CSE461 - Introduction to Robotics Lab Introduction to Robotics Lab

Lab Report 04

Section: 06

FALL – 2023

Submitted by:

Farhan Akbor Khan — [redacted]

Title : Turtlebots Control using Ubuntu

Task Description / Overview :

This lab report focuses on moving the turtle in particular distances with a ROS software. We used the Ubuntu operating system to install ROS which helps us to control turtlebots. This lab's main goal is to demonstrate how to utilize turtlebots' control and move the turtle in a precisely measured distance. This lab practice was really helpful in a number of ways. First of all, it was a priceless learning opportunity that gave me a hands-on understanding of how powerful ROS is at directing the actions of robotic platforms, especially TurtleBots. Second, it established a strong basis for knowledge about the exact control of robotic motions, which is essential to the larger field of robotics research.

Code

Task 01 : Move the turtle to create a rectangular path with height and width.

```
#!/usr/bin/python3
import rospy # Communication
from geometry_msgs.msg import Twist # Message: position, angle etc

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

    #Receiveing the user's input
    print("Let's move the robot or turtle in a rectangle area.")
    speed = input("Input your speed:") # Say 1
    distance = input("Type Rectangle Width:") # Say 1
    height= input("Type Rectangle Height:")
    speed = float(speed)
    distance = float(distance)
    height = float(height)

    #Checking if the movement is forward or backwards

    vel_msg.linear.x = abs(speed)


    #Since we are moving just in x-axis
```

```
vel_msg.linear.y= 0
vel_msg.linear.z = 0
vel_msg.angular.x = 0
vel_msg.angular.y = 0
vel_msg.angular.z = 0

for i in range(1):

 # Movement forward x-axis

    #Setting the current time for distance calculus
    t0 = rospy.Time.now().to_sec()
    current_distance = 0

    #Loop to move the turtle in an specified distance
    while(current_distance < distance):
      #Publish the velocity
      velocity_publisher.publish(vel_msg)
      #Takes actual time to velocity calculus
      t1=rospy.Time.now().to_sec()
      #Calculates distancePoseStamped
      current_distance= speed*(t1-t0)
    #After the loop, stops the robot
    vel_msg.linear.x = 0
    #Force the robot to stop
    velocity_publisher.publish(vel_msg)

# Movement upward y-axis

    t2 = rospy.Time.now().to_sec()
    current_height = 0
```

```
    vel_msg.linear.y= abs(speed)

    while(current_height < height):
      #Publish the velocity
      velocity_publisher.publish(vel_msg)
      #Takes actual time to velocity calculus
      t3=rospy.Time.now().to_sec()
      #Calculates distancePoseStamped
      current_height= speed*(t3-t2)
    #After the loop, stops the robot
    vel_msg.linear.y = 0
    #Force the robot to stop
    velocity_publisher.publish(vel_msg)

  # Movement backward x-axis

    #Setting the current time for distance calculus
    t0 = rospy.Time.now().to_sec()
    current_distance = 0
    vel_msg.linear.x = -abs(speed)

    #Loop to move the turtle in an specified distance
    while(current_distance < distance):
      #Publish the velocity
      velocity_publisher.publish(vel_msg)
      #Takes actual time to velocity calculus
      t1=rospy.Time.now().to_sec()
      #Calculates distancePoseStamped
      current_distance= speed*(t1-t0)
    #After the loop, stops the robot
    vel_msg.linear.x = 0
```

```
        #Force the robot to stop
        velocity_publisher.publish(vel_msg)

    # Movement downward y-axis

        t6 = rospy.Time.now().to_sec()
        current_height = 0
        vel_msg.linear.y= -abs(speed)

        while(current_height < height):
          #Publish the velocity
          velocity_publisher.publish(vel_msg)
          #Takes actual time to velocity calculus
          t7=rospy.Time.now().to_sec()
          #Calculates distancePoseStamped
          current_height= speed*(t7-t6)
        #After the loop, stops the robot
        vel_msg.linear.y = 0
        #Force the robot to stop
        velocity_publisher.publish(vel_msg)

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

Task 02 : Move the turtle in an outward spiral path.

```
#!/usr/bin/python3
import rospy # Communication
from geometry_msgs.msg import Twist # Message: position, angle etc

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

    #Receiveing the user's input
    print("Let's move the turtle in an outward spiral path.")
    speed = input("Input your speed:") # Say 1
    speed = float(speed)
    distance = int(-1)
    height = int(-1)

    #Checking if the movement is forward or backwards

    vel_msg.linear.x = abs(speed)



    #Since we are moving just in x-axis
    vel_msg.linear.y= 0
    vel_msg.linear.z = 0
    vel_msg.angular.x = 0
    vel_msg.angular.y = 0
    vel_msg.angular.z = 0
```

```
x=5
for i in range(1):

 # Movement forward x-axis

    while int(distance)<=x:
      #Setting the current time for distance calculus
      t0 = rospy.Time.now().to_sec()
      current_distance = 0
      vel_msg.linear.x = abs(speed)
      #Loop to move the turtle in an specified distance
      distance=int(distance)+1
      while(current_distance < int(distance)+1):
        #Publish the velocity
        velocity_publisher.publish(vel_msg)
        #Takes actual time to velocity calculus
        t1=rospy.Time.now().to_sec()
        #Calculates distancePoseStamped
        current_distance= speed*(t1-t0)
      #After the loop, stops the robot
      vel_msg.linear.x = 0
      #Force the robot to stop
      velocity_publisher.publish(vel_msg)

# Movement upward y-axis

      t2 = rospy.Time.now().to_sec()
      current_height = 0
      vel_msg.linear.y= abs(speed)
      height=int(height)+1
      while(current_height < int(height)+1):
```

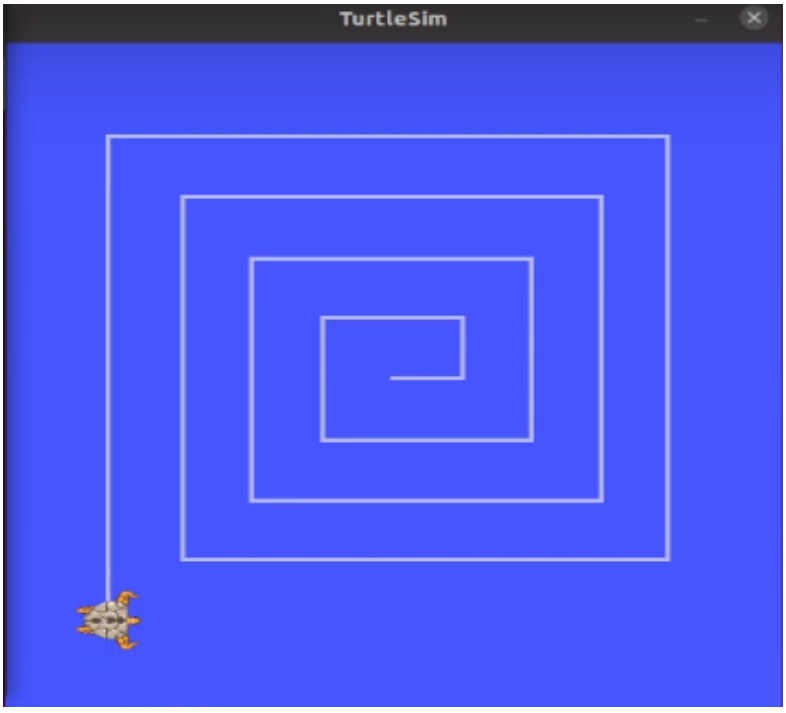
```
        #Publish the velocity
        velocity_publisher.publish(vel_msg)
        #Takes actual time to velocity calculus
        t3=rospy.Time.now().to_sec()
        #Calculates distancePoseStamped
        current_height= speed*(t3-t2)
    #After the loop, stops the robot
    vel_msg.linear.y = 0
    #Force the robot to stop
    velocity_publisher.publish(vel_msg)

# Movement backward x-axis

    #Setting the current time for distance calculus
    t0 = rospy.Time.now().to_sec()
    current_distance = 0
    vel_msg.linear.x = -abs(speed)

    #Loop to move the turtle in an specified distance
    while(current_distance < int(distance)+2):
        #Publish the velocity
        velocity_publisher.publish(vel_msg)
        #Takes actual time to velocity calculus
        t1=rospy.Time.now().to_sec()
        #Calculates distancePoseStamped
        current_distance= speed*(t1-t0)
    #After the loop, stops the robot
    vel_msg.linear.x = 0
    #Force the robot to stop
    velocity_publisher.publish(vel_msg)
```

```
    # Movement downward y-axis

        t6 = rospy.Time.now().to_sec()
        current_height = 0
        vel_msg.linear.y= -abs(speed)

        while(current_height < int(height)+2):
          #Publish the velocity
          velocity_publisher.publish(vel_msg)
          #Takes actual time to velocity calculus
          t7=rospy.Time.now().to_sec()
          #Calculates distancePoseStamped
          current_height= speed*(t7-t6)
        #After the loop, stops the robot
        vel_msg.linear.y = 0
        #Force the robot to stop
        velocity_publisher.publish(vel_msg)
        distance=int(distance)+1
        height=height+1



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

Final Output

task01

task02



Conclusion

The use of Python programming allowed for the delivery of exact commands, guaranteeing that the TurtleBot's motions were accurately measured and verified. This lab gave a basic knowledge of ROS's function in coordinating robotic motions in addition to demonstrating control over TurtleBots. In conclusion, this project demonstrates how effective ROS is at enabling fine-grained control over robotics systems and lays the groundwork for future investigations and developments in the field of robotics research.

Question - Answer

1. **How does the communication between the controller and the turtle bot happen?**

   Communication in a ROS-driven system that manages a TurtleBot takes place via a publish-subscribe approach that makes use of ROS topics and messages. The controller is often a Python script or a specialized ROS node. The TurtleBot is instructed to go ahead, turn, or halt by means of these communications. In parallel, the TurtleBot subscribes to these topics and watches for incoming messages using its ROS node. When the TurtleBot receives commands from subscribed subjects, it decodes these messages, follows the instructions, and moves in the appropriate direction. With the help of this communication architecture, the TurtleBot and the controller can communicate with precision, allowing for autonomous behavior or precise remote control based on commands sent via ROS topics and messages.

2. What are the challenges faced in this lab?

The TurtleBot's motions are difficult to manage precisely because of things like software calibration, sensor precision, and possible communication delays. Precisely aligning software orders with the robot's physical reaction is necessary to achieve the desired distances, which calls for thorough debugging and synchronization. And, I faced a lot of problems while installing a virtual machine on my computer.

Youtube Link

https://www.youtube.com/watch?v=YPgaAO7r3L8