Design Principles and Methods

ECSE211

Tony Ou

260867785

Hongtao Xu

260773785

Lab 4 – Localization

Group 27

Design Evaluation

Hardware Design

For the hardware portion, we followed the same base design as all other labs, but added parts to incorporate the ultrasonic and color sensor. We used the same mechanism as lab 2 to mount the color sensor to the front and new parts to mount the ultrasonic sensor above the color sensor. The ultrasonic sensor had to be at the right height and to the center of the robot for maximum precision. If it was too high, it would not detect the walls and if it was not centered, the robot would turn too much or not enough when calculating the angles.

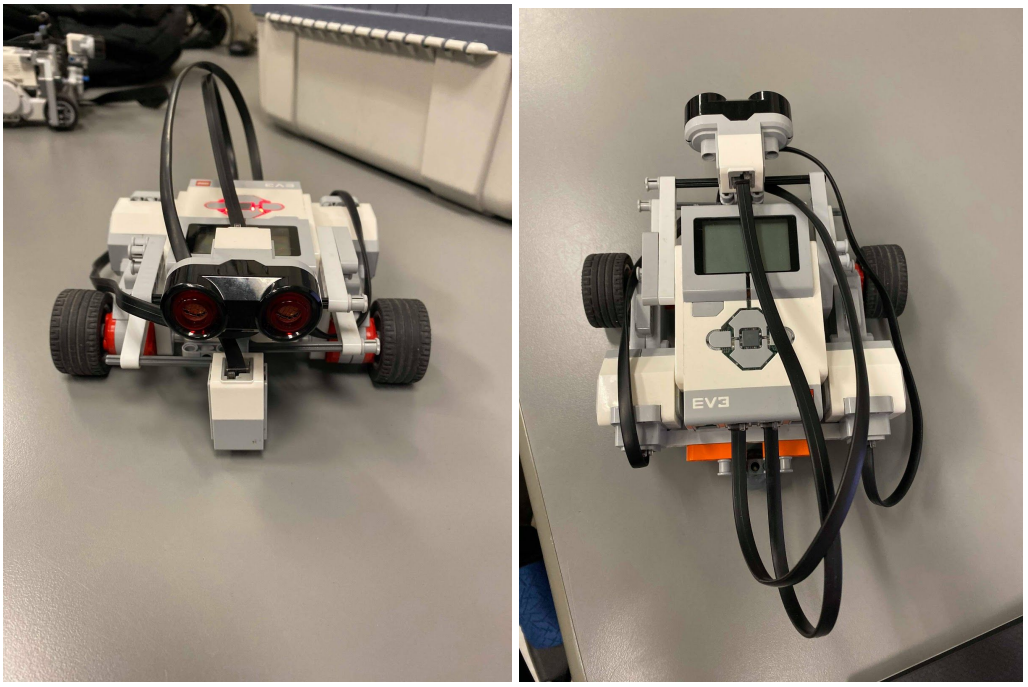

Figure 1.1 Final design of the localization robot

Software Design

After gathering all relevant classes from previous labs, we cleaned the code by removing any functions or variables that won't be used for this lab. We imported the Odometer, which updates the robot's position; the Display, which shows the odometer's

values on the brick and the Resources class, which contains all relevant constants that are added along the way. Once an instance of UltrasonicLocalizer is created, the “localize” function is executed: it first sets the motor speeds and then finds 2 reference angles using helper functions depending on the method used. If it is rising edge, the robot starts turning left until a wall is detected. It keeps turning until an open space is detected after the wall, where it records the value of the first angle. The robot then turns in the other direction following the same detection pattern: a wall followed by an open space, where it takes the second angle. The falling edge method uses the opposite of the rising edge method’s logic to find the 2 angles. The robot starts turning right until an open area is detected. It keeps turning until a wall is detected, where it records the value of the first angle. The robot then turns in the other direction until it detects an open area. It keeps turning until a wall is detected as well, where it records the second angle. Afterwards, the 2 angles are used to compute deltaT using the formula from the slides and the resulting value is added to the current theta from the odometer. The odometer is updated with the value, which contains the correct angle that the robot is currently facing. Finally, we make the robot turn to the 0 degree angle using functions from the Navigation lab and stop the motors.

The second part of the code is executed when the user presses the down button on the brick after the ultrasonic localizer is done. It creates an instance of the light localizer and executes its localize function as well. The robot first travels to an estimated origin position using the goToOrigin helper function. The robot turns 225 degrees to face a location around the back left corner of the wall and backs off until a line is detected. It then slightly moves forward to correct the error caused by the distance between the color sensor and the wheels. Once the robot is positioned at the estimated origin point, it starts rotating right until the angles of all 4 lines are recorded in an array. These angles are then used to calculate the robot’s current position relative to the origin. The odometer is updated with accurate readings from the previous calculations and the travelTo function is called to move the robot to the correct origin point. Finally, the robot turns until it points north using a while loop to block the halt of the motors as long as the correct angle has not been read.

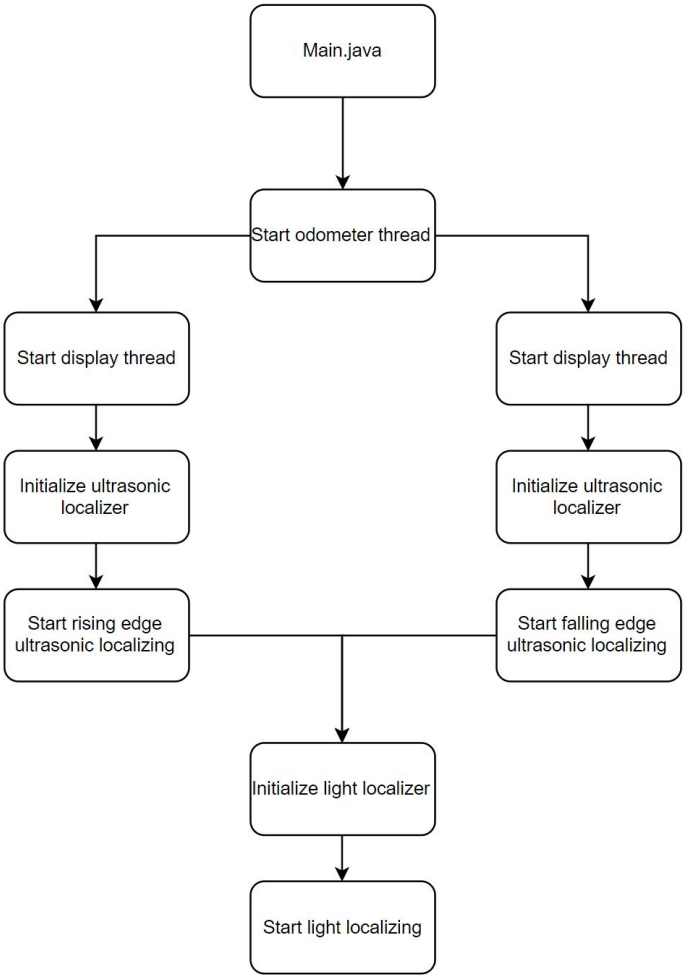

Figure 1.2. Software flowchart

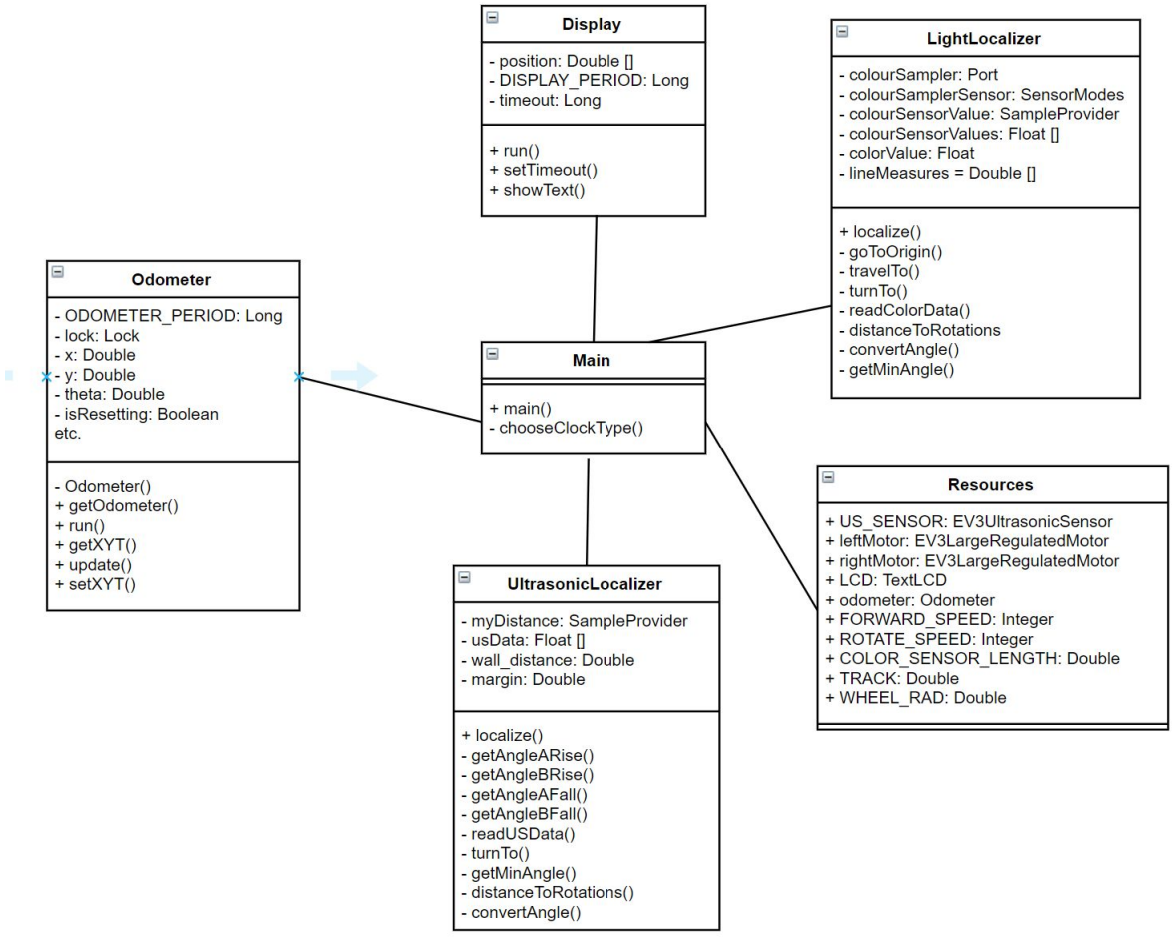

Figure 1.3. Software class diagram

Workflow

We started the design by removing all additional pieces and kept the base design of the robot. We then added the color sensor using the same procedure as lab 2. We tried different designs to fit the ultrasonic sensor above the color sensor and settled on the most solid one. Afterwards, we imported all relevant classes from previous labs (Odometer, Display, Resources) and started implementing the code for the ultrasonic localizer. After tuning the wall distance and testing the code, we wrote the code for the light localizer, which we also tested separately. Finally, we combined both modules into the main class and ran the code using different configurations to make sure everything worked correctly.

Test data

Figure 2.1. Localization tests using rising edge

Expected Ultrasonic Angle: 0°

Expected final position: (x, y, t) = (30.48 cm, 30.48 cm, 0°)

Run Number	Ultrasonic Angle (°)	Final X (cm)	Final Y (cm)	Final Angle (°)
1	358	32.54	34.44	358
2	356	31.32	30.50	355
3	358	30.75	30.64	356
4	1	31.72	30.50	357
5	352	30.56	30.52	354
6	5	31.18	33.65	4
7	357	30.57	31.54	358
8	359	30.6	33.61	356
9	2	30.51	30.57	4
10	357	30.50	30.49	355

Figure 2.2. Localization errors using rising edge

Run Number	Ultrasonic Angle Error (°)	Euclidean Distance Error (cm)	Final Angle Error (°)

1	2	4.46	2
2	4	0.84	5
3	2	0.31	4
4	1	1.24	3
5	8	0.09	6
6	5	3.25	4
7	3	1.06	2
8	1	3.13	4
9	2	0.09	4
10	3	0.02	5

Figure 2.3. Localization tests using falling edge

Expected Ultrasonic Angle: 0°

Expected final position: (x, y, t) = (30.48 cm, 30.48 cm, 0°)

Run Number	Ultrasonic Angle (°)	Final X (cm)	Final Y (cm)	Final Angle (°)
1	3	31.60	32.09	2
2	5	30.51	31.54	3
3	4	31.02	32.69	5
4	5	33.51	31.59	6
5	2	30.52	34.84	4

6	4	30.49	30.52	2
7	3	31.71	31.49	2
8	6	31.05	30.75	5
9	2	30.94	30.64	2
10	1	30.53	31.26	3

Figure 2.4. Localization errors using falling edge

Run Number	Ultrasonic Angle Error (°)	Euclidean Distance Error (cm)	Final Angle Error (°)
1	3	1.96	2
2	5	1.06	3
3	4	2.28	5
4	5	3.23	6
5	2	4.36	4
6	4	0.04	2
7	3	1.59	2
8	6	0.63	5
9	2	0.49	2
10	1	0.78	3

Test analysis

Figure 3.1. Mean and standard deviation for error e (Rising edge)

Variable	Mean	Standard deviation
Ultrasonic Angle Error (°)	3.10	2.02
Euclidean Distance Error (cm)	1.45	1.51
Final Angle Error (°)	3.90	1.22

Mean value = sum of error in 10 trials / number of trials

Sample calculation (Ultrasonic angle error) = (2 + 4 + 2 + 1 + 8 + 5 + 3 + 1 + 2 + 3) / 10 = 3.10

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

Sample calculation (Ultrasonic angle error) = (2.00-3.10)^2+(4.00-3.10)^2+(2.00-3.10)^2+(1.00-3.10)^2+(8.00-3.10)^2+(5.00-3.10)^2+(3.00-3.10)^2+(1.00-3.10)^2+(2.00-3.10)^2+(3.00-3.10)^2 = 40.90

40.90 / 10 = 4.09

sqrt(4.09) = 2.02

Figure 3.2. Mean and standard deviation for error e (Falling edge)

Variable	Mean	Standard deviation
Ultrasonic Angle Error (°)	3.50	1.50
Euclidean Distance Error (cm)	1.64	1.28
Final Angle Error (°)	3.40	1.43

Mean value = sum of error in 10 trials / number of trials

Sample Calculation (Ultrasonic angle error) = (3 + 5 + 4 + 5 + 2 + 4 + 3 + 6 + 2 + 1) / 10 = 3.50

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

Sample calculation (Ultrasonic angle error) = (3.00-3.50)^2+(5.00-3.50)^2+(4.00-3.50)^2+(5.00-3.50)^2+(2.00-3.50)^2+(4.00-3.50)^2+(3.00-3.50)^2+(6.00-3.50)^2+(2.00-3.50)^2+(1.00-3.50)^2 = 22.50

22.50 / 10 = 2.25

sqrt(2.25) = 1.50

Observations and conclusions

1. Which of the two localization routines performed the best?

When doing experimental tests while debugging the code, none of the routines performed better than the other. However, the rising edge method would usually have a final ultrasonic angle less than 0, while the falling edge method would have an angle above 0. Other than that, they performed exactly the same. If we rely solely on the test results taken for the report, we can say that the rising edge method is a bit more precise, having a mean of 3.10 for the final ultrasonic angle, which is 0.40 less than the falling edge method.

2. Was the final angle impacted by the initial ultrasonic angle?

The final angle of the robot was impacted by the ultrasonic angle, as we update the odometer's theta to 0 once the ultrasonic localization is complete. Therefore, the initial error was carried on to the light localization because it uses the last position of the robot after executing the ultrasonic localization as reference for the right odometer theta. The robot turns on itself until it reaches angle 0 on the odometer after correcting its x/y position. We compensated for this problem in the code by augmenting the threshold angle where the robot

stops turning (the robot stops between 360 and 10 degrees), which is why the experimental errors for the final light angle were significantly lower.

3. What factors do you think contributed to the performance of each method?

We expect the motor force and friction to the wheels to play a small role in the error for each method, as the robot turns a large distance (basically 2 arcs) and wheel slippage could have occured. However, the major contributing factor of the error would be the delay when the ultrasonic sensor readings are passed to the robot. For example, it is improbable that the robot stops right away once a wall is detected. There will be a short lag once the ultrasonic sensor detects the wall and the robot will keep turning for that short amount of time before stopping. The ultrasonic sensor also updates its readings at a specific frequency, which adds on to the lag: the distance might already been surpassed, but we need to wait for the next reading to stop the robot.

4. How do changing light conditions impact the light localization?

It will prevent the robot to accurately read lines, which is a crucial factor in light localization. The robot uses the lines to stop itself when travelling towards the estimated origin location and to correct its location when scanning the 4 lines. Therefore, if light conditions are changed, we will have to adjust the threshold that determines if a line is detected. Furthermore, if we were using a specific light color (i.e. red, yellow, blue), changing the ambient light to that same color will make it difficult for the robot to read the lines.

Further improvements

1. Propose a software or hardware way to minimize errors in the ultrasonic sensor.

We could use a function to predict the next value so that the speed of the robot can be reduced if an expected value is about to be met during its trajectory to make sure it faces the right direction before recording an angle (i.e. when turning away from the wall, it will expect

the distance to readings to increase; when turning towards the wall, it will expect the distance readings to decrease). For example, when the robot sees that its distance readings are steadily increasing, it will start reducing its speeding when almost at the threshold. Therefore, the robot will travel less after the ultrasonic sensor detects a distance at the threshold. We assume that there is a delay when the sensor readings are passed to the robot.

2. Propose another form of localization other than rising-edge or falling-edge.

There exists the Markov localization, which uses a probability distribution over the space of all hypotheses on where the robot might be instead of maintaining a single hypothesis. More precisely, the robot maps a uniform distribution over all its positions and raises the probability of a given location once detected. We then map a different set of probabilities after the robot moves a short distance. Finally, we multiply both sets of probabilities and keep the highest one as the most probable position of the robot.

3. Discuss how and when light localization could be used outside of the corner to correct Odometry errors, e.g. having navigated to the middle of a larger floor.

Given that the floor has grid lines, we can use the light localization to correct any Odometry errors if it loses track of the lines as long as the robot finds a way to detect the coordinates that it will be heading to. We will also need the robot to point around the north direction, the more precise the better. We can then execute the light localization normally, as it will always back off to the top right corner of the tile and update its odometer values with the previously detected coordinates. Therefore, as long as we have the coordinates of the top right corner of the current tile the robot is in and have it turn north, we can use light localization to correct Odometry errors.