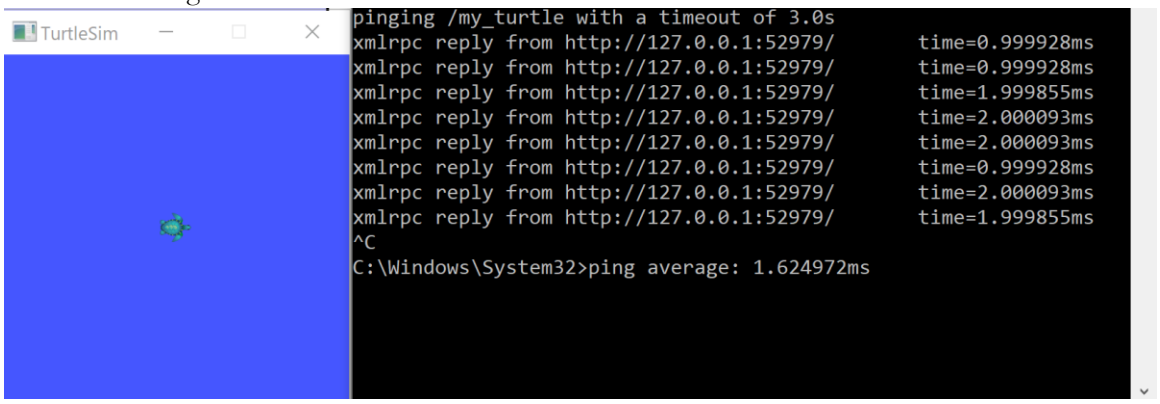
ME 459/5559 – Robotics and Unmanned Systems HW #3: DUE February 18th, 2022

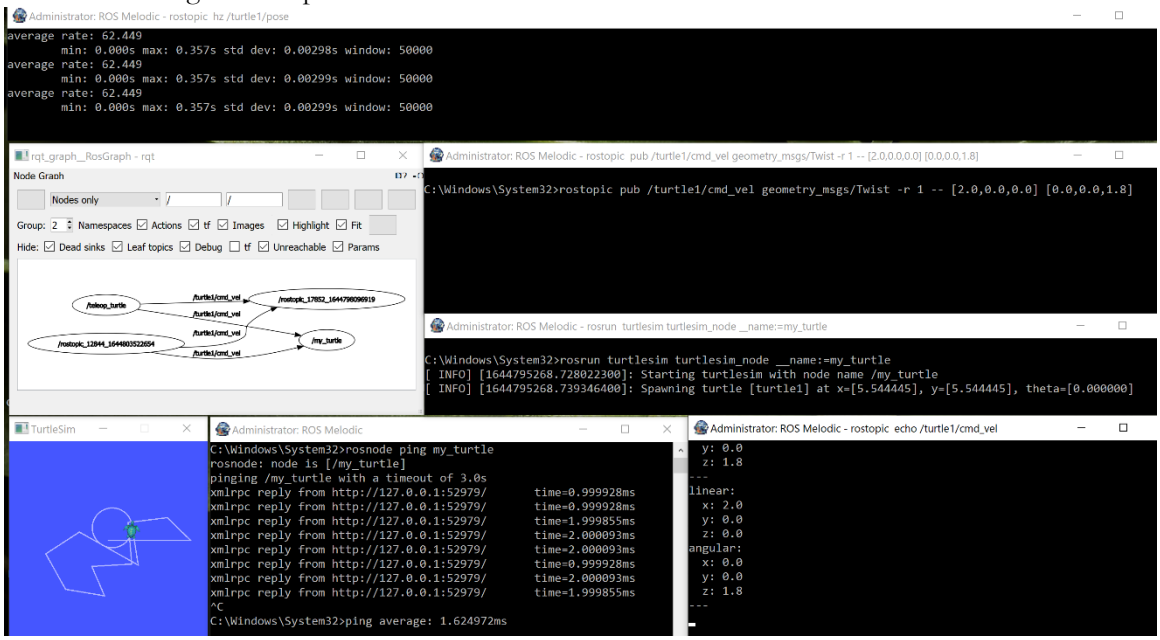
Problem 1:

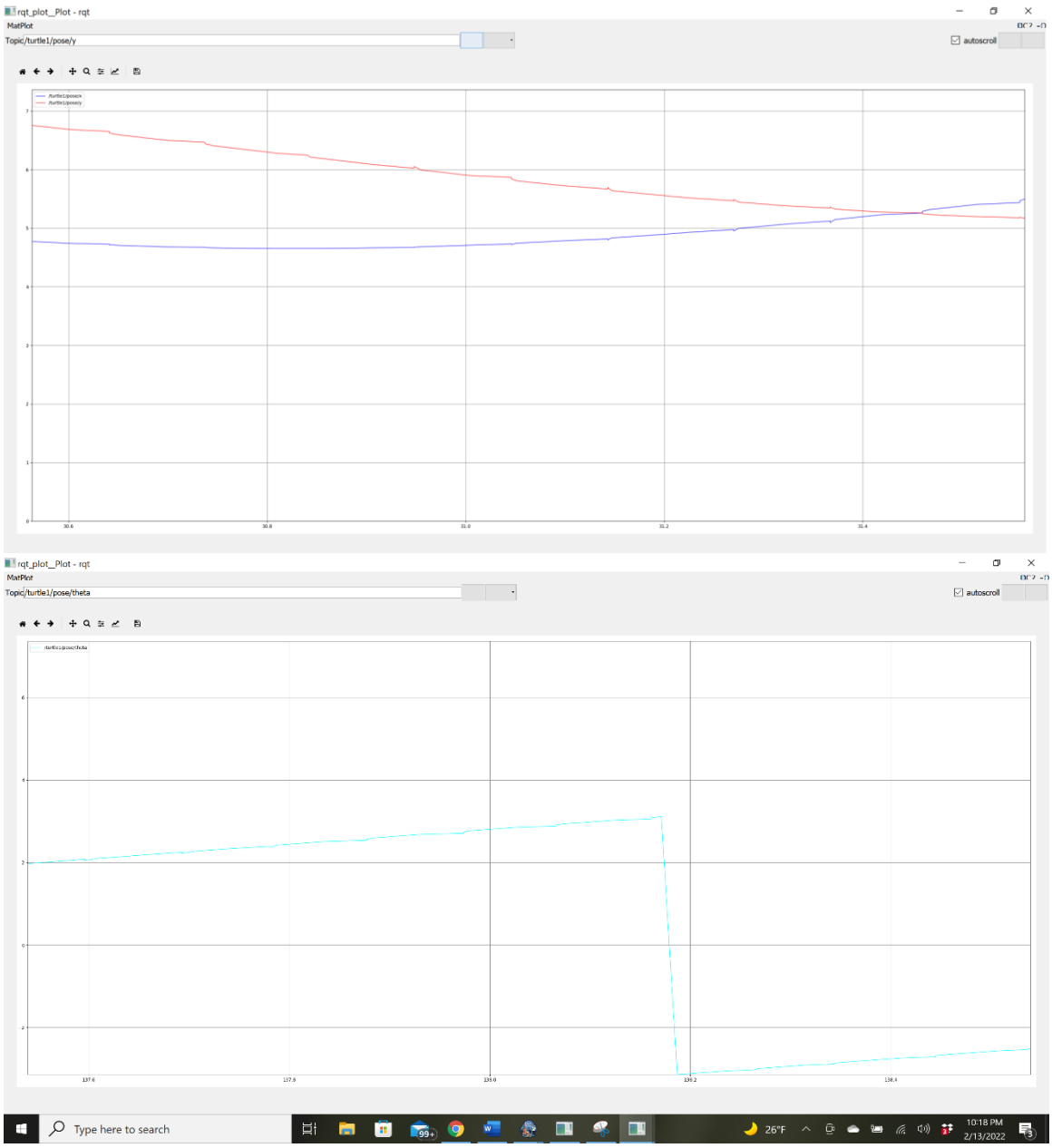
Complete the following ROS tutorials at http://wiki.ros.org/ROS/Tutorials

- 5. Understanding ROS Nodes



- 6. Understanding ROS Topics





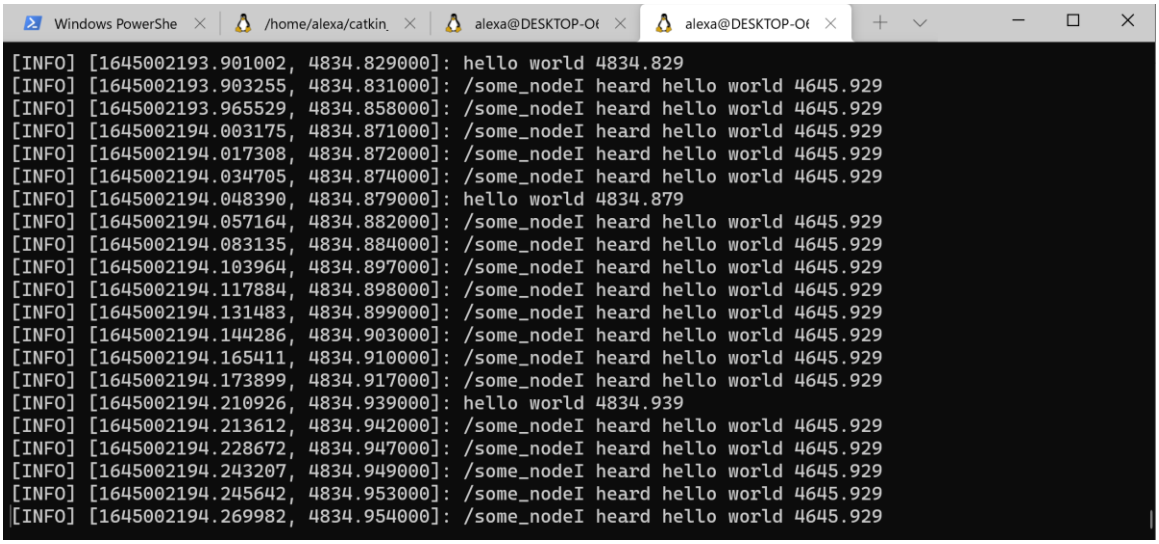
Save a screenshot of the Turtlebot simulator and show/print the screenshot.

Problem 2:

Complete the following ROS tutorials at http://wiki.ros.org/ROS/Tutorials

- 12. Writing a Simple Publisher and Subscriber (Python)
- 13. Examining the Simple Publisher and Subscriber

Save a screenshot of the running code in tutorials 13, and print these screenshots to turn in.



Problem 3:

For this problem you will be using ROS and Gazebo to simulate the Turtlebot3 Burger platform. Help in loading the simulation can be found at http://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/. Make sure to go to the Gazebo part of the E-manual. When setting the model type, replace "${TB3_MODEL}" with "burger."

Create a ROS node that makes the Turtlebot3 travel forward at 1.5 m/s (remember this speed is much faster than it can do in real life) for 5 seconds, then turns to the right at 0.15 rad/s for 2 seconds, then continues forward at 1.5 m/s for another 5 seconds before stopping. Log the position data, and velocity & angular velocity commands.

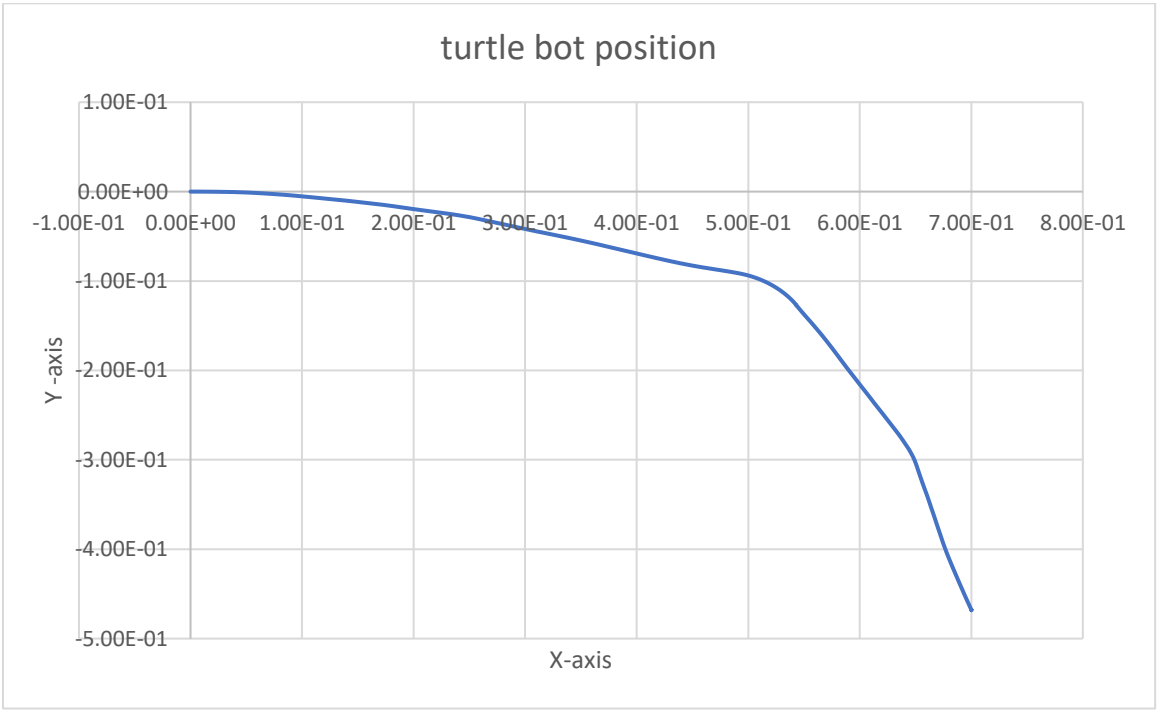
Create a subplot of the x & y position data, velocity & angular velocity commands versus time.

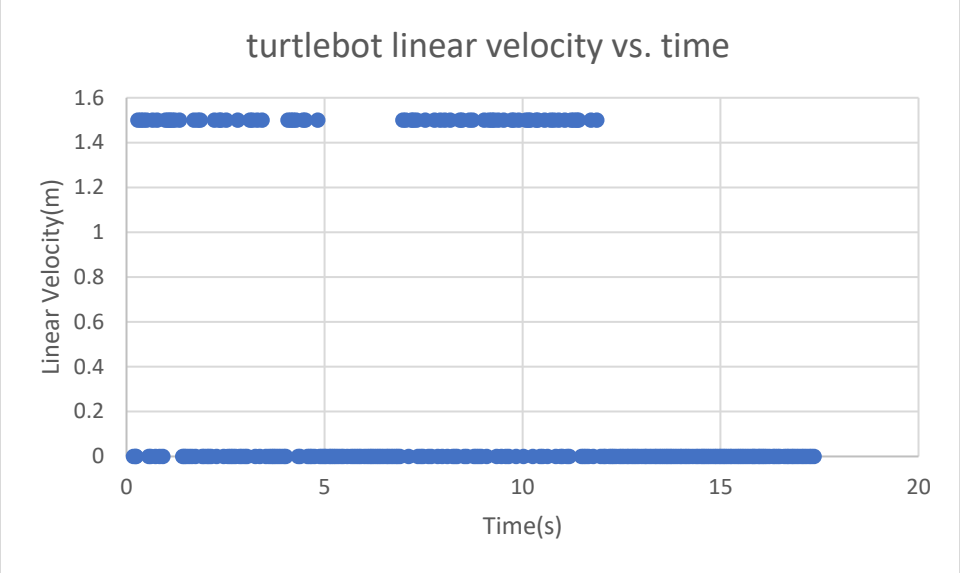
Submit your Python code.

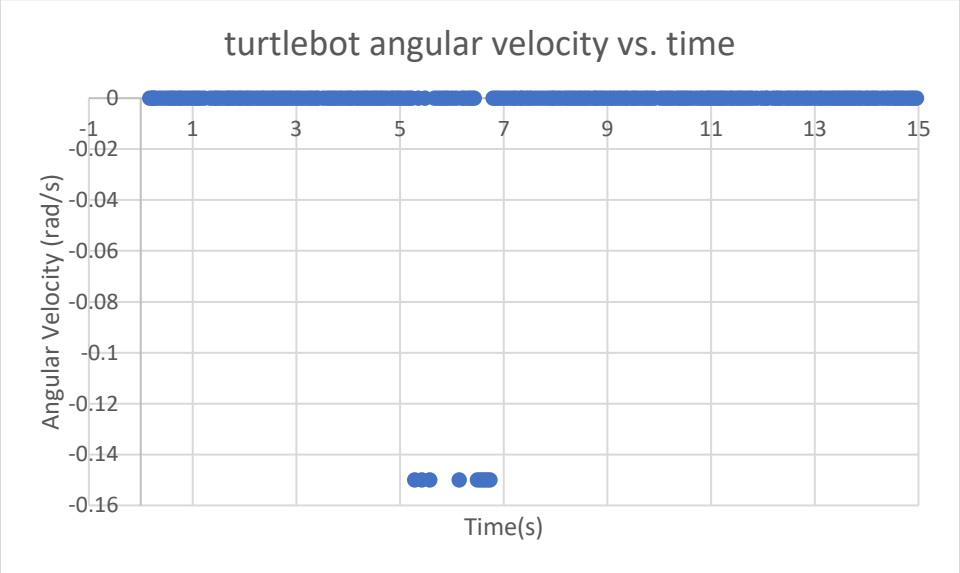
```python
#!/usr/bin/env python3

import rospy
import time
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
from tf.transformations import euler_from_quaternion,
quaternion_from_euler

class Turtlebot():
    def __init__(self):
        self.odom_x = None
        self.odom_y = None
        self.odom_z = None

        #declare publishers and subscribers
```

```
        self.odom_sub = rospy.Subscriber('odom', Odometry, self.odom_cb)
        self.vel_publisher = rospy.Publisher("cmd_vel", Twist,
queue_size=5)

    def odom_cb(self,msg):
        """recieve message of odom"""
        self.odom_x = msg.pose.pose.position.x
        self.odom_y = msg.pose.pose.position.y
        self.odom_z = msg.pose.pose.position.z

    def go_forward(self, speed):
        """commands the turtle to begin going forward at a certain
speed"""
        twist = Twist()
        twist.linear.x = speed
        self.vel_publisher.publish(twist)

    def go_turn(self, turn_speed):
        """commands the turtle to begin turning at a angular speed"""
        twist = Twist()
        twist.angular.z = turn_speed
        self.vel_publisher.publish(twist)
    """ def heading(self, msg):
        self.orientation = msg.pose.pose.orientation
        orientation_list = [self.orientation.x, self.orientation.y,
self.orientation.z]
        (roll, pitch, yaw) = euler_from_quaternion(orientation_list)
        self.vel_publisher.publish(roll, pitch, yaw) """
start_time = time.time()
stop_time_1 = 5
stop_time_2 = 5
turn_time = 2
speed = 1.5
if __name__ == '__main__':


    #initiate node
    rospy.init_node('turtlebot_go_python_class', anonymous=False)

    #control the rate of the script to recieve messages and or send
messages
    rate = rospy.Rate(20)

    #instantiate Turtlebot or create an object of class Turtlebot
    turtlebot = Turtlebot()

    #this is the main loop you can either use this or use rospy.spin
```

```
    while not rospy.is_shutdown():
        #print("turtle bot position", turtlebot.odom_x, turtlebot.odom_y,
turtlebot.odom_z)
        elapse_time = time.time() - start_time
        print("time is", elapse_time)

        if elapse_time >= 0 and elapse_time <= stop_time_1 :
            turtlebot.go_forward(1.5)
        if elapse_time >= stop_time_1 and elapse_time <= (stop_time_1 +
turn_time) :
            turtlebot.go_forward(0.0)
            turtlebot.go_turn(-.15)
        if elapse_time >= (stop_time_1 + turn_time) and elapse_time <=
(stop_time_1 + turn_time + stop_time_2) :
            turtlebot.go_turn(0.0)
            turtlebot.go_forward(1.5)
        else:
            turtlebot.go_forward(0.0)

    rate.sleep() # sleep at this controlled rate
```







Problem 4:

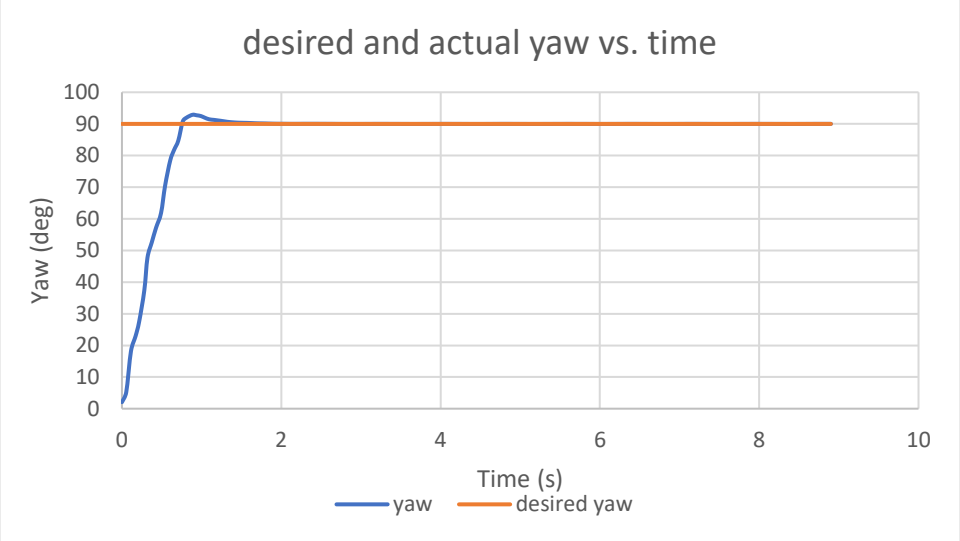
Create a ROS node/script that uses a feedback controller to control the heading of the Turtlebot3. Once you have adequately tuned the controller, collect the data (by writing to a log file) from a 90 degree step input (use a forward speed of 0.15 m/s).

Create a plot of the desired and actual heading versus time. What is the rise time, settling time, and percent overshoot of your controller?

Submit your Python code.


```
#!/usr/bin/env python3
import rospy
from nav_msgs.msg import Odometry
```

```
from tf.transformations import euler_from_quaternion
from geometry_msgs.msg import Twist
import math
import numpy as n
roll = pitch = yaw = 0.0
target_angle = 90
kP = 2.8

def get_rotation (msg):
    global roll, pitch, yaw
    orientation_q = msg.pose.pose.orientation
    orientation_list = [orientation_q.x, orientation_q.y, orientation_q.z,
orientation_q.w]
    (roll, pitch, yaw) = euler_from_quaternion (orientation_list)


rospy.init_node('my_quaternion_to_euler')

sub = rospy.Subscriber ('/odom', Odometry, get_rotation)
pub = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
command= Twist()
r = rospy.Rate(10)
while not rospy.is_shutdown():

    command.linear.x = .15
    lin_x = command.linear.x
    #print(lin_x)
    target_rad = target_angle * math.pi/180
    command.angular.z = kP  * (target_rad - yaw)
    turn_z = command.angular.z
    yaw_deg = n.rad2deg(yaw)
    print (yaw_deg,turn_z)
    pub.publish(command)
    r.sleep()
```



$$Rise\ Time = \ Time_{Yaw\ 90\%} - Time_{Yaw\ 10\%} = 0.65467\ sec - .05655\ sec = .59814\ sec$$

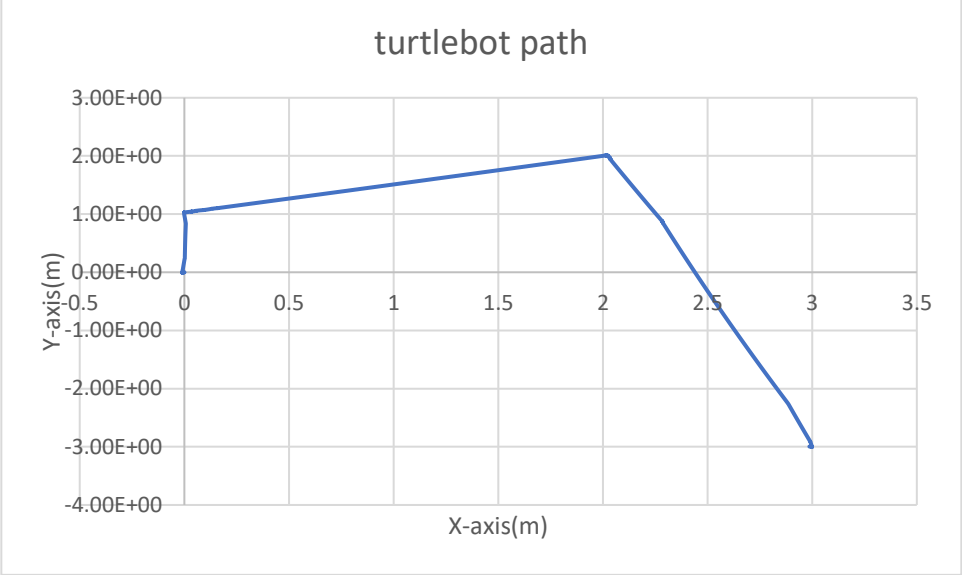
$$Settling\ Time = \ 2.0923192\ sec$$

$$\%OS = \frac{Yaw_{peak} - Yaw_{Steady\ State}}{Yaw_{Steady\ State}} \times 100\% = \frac{92.8829° - 90°}{90°} \times 100\% = 3.203\%$$

Problem 5:

Create a ROS node that follows the given set of prescribed waypoints: [0,0], [0,1], [2,2], [3, -3]. Start your robot at [0,0]. Create a plot of the X, Y coordinates to show how well your robot follows the desired path. Use a maximum translational speed of 0.15 m/s.

Submit your Python code.




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

import rospy
from nav_msgs.msg import Odometry
from tf.transformations import euler_from_quaternion
from geometry_msgs.msg import Point, Twist
from math import atan2

x = 0.0
y = 0.0
yaw = 0.0

def newOdom(msg):
    global x
    global y
    global yaw

    x = msg.pose.pose.position.x
    y = msg.pose.pose.position.y

    rot_q = msg.pose.pose.orientation
    (roll, pitch, yaw) = euler_from_quaternion([rot_q.x, rot_q.y, rot_q.z,
rot_q.w])

#rospy.init_node("speed_controller")
rospy.init_node("me_459_hw3_5")

sub = rospy.Subscriber("/odom", Odometry, newOdom)
pub = rospy.Publisher("/cmd_vel", Twist, queue_size = 1)

speed = Twist()
```

```
r = rospy.Rate(20)

#goal = Point()
#goal.x = 5
#goal.y = 5
end_point =[3.0,-3.0]
path_list =[[0.0,0.0],[0.0,1.0],[2.0,2.0],[3.0,-3.0]]
while not rospy.is_shutdown():
    #while abs(end_point[0] - x) < .15 and abs(end_point[1] - x) < .15:
    #    speed.linear.x(0.0)
    #    pub.publish(speed)
    #    print("you made it!")
    #    r.sleep()
    for point in path_list:
        while abs(point[0]-x) > .01 or abs(point[1]-y) > .01 :
            inc_x = point[0] - x
            inc_y = point[1] - y

            angle_to_goal = atan2(inc_y, inc_x)

            if (angle_to_goal - yaw) > 0.05:
                speed.linear.x = 0.0
                speed.angular.z = 0.2
            elif (angle_to_goal - yaw) < -0.05:
                speed.linear.x = 0.0
                speed.angular.z = -0.2

            else:
                speed.linear.x = 0.15
                speed.angular.z = 0.0

            pub.publish(speed)
            r.sleep()
    #    break

    #if abs(end_point[0] - inc_x) > .15 and abs(end_point[1] - inc_y) >
.15:
    #    speed.linear.x(0.0)
    #    pub.publish(speed)
    #    print("you made it!")
    #    break
```