Design Principles and Methods

ECSE211

Tony Ou

260867785

Hongtao Xu

260773785

Lab 3 – Navigation & Obstacle Avoidance

Group 27

Design Evaluation

Hardware Design

For the hardware section of the robot, we reused most of the design from previous labs, which included the EV3 brick and large motors setup. However, we could not use a still sensor, as it would not be able to detect all obstacles or follow a wall correctly on either side. We tried using the medium motor at first, but the rotational force was not enough to move the sensor. Therefore, we added an additional large motor instead to the front of the robot using a solid mounting system.

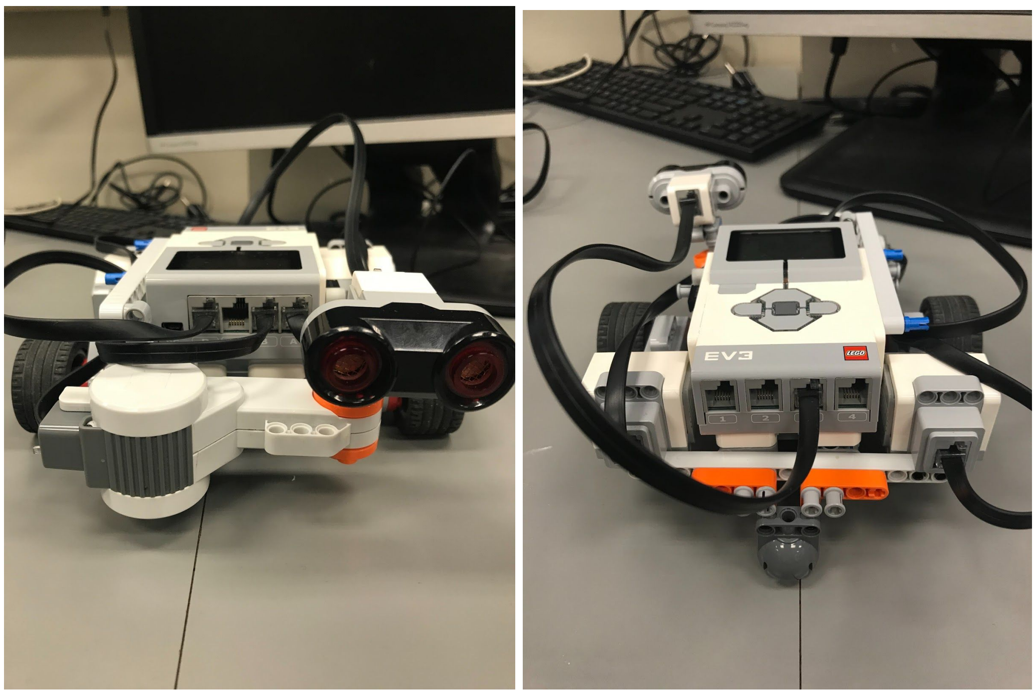

Figure 1.1 Hardware of the navigation robot

Software Design

For the software design, we first gathered the base code from previous labs, as we did not have any starter code. We modified the display, main and odometer classes by removing any useless code and adapting it to this lab, such as the display information on the brick. We also have a resources class, where we add any important constants, such as the odometer, sensor ports or motor speeds. There are 4 main functions, which include the travelTo, turnTo, avoidObstacle and run functions. We start off by inputting the waypoint coordinates in the run function by repeatedly calling the travelTo function at different points. The travelTo function then calculates the distance and angle needed to get to a point. It first makes the robot turn in the right direction and then make the right amount of rotations to get to the waypoint. If at any time, the robot's ultrasonic sensor readings are below the band center, it runs the obstacle avoidance function, which prepares it for wall following by making the robot turn 90 degrees left from its current position and by making the sensor turn 50 degrees towards the wall. The function then uses the bang-bang controller logic to avoid the obstacle until it reaches a calculated end angle. After the obstacle is dodged, the travelTo function is called again to finally reach the waypoint. Many helper functions were used as well for many different tasks, such as unit conversion and angle calculations.

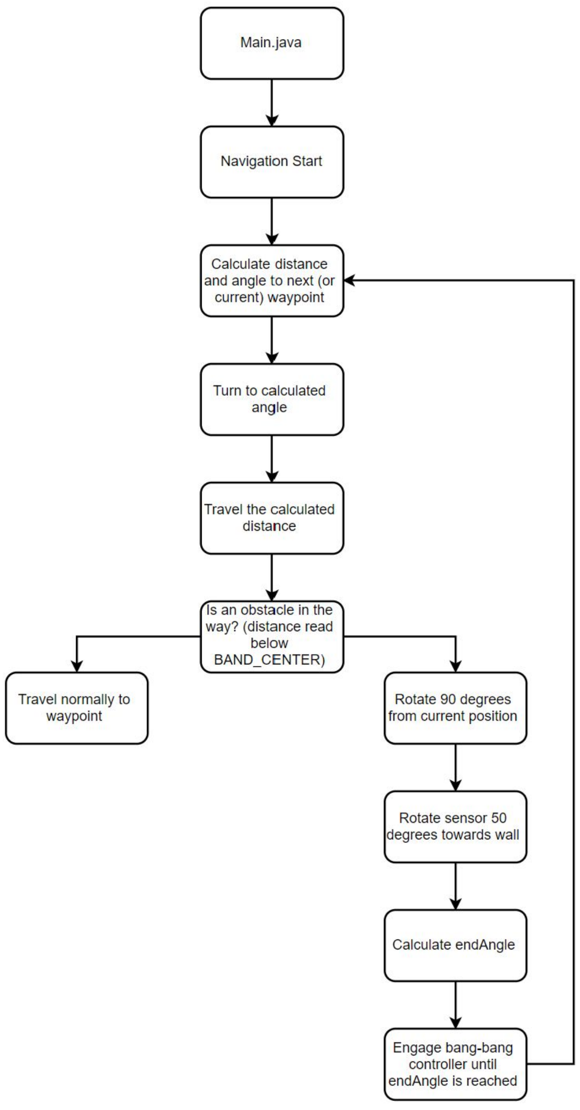
Main.java
Navigation Start
Calculate distance and angle to next (or current) waypoint
Turn to calculated angle
Travel the calculated distance
Is an obstacle in the way? (distance read below BAND_CENTER)
Travel normally to waypoint
Rotate 90 degrees from current position
Rotate sensor 50 degrees towards wall
Calculate endAngle
Engage bang-bang controller until endAngle is reached

Figure 1.2. Software flowchart

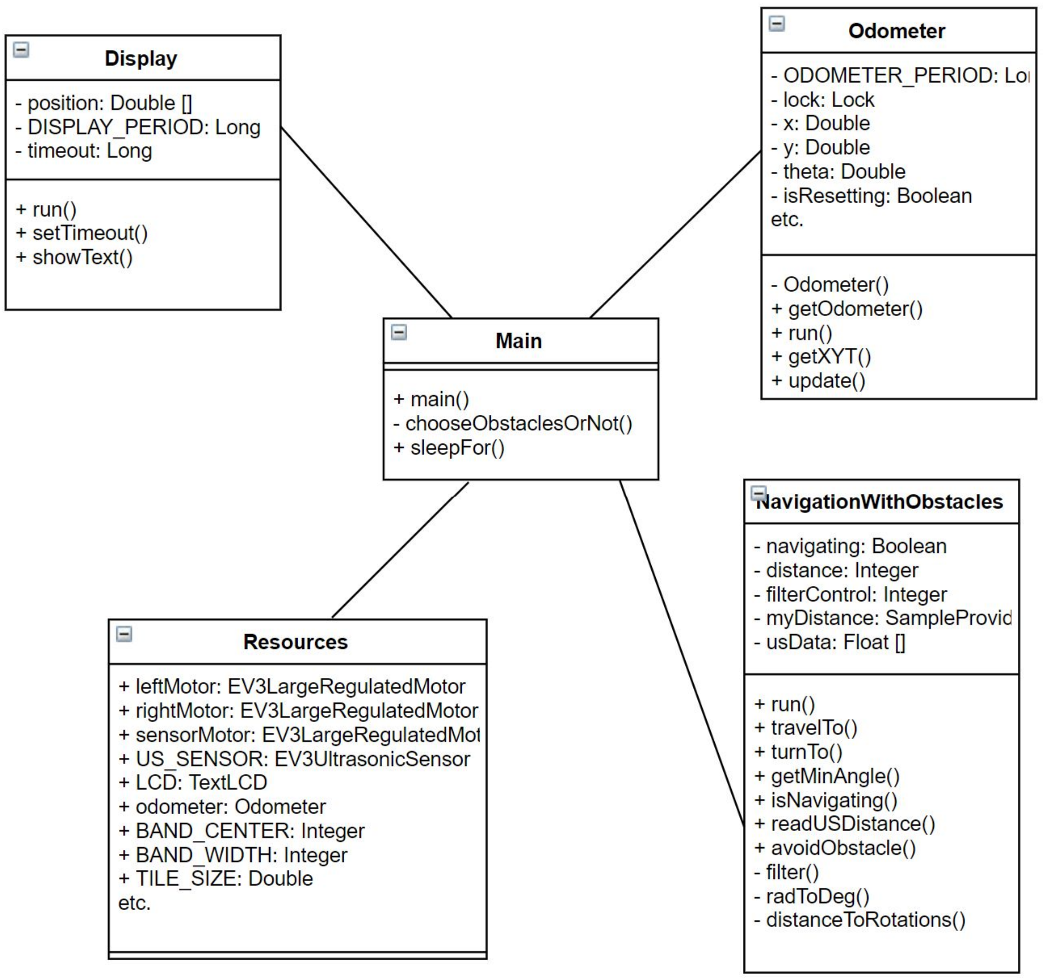

Figure 1.3. Software class diagram

Workflow

For our workflow, we started by removing all previous sensors from the robot. We then tried different designs to implement a moving sensor. After multiple failed attempts using the medium motor, we opted to mount the ultrasonic sensor at the front using a large motor. We then layed out the base software from previous labs, which included the display, main, odometer and resources classes. We implemented the three main classes for Navigation (TurnTo and TravelTo) first and added helper functions along the way. After determining that the navigation class worked correctly with a few test runs, we implemented an additional class to avoid obstacles using the code from the Bang-bang controller. Finally, we ran the code on each demo maps using different block setups to make sure that the obstacle avoidance system worked correctly.

Test data

Figure 2.1. Navigation tests

Theoretical destination waypoint - X: 91.44 cm - Y: 30.48 cm

Run Number	Destination X (cm)	Destination Y (cm)	Final X (cm)	Final Y (cm)
1	91.44	30.48	91.14	30.36
2	91.44	30.48	91.54	30.59
3	91.44	30.48	91.38	30.46
4	91.44	30.48	91.59	30.59
5	91.44	30.48	91.37	30.50
6	91.44	30.48	91.21	30.45
7	91.44	30.48	91.45	30.56
8	91.44	30.48	91.48	30.53
9	91.44	30.48	91.19	30.38
10	91.44	30.48	91.25	30.43

Test analysis

Figure 3.1 Eucledean error distance e of each trial

Run Number	Eucledean error distance e
1	0.323

2	0.149
3	0.063
4	0.186
5	0.073
6	0.232
7	0.081
8	0.064
9	0.269
10	0.196

Figure 3.3 Mean and standard deviation for error e

Mean	Standard deviation
0.1636	0.08833

Mean value of E = sum of error in 10 trials / trial number
= 1.636/10
= 0.1636

Standard dev of E = $\sqrt{\frac{\Sigma (x-\overline{x})^2}{n}}$

(0.323 - 0.1636)^2 + (0.149 - 0.1636)^2 + (0.063 - 0.1636)^2 + (0.186 - 0.1636)^2 + (0.073 - 0.1636)^2 + (0.232 - 0.1636)^2 + (0.081 - 0.1636)^2 + (0.064 - 0.1636)^2 + (0.269 - 0.1636)^2 + (0.196 - 0.1636)^2 = 0.0780324

0.0780324 / 10 = 0.00780324

sqrt(0.00780324) ≈ 0.08833

Observations and conclusions

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

We determined that any error would be caused by the odometer, as the navigator uses the odometer's values to calculate its turning point and travel distance. The formulas used are theoretical and are a lot less error prone than the readings of the odometer, which can be affected by wheel slippage. Wheel slippage can be caused by dust on the wheels or travel speeds that are too high. The regulated motor's power can also degrade over time and provide less performance at small battery percentages.

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

The navigation controller moves the robot very accurately to the destination waypoint, as the distance error averaged only around 0.16 with a low standard deviation of around 0.08833. Therefore, the controller constantly navigates the robot to the right destination with minimal error.

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

The robot almost never oscillates when reaching a waypoint. It usually arrives at the destination, stops, and rotates to the next point. The same behavior occurs when it reaches its final destination. It settles instantly upon reaching the final point, keeping the same direction it had previously.

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

Increasing the speed of the robot will increase the friction of the wheels with the surface. Therefore, there are more chances for wheel slippage to occur and affect the accuracy negatively. Also, due to the fact that our robot covers a distance using wheel rotations and that it does not slow down when approaching a waypoint, the stopping force could affect its position as well (at larger speeds).

Further improvements

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

A possible hardware improvement would be to increase the wheel size of the robot and possibly the motor size as well to compensate. Therefore, there will be a larger grippy surface in contact with the ground, which will keep it in the right trajectory. A pair of new bigger motors will also provide more power for the wheels and have less chances to underperform.

A software solution would be to reduce the traveling speed and turning speed of the robot. As mentionned before, the small error was most likely caused by wheel slippage, which is affected by friction. Reducing the overall speed of the robot would increase its friction when traveling and therefore reduce the chances of it going slightly off path. We could also decrease the speed when approaching a waypoint to avoid having the EV3 slipping when suddenly stopping.