ECSE 211 Design Principles and Methods

Lab 3 Navigation & Obstacle Avoidance

Group 53
Spencer Handfield
260805699
Yi Zhu
260716006

1. Design evaluation

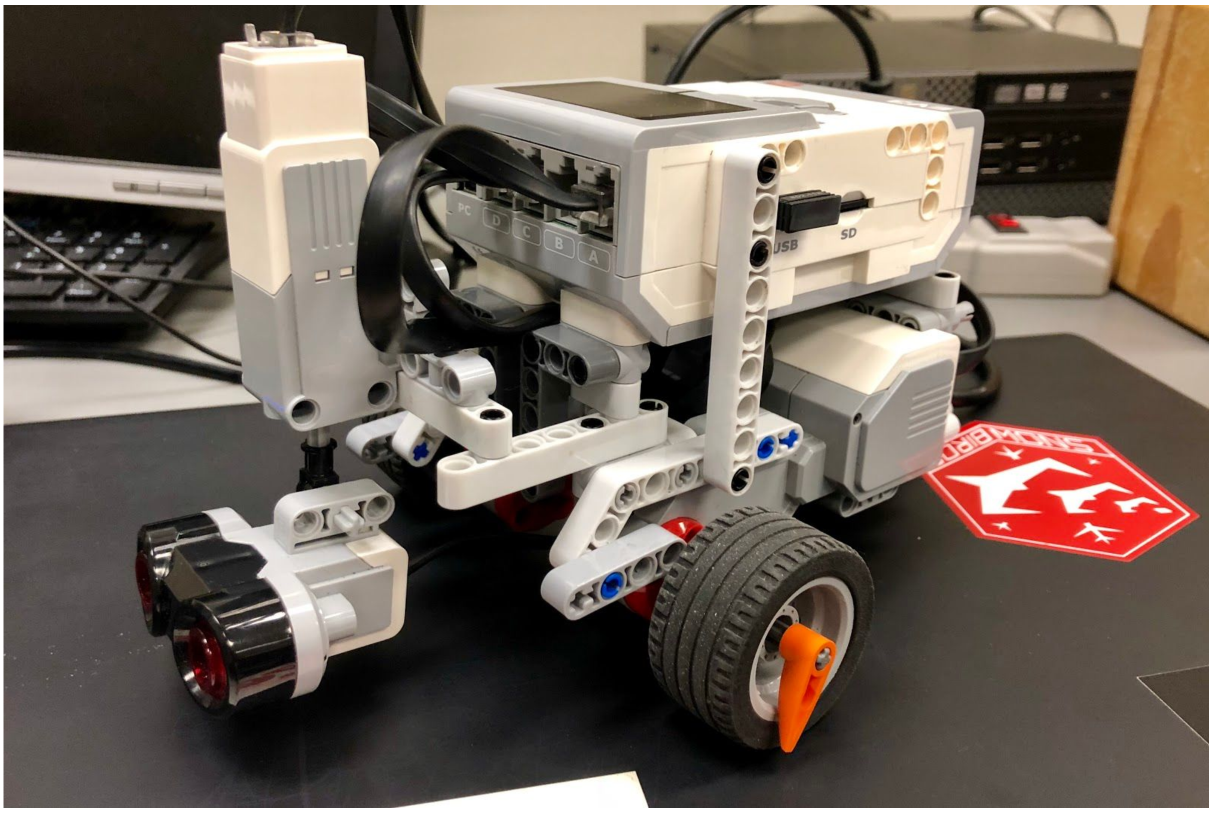


Figure 1

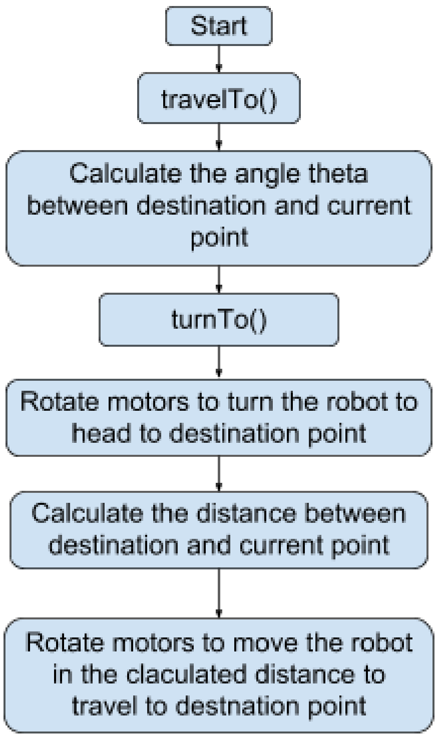
The robot is composed of 2 large motors controlling the wheels and the brick supported just above it so as to minimize its width. The wires are all tucked underneath the brick, between the support beams and above the motors to prevent them from interfering with obstacles. A support platform was extended out in front of the robot upon with a medium motor was placed, facing downward and the ultrasonic sensor was attached to it. It is placed at the bottom due to the efficiency this method displayed in the first lab in terms of accuracy of reading. More importantly it is attached to the medium motor to allow for the sensor to rotate. This hardware will allow the robot not only to see obstacles directly in front of it but also to trace its progress of avoiding said obstacles (thus requiring it to poll to its sides rather than solely in front) via the bang bang controller which will be elaborated upon within its relevant software discussion section.

As for the software, it consists of two classes: Navigation and Avoid. The Navigation part allows the robot to navigate through some given X and Y position. In this part, an array of integers represents the X and Y coordinates of the target waypoint and a counter curPoint will be used to count the way point it traveled. While the robot has not finished traveling to all waypoints, it will call travelTo() method. The travelTo () method in navigation uses the odometer to get the current X and Y coordinates. To navigate, the robot uses the coordinates of current and destination position to calculate the minimal angle of rotation (θ). The rotation angle was determined by using arctan in the following equation:

$$\theta = \tan^{-1}\frac{dy}{dx}$$

dy = the distance between the target point and current position of Y
dx= the distance between the target point and current position of X
. The Avoid class was originally attempted by simply adding an US poller into the navigating class, interrupting its travelling while it avoided and return to the process. The travelTo() method followed the almost same flow as the navigation class but rather a while loop that continuously moved the robot forward for distance it calculated, it was would be broken when the robot arrived within 0.4cm from the checkpoint. All the while, the US sensor is pointed in front of the robot, polling for a distance. When an obstacle be detected, it would rotate the robot 45 degrees and rotated the sensor 60 degrees as well to allow optimal undertaking of a modified bang bang method from the first lab. The bangbang would run on a timer that was tested to be suitable to avoid almost all obstacles. After avoiding was done, the sensor would be readjusted and a new travelTo() method would be called to return the trip towards the target checkpoint. At that point the entire process would repeat until the arrival threshold was met instead of being limited by a literal distance to travel.

The flow chart for navigation is shown below in figure 2 and for avoid is shown below in figure 3.


Start
travelTo()
Calculate the angle theta between destination and current point
turnTo()
Rotate motors to turn the robot to head to destination point
Calculate the distance between destination and current point
Rotate motors to move the robot in the claculated distance to travel to destnation point
Figure 2

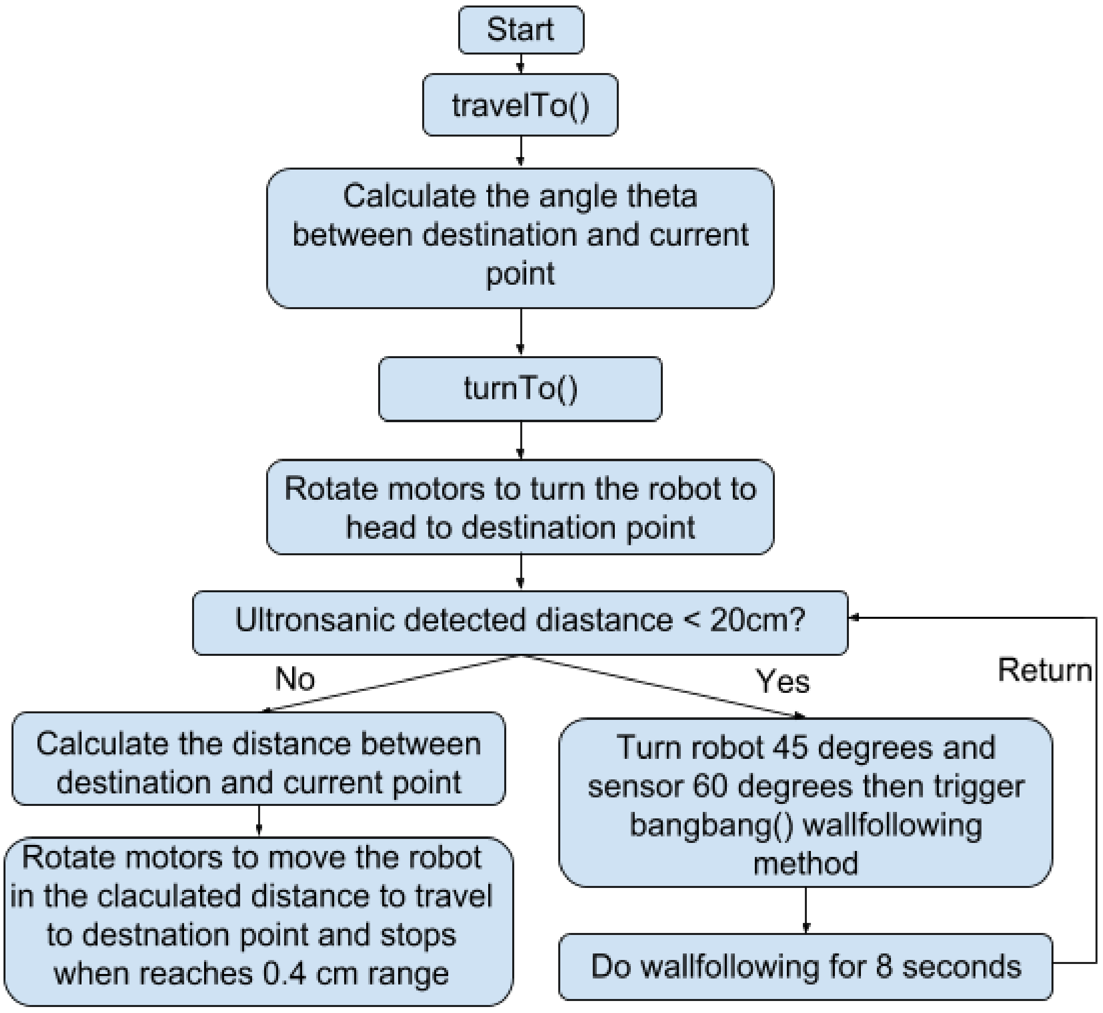

Start
travelTo()
Calculate the angle theta between destination and current point
turnTo()
Rotate motors to turn the robot to head to destination point
Ultronsanic detected diastance < 20cm?
No
Yes
Return
Calculate the distance between destination and current point
Rotate motors to move the robot in the claculated distance to travel to destnation point and stops when reaches 0.4 cm range
Turn robot 45 degrees and sensor 60 degrees then trigger bangbang() wallfollowing method
Do wallfollowing for 8 seconds

Figure 3

2.Test data

Navigation to (2,1) (1,1) (1,2) and (2,0)					
	Destination waypoint (2,0)		Actual position (measured)		Euclidean error
Trial	**x(cm)**	**y(cm)**	**x(cm)**	**y(cm)**	**e**
1	60.96	0.00	57.46	-1.21	3.70
2	60.96	0.00	57.82	-1.19	3.36
3	60.96	0.00	57.35	-0.20	3.62
4	60.96	0.00	57.66	-1.11	3.48
5	60.96	0.00	57.74	0.00	3.22
6	60.96	0.00	57.02	-0.12	3.94
7	60.96	0.00	57.83	0.00	3.13
8	60.96	0.00	57.36	0.20	3.14
9	60.96	0.00	57.96	0.00	3.00
10	60.96	0.00	57.96	0.00	3.00

Chart 1

3.Test Analysis

Mean	**Standard Deviation**
3.36	0.32

Chart 2

Mean:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Standard Deviation:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

Euclidean error:

$$\epsilon = \sqrt{(X - X_F)^2 + (Y - Y_F)^2}$$

4. Observations and conclusions

4.1 Are the errors you observed due to the odometer or navigator? What are the main sources?

Our significant errors mainly came from the odometer. We believe physical limitation such as curved axle or slippery tires causes these errors. These errors increased while traveling, especially in straight moving. It will lead to wrong turning angle and inaccurate calculation of the distance from the current position to the destination.

4.2 How acuurately does the navigation controller move the robot to its destination?

The mathematical calculation theoretically should be very accurate. However, due to the physical limitations, we have errors when the robot reaches the final point. The mean of errors is 3.36. The x coordinates in most tests have an error around 3-4 cm. The y coordinates have a relative low error with 0-1.2 cm.

4.3 How quickly does it settle on its destination?

The robot stops immediately. Since there is no correction, no oscillation will happen to the robot. No final destination check was included upon arriving at its final checkpoint, there is simply a single and immediate stop.

4.4 How would increasing the speed of the robot affect the accuracy of your navigation?

Increasing the speed will increase tire slippage on the surface. It will cause more errors during the turning or stopping movement. It will decrease the accuracy of our navigation. It will give less time for the limited physical components to make their readings, pass it to the software and receive instructions thus introducing further error sources.

5. Further improvements

5.1 Hardware

Since our errors mostly caused by physical limitation, we can reduce the length of axle to reduce the risk of bending. Along with increasing the strength of the support structure attaching the wheels to the motor and the motor to the brick. We can also implement a light sensor to detect the grid lines to make odometer correction. Accompanied with certain moving angles, we can let the odometer update certain values of X and Y correctly when the light sensor detects the grid lines in similar fashion to lab2 correction with added

5.2 Software

Since there is a risk of tire slippage when motors moving too fast. We can reduce the acceleration and speed of motors in code. To reduce the deviation during straight line moving, we can use synchronized method for both motors as well as adding speed offset for the motor which starts later.

To build on the idea of the light sensor, software implementations would have to match, such as providing a filter to ensure when it scans multiple lines in short succession (when traversing a corner of 4 intersecting lines at an improper angle). This filter could implement multiple functions to ensure accuracy such as the counter seen in the US sensor, as well as communicating with the array of checkpoints to see what corner it should theoretically be most close to and adjust to those corrected values (if the coordinates are known, the trajectory and thus the amount of corners crossed would be known and executed upon in a for loop counting through the filtered corner detector, or simple line detector if it does travel straight).