Group 26
Rene Gagnon 260801777
LiLy Li 260664379

Lab 3 : Navigation

Section 1-Design evaluation:

For this lab, we basically kept the same design as the previous one (simple three-wheel design). We simply added an ultrasonic sensor to the front. This sensor was mounted on a medium motor to allow rotation along a vertical axis. We kept the light sensor from the previous lab mounted between the 2 motors even thought we ended up not using it. In the previous lab, we had an issue with the robot not going straight. We tried using different port for the motors and we reinforced the arms around the wheels. Also, we added some Lego pieces on the right side to increase the friction on this side (the robot was deviating to the left). These adjustments made the robot go straight enough for this lab. The hardware was adjusted progressively during the lab.

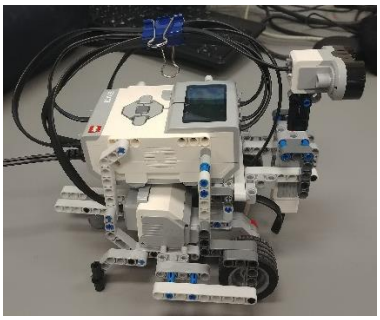

Figure 1: Right Side

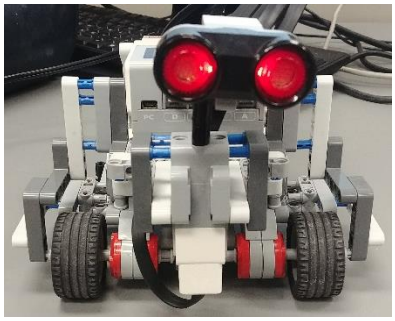

Figure 2: Front

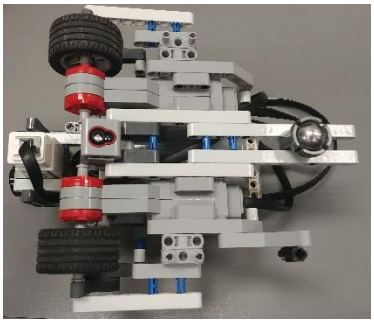

Figure 3: Bottom

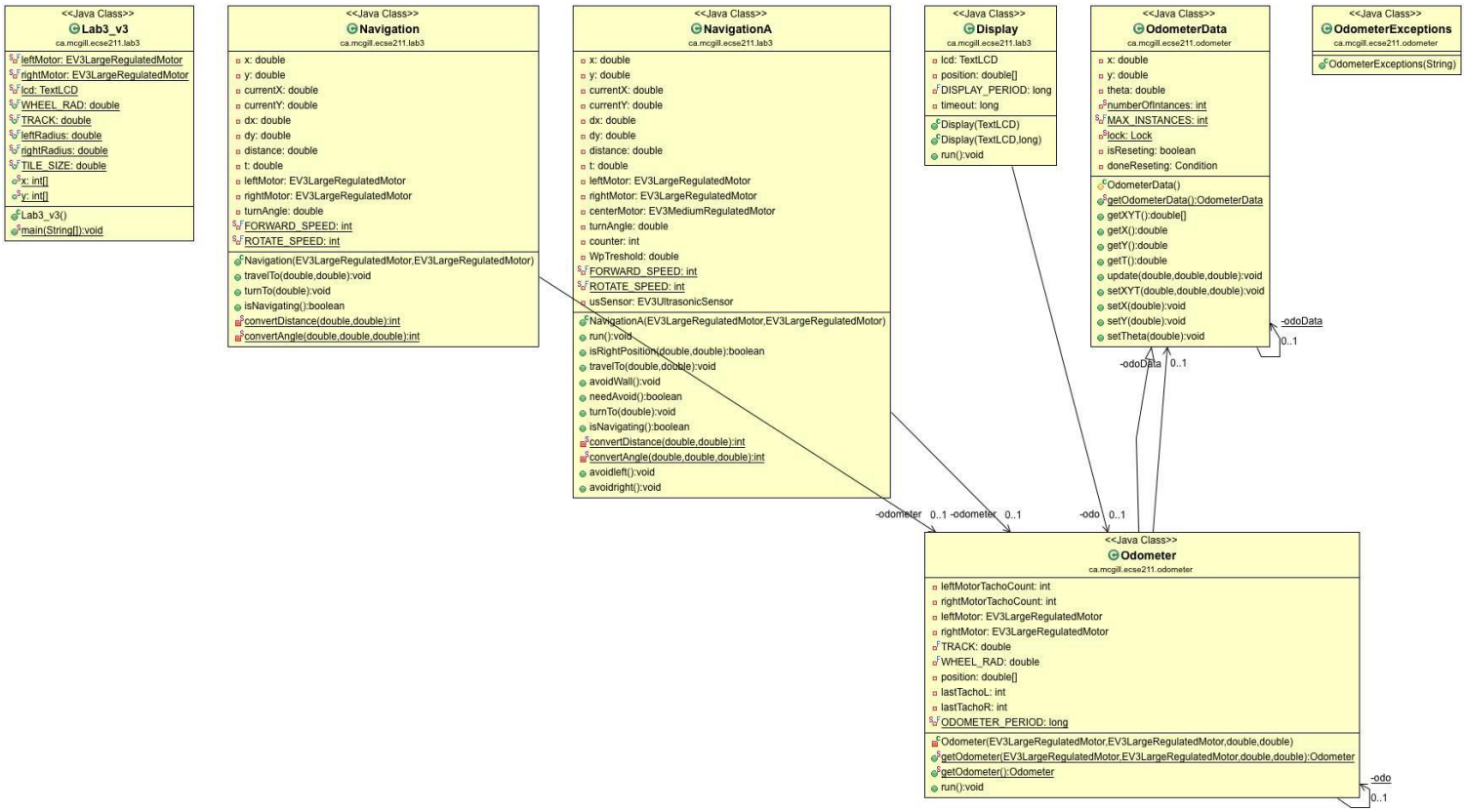

Figure 4: class diagram

For both simple navigation and navigation with obstacle avoidance, the calculation of the distance the robot needs to travel to is the same. First, three more methods are added in the odometer class to get the x, y and theta values individually. In the travelTo method in the navigation class, we simply get the current x and y values from the odometer and calculate the difference from the destination position to the current position, then use the distance equation to calculate the value the robot needs to travel. Before the motors start rotating to the destination, use the turnTo method to change the heading of the robot. To calculate the angle the robot needs to turn to, we used Math.atan2 of the difference in x and difference in y, and then subtract this angle from the current theta to get the angle the robot needs to rotate to. For navigation with obstacle avoidance, we added a needAvoid method which returns a boolean to indicate whether the robot needs to execute the avoidWall method. In the travelTo method, before the robot starts navigating, check if the ultrasonic sensor detects a block in front of it, if not then start both motors to navigate to the next waypoint. By setting immediate return to true, the program continues executing while the robot is navigating. We used a while loop to check if the robot needs to avoid a block. Then, we used a method called isRightPosition to check if the robot has reached its destination, if it is, then increment the counter to keep track of which waypoint the robot is on. For avoiding an obstacle, in the avoidWall method, first, check the position of the robot to determine whether the robot should avoid from the left or from the right. If the robot needs to avoid from the right, first of all keep the distance from the ultrasonic sensor, then turn the robot 90 degrees and then turn the sensor –90 degrees so that the sensor is facing the block, first, we make the robot to move 25 cm and check again if it still sees the block, if not then turn the robot by –90 degrees and make it move 2 times the distance we kept before, so that the robot can completely pass the obstacle.

Section 2- Test data:

Robot travelling to waypoints (2,1), (1,1), (1,2), (2,0)

Test	Battery Voltage (V)	Last Waypoint X_D (cm)	Last Waypoint Y_D (cm)	Robot X_F (cm)	Robot Y_F (cm)	Euclidean distance error (cm)
1	7.7	60.96	0	57.66	-4.5	5.580322571
2	7.7	60.96	0	59.00	-2.0	2.800285700
3	7.6	60.96	0	56.90	-3.0	5.048128366
4	7.6	60.96	0	57.80	-3.0	4.357246837
5	7.6	60.96	0	58.00	-1.0	3.124355934
6	7.7	60.96	0	56.50	-2.0	4.887903436
7	7.8	60.96	0	57.00	-2.0	4.436394933
8	7.6	60.96	0	58.00	-0.5	3.001932711
9	7.5	60.96	0	61.00	0.5	0.501597448
10	7.5	60.96	0	61.00	0.2	0.203960781

Table 1: Data from 10 independent navigation runs

For all these tests, the forward speed was 200 deg/sec and the turning speed 100 deg/sec. We found that these speeds worked the best for our robot. It did not slip too much, and it reached the waypoints relatively quickly. The wheel base was 12.4 cm and the wheel radius were 2.156 cm. These values were obtained through testing. We added a correction factor of 0.6 cm to any distance traveled to account for slipping and the odometer errors.

Section 3-Test analysis:

Euclidean error distance mean (cm)	Euclidean error distance standard deviation (cm)
3.394212872	1.852433529

Table 2: Mean and standard deviation of the 10 errors shown in table 1

Sample calculations:

- Euclidean error distance for test 1:
  $\varepsilon_1 = \sqrt{(X_D - X_F)^2 + (Y_D - Y_F)^2} = \sqrt{(60.96 - 57.66)^2 + (0 - (-4.5))^2} = 5.580322571\ cm$
- Euclidean error distance mean:
  $mean = \bar{\varepsilon} = \frac{\sum_1^n \varepsilon_i}{n} = \frac{\sum_1^{10} \varepsilon_i}{10} = \frac{33.94212872}{10} = 3.394212872\ cm$
- Euclidean error distance standard deviation:
  $Standard\ deviation\ of\ \varepsilon = \sqrt{\frac{\sum_1^n |\epsilon_i - \bar{\epsilon}|^2}{n}} = \sqrt{\frac{\sum_1^{10} |\epsilon_i - \bar{\epsilon}|^2}{10}} = 1.852433529\ cm$

Section 4-Observations and conclusions:

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

The errors observed are mostly due to the odometer since the navigation used the odometer data to calculate the distance that needed to be traveled. The odometer was mainly off because the robot did not go straight. Slipping also caused a bit of errors, especially when turning. This would make the robot think it is at a different location then it is. The navigation would then calculate wrong distances and orientations, creating errors in the final position.

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

At first, the navigation controller always moved the robot to the destination within a margin of Euclidean error smaller than 10 cm. With some adjustments we were able to bring this down to around 1 or 2 cm of error.

- *How quickly does it settle on its destination?*

With our design, the robot does not oscillate while navigating to a waypoint because before the robot starts moving to the destination point, the robot turns to head to the destination point. Once the robot finish rotating to the desired heading, then it simply goes straight until it reached the waypoint. Also, after avoiding a block, before calculating is trajectory to current waypoint, we called a method that checked if we were within 4 cm of the waypoint. This allowed the robot to start navigating to the next waypoint and not waste anytime on the previous waypoint.

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

From our testing, we found that larger the speed was, the greater the error was. With higher speeds, when the robot would slip, the distance that the wheel rotated while slipping was bigger. This would result in a bigger error in the X and/or Y positions. Also, if only one of the wheel slipped, then the robot would be deviating by a lot from its trajectory. This would again create errors in the odometer.

Section 5-Further improvements:

To reduce the Euclidean error distance, we could implement improvements in the software part or the hardware part of the robot. For the software, we could implement odometry correction from the previous lab. The algorithm would have to be slightly modified because now, the light sensor might also follow lines, instead of simply crossing them. We could account for this scenario by only correcting the X or Y when the light sensor sees a sequence of wood, black line, wood colors within a certain time frame. But, except for this, the odometry correction would work exactly as in the previous lab, meaning it would correct the position based on the direction and how many lines it has crossed it this direction before. This solution would compensate for the inaccuracy of the odometer.

On the hardware side, we could try adding a second light sensor. The second light sensor would be placed in the back of the robot. With this second sensor, we could correct the odometer of the robot twice at every line. Knowing the distance between the front light sensor and the back one, we could adequately correct the X or Y coordinate of the sensor. This would surely increase the precision of the odometer since the frequency at which we correct it, is higher. An obvious improvement would also be to clean the surface and the robot's wheels. This would prevent any slipping from happening, resulting in a more precise odometer.