Lab report for CISC 3060 Lab Assignment #2

Fall 2022

Jan C. Bierowiec

Bethany Fernandez and Mohammed Muntasir

1.0 Objective

The objective of lab assignment #2 was to develop and create a program allowing a ROS robot to rotate about itself 360 degrees at its starting point at an angular velocity set to be 0.1. As the robot would rotate, its angle would be stored in an array called H in the code, creating a bar graph as a result. Once that is done, the next step is to see how the robot would use this program and record the angles stored in the array H at all four corners and at the middle of the enclosed world simulation. This would all be done when pMax would equal 360. Then once pMax would be changed to 18, different results of the graphs would output onto the screen. Lastly a function was created which would return the index into H reflecting the furthest open space around the robot. This function would cause the robot to rotate to its heading, and move forward for 0.5 seconds at a speed of 0.5m/s. Section 2 of the report discusses the implementation of the code in the program to make the robot carry out the four steps. Section 3 goes over the analysis of the graphs produced in the program. Section 4 is the appendix of the report containing all the pictures of the code and the running robot in the enclosed world simulation. Lastly Section 5 is a summary of the results obtained from the lab.

2.0 Design and Implementation the Robot Rotation Algorithm

2.1 Step 1 of the code

To have the robot rotate about itself at an angular velocity of 0.1, some code was changed to the original Spin.py program. In Figures 1 and 2, we can see how pi and the angular velocity is added to the program. The purpose of doing so is for the robot to rotate from 0 to 360 degrees. The fixed angular velocity set to 0.1 determines the rate of rotation of the robot. It can also be noticed that when the robot rotates, it is offset by a value of 0.2. This value is in radians and the purpose of adding this value into the program is for the robot to start at a very small angle as opposed to absolute 0, which is equivalent to 360 or 2pi. This offset is therefore essential in the program because this is what will allow the robot to rotate in the first place.

```
96    # the initial position of the robot is not where we are
97    # as the robot rotates the initial position is going to increase
98    # the inital angle when the robot rotates is offset by -0.2 radians
99    initPosition = gPose[2] + math.pi - 0.2
100   print(abs(gPose[2] + math.pi - initPosition))
```

Figure 1: Code that implements the addition pi for the robot to rotate from 0 to 2pi, along with the 0.2 radians offset to start the rotation of the robot

```
105   while not rospy.is_shutdown() and abs(gPose[2] + (math.pi) - initPosition) > accuracy:
106       msg.linear.x,msg.angular.z=0.0,0.1  #angular velocity is a constant value, and it is set to 0.1
107       vel_pub.publish(msg)
```

Figure 2: Code that implements the angular velocity to be 0.1, to ensure a constant rate of rotation of the robot

Once this was completed, for each step of the rotation, gFrontDistance was added to an array H, which indexed the angle in degrees rotated of the robot. Pi was then added to the gPose[2] orientation in order to ensure that the robot would rotate under a positive orientation. Without the addition of pi, the robot would rotate about itself from negative pi to positive pi. With the addition of pi, the robot now rotates from 0 to 2pi. Once the robot would complete a rotation, it would sleep for a little bit in order to then in step 2 of the lab move the robot towards the four corners and center to give the robot time to start rotating again from those respective coordinates. This implementation of the code can be seen in Figure 3 below.

```
109       # the H array is used as a list here for plotting the gFrontDistance
110       # we add pi in the program to keep the robot in range from 0 to 2pi
111       # this is to ensure the robot has a positive orientation
112       # not to rotate from -pi to +pi
113       degree = int(math.degrees(gPose[2] + math.pi))
114       H[degree] = gFrontDistance
115
116       rate.sleep()
```

Figure 3: degree is declared to be an integer in degrees with the addition of gPose[2] to pi, resulting in a positive orientation of the robot from 0 to 2pi. The array H then passes in degrees which outputs the gFrontDistance.

In order to produce a graph after the 360 rotation of the robot, the following code portrayed in Figure 4 was implemented. The code looks at the arrays H and h, which plot two bar graphs of the laser distance of the robot before reflecting back to the robot, which is in meters, against

the angle of rotation by the robot. The array h, plots a small bar graph showing the peaks of where the rotation angle of the robot has the most significant laser distance covered, whereas the plot H demonstrates the angle of rotation of the robot and laser distance in general during the 360 degree rotation of the robot. The bar graph plots are located in Section 3 of the report.

```
128    # code to plot the small bar graph of the angles
129    plt.bar(a, h)
130    plt.xlabel("Angle of Rotation in Degrees")
131    plt.ylabel("Laser Distance Before Reflecting (m)")
132    plt.title("Compressed Bar Graph of Laser Range Distances at a Rate of 0.1")
133    plt.ylim(0)
134    plt.show()
135
136    # code to show the bar graph in more detail
137    plt.bar(range(len(H)),H)
138    plt.ylim(0)
139    plt.xlabel("Angle of Rotation in Degrees")
140    plt.ylabel("Laser Distance Before Reflecting (m)")
141    plt.title("Bar Graph of Laser Range Distances at a Rate of 0.1")
142    plt.show()
143
144    # outputs a message of where the spin of the robot was completed
145    print ("Spin complete at ",gPose[2])
146    msg.linear.x,msg.angular.z=0.0,0.0 # stop now
147    vel_pub.publish(msg)
148
149    return
```

Figure 4: Code for plotting the bar graphs for the rotation of the robot and the laser distance before reflection in meters.

pMax for the rotation of the robot is defined in Figure 5. This code contains array H multiplied by 360, to store each angle the robot rotates as it completes the 360 rotation. The rate in Hertz at which the robot initially rotates is set to 10 on line 92 in the code. The while loop from lines 105 to 116 is where the gFrontDistance is added to each degree change of the robot during the rotation. Lastly, small h is the array that will output the bar graph comparing the angle the robot rotates and the laser distance before reflection. Figure 6 on the other hand references pMax and sets it to 8 within the main program.

```
def spin_node(pMax):
    '''Spins the robot 360 degrees'''
    rospy.init_node('Spin_Node', anonymous=True)

    global H # the H array is created to be a gloal variable
    H = [0] * 360 # a bar graph plot is initialized for each degree of the robot's rotation
    # register publisher and two subscribers
    vel_pub = rospy.Publisher(motionTopic, Twist, queue_size=0)
    scan_sub = rospy.Subscriber(laserTopic, LaserScan, callback_laser)
    pose_sub = rospy.Subscriber(poseTopic, Odometry, callback_pose)
    rospy.sleep(1) # allow callbacks to happen before proceeding

    rate = rospy.Rate(10) # Hz for main loop frequency
    msg = Twist() # empty new velocity message
    accuracy = 0.1 # radians

    # the initial position of the robot is not where we are
    # as the robot rotates the initial position is going to increase
    # the inital angle when the robot rotates is offset by -0.2 radians
    initPosition = gPose[2] + math.pi - 0.2
    print(abs(gPose[2] + math.pi - initPosition))

    '''The Condition for the while loop:
       gPose[2]'s angle within the 0 - 2pi range (gPose[2] + pi) must be
       less than the termination threshold of .1 radians'''
    while not rospy.is_shutdown() and abs(gPose[2] + (math.pi) - initPosition) > accuracy:
        msg.linear.x,msg.angular.z=0.0,0.1  #angular velocity is a constant value, and it is set to 0.1
        vel_pub.publish(msg)

        # the H array is used as a list here for plotting the gFrontDistance
        # we add pi in the program to keep the robot in range from 0 to 2pi
        # this is to ensure the robot has a positive orientation
        # not to rotate from -pi to +pi
        degree = int(math.degrees(gPose[2] + math.pi))
        H[degree] = gFrontDistance

        rate.sleep()

    # small h is an array multiplied by the pMax
    h = [0] * pMax
    # aMult is an integer of 360 divided by the pMax
    # this will create the difference in the graphs
    aMult = int(360/pMax)
    a = [0] * pMax
    for i in range(pMax):
        h[i] = H[i * aMult]
        a[i] = i * aMult
```

Figure 5: Definition of pMax which is used as an index to the array H which will be used to create a bar graph portraying the rotation of the robot in relation to the laser reflection in meters.

```
165
166 #------------------------------MAIN  program---------------------
167 if __name__ == '__main__':
168     try:
169         rospy.on_shutdown(callback_shutdown)
170         for r in (0,4):
171             spin_node(8)
172             # spin_node(16)
173     except rospy.ROSInterruptException:
174         pass
```

Figure 6: Here spin_node, which defines pMax holds 8, setting the index of the array to hold 8, which will then be graphed once the robot completes a 360 degree rotation.

Lastly, this process was repeated once more, however this time at a rate of 0.1 Hz By definition a Hertz is 1/s. This means that repeating this process with 0.1 Hz would take 10 seconds since 1/0.1 = 10. This code can be seen in Figure 7 below.

```
92     rate = rospy.Rate(10) # Hz for main loop frequency
93     msg = Twist() # empty new velocity message
94     accuracy = 0.1 # radians
```

Figure 7: Code snippet that holds the 0.1 Hz rate for the rotation of the robot, which will be 10 seconds.

Step 2 of the code

For step two of the lab, using the ROS enclosed.world simulation world, the robot was moved dragging the mouse across the simulation to all of the four corners of the simulation, as well as the middle of the simulation, by the middle wall. Then the program for the robot to rotate and register the angles was run, and different graphs were outputted everytime the robot rotated around where it was placed. The appearance of each graph can be seen in Section 3, followed by an analysis of the differences and similarities between the graphs outputted. Below in Figure 8, are screenshots of the robot in rotation in all four corners as well as the middle of the enclosed world simulation.

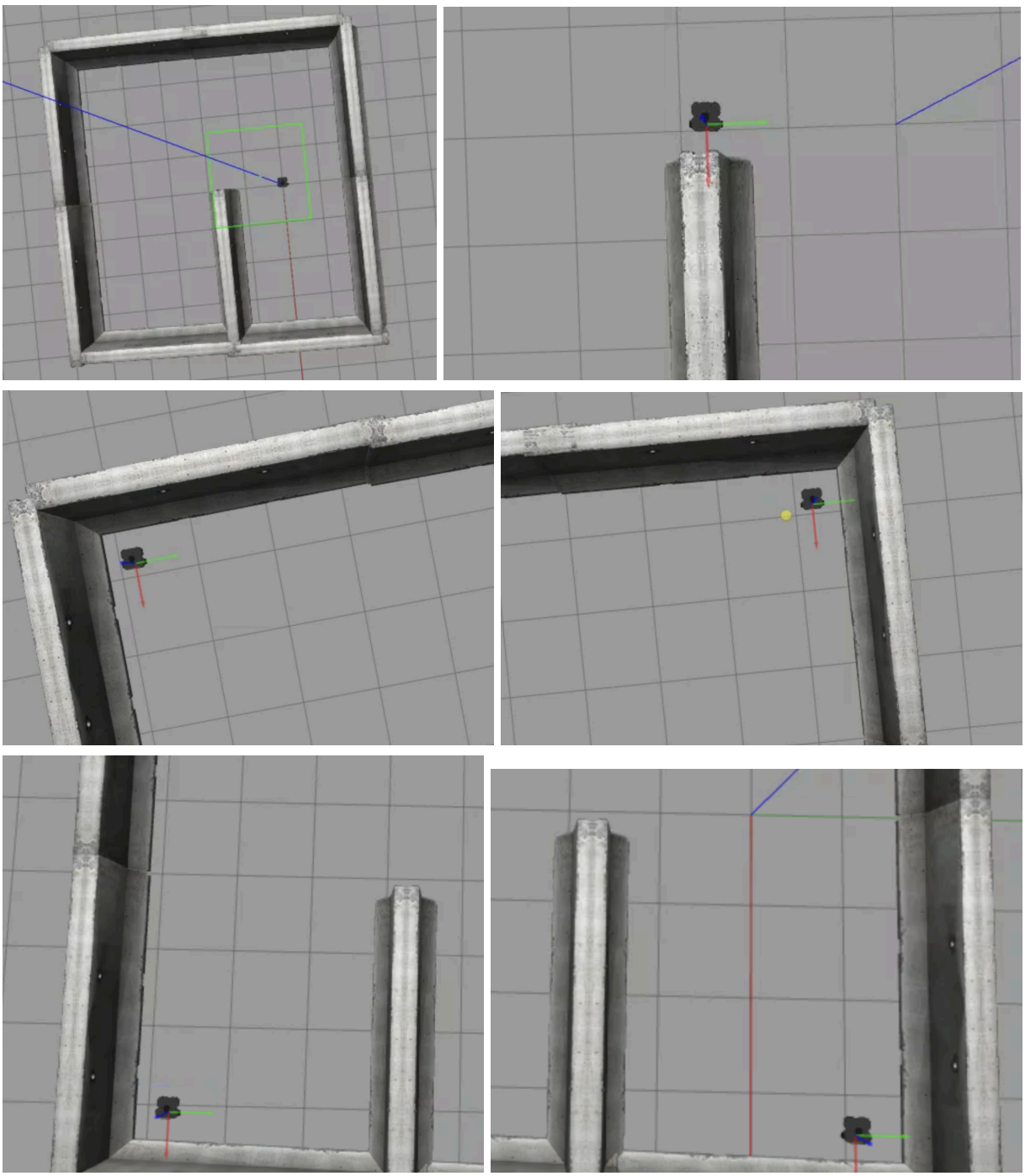

Figure 8: Images Portray the rotation of the robots from all four corners, as well as the center of the enclosed world simulation.

Step 3 of the code

For step three of the lab, this same procedure and process was repeated, changing however the pMax in the program to 16. This generated new and different graphs which will be further discussed in Section 3. The code for changing pMax from 8 to 16 can be seen in Figure 9 below.

```
165
166 #------------------------------MAIN  program----------------------
167 if __name__ == '__main__':
168     try:
169         rospy.on_shutdown(callback_shutdown)
170         for r in (0,4):
171             # spin_node(8)
172             spin_node(16)
173     except rospy.ROSInterruptException:
174         pass
175
```

Figure 9: Here the spin_node(8) is commented and below that is spin_node(16) which will now rotate the robot at pMax 16.

Step 4 of the code

For Step 4 of the code, the ROS spin_node code was supposed to be modified by adding a function that when given H as an argument, it will return the index into H that reflects the furthest open space around the robot. Code would then be added to rotate the robot to this heading and move forward for 0.5 seconds at *0.5 m/s.* Unfortunately, this part of the lab assignment was not figured out in time, and therefore has not been completed. This code would be changed most likely in the spi_node(pMax) function embedding a function within that function in order to find the furthest open space from the robot as opposed to the laser reflection of the robot as it rotates. Once that function would be created, some code would be added allowing the robot to stop at that point of furthest open space and then move towards that open space in 0.5 seconds at a speed of *0.5 m/s.* Figure 10 portrays where this code would most likely be added.

```
80 def spin_node(pMax):
81     '''Spins the robot 360 degrees'''
82     rospy.init_node('Spin_Node', anonymous=True)
83 
84     global H # the H array is created to be a gloal variable
85     H = [0] * 360 # a bar graph plot is initialized for each degree of the robot's rotation
86     # register publisher and two subscribers
87     vel_pub = rospy.Publisher(motionTopic, Twist, queue_size=0)
88     scan_sub = rospy.Subscriber(laserTopic, LaserScan, callback_laser)
89     pose_sub = rospy.Subscriber(poseTopic, Odometry, callback_pose)
90     rospy.sleep(1) # allow callbacks to happen before proceeding
91 
92     rate = rospy.Rate(10) # Hz for main loop frequency
93     msg = Twist() # empty new velocity message
94     accuracy = 0.1 # radians
95 
96     # the initial position of the robot is not where we are
97     # as the robot rotates the initial position is going to increase
98     # the inital angle when the robot rotates is offset by -0.2 radians
99     initPosition = gPose[2] + math.pi - 0.2
100    print(abs(gPose[2] + math.pi - initPosition))
101
102    '''The Condition for the while loop:
103       gPose[2]'s angle within the 0 - 2pi range (gPose[2] + pi) must be
104       less than the termination threshold of .1 radians'''
105    while not rospy.is_shutdown() and abs(gPose[2] + (math.pi) - initPosition) > accuracy:
106        msg.linear.x,msg.angular.z=0.0,0.1  #angular velocity is a constant value, and it is set to 0.1
107        vel_pub.publish(msg)
108
109        # the H array is used as a list here for plotting the gFrontDistance
110        # we add pi in the program to keep the robot in range from 0 to 2pi
111        # this is to ensure the robot has a positive orientation
112        # not to rotate from -pi to +pi
113        degree = int(math.degrees(gPose[2] + math.pi))
114        H[degree] = gFrontDistance
115
116        rate.sleep()
117
118    # small h is an array multiplied by the pMax
119    h = [0] * pMax
120    # aMult is an integer of 360 divided by the pMax
121    # this will create the difference in the graphs
122    aMult = int(360/pMax)
123    a = [0] * pMax
124    for i in range(pMax):
125        h[i] = H[i * aMult]
126        a[i] = i * aMult
```

Figure 10: In the spin_node(pMax) function, this is where the code for step would most likely be added for the robot to find the point of furthest open space, and to go to it in 0.5 seconds at a speed of 0.5 m/s.

3.0 Graph Analysis

For the first bar graph produced in the lab, the default position of the robot was used in the simulation. As we can see below in Figure 10, as the robot rotates from 0 to about 80 degrees there is no laser distance reflection. Once the robot reached between 80 and 160 degrees, there was an upward arc in the graph, showing an exponential growth of the laser reflection distance. Then there was a pause between 160 degrees and 200 degrees where there was no laser reflection distance and once the robot went between 200 and 340 degrees the reflected

distance of the laser took on a parabolic shape. Between 340 and 360, there was no laser reflection distance. This arc was created setting pMax to 8. The compressed bar graph on the left shows the maxima and minima of the graph on the right.

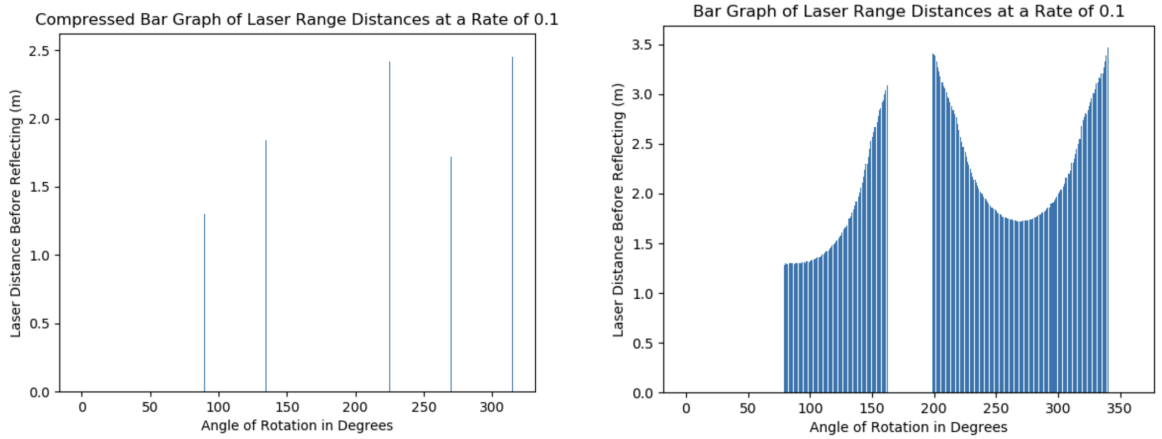

Figure 10: Exponential curve between 80 and 160 degrees, and a parabolic curve is between 200 and 340 degrees portraying the laser distance reflection. The left graph has three maxima and two minima.

At pMax 8 this is the graph when the robot was placed at the northeast corner of the enclosed world simulation. As can be seen in the graphs below, the laser distance before reflection was in between the angles of 50 and 90, and 175 and 215. The compressed bar graph on the left shows the maxima of the graph on the right. When taking the difference of the highest and lowest value between each angle where the reflected laser distance was measured, the difference is 40 degrees in both cases. The graph can be seen below in Figure 11.

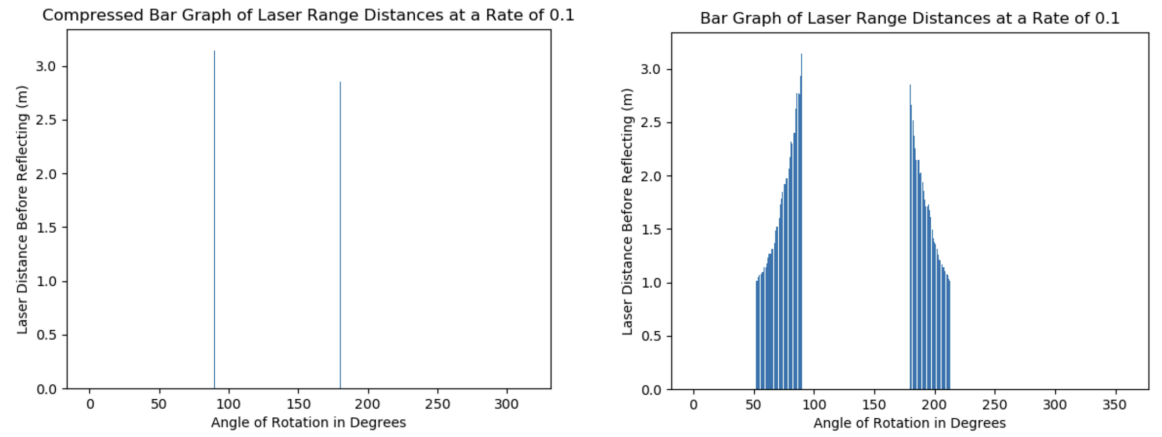

Figure 11: Exponential curve between 50 and 90 degrees, and a decreasing exponential curve

between 175 and 215 degrees portraying the laser distance reflection. The left graph has two maxima.

Similarly, at the northwest corner, at pMax 8 the laser distance before reflection was in between the angles of 145 and 185, and 275 and 315. The compressed bar graph on the left shows the maxima of the graph on the right. When taking the difference of the highest and lowest value between each angle where the reflected laser distance was measured, the difference is 40 degrees in both cases. It is also worth noting noticeably that the graph on the left has only one maxima as opposed to two. This is because in this case when the robot was at the northeast corner, the two maxima were at a close distance to each other, whereas here there is a greater gap between the maxima on th graphs on the right, therefore the highest maxima was taken into account in the graph on the left, omitting the right one. It is also worth noting that when comparing Figure 12 to Figure 11, the graphs are similar to each other, with their arcs, however their maxima differ and there is an angle shift between them of about 50 degrees.

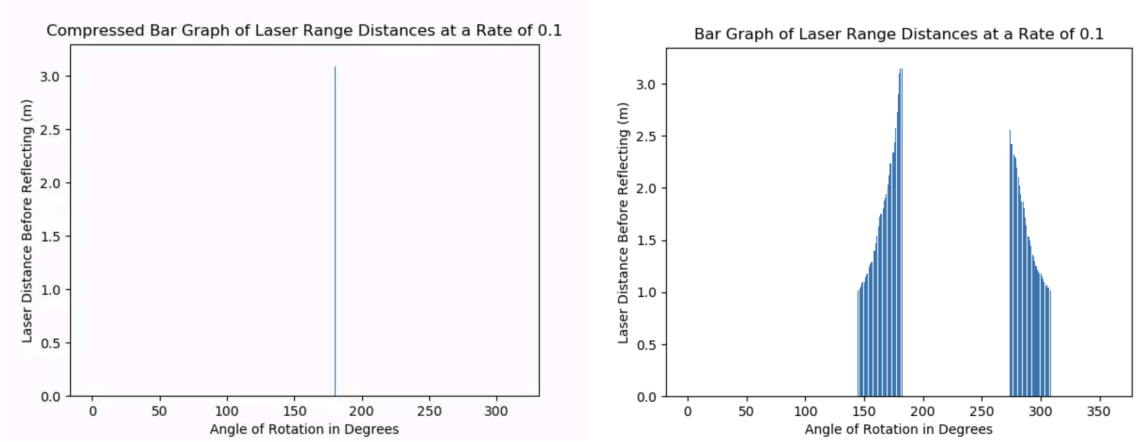

Figure 12: Exponential curve between 145 and 185 degrees, and a decreasing exponential curve between 275 and 315 degrees portraying the laser distance reflection. The left graph has one maxima.

At pMax 8 this is the graph when the robot was placed at the southwest corner of the enclosed world simulation. As can be seen in the graphs below, the laser distance before reflection was in between the angles of 0 and 40, 230 and 315, and 360. The compressed bar graph on the left shows the maxima of the graph on the right. The graph can be seen below in Figure 13.

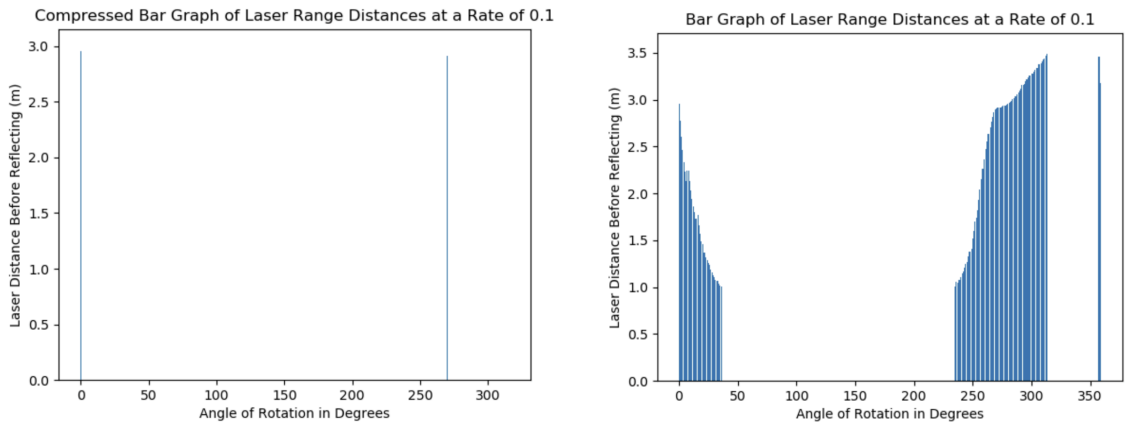

Figure 13: Exponential decreasing curve between 0 and 40 degrees, and an increasing exponential curve between 230 and 315 degrees, lastly there seems to be one final spike upwards at 360 degrees portraying the laser distance reflection. The left graph has two maxima.

Similarly, at the southeast corner, at pMax 8 the laser distance before reflection was in between the angles of 0, 55 and 135, and 330 and 360. The compressed bar graph on the left shows the maxima of the graph on the right. It is also worth noting noticeably that the graph on the left has two maxima, which is similar to the graph above which also has two maxima. It is also worth noting that when comparing Figure 12 to Figure 11, the graphs are similar to each other, with their arcs, however their maxima differ and there is an angle shift between them where the southwest graph has a spike at 360, whereas the graph at the southeast corner has a spike at 0. When looking at the areas of the curves as well, they are visually flipped. This can be explained based on the orientation of the walls around the robot since at the southwest corner the walls were facing north and east, whereas at the southeast corner the walls were facing north and west, so because of this east and west opposite direction, this can cause a sort of reflection in the graphs.

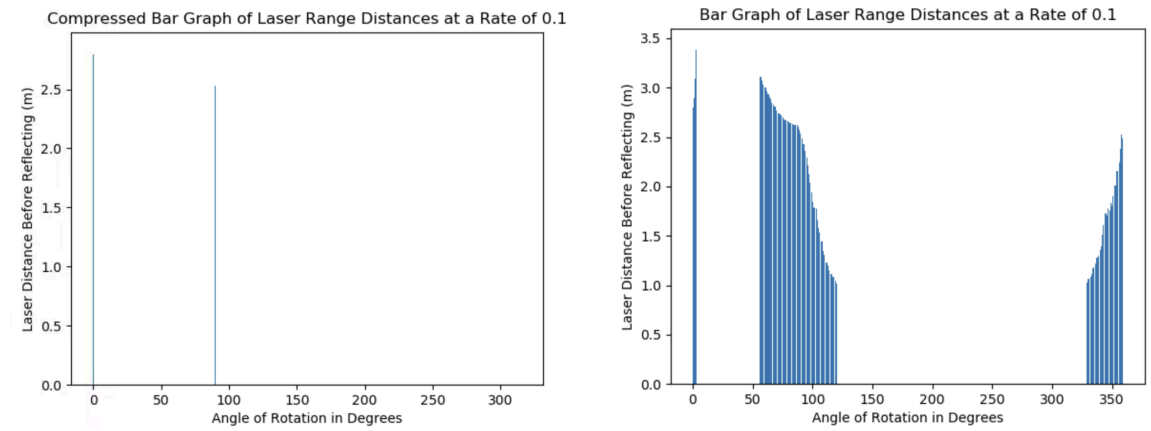

Figure 14: Spike at 0 degrees, an exponential decreasing curve between 55 and 135 degrees, and an increasing exponential curve between 330 and 360 degrees, portraying the laser distance reflection. The left graph has two maxima.

At pMax 8 this is the graph when the robot was placed in the middle of the enclosed world simulation. As can be seen in the graphs below, the laser distance before reflection was in between the angles of 0 and 25, 240 and 300, and between 335 and 360. The compressed bar graph on the left shows the maxima of the graph on the right. The graph can be seen below in Figure 15. The leftmost and rightmost graphs in the bar graph on the right seem to be reflections of each other and both share a difference of angles that is 25 degrees. What is worth noticing also is the inversion of these bar graphs at the bar graph in the middle. This can be potentially explained by how the robot was placed in the middle position when it was rotating causing a skew towards the right as the angles rotated started to increase nearing 360. There are two maxima in the compressed graph on the left.

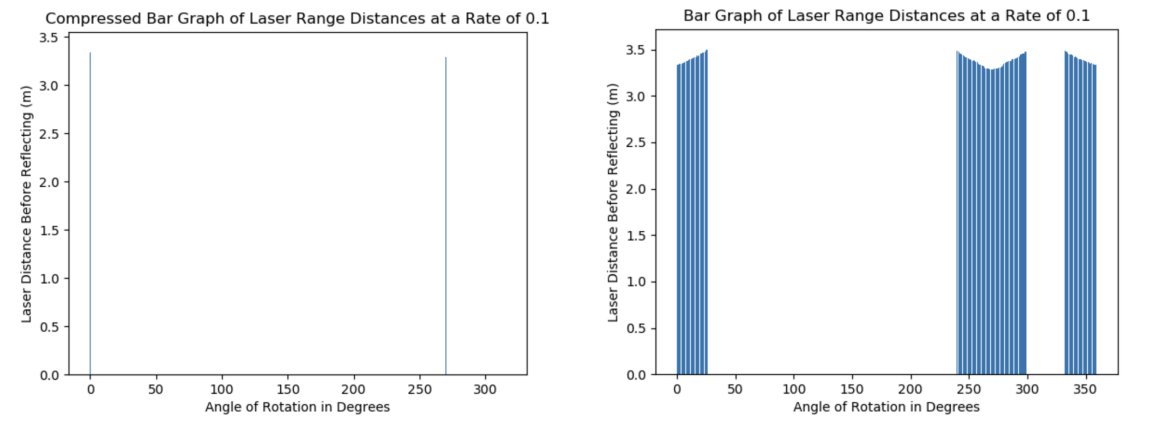

Figure 15: There is a bar graph on the right between 0 and 25 degrees, 240 and 300 degrees,

and 335 and 360 degrees. There is also a compressed bar graph on the left showing the two maxima of the graph on the right.

At pMax 16 this is the graph when the robot was placed at the northeast corner of the enclosed world simulation. As can be seen in the graphs below, the laser distance before reflection was in between the angles of 60 and 90, and 175 and 215. The compressed bar graph on the left shows the maxima of the graph on the right. When taking the difference of the highest and lowest value between each angle where the reflected laser distance was measured, the difference range is between 30 and 40 degrees. The graph can be seen below in Figure 16. There is very little difference between these graphs in Figure 16 and Figure 11, only that the difference angle range in Figure 16 is between 30 and 40 degrees, whereas in Figure 11 it is 40 flat. In this case, for the northeast, there is no significant difference between pMax being set to 8 or pMax being set to 16.

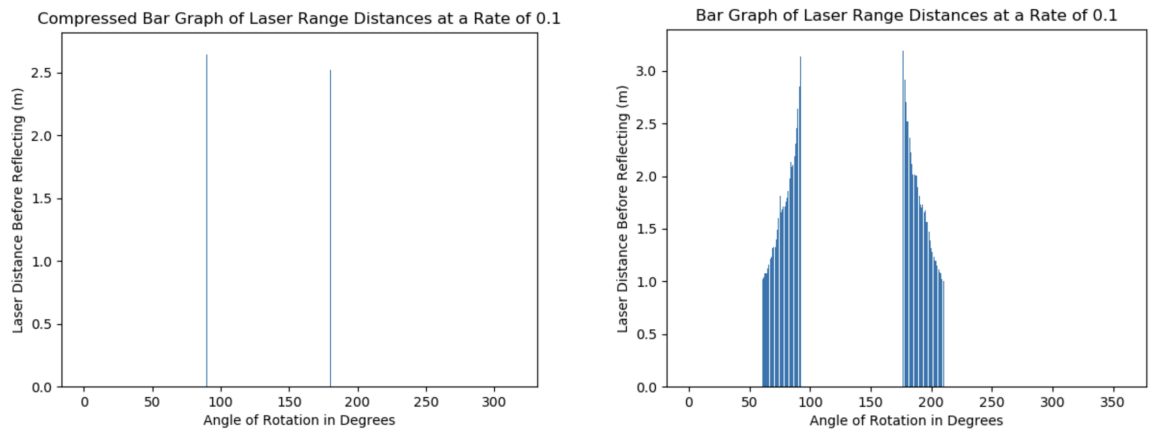

Figure 16: Exponential curve between 60 and 90 degrees, and a decreasing exponential curve between 175 and 215 degrees portraying the laser distance reflection. The left graph has two maxima.

Similarly, at the northwest corner, at pMax 16 the laser distance before reflection was in between the angles of 145 and 180, and 265 and 280. The compressed bar graph on the left shows the maxima of the graph on the right. When taking the difference of the highest and lowest value between each angle where the reflected laser distance was measured, the differences are not even close to each other with the first having a difference of 35 degrees, and the second of 15, so here we can see how changing the pMax from 8 to 16 starts to take effect here. It is also worth noting that when comparing Figure 12 to Figure 17, the graphs are similar to each other, with their arcs, however Figure 17 has a maxima and a minima, whereas Figure 12 only has one maxima. This might hint that increasing the Pmax value will create a more

accurate compressed graph which can be used to better analyze what ranges a robot rotates when placed in a specific location to rotate in.

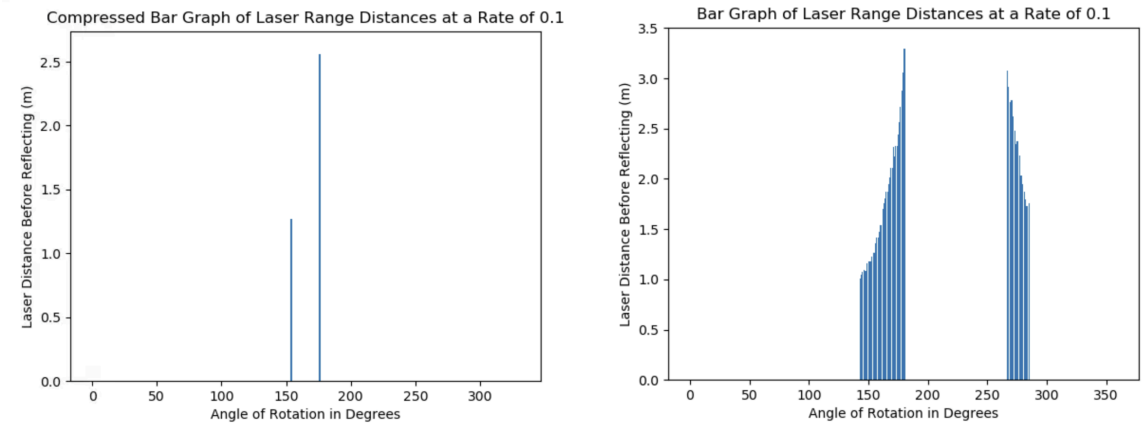

Figure 17: Exponential curve between 145 and 180 degrees, and a decreasing exponential curve between 265 and 280 degrees portraying the laser distance reflection. The left graph has one maxima and one minima.

At pMax 16 this is the graph when the robot was placed at the southwest corner of the enclosed world simulation. As can be seen in the graphs below, the laser distance before reflection was in between the angles of 0 and 30, 240 and 310, and 360. The compressed bar graph on the left shows the maxima of the graph on the right. The graph can be seen below in Figure 18. When comparing these graphs in Figure 18, to the graphs in Figure 13, what can be noticeable is that the angle ranges between the graphs of the figures are similar to each other, both for example end at 360 degrees, however they do differ slightly in the other angles as there are some discrepancies there. Both Figures however have two maxima.

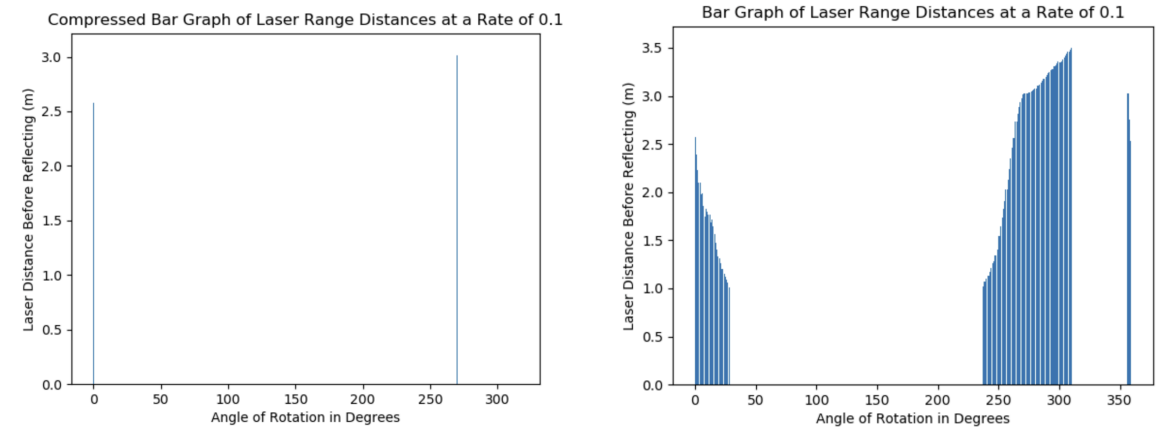

Figure 18: Exponential decreasing curve between 0 and 30 degrees, and an increasing exponential curve between 240 and 310 degrees, lastly there seems to be one final spike upwards at 360 degrees portraying the laser distance reflection. The left graph has two maxima.

Similarly, at the southeast corner, at pMax 16 the laser distance before reflection was in between the angles of 0, 45 and 115, and 330 and 360. The compressed bar graph on the left shows the maxima of the graph on the right. It is also worth noting noticeably that the graph on the left has two maxima, two minima, as well two points where the bar graph changes slope. In Figure 14, something like this was not present, whereas once pMax was changed to 16, the compressed bar graph starts to show more accurate points defining the slope of the bar graph on the right. It is also worth noting that when comparing Figure 14 to Figure 19, the graphs are similar to each other, with their arcs, however their maxima differs based on number and as mentioned earlier there are minima and other points present to help better define the arc of the bar graph. Similarly to Figures 13 and 14 when pMax was set to 8, when pMax was set to 16, when looking at the areas of the curves as well, they are visually flipped. This can be explained based on the orientation of the walls around the robot since at the southwest corner the walls were facing north and east, whereas at the southeast corner the walls were facing north and west, so because of this east and west opposite direction, this can cause a sort of reflection in the graphs.

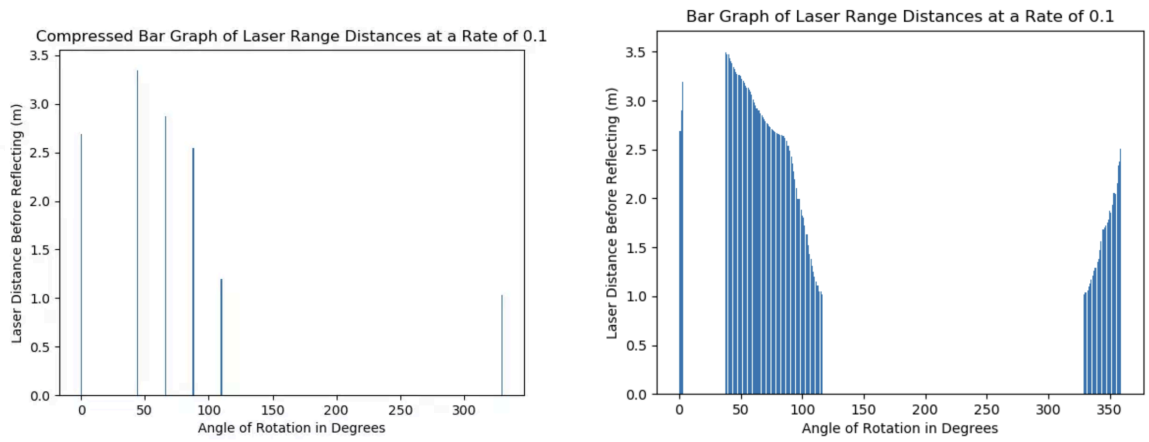

Figure 19: Spike at 0 degrees, an exponential decreasing curve between 45 and 115 degrees, and an increasing exponential curve between 330 and 360 degrees, portraying the laser distance reflection. The left graph has two maxima, two minima, and two intermediate points between one of the minima and maxima, helping define the slope of the bar graph.

At pMax 16 this is the graph when the robot was placed in the middle of the enclosed world simulation. As can be seen in the graphs below, the laser distance before reflection was in between the angles of 0 and 20, 240 and 300, and between 330 and 360. The compressed bar graph on the left shows the maxima of the graph on the right. The graph can be seen below in Figure 20. The leftmost and rightmost graphs in the bar graph however in Figure 20 differ in comparison to Figure 15, where Figure 15 had a constant range difference between the angles of rotation, whereas here there is a visual discrepancy between these two bar graphs. What is worth noticing also is the inversion of these bar graphs at the bar graph in the middle, which is similar to the case in Figure 15. This can be potentially explained by how the robot was placed in the middle position when it was rotating causing a skew towards the right as the angles rotated started to increase nearing 360. There are two maxima in the compressed graph on the left. This is also the case for the pMax is 8 case.

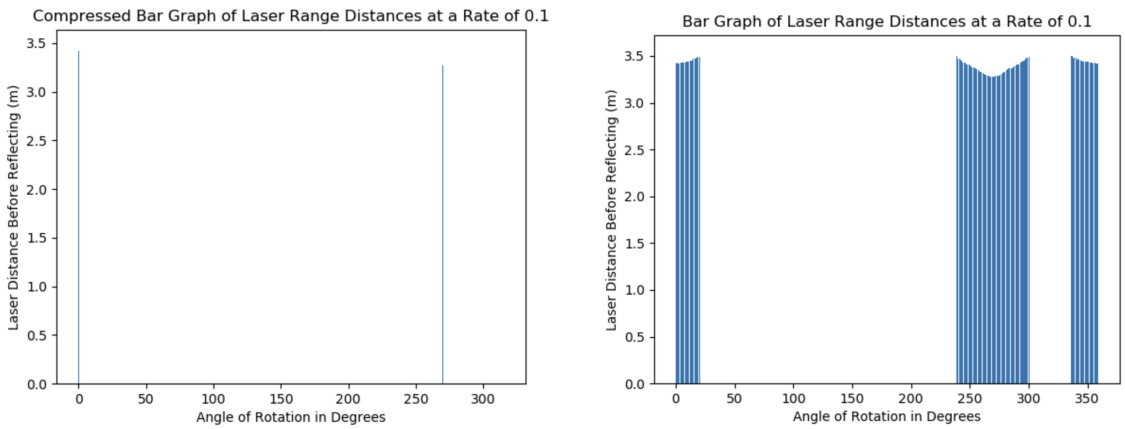

Figure 20: There is a bar graph on the right between 0 and 20 degrees, 240 and 300 degrees, and 330 and 360 degrees. There is also a compressed bar graph on the left showing the two maxima of the graph on the right.

Based on these observations of the graphs, when the robot is placed in the corners there is usually a angle range of 40 degrees where the robot measures the laser distance before reflection, otherwise, when those angles are passed, there is no laser distance before reflection, because that is where the laser bouncing off of the walls in the simulation is reflected back to the robot. Oddly enough when the robot rotated from its default position, that was the largest plot of the laser distance before reflection for the robot. This can be explained by the fact that at the default position, the robot had a lot of space around it, so the laser distance before reflection had a higher probability of not reflecting back to the robot. At the corners and at the middle, as the robot rotated, it would be close to the walls in the enclosed world simulation,

causing the laser to reflect back to the robot, therefore nullifying many possible angles for the robot to rotate where the laser reflection distance does not affect the robot.

4.0 References

```python
#!/usr/bin/env python
# license removed for brevity
#
# Example program that
# spins in place
# dml 2016
#
#

import sys
import math
import random
import matplotlib.pyplot as plt

import rospy # needed for ROS
from geometry_msgs.msg import Twist      # ROS Twist message
from sensor_msgs.msg import LaserScan    # ROS laser scan message
from nav_msgs.msg import Odometry        # ROS pose message

from tf.transformations import euler_from_quaternion # rotations are quaternions

# ROS topic names for turtlebot_gazebo

motionTopic ='/cmd_vel'
poseTopic   = '/odom'
laserTopic  = '/scan'

# Global variables

gFrontDistance=0      # average laser range distance in front of robot
gPose = [0,0,0]
H = []                # H array will create the bar graph for gFrontDistance

# callback to accept pose information
# orientation is stored as a quaterion
# and needs to be translated to angles
#
def callback_pose(msg):
   global gPose
   orient = msg.pose.pose.orientation
   quat = [axis for axis in [orient.x, orient.y, orient.z, orient.w]]
   (roll,pitch,yaw)=euler_from_quaternion(quat)
   gPose[0]= msg.pose.pose.position.x
   gPose[1]= msg.pose.pose.position.y
   gPose[2]= yaw
   return



# callback for the laser range data
# calculates a front distance as an average
# laser reading over a few measurements in front
#
def callback_laser(msg):
   '''Call back function for laser range data'''
   global gFrontDistance
```

Page 1 of the code

```
# laser reading over a few measurements in front
#
def callback_laser(msg):
    '''Call back function for laser range data'''
    global gFrontDistance

    center = 0
    width = 10 # region in front to average over
    dist=0.0
    count=0

    for i in range(center-width,center+width):
        if not math.isnan( msg.ranges[i] ) and \
           not math.isinf( msg.ranges[i] ) and \
           msg.ranges[i]>1.0:
            dist += msg.ranges[i]
            count += 1
    if (count>0) :
        gFrontDistance = dist/float(count)
    else:
        gFrontDistance=-1 # a big number, nothing in view
    return


# spin_node
# will spin the robot aroud to reach the angle given as argument
# at speed given as second argument in rps

def spin_node(pMax):
    '''Spins the robot 360 degrees'''
    rospy.init_node('Spin_Node', anonymous=True)

    global H # the H array is created to be a gloal variable
    H = [0] * 360 # a bar graph plot is initialized for each degree of the robot's rotation
    # register publisher and two subscribers
    vel_pub = rospy.Publisher(motionTopic, Twist, queue_size=0)
    scan_sub = rospy.Subscriber(laserTopic, LaserScan, callback_laser)
    pose_sub = rospy.Subscriber(poseTopic, Odometry, callback_pose)
    rospy.sleep(1) # allow callbacks to happen before proceeding

    rate = rospy.Rate(10) # Hz for main loop frequency
    msg = Twist() # empty new velocity message
    accuracy = 0.1 # radians

    # the initial position of the robot is not where we are
    # as the robot rotates the initial position is going to increase
    # the inital angle when the robot rotates is offset by -0.2 radians
    initPosition = gPose[2] + math.pi - 0.2
    print(abs(gPose[2] + math.pi - initPosition))

    '''The Condition for the while loop:
       gPose[2]'s angle within the 0 - 2pi range (gPose[2] + pi) must be
       less than the termination threshold of .1 radians'''
    while not rospy.is_shutdown() and abs(gPose[2] + (math.pi) - initPosition) > accuracy:
        msg.linear.x,msg.angular.z=0.0,0.1  #angular velocity is a constant value, and it is set to 0.1
        vel_pub.publish(msg)
```

Page 2 of the code

```
103       gPose[2]'s angle within the 0 - 2pi range (gPose[2] + pi) must be
104       less than the termination threshold of .1 radians'''
105    while not rospy.is_shutdown() and abs(gPose[2] + (math.pi) - initPosition) > accuracy:
106        msg.linear.x,msg.angular.z=0.0,0.1  #angular velocity is a constant value, and it is set to 0.1
107        vel_pub.publish(msg)
108
109        # the H array is used as a list here for plotting the gFrontDistance
110        # we add pi in the program to keep the robot in range from 0 to 2pi
111        # this is to ensure the robot has a positive orientation
112        # not to rotate from -pi to +pi
113        degree = int(math.degrees(gPose[2] + math.pi))
114        H[degree] = gFrontDistance
115
116        rate.sleep()
117
118    # small h is an array multiplied by the pMax
119    h = [0] * pMax
120    # aMult is an integer of 360 divided by the pMax
121    # this will create the difference in the graphs
122    aMult = int(360/pMax)
123    a = [0] * pMax
124    for i in range(pMax):
125        h[i] = H[i * aMult]
126        a[i] = i * aMult
127
128    # code to plot the small bar graph of the angles
129    plt.bar(a, h)
130    plt.xlabel("Angle of Rotation in Degrees")
131    plt.ylabel("Laster Distance Before Reflecting (m)")
132    plt.title("Compressed Bar Graph of Laser Range Distances at a Rate of 0.1")
133    plt.ylim(0)
134    plt.show()
135
136    # code to show the bar graph in more detail
137    plt.bar(range(len(H)),H)
138    plt.ylim(0)
139    plt.xlabel("Angle of Rotation in Degrees")
140    plt.ylabel("Laster Distance Before Reflecting (m)")
141    plt.title("Bar Graph of Laser Range Distances at a Rate of 0.1")
142    plt.show()
143
144    # outputs a message of where the spin of the robot was completed
145    print ("Spin complete at ",gPose[2])
146    msg.linear.x,msg.angular.z=0.0,0.0 # stop now
147    vel_pub.publish(msg)
148
149    return
150
151 #
152 # Callback function for a controlled shutdown
153 #
154
155 def callback_shutdown():
156    print("Shutting down")
157    vpub = rospy.Publisher(motionTopic, Twist, queue_size=10)
```

Page 3 of the code

```
120    # aMult is an integer of 360 divided by the pMax
121    # this will create the difference in the graphs
122    aMult = int(360/pMax)
123    a = [0] * pMax
124    for i in range(pMax):
125        h[i] = H[i * aMult]
126        a[i] = i * aMult
127
128    # code to plot the small bar graph of the angles
129    plt.bar(a, h)
130    plt.xlabel("Angle of Rotation in Degrees")
131    plt.ylabel("Laster Distance Before Reflecting (m)")
132    plt.title("Compressed Bar Graph of Laser Range Distances at a Rate of 0.1")
133    plt.ylim(0)
134    plt.show()
135
136    # code to show the bar graph in more detail
137    plt.bar(range(len(H)),H)
138    plt.ylim(0)
139    plt.xlabel("Angle of Rotation in Degrees")
140    plt.ylabel("Laster Distance Before Reflecting (m)")
141    plt.title("Bar Graph of Laser Range Distances at a Rate of 0.1")
142    plt.show()
143
144    # outputs a message of where the spin of the robot was completed
145    print ("Spin complete at ",gPose[2])
146    msg.linear.x,msg.angular.z=0.0,0.0 # stop now
147    vel_pub.publish(msg)
148
149    return
150
151 #
152 # Callback function for a controlled shutdown
153 #
154
155 def callback_shutdown():
156    print("Shutting down")
157    vpub = rospy.Publisher(motionTopic, Twist, queue_size=10)
158    msg = Twist()
159    msg.angular.z=0.0
160    msg.linear.x=0.0
161    vpub.publish(msg)
162    return
163
164
165
166 #------------------------------MAIN  program---------------------
167 if __name__ == '__main__':
168    try:
169        rospy.on_shutdown(callback_shutdown)
170        for r in (0,4):
171            spin_node(8)
172            # spin_node(16)
173    except rospy.ROSInterruptException:
174        pass
175
```

Page 4 of the code
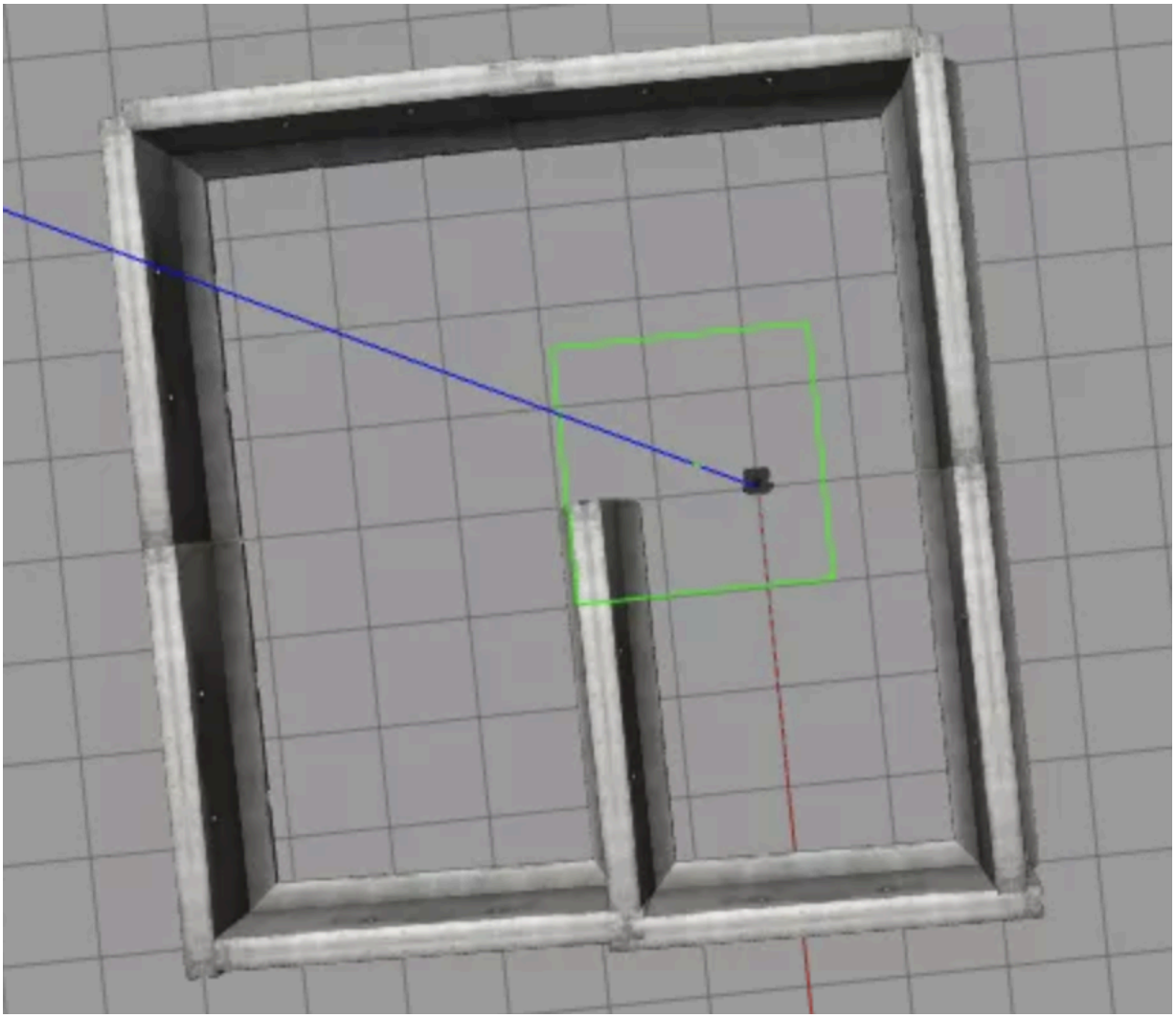
Image of the robot Rotating

Images of the code, and turtle bot

5.0 Summary

Although the program may have been flawed and not everything is correct, the robot's rotation, as well as graphs produced as a result from running the program worked. As the program started the robot started rotating at a constant angular speed of 0.1. Once the robot completed its rotation it outputted a graph that portrayed the distance before the laser reflected back onto the robot and the angles as the robot rotated. Code snippets are 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.