Lab 3: Navigation and Obstacle Avoidance

Report written by
Tian Han Jiang (ID: 260795887)
and Yicheng Song (ID: 260763294)
As part of Group 51

For ECSE 211: Design Principles and Methods

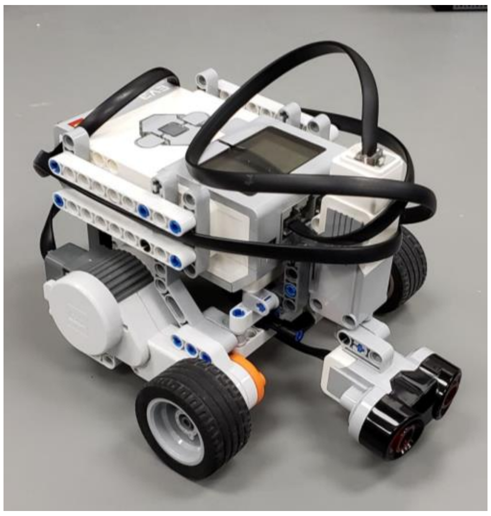
Due February 7th at 11:59PM

McGill University

Design evaluation

Hardware

From a hardware perspective, changes have been minimal. Similar as to lab 2, only the frontal sensor area has been modified to support an ultrasonic sensor attached to a small motor. This motor is connected at 4 points providing a sturdy platform for the sensor to rotate freely with minimal disturbance. **Figure 1** and **Figure 2** show in depth the modified area.



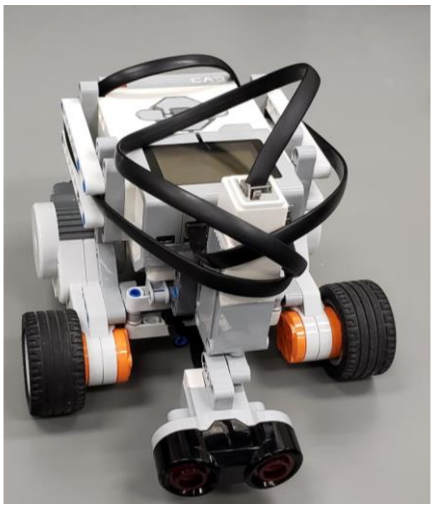


Figure 1 and Figure 2. Ultrasonic sensor mounted on upside down motor.

Software

Software implementation relied heavily on the previous two labs' code. This is exceptionally true concerning lab 2 as the core classes (Display and EV3Navigation are practically identical in format since functionality required are identical.

For the odometer class, inspiration came from previous year works as functionality was condensed into a single class whereas the code in lab 2 was spread across four classes (Odometer.java, OdometerData.java and OdometerExceptions,java). However, design logic for odometry remained identical and all the calculations were imported from lab 2.

As much as possible, constants needing to be fine tuned were declared and initialized in EV3Navigation.java for simplifications while maintaining the robot. Constants for simple navigation and obstacle avoidance were declared and initialized on their respective classes.

Navigation.java implements three major methods: travelTo, turnTo and isNavigating along with helper methods to convert between units as well as setters and getters. In order to travel to a waypoint, hypotenuse calculations as well as basic trigonometric formula are applied based on the positioning of the robot from its odometer and waypoint coordinates. From these calculations, distance in a straight line as well as the appropriate minimum angle to turn can be found. To know exactly how much to turn, the angle is found by taking the difference between the angle calculated from hypothenuse and the odometer's theta. **Figure 3** describes the logic behind the class and its methods.

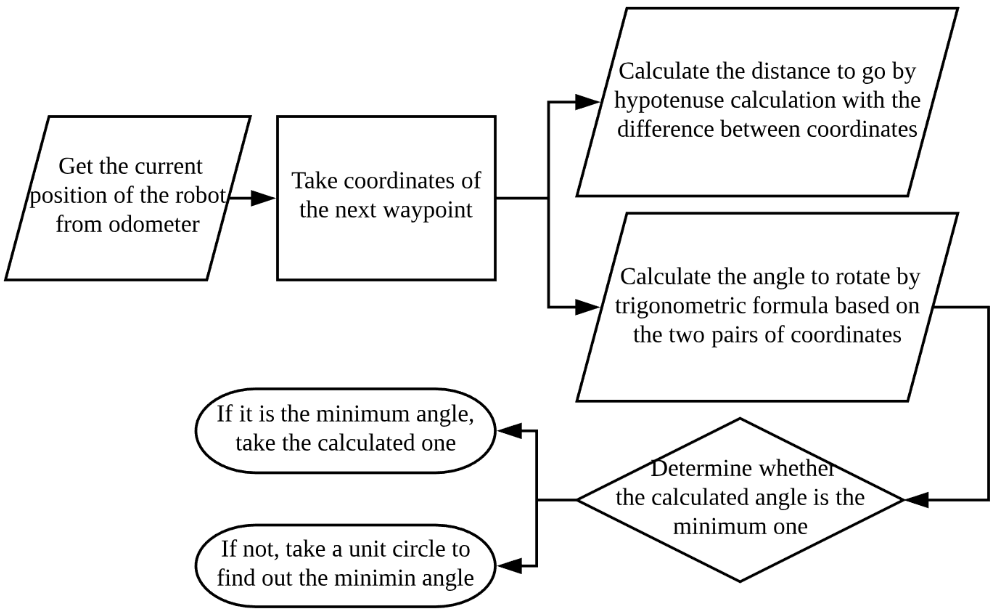

Get the current position of the robot from odometer
Take coordinates of the next waypoint
Calculate the distance to go by hypotenuse calculation with the difference between coordinates
Calculate the angle to rotate by trigonometric formula based on the two pairs of coordinates
Determine whether the calculated angle is the minimum one
If it is the minimum angle, take the calculated one
If not, take a unit circle to find out the minimin angle

Figure 3. Flowchart of Navigation.java

ObstacleAvoidanceNavigation.java improves on Navigation.java. As such, everything in Navigation.java is present in this class. However, while navigating towards a waypoint, the ultrasonic sensor is active and keeps scanning in a 110-degree cone aided by a motor installed upside down on the front of the robot. Data from this sensor is filtered through a basic filter set up identically to that in lab 1. If, the sensor detects an obstacle that is within the band center of 15 cm while navigating to a waypoint, the obstacle avoidance method is called which activates a Bang-Bang type controller that tries to avoid the obstacle by turning 90-degrees left and driving in an extended arc. It is to be noted that the ultrasonic sensor shifts to a 45-degree angle to better measure the distance between the robot and the obstacle as to prevent from crashing in it while avoiding it, therefore correcting its extended arc. **Figure 4** describes the logic behind the class and its methods.

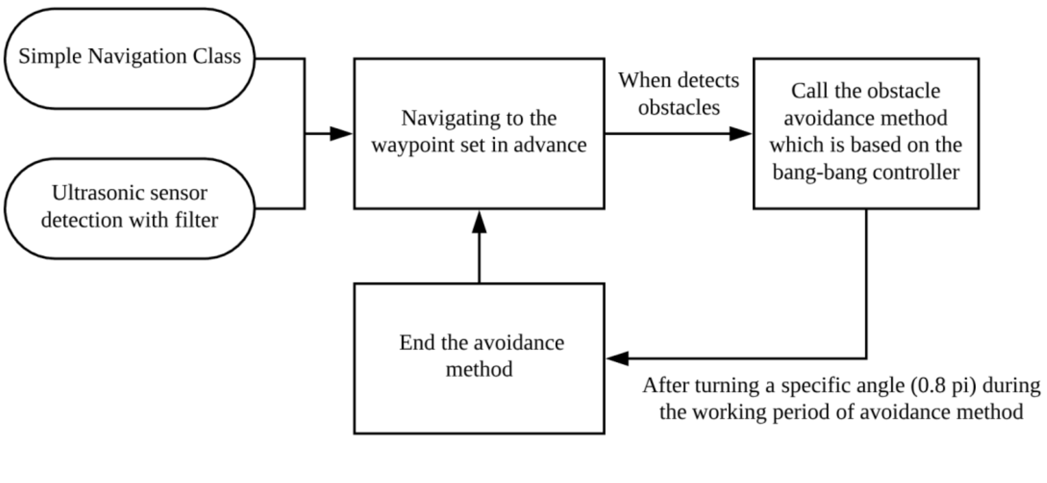

Simple Navigation Class
Ultrasonic sensor detection with filter
Navigating to the waypoint set in advance
When detects obstacles
Call the obstacle avoidance method which is based on the bang-bang controller
End the avoidance method
After turning a specific angle (0.8 pi) during the working period of avoidance method

Figure 4. Flowchart of ObstacleAvoidanceNavigation.java

Test data

Table 1. Navigation Test

Trials	X_D	Y_D	X_F	Y_F
1	60.96	0.00	60.75	0.15
2	60.96	0.00	60.65	0.10
3	60.96	0.00	60.72	0.18
4	60.96	0.00	60.70	0.15
5	60.96	0.00	60.75	0.13
6	60.96	0.00	60.58	0.10
7	60.96	0.00	60.62	0.05
8	60.96	0.00	60.78	0.16
9	60.96	0.00	60.54	0.24
10	60.96	0.00	60.81	0.22

The robot starts at the origin point and should end at waypoint (2,0)

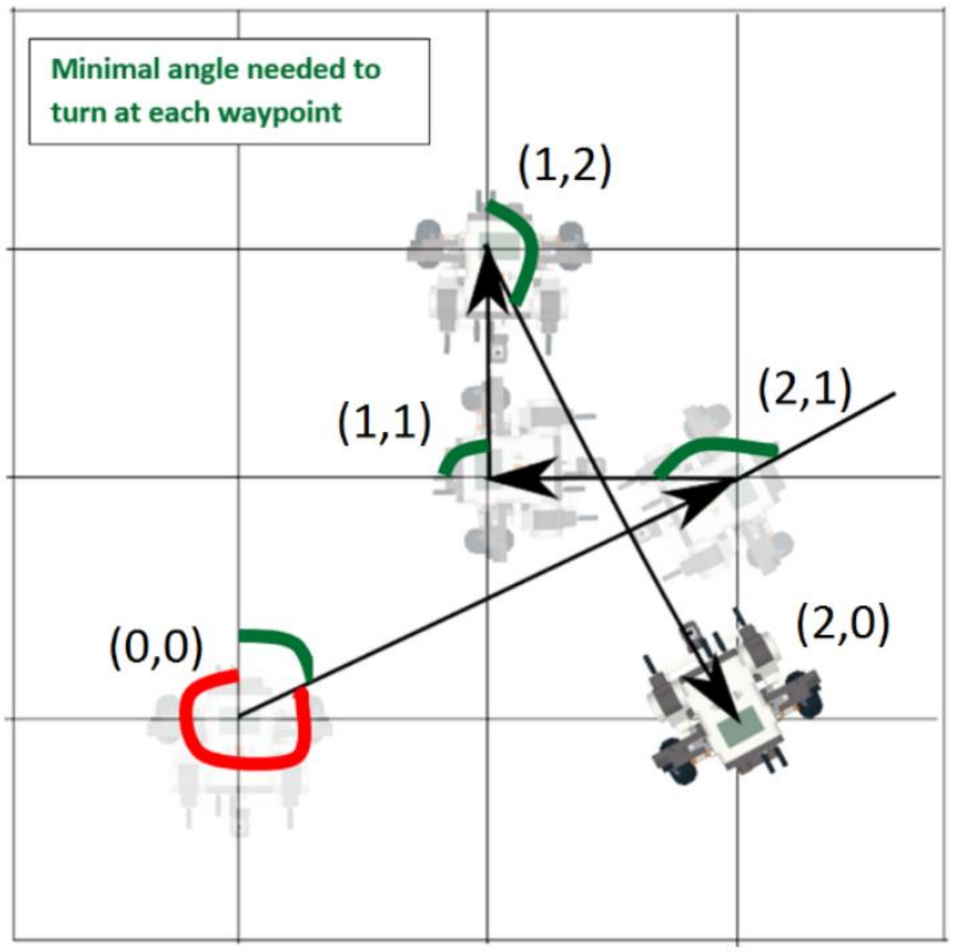


Figure 5. Testing waypoints and trajectory

Group: 51

Test analysis

Euclidean error distance ε

$$\epsilon = \sqrt{(X_D - X_F)^2 + (Y_D - Y_F)^2}$$

Table 2. Error distances

Trials	1	2	3	4	5
ε	0.26	0.33	0.30	0.30	0.25
Trials	6	7	8	9	10
ε	0.39	0.34	0.24	0.48	0.27

Table 3. Mean and standard deviation

Sample mean	0.32
Sample standard deviation	0.222

Sample calculation for ε, take trial 1 as the sample:

The difference on X-axis:

$$\Delta X = X_D - X_F$$

Then $\Delta X = 60.96 - 60.75 = 0.21$

The difference on Y-axis:

$$\Delta Y = Y_D - Y_F$$

Then $\Delta Y = 0.00 - 0.15 = -0.15$

So the Euclidean error distance ε for trial 1 is:

$\epsilon = \sqrt{(0.21)^2 + (-0.15)^2} = 0.26$

Sample calculation for the mean value of ε,

$$\mu = \frac{1}{n}\sum \epsilon$$

So

$$\mu = \frac{1}{10}\sum \epsilon$$
$$= \frac{1}{10}(0.26 + 0.33 + 0.30 + 0.30 + 0.25 + 0.39 + 0.24 + 0.34 + 0.48 + 0.27)$$
$$= 0.32$$

Group: 51

Sample calculation for the standard deviation of ε,

$$\boldsymbol{\sigma} = \sqrt{\frac{\sum(\boldsymbol{\epsilon} - \boldsymbol{\mu})^2}{\boldsymbol{n}}}$$

So

$$\sigma = \sqrt{\frac{\sum(\boldsymbol{\epsilon} - \mathbf{0.32})^2}{\mathbf{10}}}$$

$$= \sqrt{\frac{(\mathbf{0.26} - \mathbf{0.32})^2 + (\mathbf{0.33} - \mathbf{0.32})^2 + \cdots + (\mathbf{0.27} - \mathbf{0.32})^2}{\mathbf{10}}}$$

$$= \mathbf{0.222}$$

Observations and Conclusions

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

The errors are observed due to the odometer. The navigator is used to tell the robot how to get to the waypoint. Some values (values of current position) that are used for calculating path and angle in the navigation class are updated from the odometer. So if the values given by the odometer is not so accurate, then the final waypoint that the navigator leads to be away from the desired destination.
The main sources for the error are the error on the TRACK value (which refers to the distance between wheels) and the radius of both wheels. If the TRACK value differs from the accurate one, then there is error on the angle which robot needs to change for the next waypoint. The error on the radius of wheels will lead to the inaccuracy between two waypoints (If radius is large than reality, the real distance robot goes between two waypoints will be larger, vice versa). Also, the emergency condition will lead to large error, such as slippery and huge bumps.

How accurately does the navigation controller move the robot to its destination?

Both simple navigation (without obstacle-detection) and advanced navigation (with obstacle-detection) are quite accurate. Distance measured between the final point where the controller sent the robot to and the desired destination is no more than 2 cm among large amount of test.

How quickly does it settle (i.e. stop oscillating) on its destination?

There was hardly any obvious oscillation observed at the final destination when implementing both navigators.

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

Two navigation methods lead to two different results. There is no difference for simple navigation which doesn’t detect the obstacle only except that the process will be faster. However, the advanced navigation will be affected quite a lot. If the speed is increased, the ultrasonic sensor may not detect the obstacle. Also, when running with high speed, the robot will have less space to react even if the sensor has detected there is an obstacle on the path. As we are using a method like bang-bang controller in Lab 1 to solve the wall-following problem, the robot has high probability to hit the wall when bypassing the obstacle due to the high turning speed.

Further improvements

What steps can be taken to reduce the errors discussed above?
Identify at least one hardware and one software solution.
Provide explanations as to why they would work.

We could use track or some wheels made up of material that won't easily deformed instead of the wheels with rubber tires. The deformation on the rubber tiers when robot putting on the ground can be obviously observed, which means that the actual value for wheel radius is not quite the same as the measured one. Wheels of more solid material or tracks which won't easily deformed will help a lot with this problem. Change the position of the wheels may also help. As both wheels of our robot are not positioned beside the very center of the robot, the TRACK value is affected and have errors with the measured value. The wheels should be adjacent to the center of mass of the robot so that the turnTo method will work more accurately.
For the software part, we can improve the obstacle avoidance method. The avoidance method we are using can only let the robot bypass the obstacle from the left side, which means if the obstacle is on the very left of the board, the robot will have the risk of falling from the edge. Therefore, we can implement a judgement on the position of the robot and let the robot choose which side to bypass the obstacle. For example, when x is smaller than 1 tile or y is bigger than 1tile, it means that the robot will probably fall from the board if choose the left side to avoid the obstacle. As a result, the avoidance should choose the left side of obstacle to move when robot's path is at the edge of the board and robot's direction is clockwise, and choose the right side of the obstacle to move vice versa. By this way the robot will always stay in the testing area.