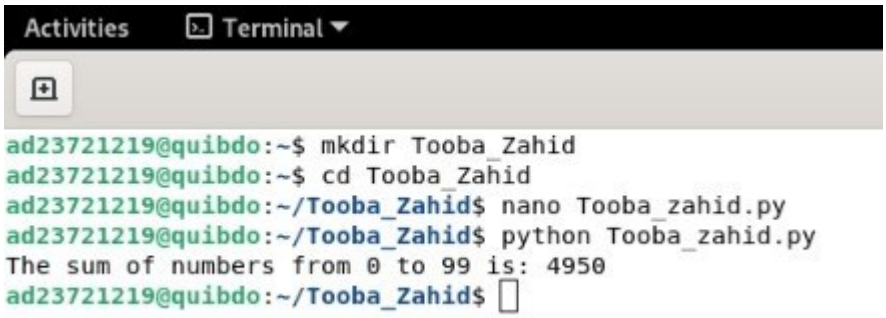
Personal Assignment 1:

Assignment 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?

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

Solution



mkdir makes a directory and cd navigates named Tooba_Zahid

Nano opens text editor to create edit file Tooba_Zahid.py.as,

calculate the sum from 0 to 99 using a for loop.

Used Ctrl+X to exit.

Python command executes the Python script Tooba_zahid.py using the Python interpreter.

****

Personal Assignment 2:

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?
```

1. **Scripts Directory**: **`~/catkin_ws_rss/src/rss_linux_pkg/scripts`**
    - **`bash_dancing_turtle_echo.sh`**: Executes Python scripts based on user input to move the TurtleBot. (Circle, forward_backward, square)
    - **`bash_dancing_turtle.sh`**: Moves the TurtleBot forward and backward or rotates it based on user input. (Forward, rotate)
    - **`bash_echo_command.sh`**: Prints a simple message to the terminal. (Hello there, Developers!)
    - **`bash_series_commands.sh`**: Shows the current path and lists directory contents.
- **Source Directory**: **`~/catkin_ws_rss/src/rss_linux_pkg/src`**
    - Contains Python scripts for specific TurtleBot movements. Like move_turtlebot_circle.py, move_turtlebot_square.py

➢ **Write a .py file to make a robot move in square.**

Modify file for square using **cp** as **(**cp move_turtlebot_circle.py move_turtlebot_square.py**)**

/home/users/zahidtoo/turtlebot/catkin_ws/src/turtlebot3_si... × ad23721219@itagui: ~

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

import rospy
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):
        """
        This is because publishing in topics sometimes fails the first time you publish.
        In continuous publishing systems there is no big deal but in systems that publish only
        once it IS very important.
        """
        while not self.ctrl_c:
            connections = self.turtlebot_vel_publisher.get_num_connections()
            if connections > 0:
                self.turtlebot_vel_publisher.publish(self.cmd)
                rospy.loginfo("Cmd Published")
                break
            else:
                self.rate.sleep()

    def shutdownhook(self):
        # works better than the rospy.is_shutdown()
        self.stop_turtlebot()
        self.ctrl_c = True

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

    # Method to move straight
    def move_straight(self, moving_time, speed):
        self.cmd.linear.x = speed
        self.cmd.angular.z = 0.0
        i = 0
        rospy.loginfo("Moving straight!")
        while not self.ctrl_c and i < moving_time:
            self.publish_once_in_cmd_vel()
            i += 1
            self.rate.sleep()


    # Method to turn
    def turn(self, turning_time, angular_speed):
        self.cmd.linear.x = 0.0
        self.cmd.angular.z = angular_speed
        i = 0
        rospy.loginfo("Turning!")
        while not self.ctrl_c and i < turning_time:
            self.publish_once_in_cmd_vel()
            i += 1
            self.rate.sleep()

    # Method to move in a square
    def move_in_square(self, side_length, speed, angular_speed):
        for _ in range(4):
            self.move_straight(side_length, speed)
            self.turn(3, angular_speed)  # Increased turning time to ensure a 90-degree turn
        self.stop_turtlebot()

if __name__ == '__main__':
    rospy.init_node('move_turtlebot_square', anonymous=True)
    moveturtlebot_object = MoveTurtleBot()
    try:
        moveturtlebot_object.move_in_square(5, 0.2, 0.5)  # Adjusted angular speed
    except rospy.ROSInterruptException:
        pass
```

• Move_in_square technique involves two movements: turning and straight.

• 1st robot moves straight for a predetermined time **(side_length)** at a predetermined speed

• robot turns at a defined angular speed (**angular_speed)** for a certain time **(turning_time).**

• 4 repetitions of straight ahead and pivoting are performed, ensure precise 90-degree turns.

• Robot stops after completing the square

➢ Modify bash_dancing_turtle_echo.sh.

/home/users/zahidtoo/turtlebot/catkin_ws/src/turtlebot3_simulations/turtlebot3_g...

```
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 in square"
   rosrun rss_linux_pkg move_turtlebot_square.py

else
echo "Please enter one of the following;
circle
forward_backward
square"
fi
~
```

Result:

Using

```
$> bash bash_dancing_turtle_echo.sh square
```

It triggers the bash script which then identifies the input parameter 'square', echoes 'moving in square', and launches the Python script.

****

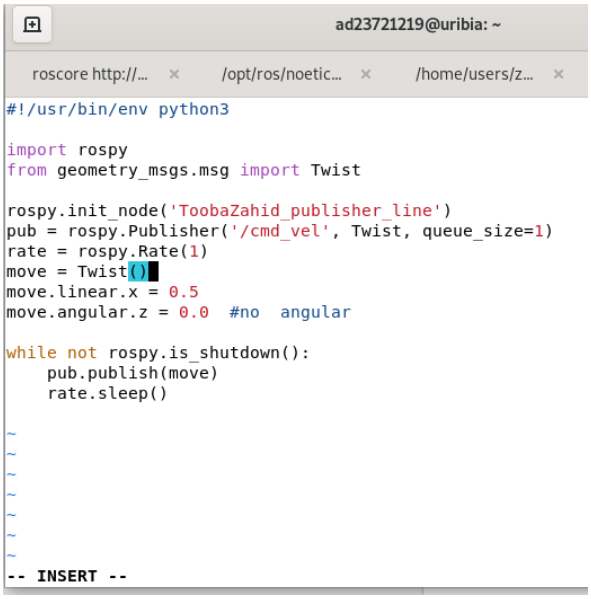
Personal Assignment 3:

Personal Assignment 3

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'.
2. Write a lauch file for 'pengwang_publisher_line.py'.
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.
4. Launch the turtelbot3 empty environment.
5. Observe and describe what happens when you launch 'pengwang_publisher.py' and 'pengwang_publisher_line.py', respectively.
6. Can you stop the robot from running? (Hint: There are two ways!)

1.ToobaZahid_publisher.py is rewritten to ToobaZahid_publisher_line.py.

- The robot moved in a circle ToobaZahid_publisher.py had both linear and angular velocities.
- The robot moves in a straight line ToobaZahid_publisher_line.py, which sets the linear velocity and sets the angular velocity to zero.



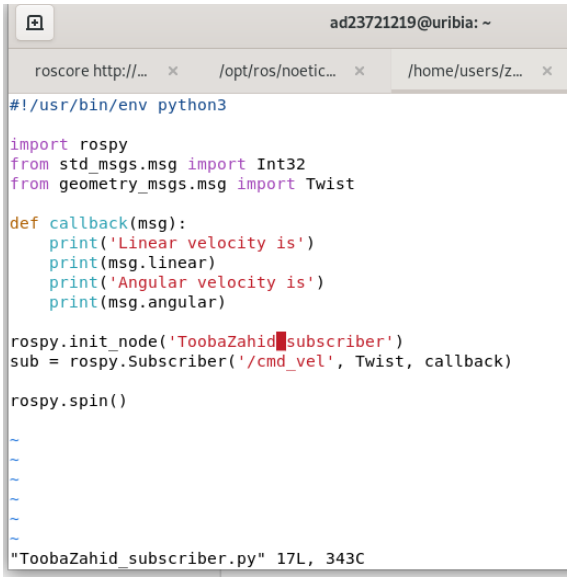
2. The launch file for ToobaZahid_publisher_line.py has been created.

The ToobaZahid_publisher_line.launch.

```
1 <launch>
2 <node name="ToobaZahid_publisher" pkg="rss_pubsub_pkg" type="ToobaZahid_publisher.py"
  output="screen"/>
3 </launch>
```

3. ToobaZahid_Subscriber.py:

- ToobaZahid_subscriber.py for listening to the subject /cmd_vel.
- It prints the linear and angular velocities being published, confirming the motion commands sent to the robot.



so

Rotational Movement: Robot is instructed to rotate around the z-axis at a 0.5 angular velocity and no linear velocity in this state. the robot stops moving forward and turns in position.

```
4296-1774-11ef-818
Linear velocity is
x: 0.5
y: 0.0
z: 0.0
Angular velocity i
x: 0.0
y: 0.0
z: 0.5
Linear velocity is
x: 0.5
y: 0.0
z: 0.0
Angular velocity i
x: 0.0
y: 0.0
z: 0.5
Linear velocity is
x: 0.5
y: 0.0
```

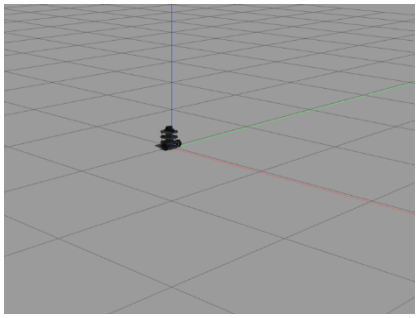
Linear Movement: The robot is instructed to move with 0 angular velocity. So, it is moving forward straight ahead without turning.

```
process[TboobaZahid_publisher_
process[TboobaZahid_subscriber
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
```

4.launch the turtlebot3 empty environment:

This command is used to launch it.

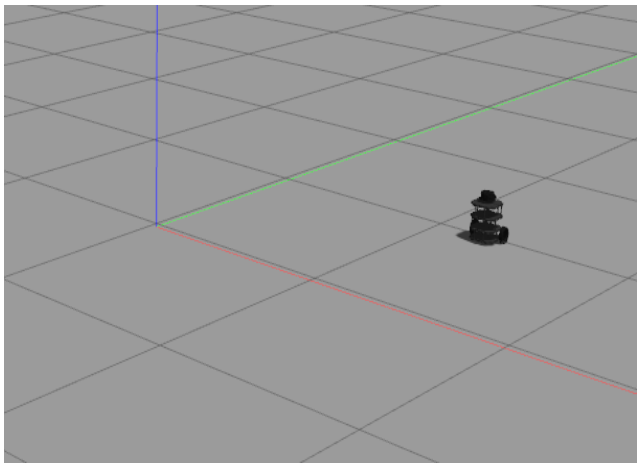
```
:$> roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```



5.Observations after running both scripts

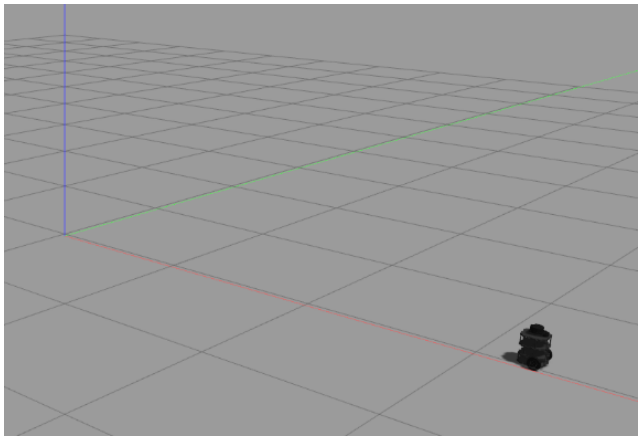
With ToobaZahid_publisher.py,

The robot angular velocity (0.5) causes it to travel continually in a circle. I just try to display my robot move in circle. :)



With ToobaZahid_publisher_line.py:

The angular velocity is 0, so the robot always travels in a straight line.



6.Stops Robot

I try to stop robot using 2 ways.

- 1st is set a command Send a Twist message to the /cmd_vel topic, setting the robot's linear and angular velocities to 0, stopping all motion

```
ad23721219@ros_noetic.sif:~/rss2_catkin_ws$> rostopic pub -1 /cmd_vel geometry_msgs/
Twist -- '[0, 0, 0]' '[0, 0, 0]'
publishing and latching message for 3.0 seconds
ad23721219@ros_noetic.sif:~/rss2_catkin_ws$>
```

- 2nd is Shutdown the ROS Node: To stop execution, use a command like Ctrl+C at the terminal where the ROS Node is running.

****

Personal Assignment 4:

Personal Assignment 4

```
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.
3. Write a launch file to start it.
4. Launch the turtelbot3 empty environment to observe if it works.
```

1.Write a ToobaZahid_Pubsub.py file:

/home/users/zahidtoo/turtlebot/rss2_catkin_ws/src/rss_pubsub_pkg/launch/ToobaZahid_Pubsub.la

roscore http://itagui:11311/ × /opt/ros/noetic/share/turtlebot3_... × /home/users/zahid

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

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

class RobotController:
    def __init__(self):

        rospy.init_node('ToobaZahid_pubsub')
        self.pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        self.sub = rospy.Subscriber('/odom', Odometry, self.odom_callback)
        self.initial_position = None
        self.goal_distance = 2.0  # Meters

    def odom_callback(self, msg):

        if self.initial_position is None:
            self.initial_position = msg.pose.pose.position

        current_position = msg.pose.pose.position
        distance = math.sqrt((current_position.x - self.initial_position.x)**2 +
                             (current_position.y - self.initial_position.y)**2)

        if distance >= self.goal_distance:
            self.publish_velocity(0, 0)  # Stop the robot
            rospy.signal_shutdown('Reached Goal Distance')
        else:
            self.publish_velocity(0.5, 0)  # Continue moving forward

    def publish_velocity(self, linear, angular):
        move_cmd = Twist()
        move_cmd.linear.x = linear
        move_cmd.angular.z = angular
        self.pub.publish(move_cmd)

if __name__ == '__main__':
    try:
        controller = RobotController()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

RobotController Class

Constructor __init__(self).

- Sets up a ROS subscriber to receive Odometry messages, which are handled by the odom_callback function.

- Sets the robot's beginning position's initial_position to None.

Method odom_callback (self, msg)

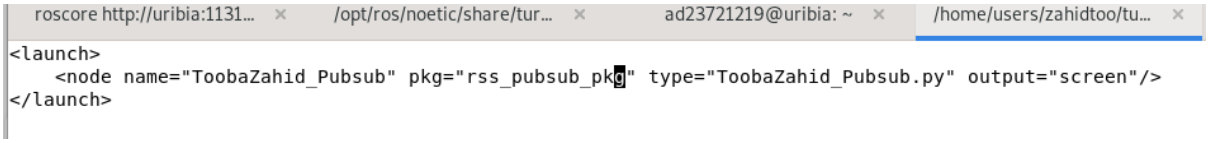
- Captures robot's starting position.
- Calculates current distance from initial position using Euclidean distance formula.
- If the calculated distance is greater than or equal to goal_distance, it stops the robot by publishing a 0 velocity and calls rospy.signal_shutdown to end the node.
- Otherwise, it publishes a forward velocity to continue moving the robot.

2.Define A Distance the robot needs to run

In above code, the robot has a 2.0 metre run distance built into its programming. The `self. goal_distance = 2.0} attribute is used to set this distance in the `__init__} function of the `RobotController` class.

3.Write a Launch File to start it (ToobaZahid_Pubsub.launch)

Here I created my launch file:



4.Launch an empty turtlebot environment:

To start the ROS node

```
s$> roslaunch rss_pubsub_pkg ToobaZahid_Pubsub.launch
```

My robot successfully moves and stops after goal reached.

****

Personal Assignment 5:

Personal Assignment 5

- Create launch files to start the publisher and subscriber in section 5.2 in Part I.
- Create launch files to start the server and client in sections 4.7 and 4.8 in Part II.
- 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

1.Create Launch files to start the publisher and subscriber for section 5.2

To launch msg_sub.py and msg_pub.py

```
s$> roslaunch rss2_msgsrv_pkg pub.launch
 roslaunch rss2_msgsrv_pkg sub.launch
```

Below is the output how it shows after running.

```
ROS_MASTER_URI=http://localhost:11311

process[msg_sub-1]: started with pid [4300]
[INFO] [1716401406.182699, 101.567000]: 05/22/24, 19:09:20
[INFO] [1716401406.183826, 101.568000]: linear:
  x: 0.5
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.1
[INFO] [1716401407.184525, 102.567000]: 05/22/24, 19:09:20
[INFO] [1716401407.185624, 102.568000]: linear:
  x: 0.5
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.1
[INFO] [1716401408.185706, 103.567000]: 05/22/24, 19:09:20
[INFO] [1716401408.186749, 103.567000]: linear:
```

1.Create Launch files to start the Server and Client

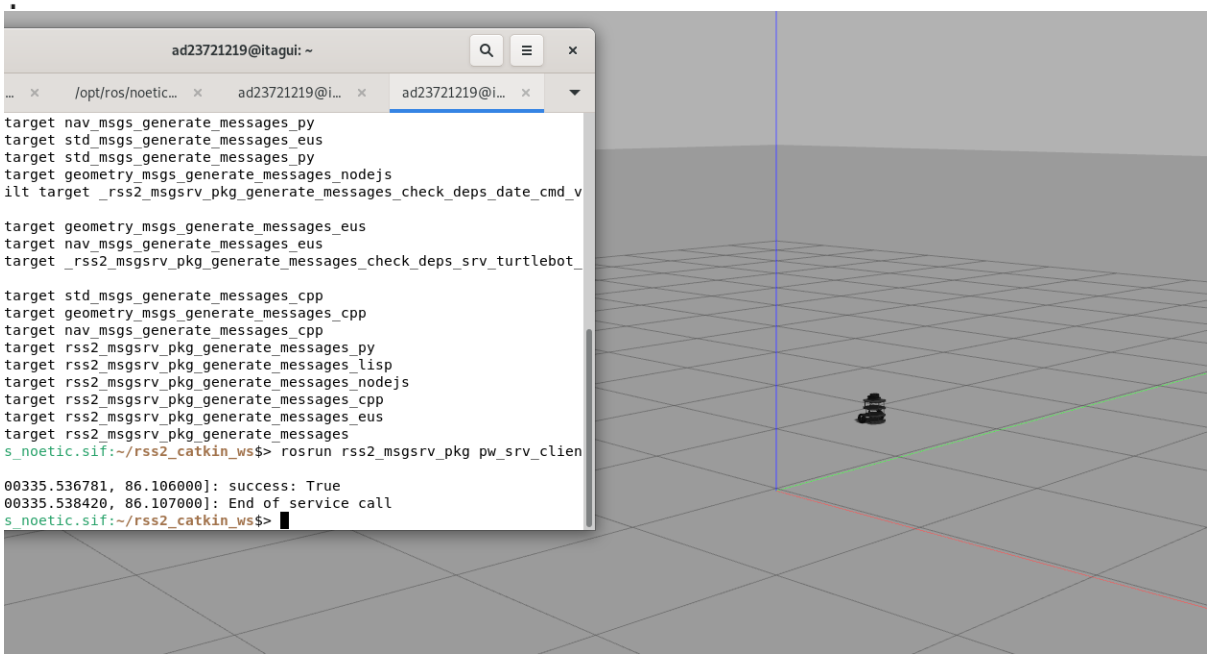
```
roslaunch rss2_msgsrv_pkg pw_srv_client.launch
roslaunch rss2_msgsrv_pkg pw_srv_server.launch
```

I used same file pw_srv_server.py and pw_srv_client_py. the robot moves in circle and stops moving after (20 sec)

```
process[pw_srv_server-1]: started with pid [7302]
[INFO] [1716402869.499811, 1562.811000]: Service /turtlebot_move_service is ready!
[INFO] [1716402875.914884, 1569.219000]: Turtlebot_move_service has been called
[INFO] [1716402875.916076, 1569.220000]: time = 0
[INFO] [1716402875.917279, 1569.221000]: time = 1
[INFO] [1716402876.919196, 1570.221000]: time = 2
[INFO] [1716402877.920965, 1571.221000]: time = 3
[INFO] [1716402878.921376, 1572.221000]: time = 4
[INFO] [1716402879.923414, 1573.221000]: time = 5
[INFO] [1716402880.924481, 1574.221000]: time = 6
[INFO] [1716402881.926124, 1575.221000]: time = 7
[INFO] [1716402882.927346, 1576.221000]: time = 8
[INFO] [1716402883.928488, 1577.221000]: time = 9
[INFO] [1716402884.929563, 1578.221000]: time = 10
[INFO] [1716402885.930092, 1579.221000]: time = 11
[INFO] [1716402886.932852, 1580.221000]: time = 12
[INFO] [1716402887.934404, 1581.221000]: time = 13
[INFO] [1716402888.935239, 1582.221000]: time = 14
[INFO] [1716402889.937028, 1583.221000]: time = 15
[INFO] [1716402890.938129, 1584.221000]: time = 16
[INFO] [1716402891.939465, 1585.221000]: time = 17
```

```
ROS_MASTER_URI=http://localhost:11311

process[pw_srv_client-1]: started with pid [7333]
[INFO] [1716402896.946356, 1590.221000]: success: True
[INFO] [1716402896.947834, 1590.222000]: End of service call
[pw_srv_client-1] process has finished cleanly
log file: /home/users/zahidtoo/turtlebot/.ros/log/10cca61c-1866-11ef-87e9-c8d9d234aa
all processes on machine have died, roslaunch will exit
shutting down processing monitor...
... shutting down processing monitor complete
```



Modify Code (4.7,4.8)

```python
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')

    # for the first 20 seconds
    vel.linear.x = 0.2
    vel.angular.z = 0.2
    total_time = 0
    while total_time < 20:
        pw_pub.publish(vel)
        rospy.loginfo('Moving in a circle, time = %d', total_time)
        rate.sleep()
        total_time += 1

    # Stop for 5 seconds
    vel.linear.x = 0.0
    vel.angular.z = 0.0
    for _ in range(5):
        pw_pub.publish(vel)
        rospy.loginfo('Stopping, time = %d', total_time)
        rate.sleep()
        total_time += 1

    # Move along x-axis for 5 seconds
    vel.linear.x = 0.2
    vel.angular.z = 0.0
    for _ in range(5):
        pw_pub.publish(vel)
        rospy.loginfo('Moving along x-axis, time = %d', total_time)
        rate.sleep()
        total_time += 1

    # Stop the robot
    vel.linear.x = 0.0
    vel.angular.z = 0.0
    pw_pub.publish(vel)
    rospy.loginfo('Stopped, time = %d', total_time)

    return srv_turtlebot_moveResponse(True)

rospy.init_node('turtlebot_move_server')

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

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

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

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

# Maintain the service
rospy.spin()
```

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

import rospy
from rss2_msgsrv_pkg.srv import srv_turtlebot_move, srv_turtlebot_moveRequest

# The service client node
rospy.init_node('turtlebot_move_client')

# Wait for the service '/turtlebot_move_service' to run
# You need to start the service first
rospy.wait_for_service('/turtlebot_move_service')

# Connect to the service '/turtlebot_move_service'
turtlebot_service_client = rospy.ServiceProxy('/turtlebot_move_service', srv_turtlebot_move)

# Create a request instance
turtlebot_request_instance = srv_turtlebot_moveRequest()
turtlebot_request_instance.duration = 30  # Total duration is 30 seconds

# Send the request to the server through the connection built
feedback = turtlebot_service_client(turtlebot_request_instance)

# Show results after the service is called
rospy.loginfo(str(feedback))

rospy.loginfo('End of service call')
```

I changed duration to 30.

1. **The first 20 seconds the robot moves in a circle.**

```python
# for the first 20 seconds
vel.linear.x = 0.2
vel.angular.z = 0.2
total_time = 0
while total_time < 20:
    pw_pub.publish(vel)
    rospy.loginfo('Moving in a circle, time = %d', total_time)
    rate.sleep()
    total_time += 1
```

For a duration of 20 seconds, the function publishes the velocity command every second and sets the linear and angular velocities for a circle.

2. **Then stops for 5 seconds.**

```python
# Stop for 5 seconds
vel.linear.x = 0.0
vel.angular.z = 0.0
for _ in range(5):
    pw_pub.publish(vel)
    rospy.loginfo('Stopping, time = %d', total_time)
    rate.sleep()
    total_time += 1
```

set its angular and linear velocities to zero and send out a stop command once every second for five seconds.

3. **Then moves along x-axis for 5 seconds**

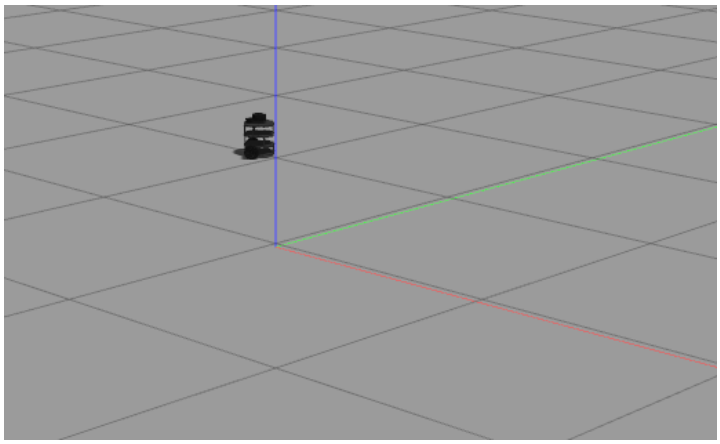
```python
# Move along x-axis for 5 seconds
vel.linear.x = 0.2
vel.angular.z = 0.0
for _ in range(5):
    pw_pub.publish(vel)
    rospy.loginfo('Moving along x-axis, time = %d', total_time)
    rate.sleep()
    total_time += 1
```

The forward instruction is set every second for five seconds, with the angular velocity set to zero and the movement set to forward.

4. Stop

```
# Stop the robot
vel.linear.x = 0.0
vel.angular.z = 0.0
pw_pub.publish(vel)
rospy.loginfo('Stopped, time = %d', total_time)

return srv_turtlebot_moveResponse(True)
```



Zeroes both angular and linear velocities.
Sends out a single stop command.

****

Personal Assignment 6:

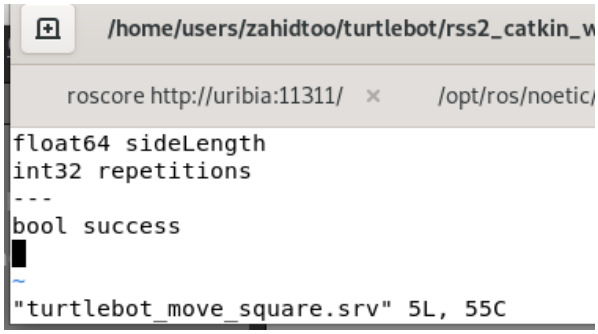
Personal Assignment 6

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.
- Modify 'CMakeLists.txt' and 'package.xml'.
- 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'.

Create a turtlebot_move_srv:



Modify "CMakeList.txt" package.xml:

```
## Generate services in the 'srv' folde
add_service_files(
  FILES
  srv_turtlebot_move.srv
  turtlebot_move_square.srv
)
```

Create a client and server:

1.

1. When the server being called, it should move along a square with side length defined by 'sideLength'.

roscore http://uribia:11311/ × /opt/ros/noetic/share/turtlebot3_gazebo/l... × /home/users/zahidtoo/tu

```
from geometry_msgs.msg import Twist
import time

def move_straight(side_length):
    vel.linear.x = 0.2
    vel.angular.z = 0.0
    move_time = side_length / 0.2
    start_time = time.time()
    while (time.time() - start_time) < move_time:
        pw_pub.publish(vel)
        rate.sleep()
    vel.linear.x = 0.0
    pw_pub.publish(vel)
    rate.sleep()

def turn_90_degrees():
    vel.linear.x = 0.0
    vel.angular.z = 0.5
    turn_time = 1.57 / 0.5  # time to turn 90 degrees (1.57 radians)
    start_time = time.time()
    while (time.time() - start_time) < turn_time:
        pw_pub.publish(vel)
        rate.sleep()
    vel.angular.z = 0.0
    pw_pub.publish(vel)
    rate.sleep()

def move_square(side_length):
    for _ in range(4):
        move_straight(side_length)
        turn_90_degrees()

def my_callback(request):
    rospy.loginfo('Turtlebot_move_square_service has been called')
    success = True
    try:
        for _ in range(request.repetitions):
            move_square(request.sideLength)
    except Exception as e:
        rospy.logerr(f"Error during movement: {e}")
        success = False
    return turtlebot_move_squareResponse(success)

rospy.init_node('turtlebot_move_square_server')

pw_service = rospy.Service('/turtlebot_move_square_service', turtlebot_move_square, my_callback)

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

rate = rospy.Rate(10)

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

rospy.spin()
```

2.

2. The robot must repeat moving along the square defined by ‘repetitions’,

I used repetitions =2

roscore http://uribia:11311/ × /opt/ros/noetic/share/turtlebot3_gazebo/l... × /home/users/zahidtoo/turtlebot/rss2_

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

import rospy
from rss2_msgsrv_pkg.srv import turtlebot_move_square, turtlebot_move_squareRequest

rospy.init_node('turtlebot_move_square_client')

rospy.wait_for_service('/turtlebot_move_square_service')

try:
    turtlebot_service_client = rospy.ServiceProxy('/turtlebot_move_square_service', turtlebot_move_square)
    request = turtlebot_move_squareRequest()
    request.sideLength = 1.0  # Example side length
    request.repetitions = 2  # Example repetitions
    response = turtlebot_service_client(request)
    rospy.loginfo('Service call result: %s', response.success)
except rospy.ServiceException as e:
    rospy.logerr('Service call failed: %s', e)
```

3.

3. When the robot finishes the movement, it should return 'True'. Otherwise, 'False'.

```
process[turtlebot_move_square_client-2]: started with pid [5849]
[INFO] [1716459910.846636, 1020.366000]: Service /turtlebot_move_square_service is ready!
[INFO] [1716459911.148715, 1020.666000]: Turtlebot_move_square_service has been called
[INFO] [1716459978.439190, 1087.867000]: Service call result: True
[turtlebot_move_square_client-2] process has finished cleanly
```

****