Design Principles and Methods Michael Li and Cecilia Jiang

ECSE 211 (Fall 2020) 260869379, 260795889, Group #36

Lab 4: Localization

Prof. Frank Ferrie

McGill Faculty of Engineering

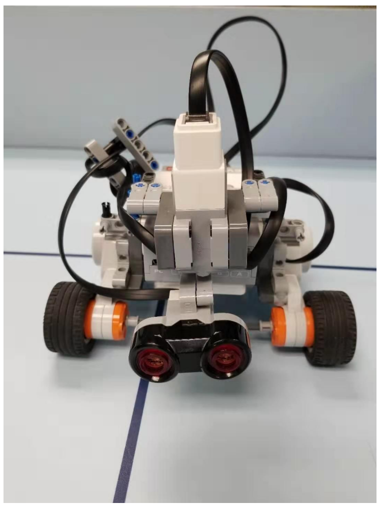
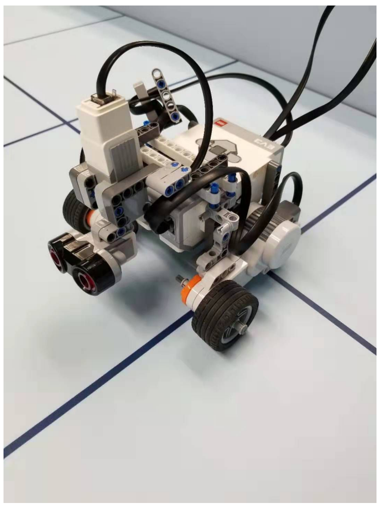
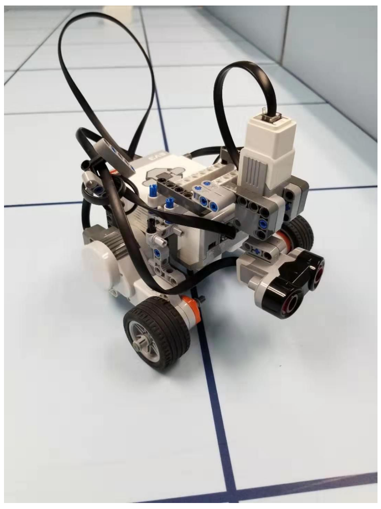
Due October 10th, 2019

Section 1: Design Evaluation

1.1 Hardware Design

The robot was designed to have a basic, wide but strong structure shown in Fig. 1 below. The main design objective is to ensure a robust robot structure. Cross shape joints were used to connect the main brick to the wheels to prevent them from falling apart or experience any minor displacement. The two front wheels are assembled to two sides of the brick and we used a single rollable steel ball as the back wheel. In fact, the reason why the steel ball was chosen instead of the plastic wheel or a third wheel is that the steel ball can rotate in all directions Thus, this design decision can significantly reduce sliding and friction. Besides, the height of the steel ball is perfect. After attaching it to the rear of the brick, the brick remained parallel to ground level.

The light sensor was fixed at the back of the robot in order to get the greatest circumference possible. In fact, when the robot is executing light sensor localization, it is rotating 360 degrees around its center of rotation which is the middle of the wheelbase. The light sensor is installed as far as possible from the center of rotation to increase the circumference of the light sensor's path when the robot is rotating. Therefore, the robot has a greater chance of detecting all four grid lines when it is executing light localization. On the other side, the ultrasonic sensor is mounted in front of the robot between the wheelbase (see Fig. 1). In fact, it was mounted at this position since it represents the orientation of the robot with respect to the Y-axis. Finally, the EV3 Medium Servo Motor from lab 3 is kept in the same place in order to hold the ultrasonic sensor at the perfect height and width.

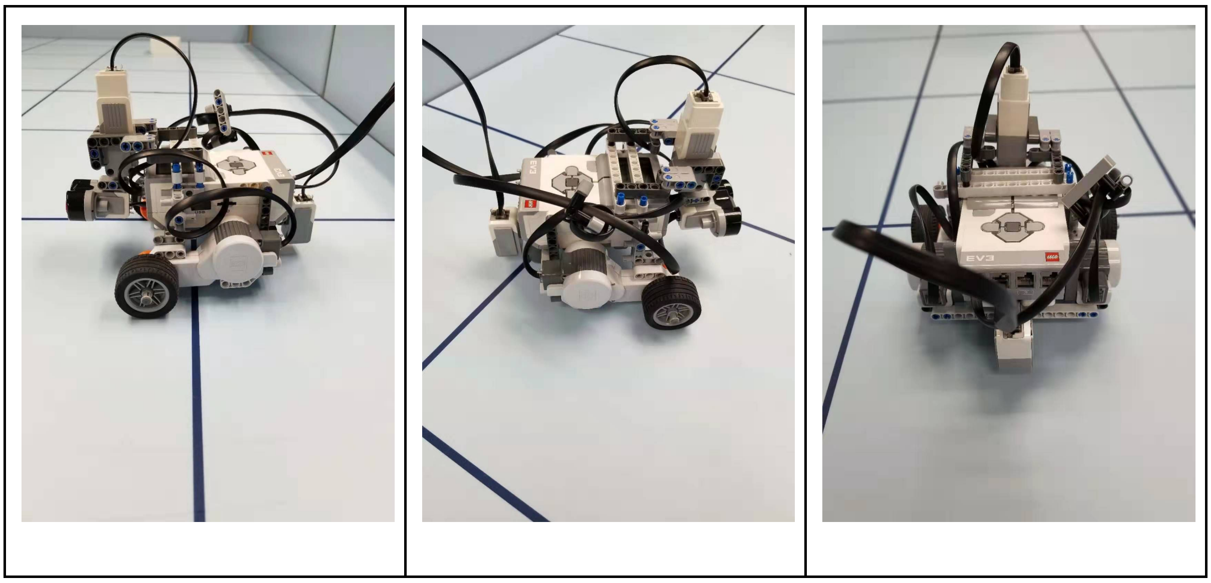


Figure 1: Robot’s hardware design viewed from 6 different angles.

1.2 Software Design

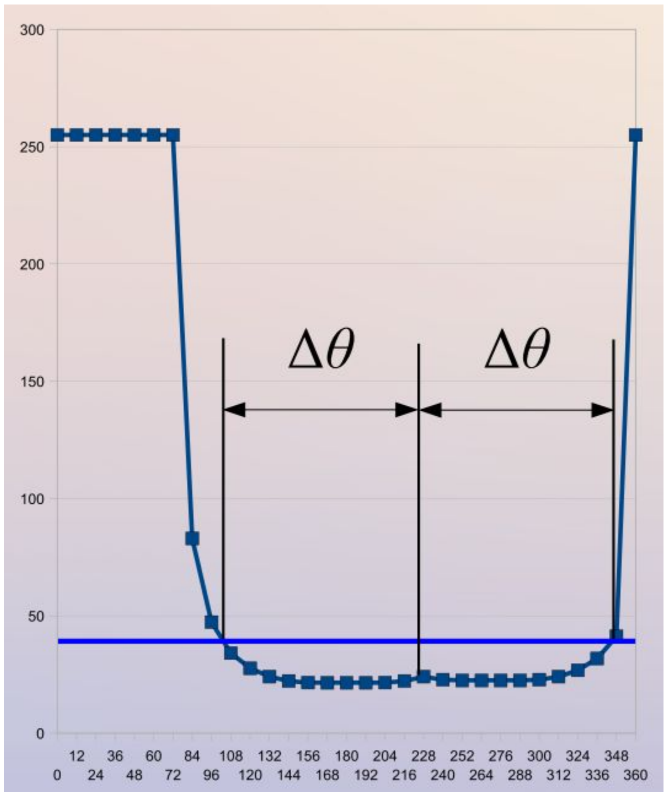


Figure 2: Graph displaying distances as a function of the robot’s orientation, θ, w.r.t the Y-axis.

For clarification purposes, the blue line represented in Fig. 2 is defined as the “threshold” in the following report.

The key assumption is that the robot will start with its center of rotation located on the diagonal line of the back-left tile of the field. Since the ultrasonic sensor contains a lot of noise, it may happen that the falling edge and the rising edge are right next to each other (see Fig. 3).

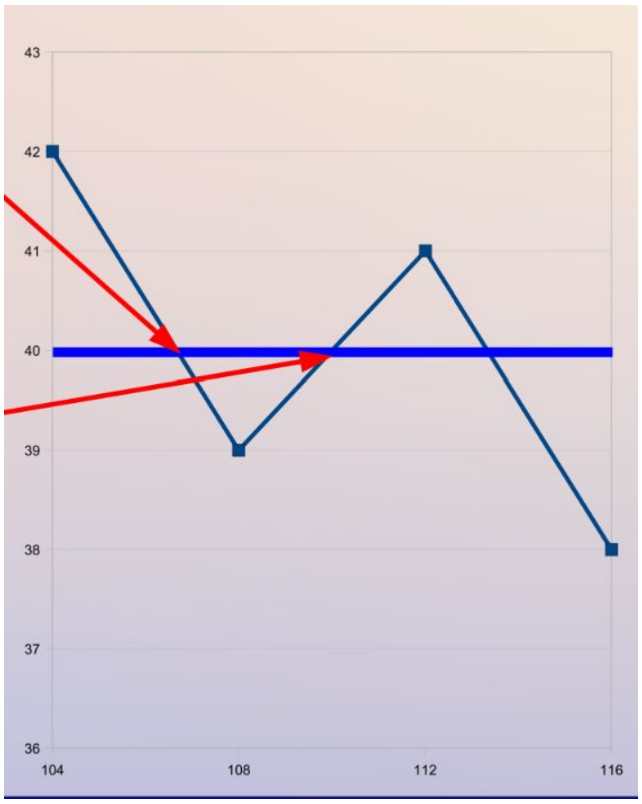


Figure 3: Closer at data collected by the ultrasonic sensor.

The ultrasonic sensor localization includes two types of execution:

1. Falling edge
2. Rising edge

For the falling edge implementation, the robot starts by turning left (i.e. counter-clockwise) if it detects a distance smaller than the addition of threshold (45 cm) and noise margin (5 cm). Therefore, the ultrasonic localization will only start if the robot is not facing the wall or under the threshold (see Fig.2). Once the latter is adjusted, it will start by rotating clockwise until it detects that it has entered the noise margin (i.e a distance smaller than the threshold (45 cm) + the noise margin (5 cm)). The robot will store the angle at which it has entered the noise margin. Afterward, the robot will continue to turn right until it detects a falling edge (i.e. a distance smaller than the threshold (45 cm) - the noise margin (5 cm)). Again, the robot will store the angle at which it has detected the falling edge. Hence, the angle at which the robot has detected the back wall is given by the average of these two angles. The robot will use the same logic for detecting the left wall. It will turn left (counter-clockwise) until it detects a distance smaller than 50 cm (threshold (45 cm) + noise margin (5 cm)). Then, the robot will store its current orientation as the angle at which it has entered the noise margin for the left wall. Afterward, it will continue to turn left until it detects a falling edge. This angle will be stored and used to compute an average with the first angle in order to represent the angle at which the left wall was detected. Finally, the angles at which the back and left walls were detected are used to compute the correction angle. The latter will be added to the odometer's current heading in order to orient the robot correctly (see Fig.4).

$$\Delta\theta = \begin{cases} 45 - \dfrac{\alpha+\beta}{2} & \alpha < \beta \\ 225 - \dfrac{\alpha+\beta}{2} & \alpha > \beta \end{cases}$$

Figure 4: Formula to compute the correction angle needed to adjust the odometer's current theta. ***α*** and β are the angles at which the back and left walls are detected respectively.

For the rising edge implementation, the robot will start by turning right until it is facing the wall. In fact, if it detects a distance bigger than 40 cm (threshold (45 cm) - error margin (5 cm)) it will turn right until the detected distance is smaller than the threshold (see Fig. 2). During the ultrasonic sensor's thread initialization, multiple distances whose value is 0 cm are detected. Hence, a while loop is added to filter out those noisy data. Once the robot is facing the wall and ready to execute localization, it will start by turning left, until it enters the noise margin (i.e. detect a distance bigger than 40 cm (threshold (45 cm) - noise margin (5 cm)). The robot will note the current orientation as the angle at which it has entered the noise margin for the back wall. Then, it will continue to turn left until it detects the rising edge (i.e. a distance bigger than 50 cm). Finally, the robot's current orientation, along with the angle at which it has entered the noise margin, is used to compute an average. The latter is considered as the angle at which the back wall was detected. The same logic is executed by the robot for detecting the left wall. It will turn right until it detects that it has entered the noise margin. Then, this angle will be used, along with the angle at which the rising edge is detected, to compute an average. The latter represents the angle at which the left wall was detected.

After the ultrasonic localization is done, the robot needs to find its current (x, y) coordinates. Assuming the robot is still on the diagonal line of the tile, its Y distance is theoretically equal to its X distance (see Fig. 5). Therefore, the robot travels forward until it detects a black line. Then, it moves back the same distance.

Since the light sensor is located at the back of the robot, the current (x, y) coordinates of the robot are equal to:

X = tile size - distance it has traveled to reach the black line - distance from the light sensor to the rotation center

Y = tile size - distance it has traveled to reach the black line - distance from the light sensor to the rotation center

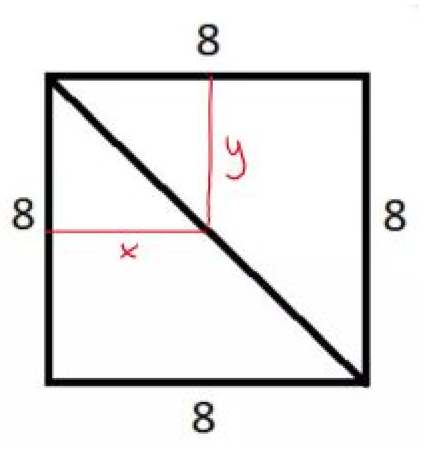


Figure 5: Figure illustrating (x, y) coordinates along the diagonal line of a tile.

Once the robot's current position is known approximately, the travelTo method from lab 3 is used to travel to the origin (1,1).

When the robot is around the origin, it uses light localization to correct its x, y and theta components. In order to optimize light sensor localization, the light sensor is placed as far as possible from the center of rotation of the robot (i.e. wheelbase). The light sensor will be used to detect the grid lines' positions relative to the robot's current. The idea is that at the origin, the robot makes a rotation with respect to its center of rotation (i.e. its wheelbase) and the light sensor has to detect all 4 grid lines as shown in Fig. 6 below.

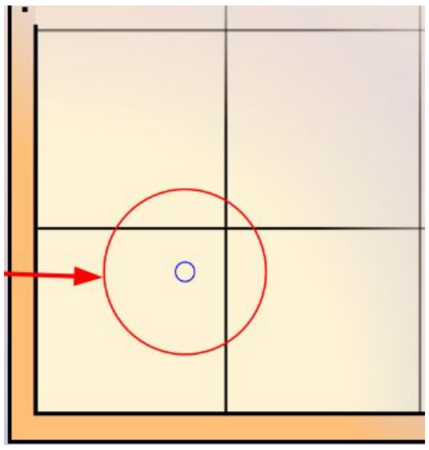


Figure 6: Path of light sensor when robot rotates w.r.t. its center of rotation.

Hence, it is possible to find the position of its center of rotation (see Fig 7).

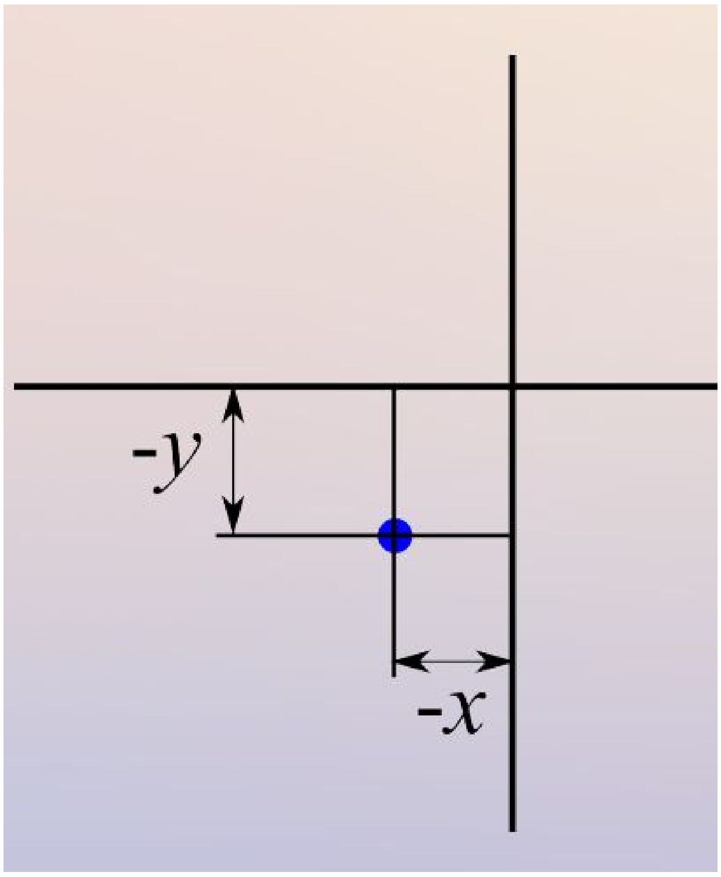


Figure 7: Location of the robot's center of rotation w.r.t. a waypoint.

The angle created by the arc connecting the intersections of the light sensor's path with the y-axis is called θy.

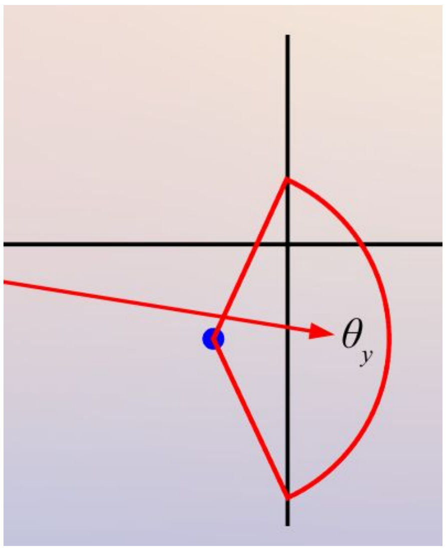


Figure 8: Angle θy created by the arc representing the intersection of the light sensor's path along the Y-axis.

It is possible to find the distance along the X-axis between the robot's center of rotation and the Y-axis using the formula below:

$$|x| = d\cos(\theta y/2)$$

where *d* is the distance from the light sensor to the center of rotation (see Fig. 9).

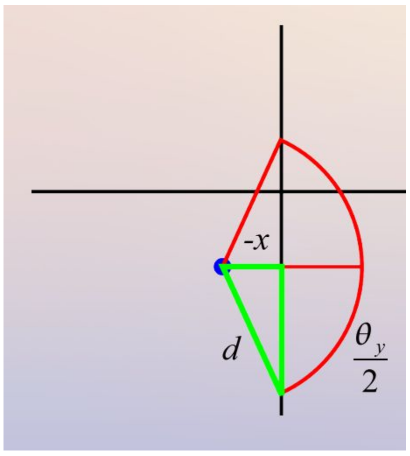


Figure 9: Finding the robot's x coordinate.

Similarly, for the X-axis, we define θx representing the angle θx formed by the intersection between the light sensor's path and the X-axis. Then, it is possible to find the distance between the center of rotation and the X-axis (see Fig. 10) using the formula below:

$$|y| = d\cos(\theta x/2)$$

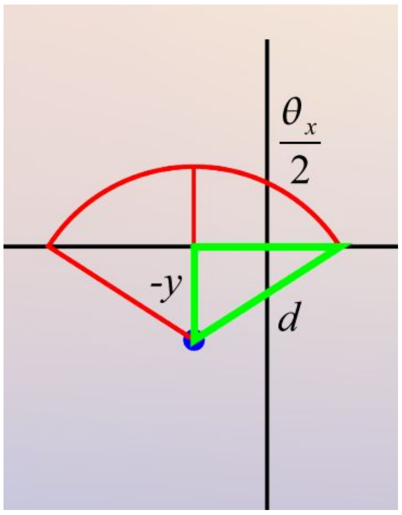
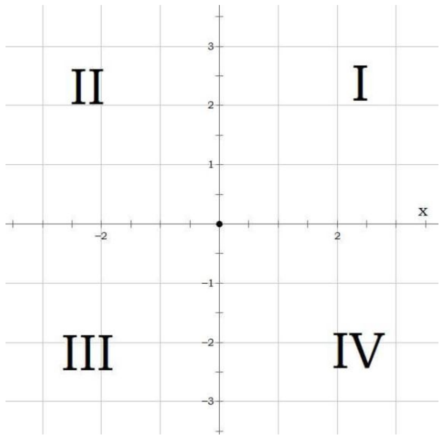


Figure 10: Finding the robot's y coordinate.

It's important to note that it is necessary to know where the robot is with respect to the X and Y axis. We find out the quadrant information by comparing θx and θy to 180 degrees. If is θx is not within 180 degrees and θy is not within 180 degrees, then it's in quadrant 1. If both of them are within 180 degrees, then the robot is in quadrant 3. If is θx is within 180 degrees and θy is not within 180 degrees, then it's in quadrant 4. If is θx is not within 180 degrees and θy is within 180 degrees, then it's in quadrant 2.

If the robot is in the first quadrant, then the real position of robot is (TILE_SIZE+X, TILE_SIZE+Y). If the robot is in the second quadrant, then the real position of robot is (TILE_SIZE-X, TILE_SIZE+Y). If the robot is in the third quadrant, then the real position of robot is (TILE_SIZE-X, TILE_SIZE-Y). If the robot is in the fourth quadrant, then the real position of robot is (TILE_SIZE+X, TILE_SIZE-Y).



Finally, if θy is the heading of the robot reported by the odometer when the light sensor's path intersects the negative y-axis, then solving for Δθ will yield the angle by which to correct the robot's heading (see Fig.11):

$$90 = \Delta\theta + (\theta y- - 180) - \theta y/2$$

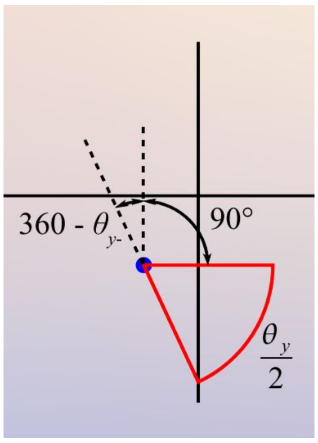


Figure 11: Correction of the robot's heading.

After the odometer’s orientation is corrected, the turnTo method from lab 3 is called to orient the robot back to 0 degrees w.r.t. the Y-axis.

1.3 Workflow

The ultrasonic sensor was implemented first, then multiple rounds of testing were executed in order to find the optimal threshold and noise margin. The original threshold was set to 20 cm. However, it was changed to 45 cm in order to avoid detecting a false positive (local maximum) during the rising edge localization. Then, a method called findCurrentPosition was implemented in the Navigation class in order to find the robot’s current position. Finally, the light sensor localization was implemented last in order to correct errors accumulated during ultrasonic sensor localization and navigation. Only the threshold and the noise margin values were modified during the design process. All hardware design choices were not changed from the beginning to the end, because they satisfied all technical requirements.

1.4 Graphics

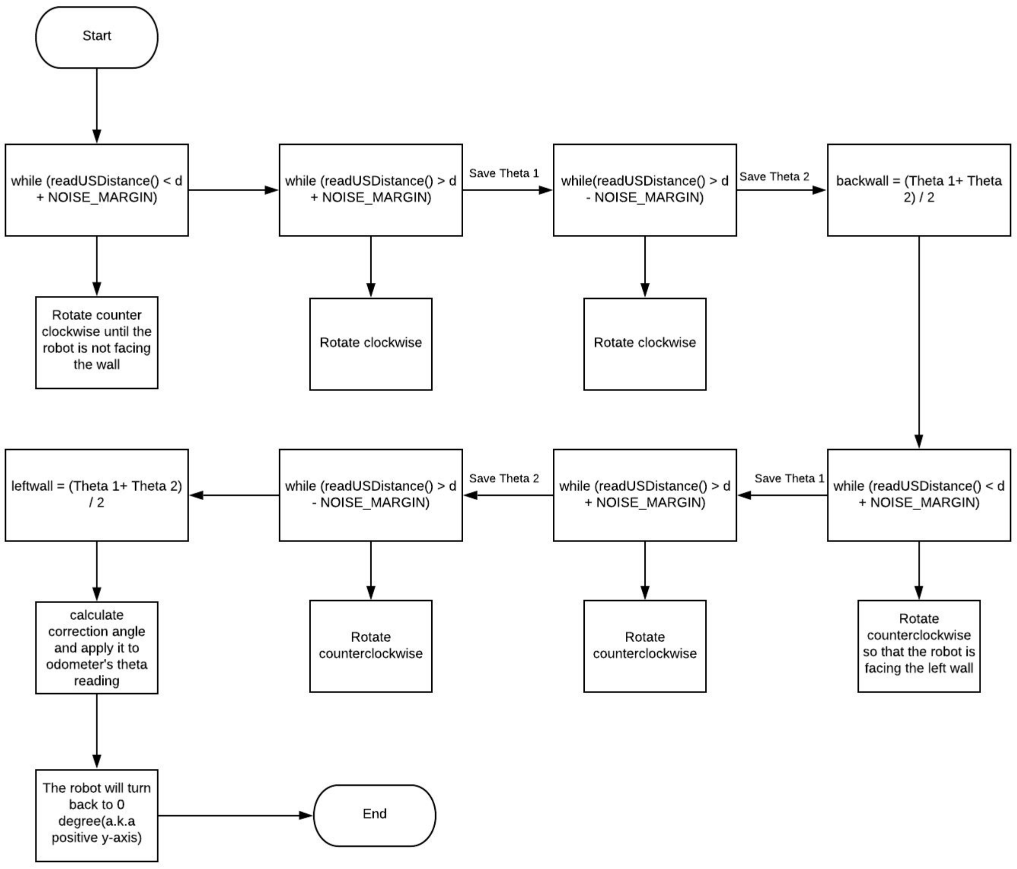
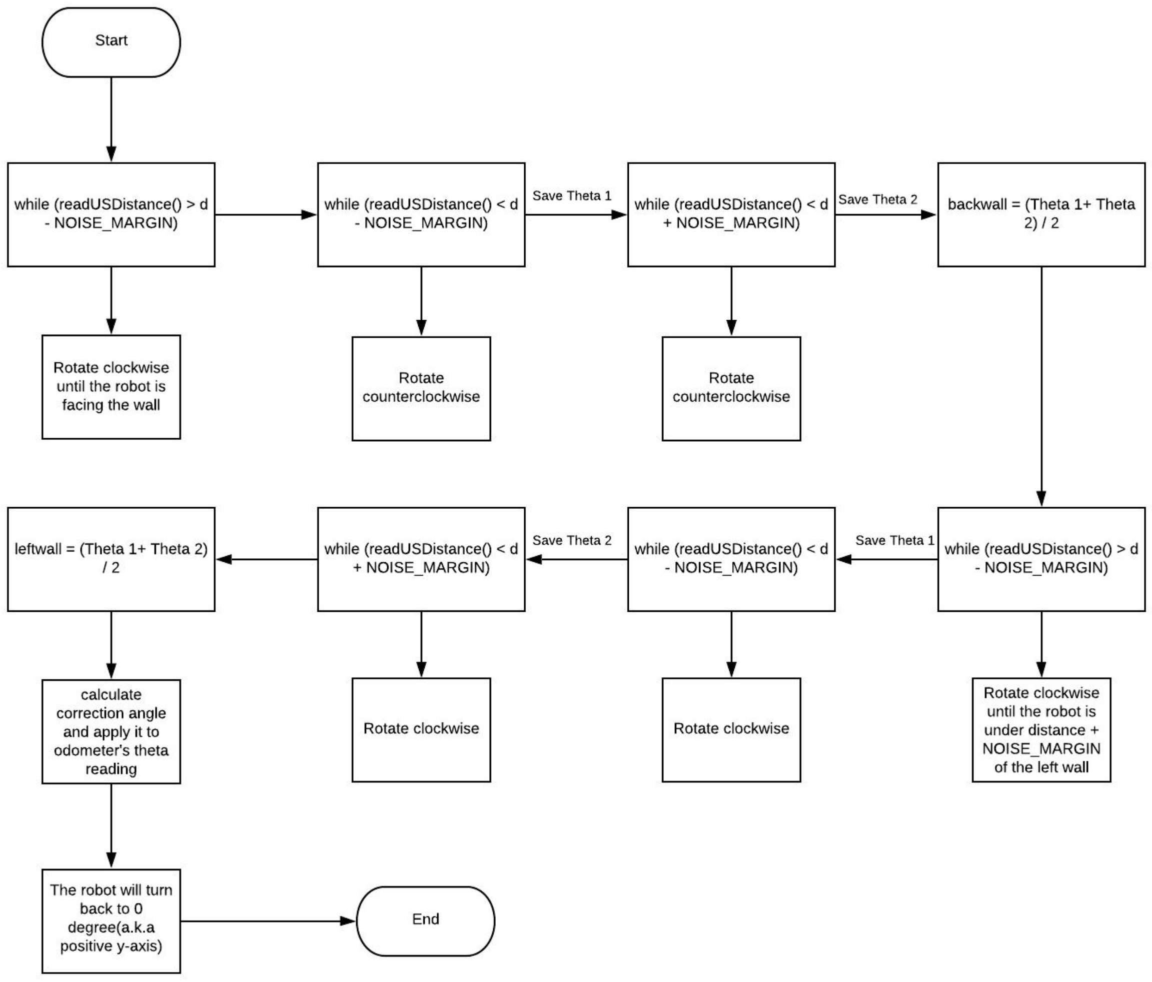

Start
while (readUSDistance() < d + NOISE_MARGIN)
Rotate counter clockwise until the robot is not facing the wall
while (readUSDistance() > d + NOISE_MARGIN)
Rotate clockwise
Save Theta 1
while(readUSDistance() > d - NOISE_MARGIN)
Rotate clockwise
Save Theta 2
backwall = (Theta 1+ Theta 2) / 2
while (readUSDistance() < d + NOISE_MARGIN)
Rotate counterclockwise so that the robot is facing the left wall
Save Theta 1
while (readUSDistance() > d + NOISE_MARGIN)
Rotate counterclockwise
Save Theta 2
while (readUSDistance() > d - NOISE_MARGIN)
Rotate counterclockwise
leftwall = (Theta 1+ Theta 2) / 2
calculate correction angle and apply it to odometer's theta reading
The robot will turn back to 0 degree(a.k.a positive y-axis)
End

Figure 12: Falling edge detection flowchart.



Start
while (readUSDistance() > d - NOISE_MARGIN)
Rotate clockwise until the robot is facing the wall
while (readUSDistance() < d - NOISE_MARGIN)
Rotate counterclockwise
Save Theta 1
while (readUSDistance() < d + NOISE_MARGIN)
Rotate counterclockwise
Save Theta 2
backwall = (Theta 1+ Theta 2) / 2
while (readUSDistance() > d - NOISE_MARGIN)
Rotate clockwise until the robot is under distance + NOISE_MARGIN of the left wall
Save Theta 1
while (readUSDistance() < d - NOISE_MARGIN)
Rotate clockwise
Save Theta 2
while (readUSDistance() < d + NOISE_MARGIN)
Rotate clockwise
leftwall = (Theta 1+ Theta 2) / 2
calculate correction angle and apply it to odometer's theta reading
The robot will turn back to 0 degree(a.k.a positive y-axis)
End

Figure 13: Rising edge detection flowchart

<u>Section 2: Test Data</u>

2.1 Falling Edge Tests

Table 1: Test localization using falling edge

Ultrasonic Angle **θ**$_U$/(degrees)	Final X Position X$_f$ /(cm)	Final Y Position Y$_f$ /(cm)	Euclidean Distance Error ε /(cm)	Final Light Angle **θ**$_L$/(degrees)
3.0	30.48	30.48	0.0	1.0
1.0	30.58	30.48	0.1	3.5
0.5	30.58	30.48	0.1	3.0
2.0	30.58	30.48	0.1	2.0

3.0	30.58	30.48	0.1	2.5
4.0	30.48	30.48	0.0	2.5
4.0	30.98	30.68	0.5385	2.5
4.0	30.48	30.58	0.1	2.0
3.5	30.78	30.58	0.3162	2.0
3.0	30.68	30.58	0.2236	3.0

2.2 Rising Edge Tests

Table 2: Test localization using rising edge

Ultrasonic Angle Error $\boldsymbol{\theta}_U$/(degrees)	Final X Position X_f /(cm)	Final Y Position Y_f /(cm)	Euclidean Distance Error ε /(cm)	Final Light Angle Error $\boldsymbol{\theta}_L$/(degrees)
5.0	30.78	30.58	0.3162	2.0
5.0	30.68	30.68	0.2828	2.0
7.0	30.48	30.58	0.1000	3.5
7.5	30.48	30.48	0.0000	1.0
7.0	30.78	30.58	0.3162	1.0
6.5	30.58	30.48	0.1000	1.0
7.0	30.48	30.48	0.0000	1.0
5.0	30.38	30.48	0.1000	2.0
9.0	30.48	30.68	0.2000	2.5
7.0	30.78	30.58	0.3162	2.0

<u>Section 3: Test Analysis</u>

Error between final waypoint and real final position:

$$\varepsilon = \sqrt{(Xf - Xd)^2 + (Yf - Yd)^2}$$

Mean formula:

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

Example: *Mean of Euclidean Distance Error = (30.78 + 30.78+ 30.68+30.48+30.48+30.78+30.58+30.48+30.38+30.48+30.78） / 10 = 0.1732*

Standard Deviation Formula:

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

Table 3: Mean and Standard deviation of ultrasonic angle error, euclidean distance error and final light angle error for falling edge

Mean of Ultrasonic Angle Error μ_U /(degrees)	Mean of Euclidean Distance Error μ_ε /(cm)	Mean of Final Light Angle Error μ_L /(degrees)	Standard Deviation of Ultrasonic Angle Error σ_U /(degrees)	Standard Deviation of Euclidean Distance Error σ_ε /(cm)	Standard Deviation of Final Light Angle Error σ_L /(degrees)
2.8	0.1578	2.4	1.1874	0.1552	0.6633

Table 4: Mean and Standard deviation of ultrasonic angle error, euclidean distance error and final light angle error for rising edge

Mean of Ultrasonic Angle Error μ_U /(degrees)	Mean of Euclidean Distance Error μ_ε /(cm)	Mean of Final Light Angle Error μ_L /(degrees)	Standard Deviation of Ultrasonic Angle Error σ_U /(degrees)	Standard Deviation of Euclidean Distance Error σ_ε /(cm)	Standard Deviation of Final Light Angle Error σ_L /(degrees)
6.6	0.1732	1.8	1.2207	0.1225	0.7810

Section 4: Observations and Conclusions

Q: Which of the two localization routines performed the best?

In the presence of noise, the mean can often serve as a good estimate of the uncorrupted signal. Hence, the falling edge routine performed the best, since the mean of its ultrasonic localization angle error is 2.8 degrees compared to 6.6 degrees for rising edge. The standard deviation is an indicator of how the error is dispersed about the mean. Therefore, it is possible to affirm that the variation of ultrasonic angle error for the falling edge method is less than the rising edge method. Their variation are respectively 1.1874 and 1.2207 degree.

However, both ultrasonic localization routines' errors will be corrected by the light localization later.

Q: Was the final angle impacted by the initial ultrasonic angle?

No, since the light sensor localization was able to correct most of the errors caused by the ultrasonic localization. In fact, the ultrasonic angle errors for falling and rising edge are respectively 2.8 and 6.6 degrees. However, the light angle errors for falling and rising edge are 2.4 and 1.8 degrees. Therefore, even though the error of the ultrasonic localization is bigger for the rising edge routine, it has a smaller final angle using the light localization. This proves that the initial ultrasonic angle does not have an impact on the final angle. In addition, both final light angle errors are similar with only 0.6 degree difference. In fact, the final light angle error can be explained by the width of the black lines and the difference in the moment at which each one of the grid lines were detected.

Q: What factors do you think contributed to the performance of each method?

The rising edge method's performance is worse than the falling edge method, because it is rotating while facing the wall. In fact, data fetched from the ultrasonic sensor contain a lot of noise. While the robot is looking for rising edges, it detects a large range of distances on the extremities which could mislead the robot to think that noisy data for rising edges. These false positives can be a source of error for the rising edge method. In addition, when the robot is turning while facing the wall, it detects the local maximum which could also be mislead for a rising edge. Hence, those factors have introduced multiple sources of errors in the rising edge method. On the opposite, the falling edge method does not turn while facing the wall. Therefore, it does not have errors caused by local maximum and false positives. For the light localization, if the circumference is not big enough to detect all four grid lines, it will not be able to correctly adjust the odometer's values. Therefore, the light localization method depends highly on where the light sensor is installed on the robot. This distance was maximized by putting the light sensor at the back of the robot.

Q: How do changing light conditions impact the light localization?

For our design, the change of light conditions does not have an impact on the light localization, since it does not rely on an absolute value. In fact, the threshold for detecting a black line is not represented by a light intensity value. Instead, a ratio representing the magnitude of difference is used to identify the presence of grid lines. This technique can factor out the influence of ambient light.

<u>Section 5: Further Improvements</u>

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

The median filter can be implemented in order to minimize errors in the ultrasonic sensor. It is is a way to reduce the impact of outliers in a sequence of data fetched by the ultrasonic sensor. Consider a sequence of numbers and a moving window of 5 samples wide:

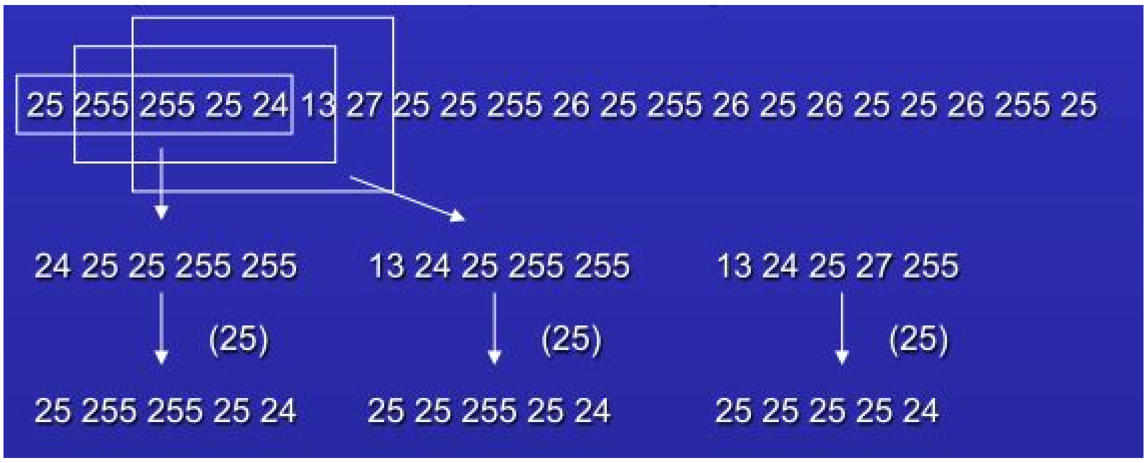


Figure 14: Median filter representation.

If a sample is bigger than the median of the window containing 5 samples, its value is replaced by the median of that window. The median is the value of the center position of an odd ordered list and the average of the 2 center elements of an ordered list (even). However, in order to choose the size of the window (W), we need to know the noise characteristics. More specifically, the number of outliers must be smaller or equal to W/2 rounded down, if W is Odd, and smaller than W/2 rounded down, if W is even.

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

Suppose we had two ultrasonic sensors that are 45 degrees w.r.t. each other. Both are mounted at the same position on the robot, one over the other. Assuming the robot's current orientation is 0 degree w.r.t. the Y-axis, the first and second sensor would be respectively installed at 22.5 degrees and 337.5 degrees w.r.t. the Y-axis. In addition, we assume that the robot will start with its center of rotation located on the diagonal line of the back-left tile of the field. Therefore, the robot turns clockwise or counter clockwise until both sensors detect the same distance +/- an error margin. Once both US sensors detect the same distance, it is possible to assume that the robot's current orientation is 225 degrees with respect to the Y-axis. Thus, it can turn 135 degrees clockwise to get to 0 degrees w.r.t. Y-axis. Once the turn is completed, we can set its current orientation to 0.

Q: 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

Light Localization can be used when the robot is traveling a specific path (i.e. lab 3). In fact, suppose the robot has to travel through 5 waypoints. Light localization can be executed at each one of the waypoints in order to correct any errors that occurred during navigation. This technique makes the robot's navigation significantly more precise. In fact, the robot knows its destination's (x,y) coordinates and its current (x,y) position. Therefore, the robot can execute light localization on any grid lines intersection to correct its odometer's x, y and theta values.