Lab report for CISC 3060 Lab Assignment #1

Fall 2022

Jan C. Bierowiec

Bethany Fernandez and Mohammed Muntasir

1.0 Objective

The objective of lab assignment #1 is to develop and implement an algorithm that navigates the ROS turtlesim robot to a series of 2D waypoints, and to conduct an exploration of the effects of several parameters used in the algorithm.

This document is laid out as follows: Section 2 describes the algorithm in detail. Section 3 presents an experimental analysis of the accuracy with which the algorithm reaches waypoints for different values of the termination threshold parameters. Section 3 shows the experimental results that were obtained throughout the lab.

2.0 Design and Implementation of the waypoint algorithm

The group decided to install pandas on the apporto terminal, which is a python library that helps store and calculate data easily. This helped plotting data and viewing results when testing the different steps within the algorithm.

For Step 1 of the lab, an array, waypoints, was declared in the main program which stored values of the points the turtle sim would travel to. This array was not limited to any size, allowing the user to change the coordinates within the array. The turtle bot goes one point at a time. In this case, the turtle bot starts at (5,7) and then goes to the next point, (1,2), and the next and so on. This process terminates when the robot reaches the last point in the waypoints array. Figure 1 below shows the waypoints array within the algorithm.

```
#-----------------------------Main Program---------------------------------
if __name__ == ("__main__"):
    try:


        waypoints = [(5,7), (1,2), (3,2), (4,6), (5,3), (6,9), (7,4), (3,1), (5,8), (4,2)]
```

Figure 1: Waypoints

For Step 2 of the lab when calculating and maintaining a running estimate of the average accuracy of the waypoints, a function eu_dist was defined. eu_dist calculated the change in x and y in the distance traveled by the robot. Those values were then used to calculate the distance traveled by the robot which was returned in the function. This function can be seen in Figure 2.

```
def eu_dist(x1, y1, x2, y2):
    ''' finds the distance between two points.
        distance is equal to the square root of the
        change in x squared plus
        the change in y squared'''

    #Calculate Delta X and Y
    delx = x2 - x1 #change in x
    dely = y2 - y1 #change in y

    #Calculate Euclidean Distance
    dist = math.sqrt((delx ** 2) + (dely **2))

    return dist
```

Figure 2: This code portrays the eu_dist function which was used to calculate the distance the turtle bot traveled, which was then used to find the accuracy of the turtle bot reaching its coordinates set in the waypoints array.

Within the while loop before the main program, accuracy was defined as the points in the eu_dist function. This accuracy was then printed to the screen after each iteration of the turtle bot from point to point. This message lets the user know the accuracy of the turtle bot reaching a certain point. accuracy is defined in the while loop and as the program ran the velocity was also returned to obtain the velocity the turtle bot was traveling. This code can be seen in Figure 3 below.

```
    while not rospy.is_shutdown() and eu_dist(goalx, goaly, gLoc.x,
gLoc.y) > t:
        print(eu_dist(goalx, goaly, gLoc.x, gLoc.y))
        del_Theta = math.atan2(goaly - gLoc.y, goalx - gLoc.x) -
gLoc.theta

        msg.angular.z = a * del_Theta

        velocity = eu_dist(goalx, goaly, gLoc.x, gLoc.y)
        getVelList.append(velocity)

        msg.linear.x = linearGain * velocity

        pub.publish(msg)

        rate.sleep()

    accuracy = eu_dist(goalx, goaly, gLoc.x, gLoc.y)
    print('Done! dist:', accuracy)

    return accuracy, getVelList
```

Figure 3: While loop prints the accuracy of the points the turtle bot traveled to, and returns the accuracy and velocity of the turtle bot in the main program.

Lastly, in the main program an array term_threshold was then declared to store the values of the different termination thresholds in the while loop of goto_node. term_threshold holds the values of how long the turtle bot travels to each point. A for loop was created to print the reached waypoint of the turtle bot from the waypoints coordinates in the array. Once the reached point was printed onto the screen, the average accuracy of the turtle not reaching that point was calculated and printed. This portrayed the accuracy of each time the turtle completed one iteration from one point to another in the while loop. Once the program is completed, a graph is generated outputting the average accuracy against the termination threshold. Figure 4 portrays the code used to print the reached waypoints and the threshold accuracy of the turtle bot from point to point. The plot of the average accuracy and threshold terms can be found in the Experimental Analysis section in Figure 8.

```
87          term_threshold = [1, .5, .25, .1, .05, .01]
88          size  = len(waypoints)
89          acc = []
90          thresh_acc = []
91
92          '''STEP 2a: Add code to calculate and maintain a running
    estimate of the average
93              accuracy with which waypoints are being reached  when
    processing a waypoint
94              list. Test this with 10 waypoints.'''
95          '''STEP 2b: Vary the termination threshold in the while loop
    of the goto node using
96              the following values [1, 0.5, 0.25, 0.1, 0.05, 0.01'''
97
98          for t in term_threshold:
99              for p in waypoints:
100                 a, v = goto_node(p[0],p[1], 1.0, 4, 1.5, 10)
101                 acc.append(a)
102                 total = sum(acc)
103                 print('Reached Waypoint ', p)
104
105             avg_acc = total/size
106             thresh_acc.append(avg_acc)
107         print('avg accuracy: ', avg_acc)
108         time.sleep(2)
```

Figure 4: The code above defines the threshold terms for each turtle bot iteration from point to point, prints the reached waypoint of the robot and the average accuracy of the robot reaching that exact point.

For Step 3 of the lab, a new waypoints array was created called, wp2. wp2 stored two coordinate points, (5,7), and (1,2). term_threshold was used in a for loop and the goto_node was then called to loop the waypoints within the algorithm. This code works however generating the graph for this was unsuccessful. Figure 5 shows the code that was written for now.

```
123         '''STEP 3:Using a waypoint list with just 1 waypoint,
    generate a graph
124         of the velocities generated on each loop of the algorithm
    for each of the
125         loop termination values in Step 2.'''
126         wp2 = [(5,7),(1,2)]
127         df2 = pd.DataFrame()
128
129         for t in term_threshold:
130             i = 0
131             goto_node(wp2[0][0], wp2[0][1], t, 4, 1.5, 10)
132             a, v = goto_node(wp2[1][0], wp2[1][1], t, 4, 1.5, 10)
133             #df2.insert(loc = i, column = str(t), value = v)
134             i+= 1
135         #df2
```

Figure 5: Code for using one waypoint to generate a graph of the velocities generated from each loop of the algorithm.

For Step 4 of the lab, two waypoints were chosen in an array declared wp3. An array ang_gain then was created to analyze the angular gain of the turtlesim. In this case the angular gain points were chosen to be [3,4,5]. A for loop was created for the algorithm to reference the goto_node, and look at the angular gain of each point in wp3. A message 'Reached' was printed on the screen when the turtle bot reached each point and then moved onto the next.

```
137         '''STEP 4: Set the termination threshold value to 1.0, and
    conduct a series
138         of movements from one waypoint to a second and back, where
    you vary the angular
139         gain to the values [3,4,5], for each movement.'''
140         wp3 = [(1,2), (5,7)]
141         ang_gain = [3,4,5]
142         print('Starting Angular Gain Tests')
143         for a in ang_gain:
144             i = 0
145             ac, v = goto_node(wp3[0][0], wp3[0][1], 1.0, a, 1.5, 10)
146             print ('Reached', wp3[i])
147             i += 1
148             ac, v = goto_node(wp3[1][0], wp3[1][1], 1.0, a,1.5, 10)
149             print('Reached', wp3[i])
```

Figure 6: The code below shows the termination threshold value set to 1.0, with two waypoints, and an angular gain set to [3,4,5]. The message reached was printed when the turtle bot reached each point in the wp3 waypoints array.

For Step 5 an array linear_gain was declared. linear_gain had the values of [0.5, 1.5, 3]. Then goto_node was called and the waypoints of wp3 were referenced for the linear velocity. In Figure 7 the upper half of the code portrays this. Once the linear_gain and waypoints in wp3 went through, step 6 began. For Step 6 an array rate was declared. rate possessed the values, [2,5,10,20]. The termination threshold value was set to equal 1, the linear velocity to 1.5, and the angular velocity gain to 4.0. These values were referenced in the goto_node function that was called. Once the function ran in the for loop the algorithm was completed.

```
151         '''STEP 5: Repeat the experiment varying the linear
    velocity[0.5, 1.5, 3]'''
152         linear_gain = [0.5, 1.5, 3]
153
154         print('Testing Linear Gain')
155         for l in linear_gain:
156             goto_node(wp3[0][0], wp3[0][1], 1.0, 4.0, l,10)
157             ac, v = goto_node(wp3[1][0], wp3[1][1], 1.0, 4.0, l, 10)
158
159         '''STEP 6: Repeat the experiment varying the sample rate.'''
160         rate = [2,5,10,20]
161
162         print('Testing Rate')
163         for r in rate:
164             goto_node(wp3[0][0], wp3[0][1], 1.0, 4.0, 1.5, r)
165             ac, v = goto_node(wp3[1][0], wp3[1][1], 1.0, 4.0, 1.5, r)
166
167     except rospy.ROSInterruptException:
168         pass
```

Figure 7: Code for step 5 and 6 respectfully. Two arrays linear_gain and rate were declared. linear_gain calculated the linear gain of the turtle bot, and rate varied the rate of the turtle bot going from point to point.

The algorithm works for the most part. One graph was produced. Some of the code may also be broken, however the idea of what is going on in the code is there. The lab is not complete due to time constraints, but for the most part, the program algorithm works resulting in some observations made in the turtle bot's behavior. Observations of this behavior are in the section below. The full code to the algorithm program is in the Appendix below.

3.0 Experimental Analysis of Termination Threshold on Accuracy

Based on the graph produced analyzing the termination threshold of the turtlesim on the accuracy, a downward slope can be observed. The smaller the termination threshold, the higher the average accuracy, the greater the termination the smaller the average accuracy. This makes sense because as the turtle bot traveled from point to point very quickly and then quickly went to the next point, the turtle bot did not focus on getting to the point accurately but somewhere around where it was supposed to go. This is in the case of a high termination threshold and a low average accuracy. The accuracy is low with a high termination threshold, the robot performs the iteration quickly, which can lead to discrepancies along the way causing a lack or precise accuracy. As the turtle bot started to have a lower termination threshold, the average accuracy of the turtle bot became more accurate. This makes sense because with a smaller termination threshold, the turtle bot took longer to get from point to point. Since the turtle bot took longer to get from point to point the turtle bot was able to follow a more uniform path, and would reach its precise accurate point. Based on this observation it would make sense to construct robots that have a termination threshold that would be low, but would provide a high accuracy. Of course this would mean that the robot would be slower. Alternatively, the average of the termination threshold can be taken and the average of the average accuracy to define a middle ground. For robots to be somewhat accurate and not take too long going from place to place.

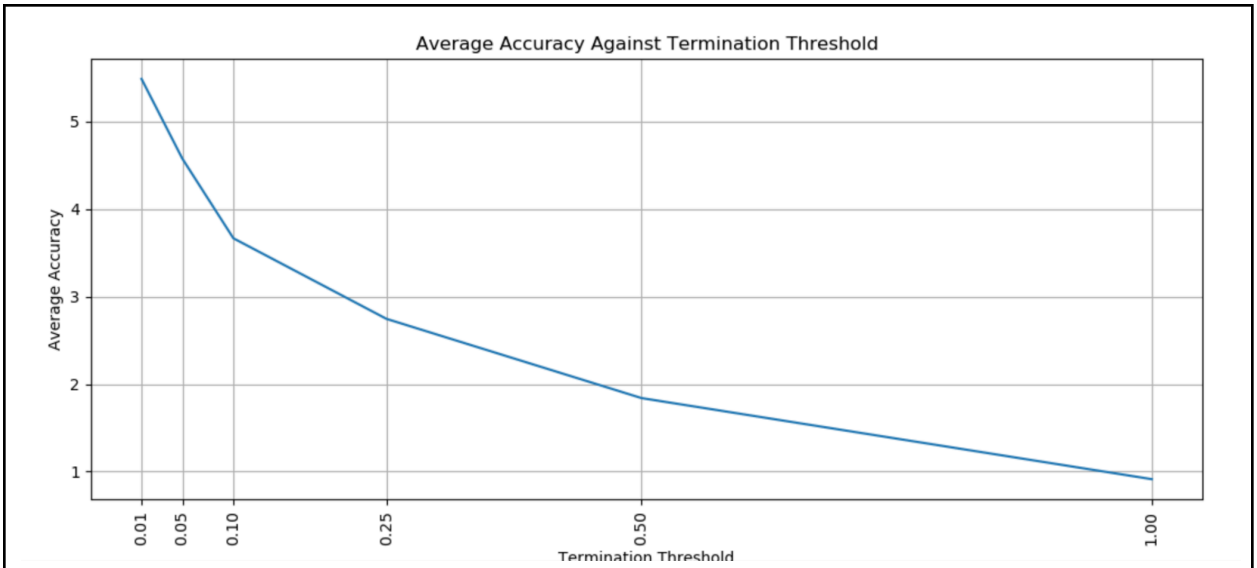


Figure 8: This graph represents the Average Accuracy vs Termination Threshold. The lower the termination threshold, the higher the average accuracy, the higher the termination threshold, the lower the average accuracy.

The average velocity of the thresholds was also calculated in the algorithm. The velocity shows a distance vs. velocity graph with the different colors on the graph representing the different

threshold values. As can be observed on the graph, despite the differences in the average accuracy and termination threshold, when comparing the distance and velocity of the turtle bot there is no significant change. This makes sense because as the turtle bot ran in the program, it maintained its track, with little adjustments. The path the turtle bot followed was ordered not chaotic. What is also worth noting is that towards the beginning of the turtle bot's movement the differences in thresholds can be observed. What can also be noted is that the turtle bot started out with a large velocity for a small distance, however as the distance increased, the turtle bot slowed down until the program ended at a single point, where all the terminal thresholds come together. In this case this number is about 47. This graph can be observed in Figure 9 below.

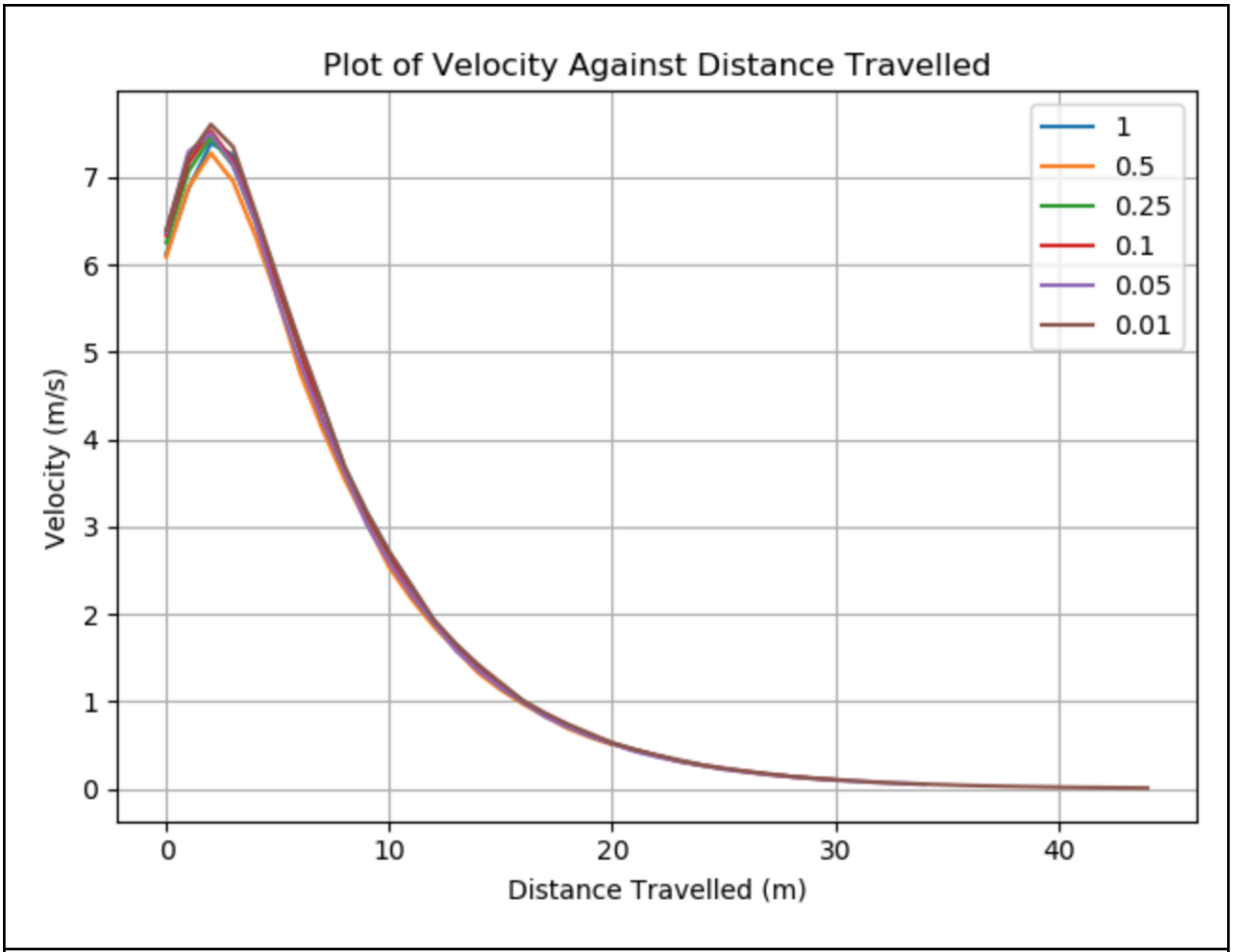
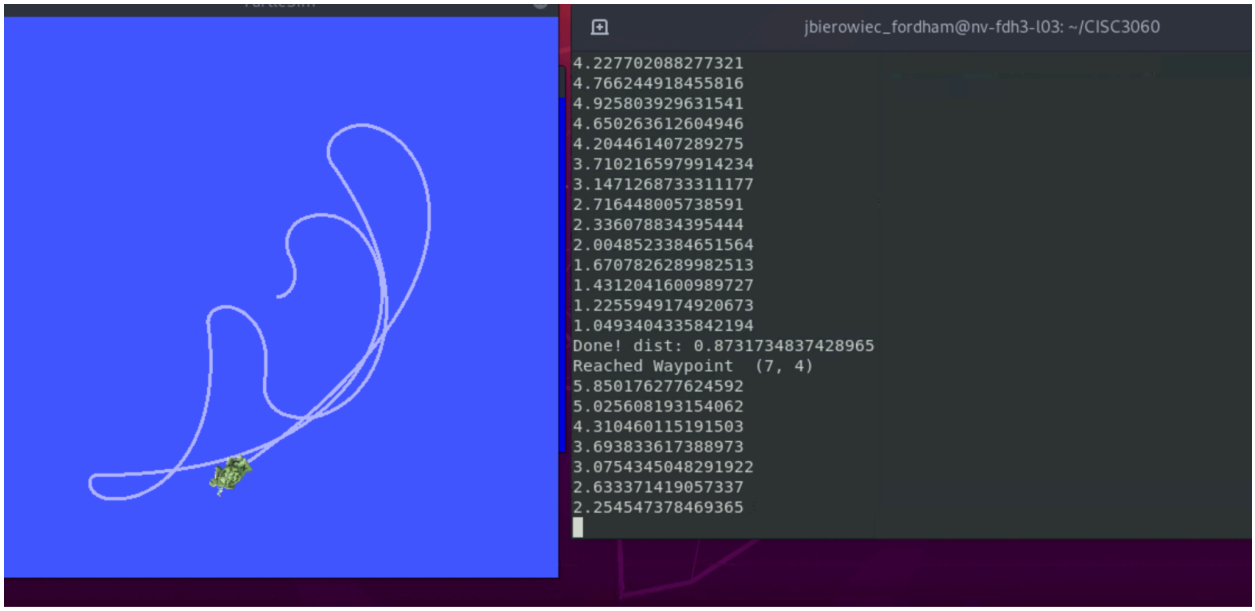
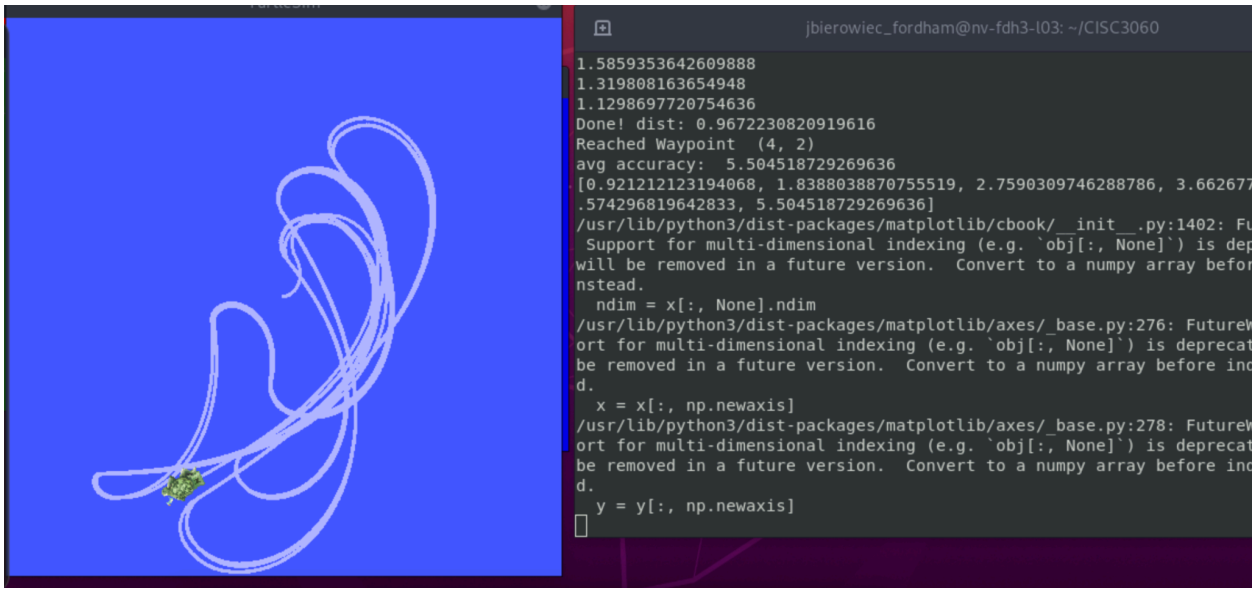
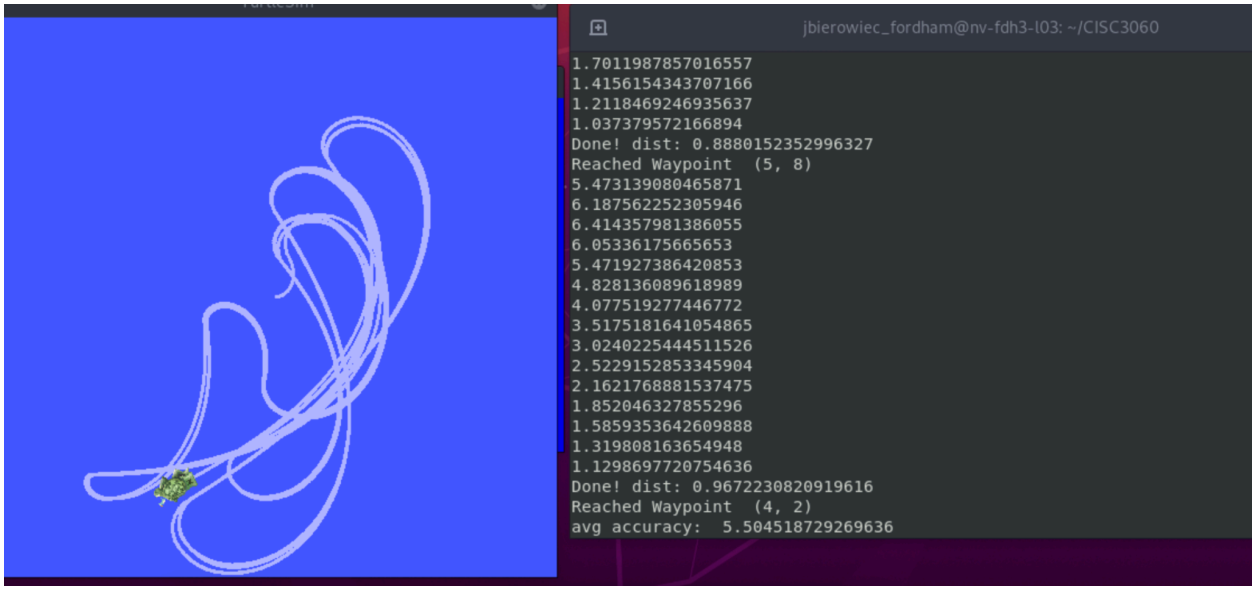


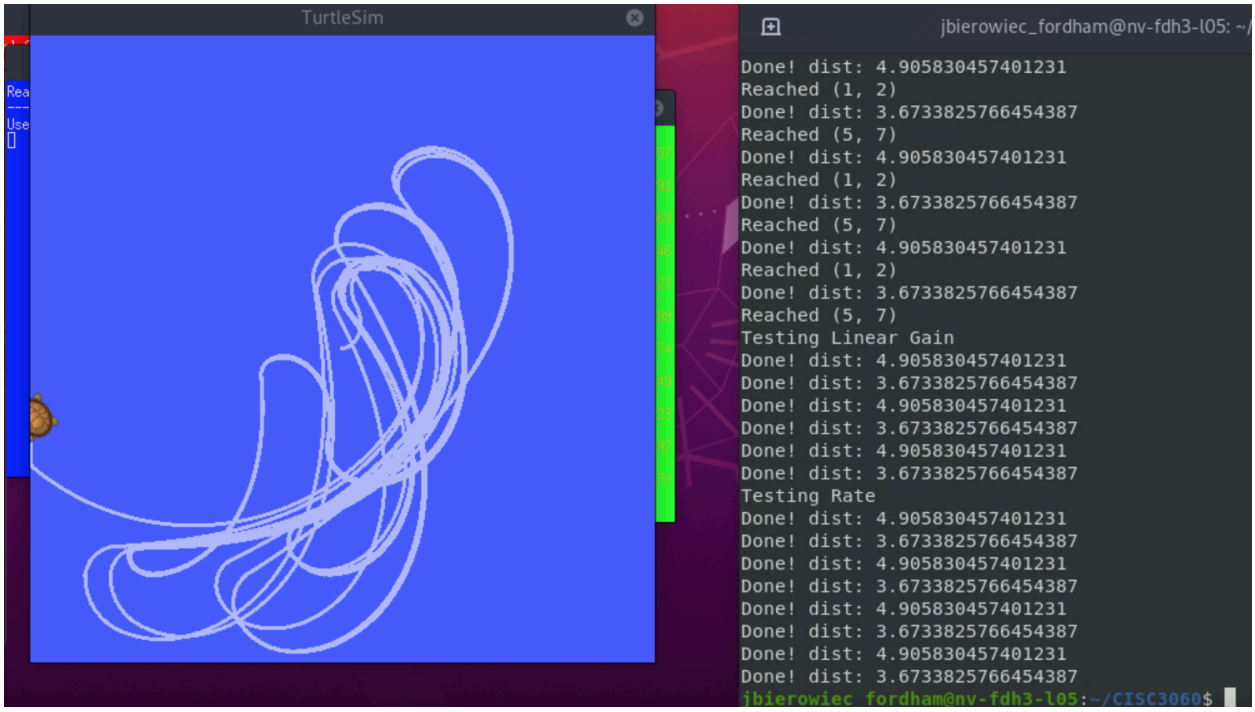
Figure 9: This graph portrays the distance traveled by the turtle bot and the velocity it had when reaching each point. The different colors on the graph represent the different threshold points the turtle bot took to get from point to point. As the graph shows, despite the difference in threshold points, the turtle bot maintained the same curvature for all cases.

4.0 References

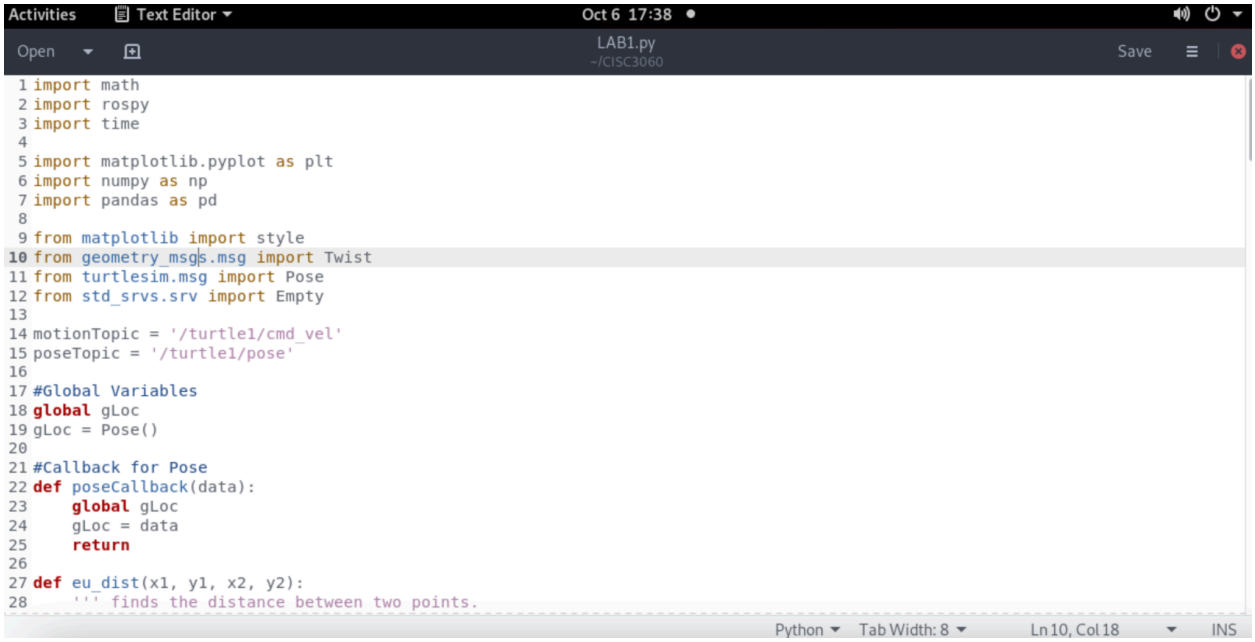
1. 

2. 

3. 
jbierowiec_fordham@nv-fdh3-l03: ~/CISC3060
1.7011987857016557
1.4156154343707166
1.2118469246935637
1.037379572166894
Done! dist: 0.8880152352996327
Reached Waypoint (5, 8)
5.473139080465871
6.187562252305946
6.414357981386055
6.05336175665653
5.471927386420853
4.828136089618989
4.077519277446772
3.5175181641054865
3.0240225444511526
2.5229152853345904
2.1621768881537475
1.852046327855296
1.5859353642609888
1.319808163654948
1.1298697720754636
Done! dist: 0.9672230820919616
Reached Waypoint (4, 2)
avg accuracy: 5.504518729269636

4. 
TurtleSim
jbierowiec_fordham@nv-fdh3-l05: ~/
Done! dist: 4.905830457401231
Reached (1, 2)
Done! dist: 3.6733825766454387
Reached (5, 7)
Done! dist: 4.905830457401231
Reached (1, 2)
Done! dist: 3.6733825766454387
Reached (5, 7)
Done! dist: 4.905830457401231
Reached (1, 2)
Done! dist: 3.6733825766454387
Reached (5, 7)
Testing Linear Gain
Done! dist: 4.905830457401231
Done! dist: 3.6733825766454387
Done! dist: 4.905830457401231
Done! dist: 3.6733825766454387
Done! dist: 4.905830457401231
Done! dist: 3.6733825766454387
Testing Rate
Done! dist: 4.905830457401231
Done! dist: 3.6733825766454387
Done! dist: 4.905830457401231
Done! dist: 3.6733825766454387
Done! dist: 4.905830457401231
Done! dist: 3.6733825766454387
Done! dist: 4.905830457401231
Done! dist: 3.6733825766454387
jbierowiec_fordham@nv-fdh3-l05:~/CISC3060$

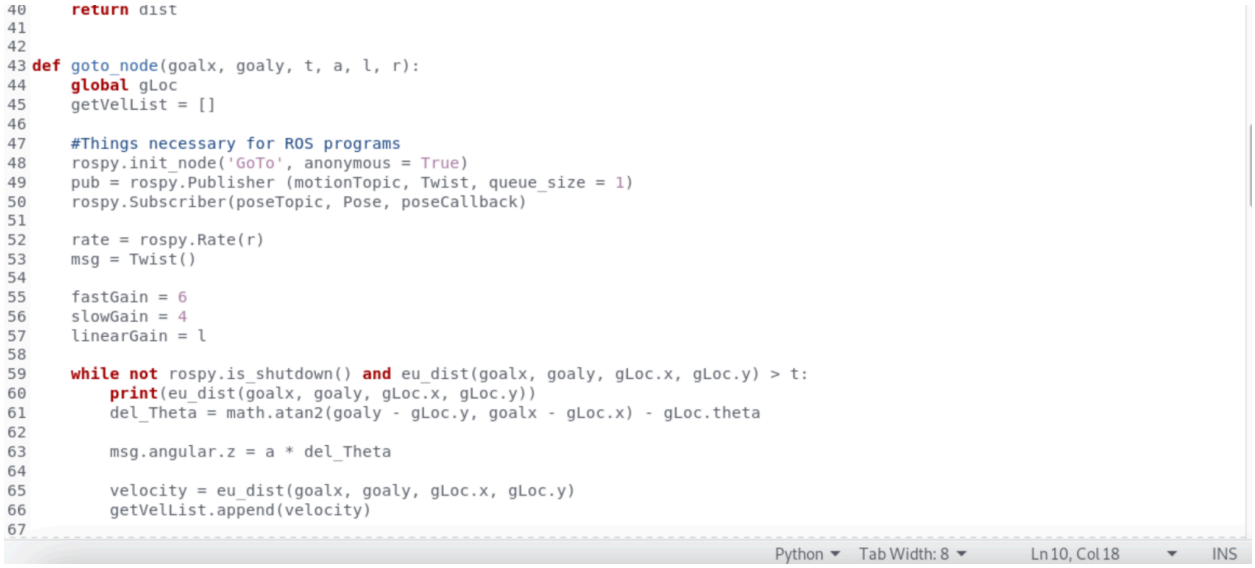
5.


Activities
Text Editor
Oct 6 17:38
Open
LAB1.py
~/CISC3060
Save
1 import math
2 import rospy
3 import time
4
5 import matplotlib.pyplot as plt
6 import numpy as np
7 import pandas as pd
8
9 from matplotlib import style
10 from geometry_msgs.msg import Twist
11 from turtlesim.msg import Pose
12 from std_srvs.srv import Empty
13
14 motionTopic = '/turtle1/cmd_vel'
15 poseTopic = '/turtle1/pose'
16
17 #Global Variables
18 global gLoc
19 gLoc = Pose()
20
21 #Callback for Pose
22 def poseCallback(data):
23 global gLoc
24 gLoc = data
25 return
26
27 def eu_dist(x1, y1, x2, y2):
28 ''' finds the distance between two points.
Python
Tab Width: 8
Ln 10, Col 18
INS

6.


27 def eu_dist(x1, y1, x2, y2):
28 ''' finds the distance between two points.
29 distance is equal to the square root of the
30 change in x squared plus
31 the change in y squared'''
32
33 #Calculate Delta X and Y
34 delx = x2 - x1 #change in x
35 dely = y2 - y1 #change in y
36
37 #Calculate Euclidean Distance
38 dist = math.sqrt((delx ** 2) + (dely **2))
39
40 return dist
41
42
43 def goto_node(goalx, goaly, t, a, l, r):
44 global gLoc
45 getVelList = []
46
47 #Things necessary for ROS programs
48 rospy.init_node('GoTo', anonymous = True)
49 pub = rospy.Publisher (motionTopic, Twist, queue_size = 1)
Python
Tab Width: 8
Ln 10, Col 18
INS

7.


40 return dist
41
42
43 def goto_node(goalx, goaly, t, a, l, r):
44 global gLoc
45 getVelList = []
46
47 #Things necessary for ROS programs
48 rospy.init_node('GoTo', anonymous = True)
49 pub = rospy.Publisher (motionTopic, Twist, queue_size = 1)
50 rospy.Subscriber(poseTopic, Pose, poseCallback)
51
52 rate = rospy.Rate(r)
53 msg = Twist()
54
55 fastGain = 6
56 slowGain = 4
57 linearGain = l
58
59 while not rospy.is_shutdown() and eu_dist(goalx, goaly, gLoc.x, gLoc.y) > t:
60 print(eu_dist(goalx, goaly, gLoc.x, gLoc.y))
61 del_Theta = math.atan2(goaly - gLoc.y, goalx - gLoc.x) - gLoc.theta
62
63 msg.angular.z = a * del_Theta
64
65 velocity = eu_dist(goalx, goaly, gLoc.x, gLoc.y)
66 getVelList.append(velocity)
67
Python
Tab Width: 8
Ln 10, Col 18
INS

8.

```
    fastGain = 6
    slowGain = 4
    linearGain = l

    while not rospy.is_shutdown() and eu_dist(goalx, goaly, gLoc.x, gLoc.y) > t:
        print(eu_dist(goalx, goaly, gLoc.x, gLoc.y))
        del_Theta = math.atan2(goaly - gLoc.y, goalx - gLoc.x) - gLoc.theta

        msg.angular.z = a * del_Theta

        velocity = eu_dist(goalx, goaly, gLoc.x, gLoc.y)
        getVelList.append(velocity)

        msg.linear.x = linearGain * velocity

        pub.publish(msg)

        rate.sleep()

    accuracy = eu_dist(goalx, goaly, gLoc.x, gLoc.y)
    print('Done! dist:', accuracy)

    return accuracy, getVelList


#----------------------------Main Program--------------------------------
```

Python ▾ Tab Width: 8 ▾ Ln 10, Col 18 ▾ INS

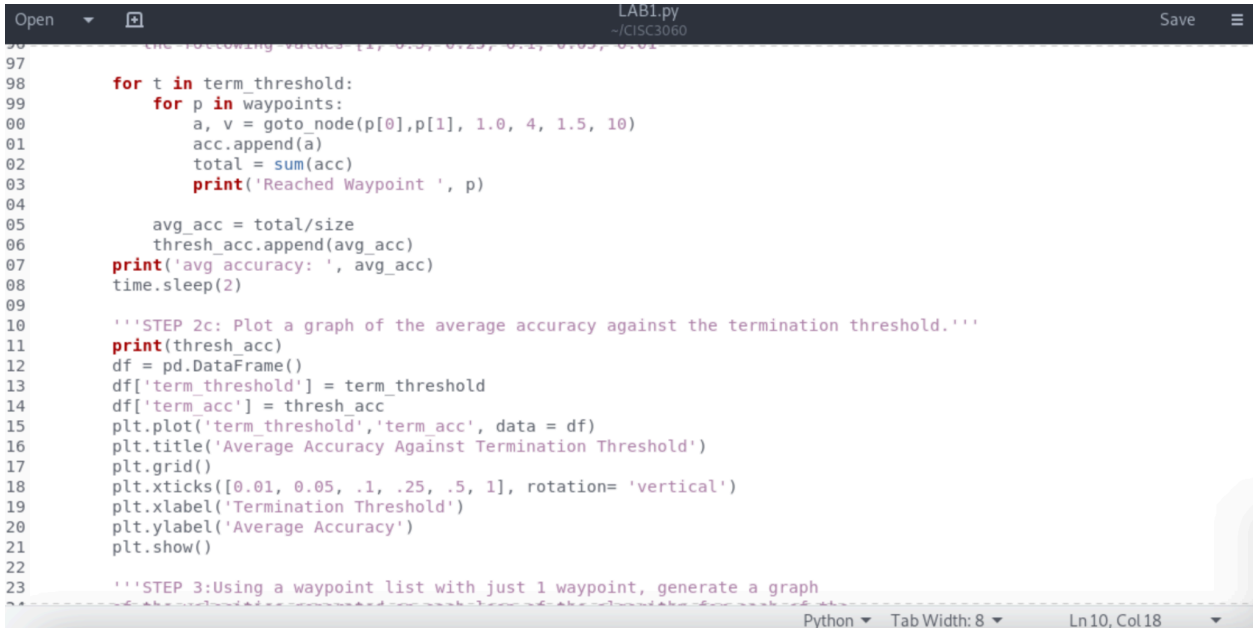
9.

Open ▾ LAB1.py ~/CISC3060 Save ☰

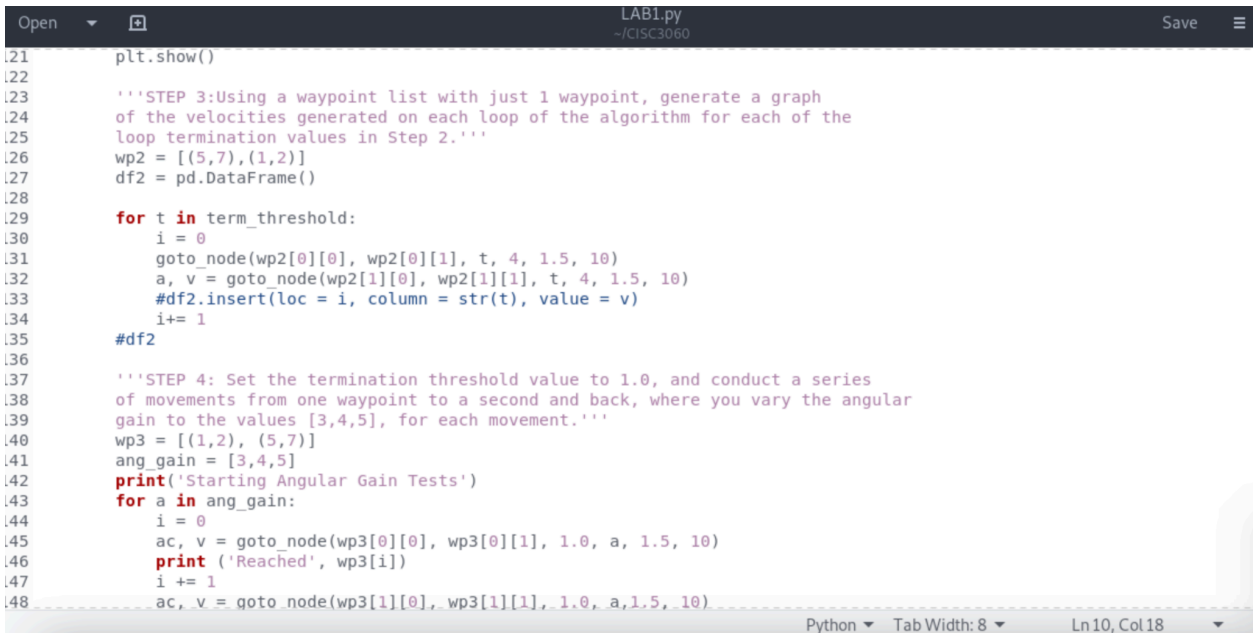
```
#----------------------------Main Program--------------------------------
if __name__ == ("__main__"):
    try:


        waypoints = [(5,7), (1,2), (3,2), (4,6), (5,3), (6,9), (7,4), (3,1), (5,8), (4,2)]
        term_threshold = [1, .5, .25, .1, .05, .01]
        size  = len(waypoints)
        acc = []
        thresh_acc = []

        '''STEP 2a: Add code to calculate and maintain a running estimate of the average
           accuracy with which waypoints are being reached  when processing a waypoint
           list. Test this with 10 waypoints.'''
        '''STEP 2b: Vary the termination threshold in the while loop of the goto node using
           the following values [1, 0.5, 0.25, 0.1, 0.05, 0.01'''

        for t in term_threshold:
            for p in waypoints:
                a, v = goto_node(p[0],p[1], 1.0, 4, 1.5, 10)
                acc.append(a)
                total = sum(acc)
                print('Reached Waypoint ', p)

            avg_acc = total/size
            thresh_acc.append(avg_acc)
```

Python ▾ Tab Width: 8 ▾ Ln 10, Col 18 ▾

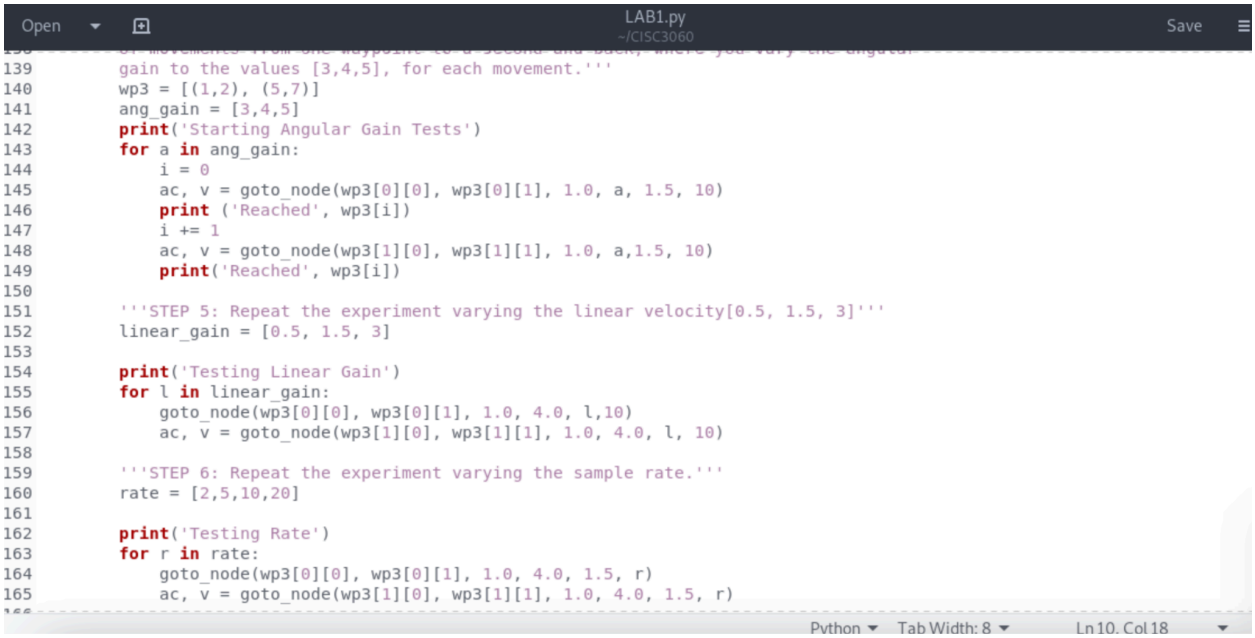
10.


Open
LAB1.py
~/CISC3060
Save
for t in term_threshold:
for p in waypoints:
a, v = goto_node(p[0],p[1], 1.0, 4, 1.5, 10)
acc.append(a)
total = sum(acc)
print('Reached Waypoint ', p)
avg_acc = total/size
thresh_acc.append(avg_acc)
print('avg accuracy: ', avg_acc)
time.sleep(2)
'''STEP 2c: Plot a graph of the average accuracy against the termination threshold.'''
print(thresh_acc)
df = pd.DataFrame()
df['term_threshold'] = term_threshold
df['term_acc'] = thresh_acc
plt.plot('term_threshold','term_acc', data = df)
plt.title('Average Accuracy Against Termination Threshold')
plt.grid()
plt.xticks([0.01, 0.05, .1, .25, .5, 1], rotation= 'vertical')
plt.xlabel('Termination Threshold')
plt.ylabel('Average Accuracy')
plt.show()
'''STEP 3:Using a waypoint list with just 1 waypoint, generate a graph
Python
Tab Width: 8
Ln 10, Col 18

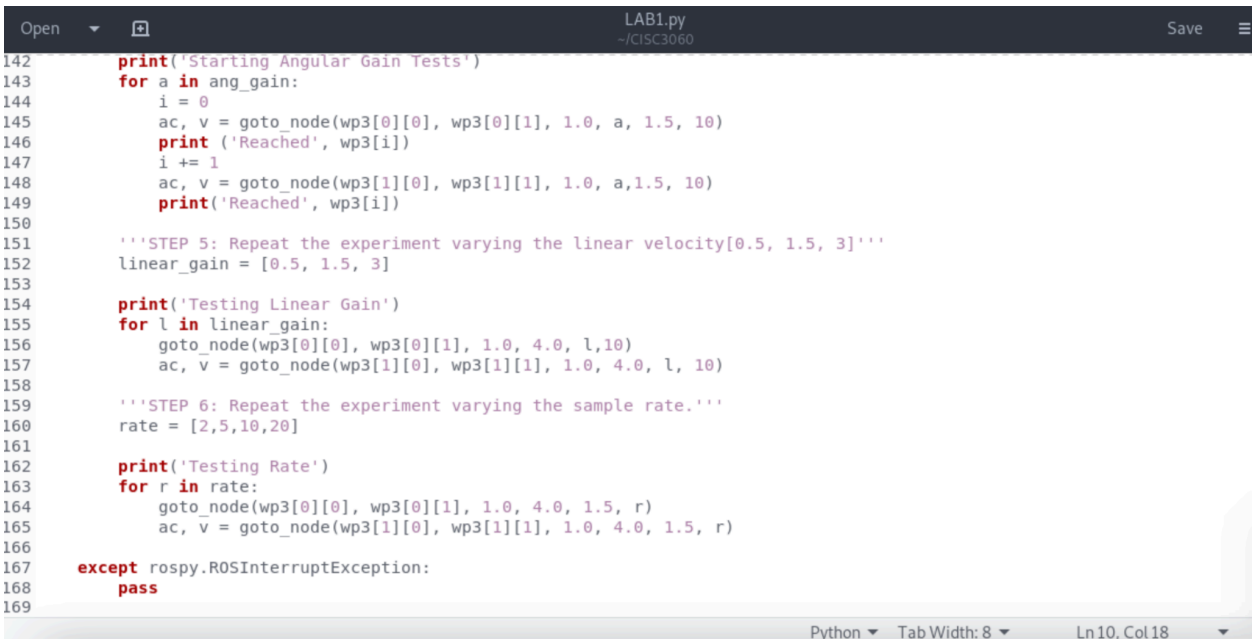
11.


Open
LAB1.py
~/CISC3060
Save
plt.show()
'''STEP 3:Using a waypoint list with just 1 waypoint, generate a graph
of the velocities generated on each loop of the algorithm for each of the
loop termination values in Step 2.'''
wp2 = [(5,7),(1,2)]
df2 = pd.DataFrame()
for t in term_threshold:
i = 0
goto_node(wp2[0][0], wp2[0][1], t, 4, 1.5, 10)
a, v = goto_node(wp2[1][0], wp2[1][1], t, 4, 1.5, 10)
#df2.insert(loc = i, column = str(t), value = v)
i+= 1
#df2
'''STEP 4: Set the termination threshold value to 1.0, and conduct a series
of movements from one waypoint to a second and back, where you vary the angular
gain to the values [3,4,5], for each movement.'''
wp3 = [(1,2), (5,7)]
ang_gain = [3,4,5]
print('Starting Angular Gain Tests')
for a in ang_gain:
i = 0
ac, v = goto_node(wp3[0][0], wp3[0][1], 1.0, a, 1.5, 10)
print ('Reached', wp3[i])
i += 1
ac, v = goto_node(wp3[1][0], wp3[1][1], 1.0, a,1.5, 10)
Python
Tab Width: 8
Ln 10, Col 18

12.



13.



Images of the code, and turtle bot

5.0 Summary

Although the program may have been flawed and not everything is correct, a general understanding of turtle bot movement can be made. As the program started, the turtle bot had a high velocity, as the turtle bot started traveling farther distances the velocity slowed and all the termination thresholds came together to form one line. In the earlier graph the average accuracy was compared to the termination thresholds. Here it was observed that the larger

threshold a turtle bot had, the lower the average accuracy, and vise versa. Code snippets were provided in the Appendix and in the explanation of the code, and although not everything ran well and correctly, these were the observations made during the lab.