DPM Lab 3 Report

Group 42
Wen Cui 260824815
Aidan Gerkis 260827581

1) Design Evaluation:

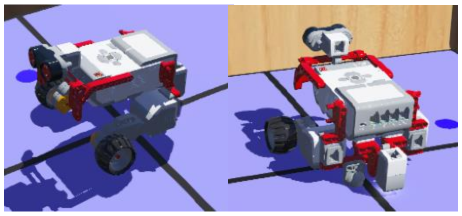
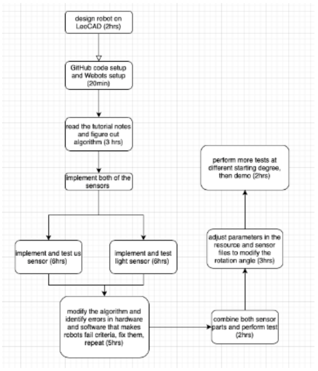


Figure 1: Two Views of the final design

Our design workflow began with the evaluation of the requirements our robot must meet to implement the localization algorithm. The goal for this lab is to rotate the robot to 0°, with respect to a chosen reference, using the ultrasonic sensor. Then, the colour sensor is used to move the robot to (1,1), and to orient the robot more accurately to 0°. A detailed design workflow diagram is shown in Figure 2.


design robot on LeoCAD (2hrs)
GitHub code setup and Webots setup (20min)
read the tutorial notes and figure out algorithm (3 hrs)
implement both of the sensors
implement and test us sensor (6hrs)
implement and test light sensor (6hrs)
modify the algorithm and identify errors in hardware and software that makes robots fail criteria, fix them, repeat (5hrs)
combine both sensor parts and perform test (2hrs)
adjust parameters in the resource and sensor files to modify the rotation angle (3hrs)
perform more tests at different starting degree, then demo (2hrs)
Figure 2: Our design workflow

Our hardware design for this lab went through three iterations, as shown in Figure 3. The first iteration was similar to the robot we built for Lab 2, with the colour sensor and ultrasonic sensor added on the same face of the robot. The first change we made was moving the sensors further away from the robot's centre of rotation, ensuring that the colour sensor would sweep a large enough radius when the robot rotates. The second change was made while we were testing our localization code. Due to an issue with the ballcaster-floor interaction in webots the robot was not flat during testing, which caused the ultrasonic sensor to be angled slightly downwards and would occasionally return a false positive as the sensor detected the ground. Originally a median filter was implemented to filter the incorrect readings, but the noise still affected the sensor value significantly. After multiple tests we came up with the solution to move the ultrasonic sensor to the other side of the robot, so it would be angled upwards. As a result, the ultrasonic sensor no longer detected the floor, and a more accurate reading was achieved for the ultrasonic sensor.

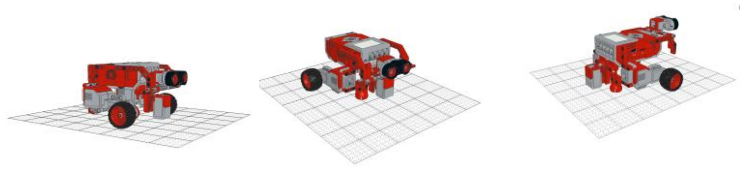


Figure 3: From left to right the three iterations of our robot design

Our software includes 6 classes: Main, Odometer, Driver, UltrasonicLocalizer, LightLocalizer, and Resources. Figure 4 illustrates the class diagram for our code, showing the software architecture we used for this lab. In Main we created a thread corresponding to the odometer class implemented in Lab 2 to track the robot's position, then ran both localization algorithms sequentially. Our ultrasonic localization method uses a falling edge detection algorithm, the procedure the localization algorithm follows can be seen in Figure 5. Compared with a rising edge algorithm, the falling edge algorithm performs better when the robot was initially facing away from the wall, for this reason we first rotate the robot to ensure it is facing away from the wall, then run our falling edge algorithm to localize the robot. After rotating the robot to 0° we orient it so that it the light sensor is inside the tile bounded by (1,1). By following the procedure shown in Figure 6 the localization method is able to move the robot from anywhere in the bottom left square to the (1,1) intersection, oriented to 0°. The Driver class contains several useful helper methods for controlling the robots motion, such as setSpeed, moveStraightFor, etc.

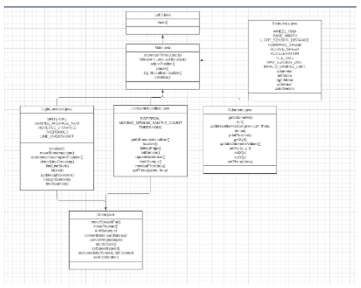
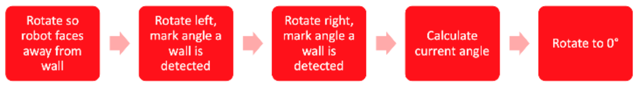
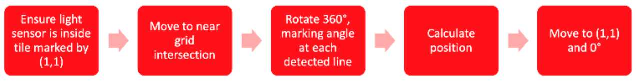


Figure 2: The class diagram for our code


Rotate so robot faces away from wall
Rotate left, mark angle a wall is detected
Rotate right, mark angle a wall is detected
Calculate current angle
Rotate to 0°
Figure 5: A breakdown of the ultrasonic localization algorithm


Ensure light sensor is inside tile marked by (1,1)
Move to near grid intersection
Rotate 360°, marking angle at each detected line
Calculate position
Move to (1,1) and 0°
Figure 6: A breakdown of the light localization algorithm

2) Test Data

Table 1: Test Data

Trial #	$\Theta_{Ultrasonic}$	Θ_{Final}	X_{Final}	Y_{Final}	ε_{XY}
1	0.01602	0.03388	0.293	0.306	0.011861
2	0.0139	0.03388	0.292	0.307	0.012988
3	0.1657	0.03388	0.306	0.298	0.006905
4	0.04386	0.03388	0.289	0.309	0.016349
5	0.03388	0.0235	0.29	0.308	0.015142

** Distances in Metres, angles in Radians*

3) Test Analysis

Table 2: Mean & Standard Deviation of Error

$\Theta_{Ultrasonic}$		Θ_{Final}		ε_{XY}	
Mean	Standard Deviation	Mean	Standard Deviation	Mean	Standard Deviation
0.054672	0.063310261	0.031804	0.004642	0.0126489	0.003662

$$\mu_X = \frac{1}{N} * (X_1 + X_2 + \ldots + X_N)$$

$$\sigma_X = \sqrt{\frac{\sum_{i=1}^{N}(X_i - \mu_X)^2}{N-1}}$$

$$\varepsilon_{\Theta_{Ultrasonic}} = |0 - \Theta_{Ultrasonic}| = \Theta_{Ultrasonic}$$

$$\varepsilon_{\Theta_{Final}} = |0 - \Theta_{Final}| = \Theta_{Final}$$

Our test data allows us to conclude that our average ultrasonic angle correction is wrong by about 0.055 rad (or 3.1°), although the large standard deviation indicates that the algorithm is not very precise, and we can expect to see variations of greater then 3.1° from the mean in our correction. The test data also shows that the error in our light sensor localization is very low, meaning a high accuracy, while the low standard deviation indicates that the algorithm is very precise. These results show that the light sensor based localization algorithm is much more robust then the ultrasonic sensor based version, as it produces more accurate results with higher precision

4) Observations & Conclusions

For our ultrasonic localization method we could have used the rising edge algorithm to orient our robot. However, from our initial experiments, the falling edge algorithm had better accuracy on average when detecting a falling edge than a rising edge. Also, from our observations, falling edge localization produces a better result when the robot is facing away from the wall initially, while rising edge localization produces a better result when the robot is facing towards the wall initially. In order to reduce the complexity of our algorithm, we came up with the solution to first rotate the robot so that it was facing away from the wall, then perform falling edge localization. For light sensor localization we could have implemented the algorithm using 2 light sensors (located on opposite sides of the robot) as opposed to one. This would have improved the accuracy of our light sensor localization algorithm, but due to the time constraints of the project, and the increased complexity of the code required to implement this algorithm, we were unable to implement this solution.

From our experiments, and our understanding of the algorithm, the initial angle of the ultrasonic sensor does not impact the final angle. Since the formulas used for determining the required adjustment in heading all use the difference between the two measured angles the initial angle will not factor into our calculated correction to heading. From our experimental observations, the initial ultrasonic angle did not have a large impact on the final angle of the robot, although we did notice some discrepancies to this. These were most likely caused by the readings from the ultrasonic sensor and their accuracy. When the sensor is looking for a falling edge while rotating a noise margin is introduced to filter out potential false positives. The value of the noise margin was determined through trial-and-error, and may not be the ideal value for all cases, which could cause inaccuracies in the new, calculated, heading. Another possible reason for these discrepancies is the initial position of the robot. The position of the robot on the 45-degree line will affect the accuracy of the localization algorithm, as we found that the closer the robot is to the wall the less error we see in the ultrasonic angle.

The median filter method used to filter the ultrasonic sensor values used in ultrasonic localization significantly improved its performance. Through trial and error a window size of 8 was found to result in the most accurate filtered result. For our robot we used only one light sensor, however, adding another would have most likely increased the accuracy of our light sensor localization algorithm. Adding one more sensor would have increased the accuracy of the x, y, and Θ calculations, allowing our robot to better locate itself.

5) Further Improvements

A possible improvement for the software in this lab is to adjust the value of k (the noise margin) according to the distance from the wall at the starting point. More specifically, we could increase k when the robot is close to the wall and decrease k when the robot is further from the wall. This would improve the performance of the sensor because as the distance from the ultrasonic sensor to the wall increases the ultrasonic sensor has a larger part of the wall at the distance d in its field of view. Therefore, decreasing k will decrease the part of wall at the distance d in the field of view, which will provide more accurate measurements for the angle. Similarly, if the robot is close to the wall at the starting position, increasing the value of k will give a more accurate Θ adjustment.

In order to use the light sensor to navigate the robot outside of the corner, we need to know that the tile edges on the floor are evenly distributed. Knowing this, once the robot has been localized the light sensor can be used to calculate the number of tiles the robot has traversed. This information can be used to calculate the real distance the robot has travelled (real distance = tile size x number of tiles traversed). By comparing this calculated distance travelled to the distance measured by the odometer we can make adjustments to the robots position to ensure the robot is where we want it to be.

We could localize the robot in the middle of the grid using a variety of different methods. Assuming we are limited to the resources we have used in our labs so far one feasible method is to localize the robot using odometry. Using an ultrasonic sensor we can localize our robot to a starting point for reference, then initialize our odometer to (0, 0) at that point. We could then use our odometer to position the robot anywhere we desire, although this would be subject to errors in the odometry calculations. Another method could be to place an object in the middle of the grid. The odometer could be used to navigate the robot to the approximate location of the object, then an algorithm could traverse the area and use a touch sensor to locate the objects exact position. This method could be more accurate then the first one described as any errors in the odometer will not result in the robots inability to localize itself.