Spring 2018

Home Assignment V

Differential Driving with ActivityBot

Course: Autonomous Robotics

Submitted By:
Taylor JL Whitaker
Pankaj Bhowmik
Ying Fan
Md Jubaer Hossain Pantho

Objective:
The objective of this work is to test the quality and accuracy of the ActivityBot's differential driving capabilities.

Problem I : Error Analysis
The goal of this exercise is to perform an error analysis on the ActivityBot's kinematic. For this purpose, we consider the influence of the following parameters on the driving accuracy: speed, floor type and distance.

We programmed the robot to drive on a straight line at various distances (8 ft, 12 ft and 16 ft) with various speed (slow, medium and fast). We tested errors for two type of surface (with carpet and without carpet). This process was repeat this process 5 times and an an error analysis is perform by calculating the average error at each distance at various speed and floor type.

Next, for 4 different distances (6 in, 12 in, 18 in, 24 in), we made the robot drive in a straight line. The start and stop positions was marked with masking tape on the tile floor (as shown in the video link).
The results are shown in tabular form in Table I ,II and III.

Table 1: Error measurement on Floor

Position(foot)	Slow	Medium	Fast
8	~	3	0
12	2	6	2
16	11	14	11

As we can see in Table 1, the error increases with the distance. When we performed the same test on carpet, it is noticeable that the error decreases drastically (shown in Table 2). We can assume that the texture of the track plays a vital role on the accuracy of the ActivityBot. We can further observe that on carpet, the error decreases drastically when we use very high speed.

Table 2: Error measurement on Carpet

Position(foot)	Slow	Medium	Fast
8	7	2.5	1
12	10	11	1
16	26	18	1

The results in Table 3 suggest that when the distance is small the performance of the ActivityBot is fairly acceptable as it shows no error at all.

Table 3: Driving Straight

Position(in)	Error
8	0
12	0
16	0
24	0

Problem II : Dead Reckoning

In this experiment, we program the ActivityBot to drive on the track provided in the assignment. The following figures show the track we used to test the performance of our bot.

Please see the video to observe the performance of our ActivityBot. As you can see in Table 4, the error increases as the bot progresses through the track. However, it is interesting to note that after some nodes the bot starts to correct itself, and somewhat reaches close to its original intended destination.

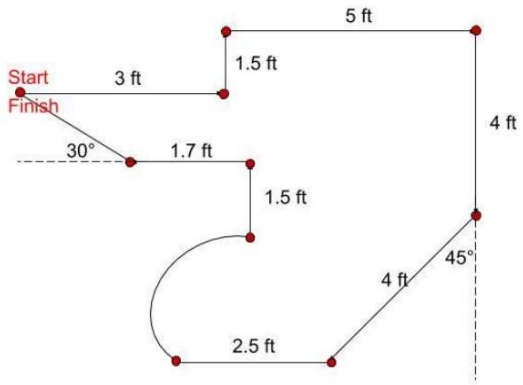

Figure: Provided track

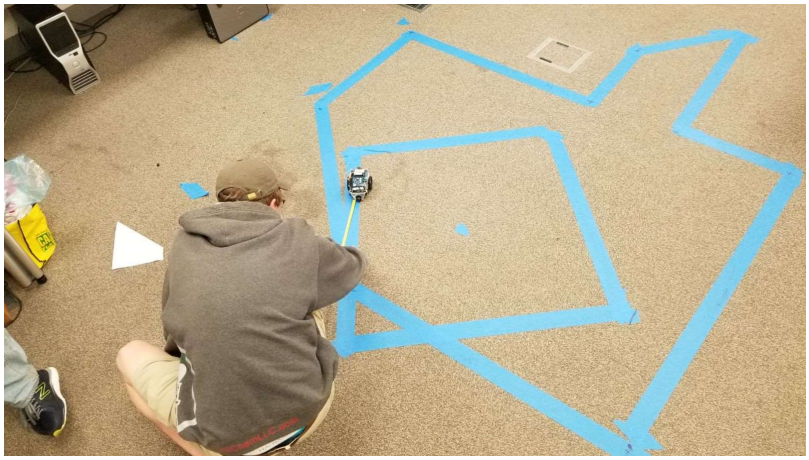

Figure: Designed track

Table 4: Error generated in different end positions of the track.

Position	Error(in inch)
1	--
2	1.6
3	11.5
4	18
5	24
6	22
7	13.5

8	14
9	15.5

Problem III: Dead Reckoning II

In this work, we program the ActivityBot to drive the track of the following figure and compute the average error per unit of distance.

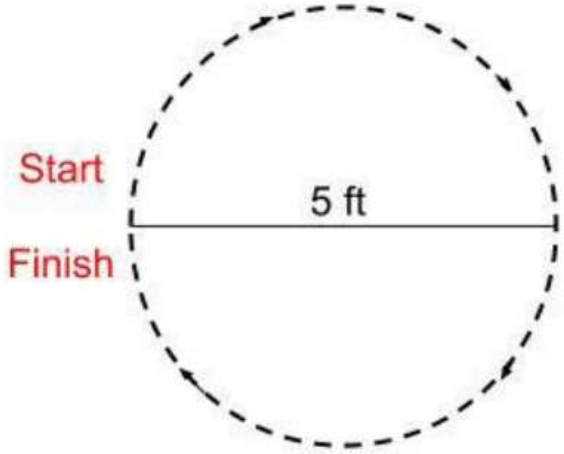

Figure: provided track (circle)

The performance of the bot can be observed in the provided video. We calculated the error after one circle and found that the bot deviates around 9 inches after one lap.

Problem IV: Dead Reckoning III (Mandatory for the Graduate Section, Bonus for the rest)

Next, we program the ActivityBot to drive the track of the following figure "8" below. We programmed the robot to always pass through the same center point of the figure 8 and compute the error observed for the center point in 4 runs.

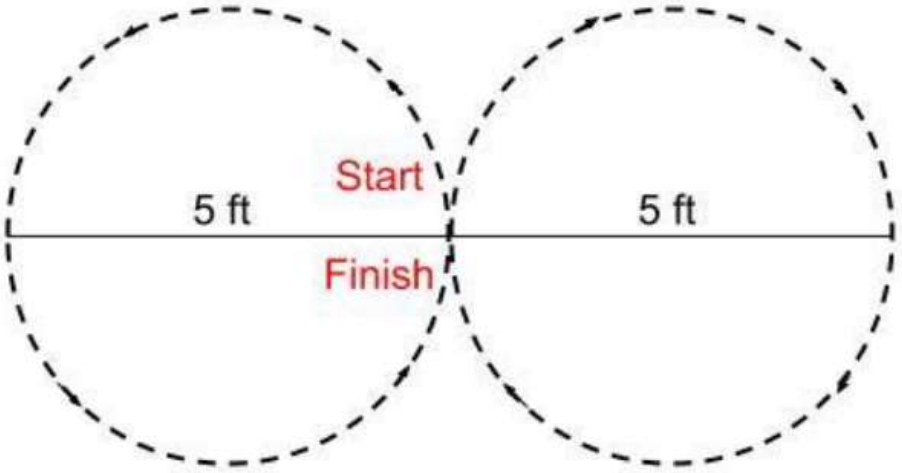

Figure: provided track (eight)

When the ActivityBot made a circle we observed an average error of 9 inch from the starting point. To complete the “8” shape, we programmed to reverse the order of the activation values on the wheels no sooner it reaches to the starting point. When the ActivityBot started its journey to make another circle in reverse order to form a route of “8” shape, it gave an average deviation of 4.5 inch from the initial position. The route has two circles of equal size with a diameter of 5 foot.

Discussion:

The code for this assignment was not very complex as we only relied on two functions the ActivityBot360 use to interface with its servo. The function *drive_goto(leftWheel, rightWheel)* simply takes values according to the number of ticks the left and right servos should move. The servos will travel the allotted distance in the same time interval. In other words, both servos will travel the specified distance though each will operate at a different speed in order to have both servos stop at the same time. When translating ticks of the servo into distances, we had to consider the distance the outer edge of the wheels traveled per tick. This value was provided by Terasic and was 3.25mm per tick. The second function we relied on was *drive_speed(leftWheel, rightWheel)* which will set the servo speeds accordingly. The arguments are provided in ticks per second.

To touch on the specific tasks, we will start with task one, to perform error analysis on the robot’s movement when programmed to go in a straight line. For all of these tests, the *drive_speed()* function was used to set both wheels at the same speed. We simply measured the deviations at the specified distances. The second task was the track configuration. Here, we utilized the *drive_goto()* function to specify the distances of each segment of the track. By calling this function sequentially with each segments distances, we were able to program the entire path. This path also included vertices in which the robot was to turn at some angle to continue to the next track segment.

Lastly, the third and fourth tasks called for the robot to travel in a circle with a 5ft diameter. We again used the *drive_goto()* function to program the circle. Considering the different speeds assigned to each wheel, discussed above, we utilized this behavior to simply set each wheel’s distance to travel. The inner wheel would travel the circumference of the 5ft circle, while the outer wheel was to travel the distance of the 5ft circle plus the width of the robot. This came out to be approximately 120mm, bringing the outer wheel’s circle radius to 5ft and 240mm. The single circle was accomplished with a single call to the *drive_goto()* function, while the Figure 8 required only one additional call with the arguments switched to complete the second circle of the track.

The source code for each of these tasks is shown below.

Appendix A - Videos Link
https://drive.google.com/drive/folders/1fODJ0xuWLX2etly0j-F3uWA7nU3Zqphc?usp=sharing

Appendix B - Source Code:

StraightLines.py

```
import time
from PiLo import PiLo
p = PiLo()

# Slow Speed
p.sendCommand(1, 50, 50)

# Medium Speed
#p.sendCommand(1, 100, 100)

# Fast Speed
#p.sendCommand(1, 200, 200)

# Stop after some time
time.sleep(20)
p.sendCommand(1, 0, 0)
```

Track.py

```
import time
from PiLo import PiLo
p = PiLo()

# The outer wheel must travel 1.1574 times the distance of the inner wheel
#p.sendCommand(0,1606,1388)
p.sendCommand(0,281,281)
time.sleep(5)
p.sendCommand(0,-25,26)
time.sleep(5)
p.sendCommand(0,143,143)
time.sleep(5)
p.sendCommand(0,26,-25)
#time.sleep(5)
```

```
p.sendCommand(0,469,469)
time.sleep(8)
p.sendCommand(0,26,-25)
#time.sleep(5)
p.sendCommand(0,375,375)
time.sleep(8)
p.sendCommand(0,13,-13)
#time.sleep(5)
p.sendCommand(0,375,375)
time.sleep(8)
p.sendCommand(0,13,-13)
#time.sleep(5)
p.sendCommand(0,234,234)
time.sleep(5)
p.sendCommand(0,26,-25)
#time.sleep(5)
p.sendCommand(0,359,359)
time.sleep(8)
p.sendCommand(0,-25,26)
#time.sleep(5)
p.sendCommand(0,159,159)
time.sleep(5)
p.sendCommand(0,9,-8)
#time.sleep(5)
p.sendCommand(0,179,179)
time.sleep(5)
```

Circle.py

```
from PiLo import PiLo
p = PiLo()

# The outer wheel must travel 1.1574 times the distance of the inner wheel
#p.sendCommand(0,1606,1388)
p.sendCommand(0,1927,1665)
```

Figure8.py

```
import time
```

```
from PiLo import PiLo
p = PiLo()

# The outer wheel must travel 1.1574 times the distance of the inner wheel
#p.sendCommand(0,1606,1388)
p.sendCommand(0,1927,1665)
time.sleep(30)
p.sendCommand(0,1665,1927)
```