ECSE 211 Lab 4 Report Group 33

Authors: Aleksas Murauskas, 260718389 and Talaal Mazhar Shafi, 260865281

1. Design Evaluation

This Paper will discuss the hardware and software design of a Robot using ultrasonic and Light sensors to localize itself to a point. First in terms of hardware, we kept the core design consistent with the previous two labs to avoid retesting to find the true track value. Our measured track value was 17.6 cm, but through previous testing it was found to truly be 18.15 cm. The Ultrasonic sensor was fixed to the front of the robot, as close to the center of rotation as possible to minimize possible error when the angles are found using the Ultrasonic Localizer. A light sensor was fixed to the rear of the robot as far as possible from the center of rotation in order to ensure it would be able to pass over all four lines leaving the origin. Originally, the light sensor was too high to properly test the table's color, so the sensor was moved lower to avoid any error caused by the sensor's distance from the table.

Figure 1

Figure 2.

Figure 3.

In terms of software, this lab was built off the skeleton of Lab 3. The LCDInfo, Navigator, BasicNavigator, Odometer, and UltrasonicPoller were reused relatively unchanged from the Obstacle Avoidance Lab. The majority of new code was introduced in two new classes, UltrasonicLocalizer and LightLocalizer. The flow of operation goes as follows: The Main class's main method is called. Then the Robot prompts the user what type of edge Ultrasonic Localizer should search for. If the button choice of left is selected the risingEdge() is called, similarly fallingEdge is called if the right button is pressed. Then several threads are created to control the odometer, US sensor, and LCD Screen. Afterwards, the UltrasonicLocalizer is created and the respective edge function is called. The localizer order of operations will be discussed later in the paper. Once the

edge function has localized the robot's theta, the robot turns to what i the robot believes 90 degrees is. The robot then waits for any button to be pressed to begin light localization. A LightLocalizer object is created and then calls the preparation() method, which corrects the odometer by crossing the axis lines similar to Lab 2's Square Driver. Then the localize() method is called to perform a secondary localization following the first. Both of these methods will be explained in depth further in the paper. Then a navigator thread is created, travelling to the origin and adjusting to 90 degrees.

The UltrasonicLocalizer is made up of two primary methods, RisingEdge() and FallingEdge(). FallingEdge() is designed to begin with the robot facing away from the wall and it will detect a sharp decline in the distance. FallingEdge functions by rotating right as long as the distance is greater than the an expected distance and buffer value. The Distance D and Buffer value K are established in Main method and were found through trial and error testing. As soon as as the the distance is less than the sum of D and K, The current theta the robot is at is recorded in the variable thetaA. It continues to rotate right until the distance falls below the distance buffer. At that moment, the current theta is stored in thetaB. Then the robot begins to rotate to its left until it detects the distance entering the buffer for the other wall, then it records thetaC. The leftward rotation continues until the distance is lower than the buffer's lower bound, at which point thetaD is found. At this point all key angles have been found and the robot's engine speeds are set to 0. The average of thetaA and thetaB become alphaAngle. The average of thetaC and thetaD become betaAngle. The necessary change in theta for correction, deltaT, is then calculated. If alpha is smaller than beta than deltaT is calculated using the equation: $deltaT = (45 - ((alpha + beta)/2))\%360$, and otherwise $deltaT = (225 - ((alpha + beta)/2))\%360$. Then the odometer's Theta is updated with the correction of sum of the current odometer theta and deltaT. Rising Edge functions similarly, except the core difference is the robot should be facing the wall at the start, and the thetaA and thetaC are recorded when the robot passes the lower bound of the buffer. This is because a rising edge begins at the bottom of the buffer.

The LightLocalizer class also has two key methods, Preparation() and localize(). Preparation occurs first. The robot rolls forward along the y axis until the rear light sensor detects a line. Once that occurs, the odometer's Y value is set to the distance from the robots center of rotation to the light sensor, stored as the vehicle's length. The robot then reverses the length and 10 cm, then turns 90 degrees to the right. The above sequence of events repeats for the X axis. Once the odometer correction has been completed for both axes, a navigator thread is created. The robot travels to (-6,-6) where it is now prepared for the localize method.

The localize method then begins by recording the initial angle the odometer is at when the method is called. As the light sensor passes a line, the current theta of the robot will be recorded. The robot begins to rotate to the left, and begins repeated polling the

light sensor to search for a line. Each time a line is detected, the current theta held by the odometer is stored into the respective variables: xcrossA,ycrossA, xcrossB, ycrossB. Once the full 360 cycle has been completed, both motor speeds are set to 0. First all cross variables are inverted using the equation: $invert = (angle + 180)\%360$. Then difference of both X cross values are found and stored in the variable X. If the difference is larger than 180, the difference X is set to $X = 360 - X$. This is repeated for the Y axis to find Y. The odometer is corrected again with the equation:
$positionInX = cos(Radians(Y)/2) * -Length$ and for Y:
$positionInY = cos(Radians(X)/2) * -Length$. The odometer's x,y, and theta is then set with new localized values. Then the method terminates.

2. Test Data

Falling Edge Trials:

Ultrasonic Trials

$US\ Angle\ Error = |US\ Robot\ Angle - US\ True\ Angle|$

Falling Edge			
Trail Number	US Robot Angle	US True Angle	US Angle Error
1	90	93	3
2	90	86	4
3	90	81	9
4	90	89	1
5	90	90	0
6	90	91	1
7	90	92	2
8	90	89	1
9	90	92	2
10	90	88	2

Light Localizer Trials

[Each trial follows the above Graphs trial, i.e. Trial 1 in Light trials begins in the position and angle where Trial 1 in Ultrasonic ended]

$Final\ Angle\ Error = |Expected\ Angle - Final\ True\ Angle|$ $Expected\ Angle = 90\ degrees$

$Euclidean\ Error = \sqrt{(Final\ True\ X)^2 + (Final\ True\ Y)^2}$

Falling Edge					
Trail Number	Final True Angle	Final True X	Final True Y	Euclidean Error	Final Angle Error
1	85	-0.4	-0.7	0.806225775	5
2	80	0.4	-1.5	1.55241747	10
3	75	0.5	-2.8	2.844292531	15
4	85	0.1	-1.5	1.503329638	5
5	86	-0.2	-1.5	1.513274595	4
6	88	-0.2	-1.8	1.811077028	2
7	85	-0.3	-1.1	1.140175425	5
8	87	-0.4	-0.7	0.806225775	3
9	86	-0.2	-0.9	0.921954446	4
10	87	-0.1	-1.2	1.204159458	3

Rising Edge Trials:

Ultrasonic Trials

$$US\ Angle\ Error\ =\ |US\ Robot\ Angle\ -\ US\ True\ Angle|$$

Rising Edge			
Trail Number	US Robot Angle	US True Angle	US Angle Error
1	90	88	2
2	90	62	28
3	90	86	4
4	90	69	21
5	90	65	25
6	90	86	4
7	90	72	18
8	90	77	13
9	90	82	8
10	90	81	9

Light Localizer Trials

[Each trial follows the above Graphs trial, i.e. Trial 1 in Light trials begins in the position and angle where Trial 1 in Ultrasonic ended]

$$Final\ Angle\ Error\ =\ |Expected\ Angle\ -\ Final\ True\ Angle|,\quad Expected\ Angle\ =\ 90\ degrees$$

$$Euclidean\ Error\ =\ \sqrt{(Final\ True\ X)^2\ +(Final\ True\ Y)^2}$$

Rising Edge					
Trail Number	Final True Angle	Final True X	Final True Y	Euclidean Error	Final Angle Error
1	82	-0.3	-1.9	1.923538406	8
2	79	1.1	-2.6	2.823118843	11
3	85	0	-1.3	1.3	5
4	60	2.2	-3.1	3.801315562	30
5	58	2.6	-4.1	4.854894438	32
6	84	0.2	-1.6	1.61245155	6
7	64	1.8	-2.8	3.328663395	26
8	70	1.6	-3.2	3.577708764	20
9	84	0.3	-1.6	1.62788206	6
10	79	1.3	-1.4	1.910497317	11

3. Test Analysis

$$Error\ Mean\ =\bar{\varepsilon}\ =\ \sum_{i=1}^{10}\frac{\varepsilon_i}{10}$$

$$Standard\ Deviation\ of\ Error\ =\ \sqrt{\frac{\sum_{i=1}^{10}|\varepsilon_i-\bar{\varepsilon}|^2}{10}}$$

Falling Edge			Rising Edge		
	Mean	Standard Deviation		Mean	Standard Deviation
US Angle Error	2.5	2.549509757	US Angle Error	13.2	9.319036669
Euclidean Error	1.410313	0.60920407	Euclidean Error	2.67601	1.180859554
Final Angle Error	5.6	3.949683532	Final Angle Error	15.5	10.54356044

4. Observations and conclusions

The light localization routine performed better than the ultrasonic, as there was less chances for false positives, and it localized both the angles and the intercepts at once. The light sensor is known to be a higher resolution sensor, and this was proven in our observations as it determined the position of the robot with higher accuracy on the field.

The final angle was heavily impacted by the initial ultrasonic angle. A small error in the angle in ultrasonic would lead to a small error in the initial error, which led to a much more significant error in the final angle. The error in by the light sensor is amplified by an error in ultrasonic.

We used two different types of localization methods: the preparation method, and the localize method. There is less likely to be a false positive in the localize method. The rising edge failed more frequently due to these false positives. The most effecive was the conjunction of the falling edge and the localized methods, which resulted in the most error free final position.

Different ambient light can result in the robot failing to tell a difference between what is a black line and what is not. For example, if the room is too dark, it will see only black lines, and if the room is bright, it may not see any at all. The ambient light in the room can affect the accuracy of its light sensor.

5. Further Improvements

For a hardware improvement, we can add a light sensor that help with odometry correction. This would correct the odometer understanding of the robot's position, as proved in our last lab,where mean error dropped from 4.3cm to 2.4 this would improve our error mean significantly.

Another form of localization used could be range-free localization. This would be based off of connectivity information, rather than distance/angle measurements. For example, 3 signal producing towers would send out signals the robot would be able to intercept. It would use triangulation to then locate itself.

Our aforementioned hardware correction could be used outside of the corners. It would correct odometry errors after every line passed, due to the 2 light sensors.