BRAC UNIVERSITY

Inspiring Excellence

FALL 2023

COURSE TITLE: Robotics
COURSE CODE: CSE461

PREPARED BY

Name: Nur-E-Jannat

ID: 21301744

Section: 6

Title: Turtle bot control

Overview:

The Turtle Bot Control project involves utilizing ROS (Robot Operating System) and Python scripting to control a turtle bot's movements. The primary objectives include creating scripts for specific tasks, executing them to observe the turtle bot's behavior, and reflecting on the overall experience.

Code:

Task_1

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

def move_rectangular_path():
    # Starts a new node
    rospy.init_node('robot_mover', anonymous=True)
    vel_pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
    vel_msg = Twist()

    # Set a default speed
    robot_speed = 1.0

    # Receive user input for height and width of the rectangle
    print("Let's move your robot in a perfect rectangular path")
    rect_height = input("Enter the height of the rectangle: ")
    rect_width = input("Enter the width of the rectangle: ")
```

```
rect_height = float(rect_height)
rect_width = float(rect_width)

# Move right
vel_msg.linear.x = abs(robot_speed)
vel_msg.linear.y = 0
vel_msg.angular.z = 0
vel_pub.publish(vel_msg)
rospy.sleep(rect_width / robot_speed)

# Stop the turtle
vel_msg.linear.x = 0
vel_msg.linear.y = 0
vel_msg.angular.z = 0
vel_pub.publish(vel_msg)

# Move up
vel_msg.linear.x = 0
vel_msg.linear.y = abs(robot_speed)
vel_msg.angular.z = 0
vel_pub.publish(vel_msg)
rospy.sleep(rect_height / robot_speed)

# Stop the turtle
vel_msg.linear.x = 0
vel_msg.linear.y = 0
vel_msg.angular.z = 0
vel_pub.publish(vel_msg)

# Move left
vel_msg.linear.x = -abs(robot_speed)
vel_msg.linear.y = 0
vel_msg.angular.z = 0
vel_pub.publish(vel_msg)
rospy.sleep(rect_width / robot_speed)
```

```
    # Stop the turtle
    vel_msg.linear.x = 0
    vel_msg.linear.y = 0
    vel_msg.angular.z = 0
    vel_pub.publish(vel_msg)

    # Move down
    vel_msg.linear.x = 0
    vel_msg.linear.y = -abs(robot_speed)
    vel_msg.angular.z = 0
    vel_pub.publish(vel_msg)
    rospy.sleep(rect_height / robot_speed)

    # Stop the turtle
    vel_msg.linear.x = 0
    vel_msg.linear.y = 0
    vel_msg.angular.z = 0
    vel_pub.publish(vel_msg)

if __name__ == '__main__':
    try:
        # Testing our function
        move_rectangular_path()
    except rospy.ROSInterruptException:
        pass
```

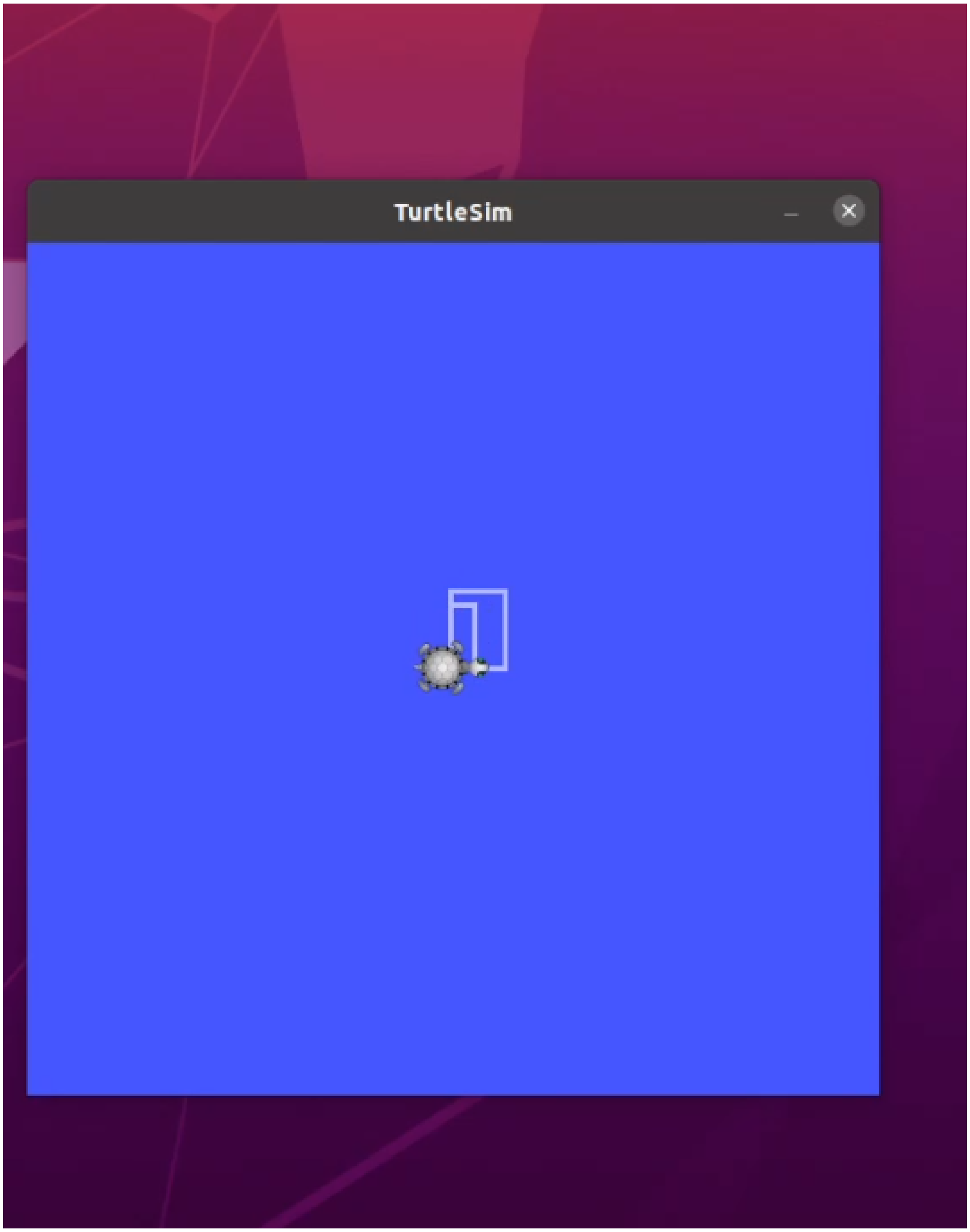
Task_2

```
#!/usr/bin/python3
import rospy
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose
import math

def pose_callback(data):
    # Callback function to get the turtle's current position
    global turtle_x
    turtle_x = data.x

def perform_archimedean_spiral():
    # Initialize a new ROS node
    rospy.init_node('cleaning_robot', anonymous=True)
    velocity_publisher = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
    pose_subscriber = rospy.Subscriber('/turtle1/pose', Pose, pose_callback)
    vel_msg = Twist()

    # Wait for the first pose message to get the initial turtle position
    rospy.wait_for_message('/turtle1/pose', Pose)

    # Collect user input for robot movement parameters
    print("Initiating Archimedean spiral movement")
    speed = input("Enter the speed of the robot: ")
    constant = input("Enter the constant for the Archimedean spiral: ")
    max_distance = input("Enter the maximum distance to travel before stopping: ")

    # Convert input values to float
    speed = float(speed)
    constant = float(constant)
    max_distance = float(max_distance)

    # Move the robot in an Archimedean spiral path
```

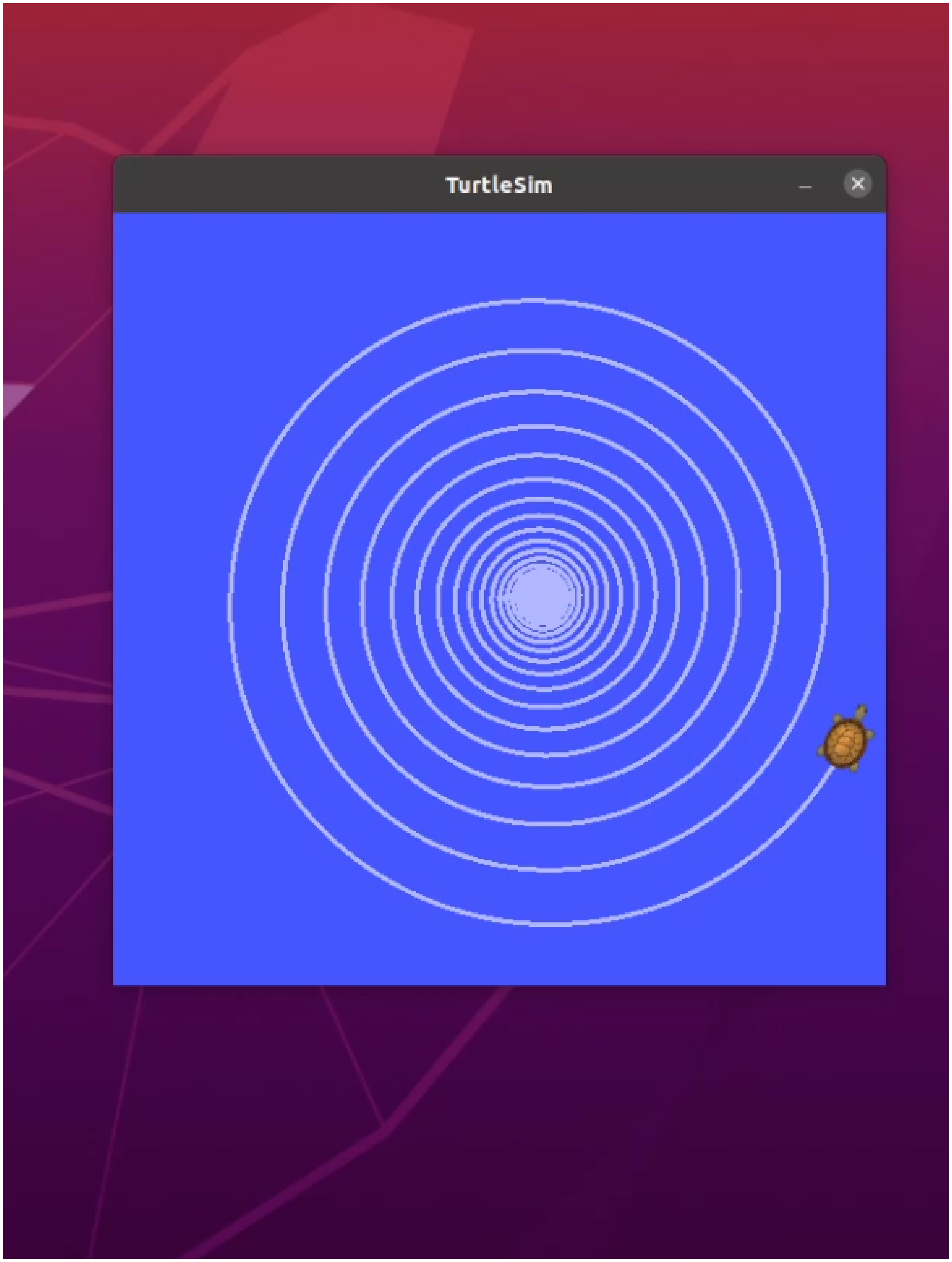
```
    rate = rospy.Rate(10)  # 10 Hz

    initial_x = turtle_x
    counter = 0

    while abs(turtle_x - initial_x) < max_distance:
        angle = math.radians(counter / 10.0)
        radius = constant * angle

        vel_msg.linear.x = speed
        vel_msg.angular.z = speed / radius if radius > 0.01 else 0

        velocity_publisher.publish(vel_msg)
        rate.sleep()

        counter += 1

    # Stop the robot after reaching the specified distance
    vel_msg.linear.x = 0
    vel_msg.angular.z = 0
    velocity_publisher.publish(vel_msg)

if __name__ == '__main__':
    try:
        # Test the Archimedean spiral movement function
        perform_archimedean_spiral()
    except rospy.ROSInterruptException:
        pass
```

Final Output Image

Task_1



Task_2



Conclusion

From this project, I learnt to control turtle bot and the operations of the ROS system. In task-1, it was difficult to make a rectangle and I faced a great problem to control the ending point of the turtle bot in the spiral path. Finally, I solved it. In a nutshell, now I am confident to work with ROS and Turtle bot.

Question-Answer

1. The communication between the controller and the turtle bot:
    - In ROS, the communication between the controller (Python scripts) and the turtle bot occurs through a publish-subscribe mechanism.
    - The controller (script) publishes messages to specific topics, which are channels for communication in ROS.
    - The turtle bot subscribes to these topics, listening for the published messages, and executes actions accordingly.
    - In the provided scripts (rectangle.py and rotate.py), rospy.Publisher is used to publish velocity commands to the /turtle1/cmd_vel topic, controlling the turtle bot's movements.

2. The challenges i faced in this lab:
    - Understanding ROS Concepts
    - Scripting Turtle Bot Movements
    - Debugging and Testing
    - Optimizing Motion Patterns

YouTube link

Task 1:
https://drive.google.com/file/d/1ocShmxWTj4gXJRBL8nyEXfDqPtcr91C8/view?usp=sharing
Task 2:
https://drive.google.com/file/d/1sGkrCWbQxNXQP3S_c57JjWw7PEG4V6XS/view?usp=drive_link