is created as above and put in the srv folder of rss2_msgsrv_pkg folder.

Question 2

- Modify ‘CMakeLists.txt’ and ‘package.xml’.

Answer

```

## Generate messages in the 'msg' folder
add_message_files(
   FILES
   date_cmd_vel.msg
#   Message2.msg
)

## Generate services in the 'srv' folder
add_service_files(
   FILES
   srv_turtlebot_move.srv
   turtlebot_move_square.srv
#   Service2.srv
)

```

The CMakeLists.txt is modified as shown to the left. The service file turtlebot_move_square.srv is added to it.

```xml
  <!--   <doc_depend>doxygen</doc_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>geometry_msgs</build_depend>
  <build_depend>nav_msgs</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>message_generation</build_depend>

  <build_export_depend>geometry_msgs</build_export_depend>
  <build_export_depend>nav_msgs</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <build_export_depend>message_runtime</build_export_depend

  <exec_depend>geometry_msgs</exec_depend>
  <exec_depend>nav_msgs</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>message_runtime</exec_depend>
```

The Package.xml file is edited as shown on the right.

Questions 3,4, and 5

- Create a client and a server that use the 'turtlebot_move_square.srv' message to do
  1. When the server being called, it should move along a square with side length defined by 'sideLength'.
  2. The robot must repeat moving along the square defined by 'repetitions', e.g., if 'repetitions=4', then the robot must move along the square 4 times.
  3. When the robot finishes the movement, it should return 'True'. Otherwise, 'False'.

Answer:

SERVER :

ish_move_srv_square_PA6.py

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

import rospy
from rss2_msgsrv_pkg.srv import turtlebot_move_square, turtlebot_move_squareResponse
from geometry_msgs.msg import Twist

class MoveTurtleBot():
    def __init__(self):
        self.turtlebot_vel_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
        self.cmd = Twist()
        self.ctrl_c = False
        self.rate = rospy.Rate(1)
        rospy.on_shutdown(self.shutdownhook)

    def publish_once_in_cmd_vel(self):
        while not self.ctrl_c:
            connections = self.turtlebot_vel_publisher.get_num_connections()
            if connections > 0:
                self.turtlebot_vel_publisher.publish(self.cmd)
                break
            else:
                self.rate.sleep()

    def shutdownhook(self):
        self.stop_turtlebot()
        self.ctrl_c = True

    def stop_turtlebot(self):
        #rospy.loginfo("Shutdown time! Stopping the robot")
        self.cmd.linear.x = 0.0
        self.cmd.angular.z = 0.0
        self.publish_once_in_cmd_vel()

    def move_turtlebot_square(self, side_length=0.5, linear_speed=0.2, angular_speed=1.6, repetitions=1):
        for rep in range(repetitions):
            rospy.loginfo(f'Executing repetition {rep+1}/{repetitions}')
            for side in range(4):
                rospy.loginfo(f'Moving along side {side+1}/4 (length: {side_length})')
                # Move forward
                self.cmd.linear.x = linear_speed
```

Python 3 Tab Width: 8 Ln 1, Col 23 INS

ish_move_srv_square_PA6.py
~/turtlebot/rss2_catkin_ws/src/rss2_msgsrv_pkg/scripts

```python
            rospy.loginfo(f"Executing repetition {rep+1}/{repetitions}")
            for side in range(4):
                rospy.loginfo(f"Moving along side {side+1}/4 (length: {side_length})")
                # Move forward
                self.cmd.linear.x = linear_speed
                self.cmd.angular.z = 0.0
                self.publish_once_in_cmd_vel()
                rospy.sleep(1 + (side_length / linear_speed))  # Move forward for side_length duration

                # Stop
                self.stop_turtlebot()

                # Turn right (90 degrees)
                self.cmd.linear.x = 0.0
                self.cmd.angular.z = angular_speed
                self.publish_once_in_cmd_vel()
                rospy.sleep(1.0)  # Adjust the duration to make a 90-degree turn

                # Stop
                self.stop_turtlebot()


def handle_move_square(req):
    rospy.loginfo(f"Moving in a square with side length {req.sideLength} and {req.repetitions} repetitions")
    mover = MoveTurtleBot()
    try:
        mover.move_turtlebot_square(side_length=req.sideLength, repetitions=req.repetitions)
        return turtlebot_move_squareResponse(True)
    except Exception as e:
        rospy.logerr(f"Error while moving: {e}")
        return turtlebot_move_squareResponse(False)

def move_square_server():
    rospy.init_node('move_square_server')
    s = rospy.Service('turtlebot_move_square', turtlebot_move_square, handle_move_square)
    rospy.loginfo("Ready to move in a square.")
    rospy.spin()

if __name__ == "__main__":
    move_square_server()
```

Python 3 Tab Width: 8 Ln 1, Col 23 INS

This script moves turtleBot in a square shape. It uses ROS, a system for controlling robots. The TurtleBot moves forward for a certain distance, then turns right, and repeats this to form a square.

This Python script defines a ROS service `turtlebot_move_square` to make a TurtleBot move in a square pattern. It initializes a class `MoveTurtleBot` to control the robot's movement, with methods for publishing velocity commands, stopping the robot, and moving it in a square. The `move_square_server` function sets up the sROS node and service.

When the service is called, it moves the TurtleBot in a square path with configurable side length, linear and angular speeds, and repetitions. Time intervals ensure precise movement. Any errors during execution are logged.

CLIENT

ish_move_client_squarePA6.py
~/turtlebot/rss2_catkin_ws/src/rss2_msgsrv_pkg/scripts

ish_move_srv_square_PA6.py | ish_move_client_squarePA6.py

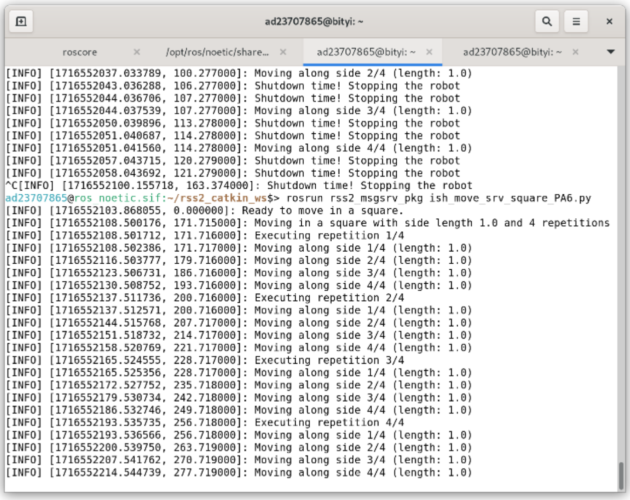
```python
#!/usr/bin/env python3

import rospy
from rss2_msgsrv_pkg.srv import turtlebot_move_square

def move_square_client(side_length, repetitions):
    rospy.wait_for_service('turtlebot_move_square')
    try:
        move_square = rospy.ServiceProxy('turtlebot_move_square', turtlebot_move_square)
        resp = move_square(side_length, repetitions)
        return resp.success
    except rospy.ServiceException as e:
        rospy.logerr(f"Service call failed: {e}")
        return False

if __name__ == "__main__":
    rospy.init_node('move_square_client')
    # Default values for side_length and repetitions
    side_length = rospy.get_param('~side_length', 1.0)
    repetitions = rospy.get_param('~repetitions', 4)
    success = move_square_client(side_length, repetitions)
    if success:
        rospy.loginfo("TurtleBot moved in square successfully")
    else:
        rospy.logerr("Failed to move TurtleBot in square")
```

Python 3 Tab Width: 8 Ln 19, Col 52 INS

This Python script creates a client to call the `turtlebot_move_square` service which we created in question 1of this assignment, allowing users to command a TurtleBot to move in a square. It initializes a ROS node and defines a function `move_square_client` to interact with the service. We specify the side length as 1 and the number of repeatitions as 4. The client logs success or failure messages based on the service response.

SERVER OUTPUT

The server file displays which side of the square robot is moving in, it also says which repetition is going on.



```
ad23707865@bityi: ~
roscore | /opt/ros/noetic/share... | ad23707865@bityi: ~ | ad23707865@bityi: ~
[INFO] [1716552037.033789, 100.277000]: Moving along side 2/4 (length: 1.0)
[INFO] [1716552043.036288, 106.277000]: Shutdown time! Stopping the robot
[INFO] [1716552044.036706, 107.277000]: Shutdown time! Stopping the robot
[INFO] [1716552044.037539, 107.277000]: Moving along side 3/4 (length: 1.0)
[INFO] [1716552050.039896, 113.278000]: Shutdown time! Stopping the robot
[INFO] [1716552051.040687, 114.278000]: Shutdown time! Stopping the robot
[INFO] [1716552051.041560, 114.278000]: Moving along side 4/4 (length: 1.0)
[INFO] [1716552057.043715, 120.279000]: Shutdown time! Stopping the robot
[INFO] [1716552058.043692, 121.279000]: Shutdown time! Stopping the robot
^C[INFO] [1716552100.155718, 163.374000]: Shutdown time! Stopping the robot
ad23707865@ros_noetic.sif:~/rss2_catkin_ws$> rosrun rss2_msgsrv_pkg ish_move_srv_square_PA6.py
[INFO] [1716552103.868055, 0.000000]: Ready to move in a square.
[INFO] [1716552108.500176, 171.715000]: Moving in a square with side length 1.0 and 4 repetitions
[INFO] [1716552108.501712, 171.716000]: Executing repetition 1/4
[INFO] [1716552108.502386, 171.717000]: Moving along side 1/4 (length: 1.0)
[INFO] [1716552116.503777, 179.716000]: Moving along side 2/4 (length: 1.0)
[INFO] [1716552123.506731, 186.716000]: Moving along side 3/4 (length: 1.0)
[INFO] [1716552130.508752, 193.716000]: Moving along side 4/4 (length: 1.0)
[INFO] [1716552137.511736, 200.716000]: Executing repetition 2/4
[INFO] [1716552137.512571, 200.716000]: Moving along side 1/4 (length: 1.0)
[INFO] [1716552144.515768, 207.717000]: Moving along side 2/4 (length: 1.0)
[INFO] [1716552151.518732, 214.717000]: Moving along side 3/4 (length: 1.0)
[INFO] [1716552158.520769, 221.717000]: Moving along side 4/4 (length: 1.0)
[INFO] [1716552165.524555, 228.717000]: Executing repetition 3/4
[INFO] [1716552165.525356, 228.717000]: Moving along side 1/4 (length: 1.0)
[INFO] [1716552172.527752, 235.718000]: Moving along side 2/4 (length: 1.0)
[INFO] [1716552179.530734, 242.718000]: Moving along side 3/4 (length: 1.0)
[INFO] [1716552186.532746, 249.718000]: Moving along side 4/4 (length: 1.0)
[INFO] [1716552193.535735, 256.718000]: Executing repetition 4/4
[INFO] [1716552193.536566, 256.718000]: Moving along side 1/4 (length: 1.0)
[INFO] [1716552200.539750, 263.719000]: Moving along side 2/4 (length: 1.0)
[INFO] [1716552207.541762, 270.719000]: Moving along side 3/4 (length: 1.0)
[INFO] [1716552214.544739, 277.719000]: Moving along side 4/4 (length: 1.0)
```

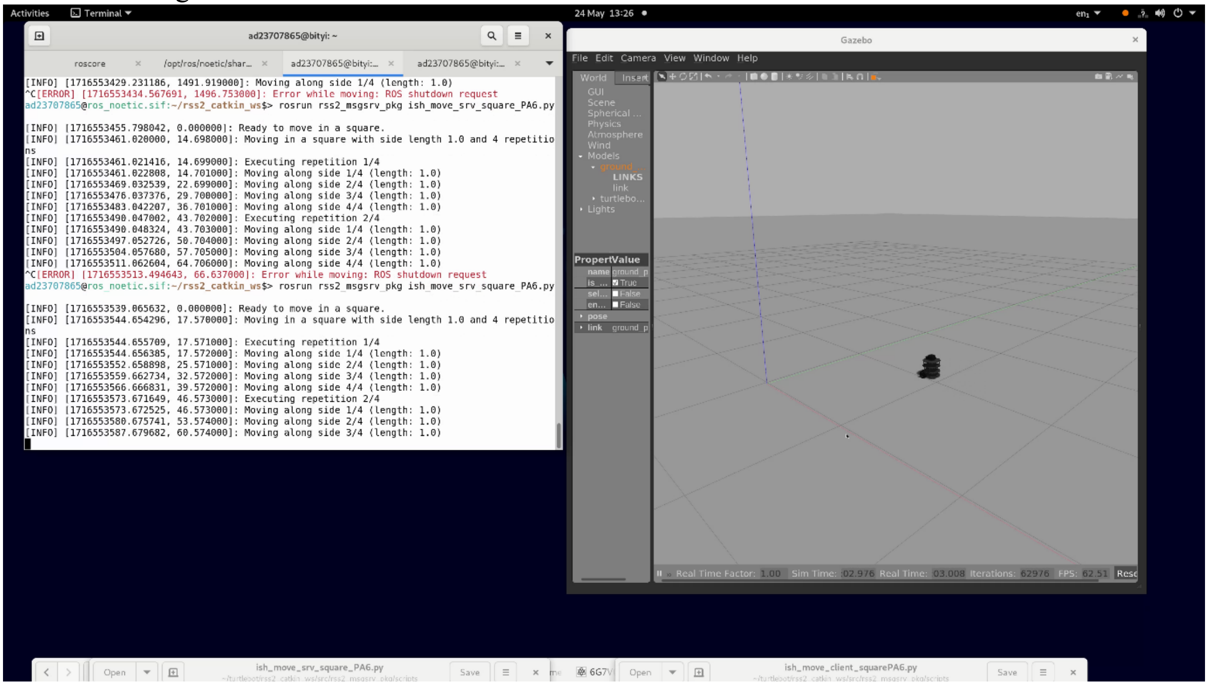
CLIENT OUTPUT

The client returns if the robot has successfully moved in the square or not. That will be printed on the terminal.

```
ad23707865@ros_noetic.sif:~/rss2_catkin_ws$> rosrun rss2_msgsrv_pkg ish_move_client_squarePA6.py
^C[ERROR] [1716551934.563981, 569.495000]: Service call failed: service [/turtlebot_move_square] r
eturned no response
[ERROR] [1716551934.564850, 569.495000]: Failed to move TurtleBot in square
ad23707865@ros_noetic.sif:~/rss2_catkin_ws$> rosrun rss2_msgsrv_pkg ish_move_client_squarePA6.py
[INFO] [1716552058.044669, 121.279000]: TurtleBot moved in square successfully
```

GAZEBO OUTPUT

In Gazebo, we can visualize that the robot is making 4 repetitions with side lengths a little higher than 1 unit. This might be due to some wheel calibration error or motor error.



Section 2. Group Assignment