ECSE 211 Lab 2 Report Group 33

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

1. Design Evaluation

The following section will entail of the hardware and software design of our Square Driver Robot. We first began by designing a new robot that would have a wider track than the previous lab. We chose this because we wanted to place the light sensor right in between the two wheels, which would avoid possible errors caused by the offset of the sensor and the robot's turning radius. The sensor's placement in between the track is visible in Figures 1 and 2. In addition a larger track size would require the wheels to turn more rotations in order to complete a 90 degree turn compared to that of a robot with a smaller track size. This would make the robot's possible underturns or overturns easier to see. The measured track length was 17.6 cm, and the radius of the wheels fit the given 2.2 cm. However after preliminary testing, it was found that the measured Track value was too small resulting in turns of roughly 85 degrees. Through trial and error, the track length of 18.15 cm was found to result in the most accurate turns. Afterwards, it was noticed that the wheels would lean away from the robot. This was due to the center of gravity being spread too wide without enough support. To alleviate the issue, two new supports were added. A low crossbar [visible in Figure 3] was fitted below the motors and a front crossbar [visible best in figure 2] was added above the motors and connected at the front of the robot. Below in Figure 1 2 and 3 is the final design used in the demo and test data trials.

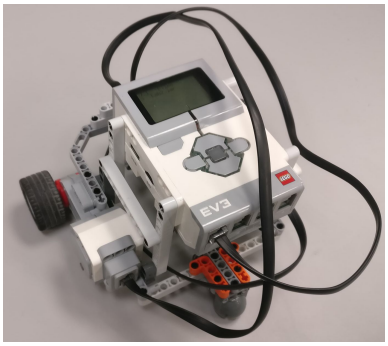


Figure 1.

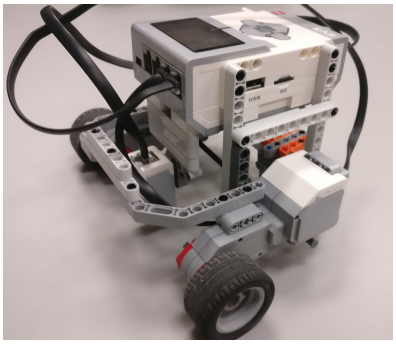


Figure 2.

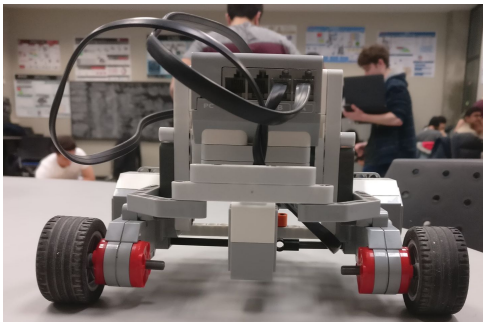
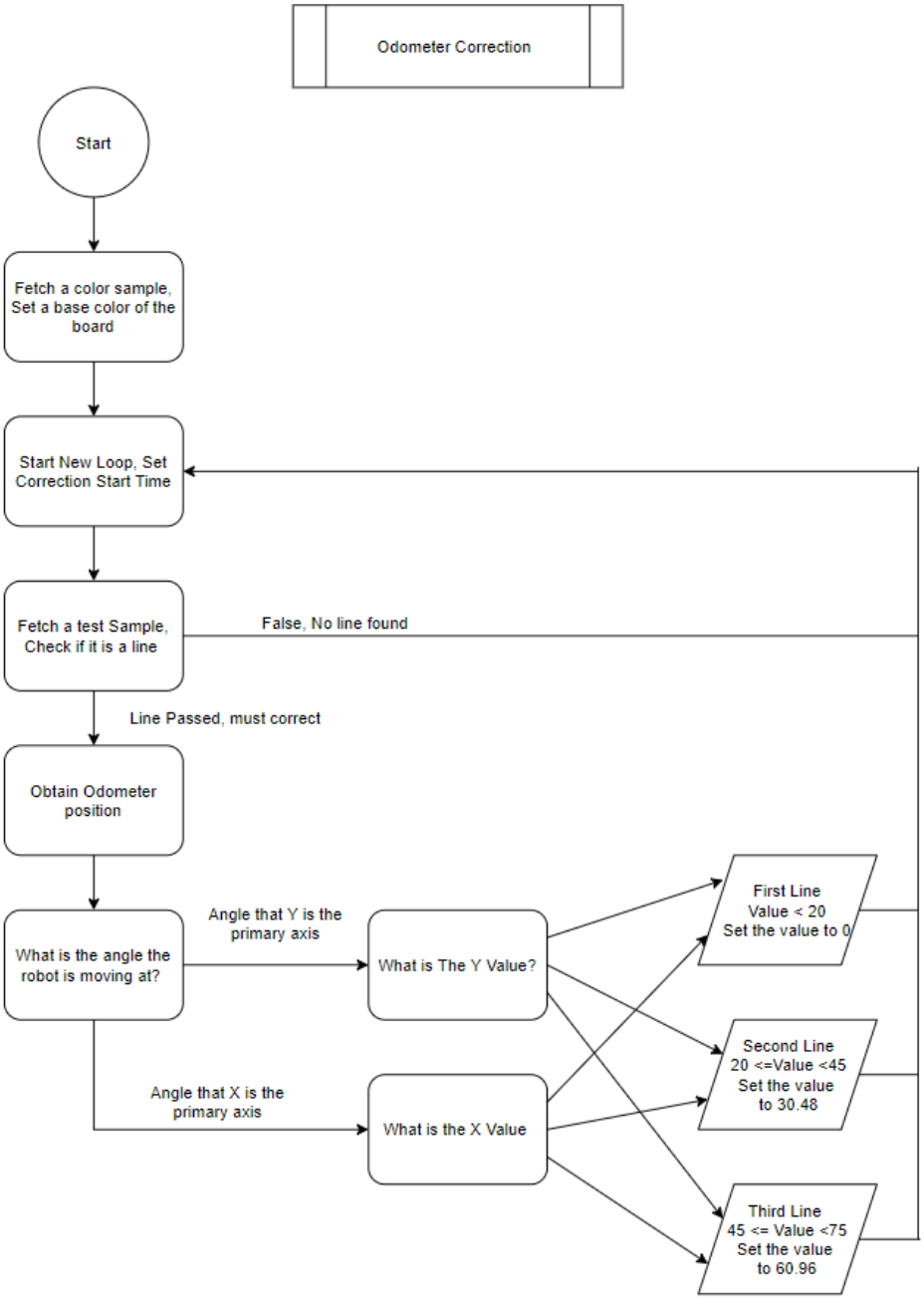


Figure 3.

In terms of software, the first thing coded was the odometer. Outside of the infinite loop our code resides in, two instance variables were made. oldRightTachoCount and oldLeftTachoCount were both initialized to zero and designed to record the tachometer count of the previous loop. Within the run method's infinite loop, first the update's start time is logged and then the code we implemented begins. The tachometer count for both left and right motors is received using the method getTachoCount(). First the distance travelled by both motors is calculated using the equation below.
$\Delta MotorDistance = (\pi/180) * WheelRadius * (MarginalTachometerCount)$
Marginal Tachometer Count is found by subtracting the OldTachoCount from the respective current TachoCount. The robot's change in distance travelled is then found by averaging the change in distance traveled by the motors. The change in Theta is calculated by the following equation: $\Delta ThetaRad = (leftDist - rightDist)/TRACK$. Right is subtracted from Left because in our design left is the positive direction for rotation. Then the current Theta is obtained and turned into radians so that the calculated change in theta can be taken into account when updating X and Y. After this, the delta for X and Y are calculated using the equations $\Delta X = \Delta distance * sin(angle)$ *and* $\Delta Y = \Delta distance * cos(angle)$ respectively. Then the update is sent to carrying delta X, delta Y and delta Theta after theta is turned into degrees. Lastly the current tachometer count's become the old tachometer counts to prepare for the next loop update.

The OdometryCorrection has the majority of our code in the the run method. A simplified flow of the program is visible in Figure 4. When first called, the color sensor gets a color sample to set a base for the board's color, to be used in a comparative analysis for a line. At the beginning of each loop a correctionStart variable is set to the current time. Then the Color sensor fetches a sample and if it is of a darkness below .20 or significantly darker than the base color, a line is passed and a boolean passline becomes true. If there is no line, the correction ends and the loop reiterates. If there is a line passed then the correction code begins and the current position is then obtained from the odometer. First the theta is checked to determine which axis the robot is following. If the angle is between 45 degrees and 315 or between 135 and 225, the robot is moving in along the y axis, x axis otherwise. Then the current value along that axis is judged, if it is less than 20, it is presumed the line passed is the starting line so the value is set to zero. If the value is between 15 and 45 the line is likely the second line so the value is set to 30.48 the length a full square on the board. If the value is between 45 and 75 it is likely the third line so the value of 60.96 is assigned to the axis value. Then the odometer is given the corrected values and the passline boolean is set to false again. The loop then reiterates. In without correction, at the end of the square the robot's display will show the perceived error. In with correction, a value relative to the (0,0) origin [the upper right corner of the starting square] is displayed.

Figure 4.


Odometer Correction
Start
Fetch a color sample, Set a base color of the board
Start New Loop, Set Correction Start Time
Fetch a test Sample, Check if it is a line
False, No line found
Line Passed, must correct
Obtain Odometer position
What is the angle the robot is moving at?
Angle that Y is the primary axis
What is The Y Value?
Angle that X is the primary axis
What is the X Value
First Line Value < 20 Set the value to 0
Second Line 20 <=Value <45 Set the value to 30.48
Third Line 45 <= Value <75 Set the value to 60.96

2. Test Data

Without Correction

Euclidean Error Distance Calculations:

$$Hand\ Error\ =\ Hand\ End\ -\ Hand\ Starting$$

$$\sqrt{(Hand\ Error\ X\ -\ Robot\ End\ X)^2 + (Hand\ Error\ Y\ -Robot\ End\ Y)^2}$$

Without Correction										
Trial #	Hand Starting X	Hand Starting Y	Hand End X	Hand End Y	Hand X Error	Hand Y Error	Robot End X	Robot End Y	Robot End Theta	Euclidean Error Distance
1	14.2	12.3	15	9.4	0.8	-2.9	0.82	-0.05	359.78	2.850070175
2	14.8	12.7	10.5	13.3	-4.3	0.6	0.37	-0.33	357.87	4.761701377
3	14.5	12.7	10.8	14.5	-3.7	1.8	-0.02	-0.25	359.87	4.212469584
4	12.2	12.1	10.2	10.5	-2	-1.6	0.44	-0.25	359.08	2.788565940
5	12.8	12.6	9	14.4	-3.8	1.8	-0.14	-0.12	359.89	4.133037624
6	13.1	12.7	8.2	14.7	-4.9	2	0.46	-0.01	359.85	5.724482509
7	13	13.8	18.5	12.5	5.5	-1.3	0.072	-0.27	359.78	5.524860541
8	13.1	12	12.7	10.5	-0.4	-1.5	0.192	0.19	359.76	1.790688136
9	10.9	12.5	11	12.2	0.1	-0.3	0.363	0	359.76	0.398959898
10	11.1	12.1	10.7	12.4	-0.4	0.3	-0.23	-0.7	359.76	1.014347081

With Correction:

Euclidean Error Distance Calculation:

$$\sqrt{(Robot\ End\ X\ -\ Hand\ End\ X)^2 + (Robot\ End\ Y\ -Hand\ End\ Y)^2}$$

With Correction								
Trial #	Hand Starting X	Hand Starting Y	Hand End X	Hand End Y	Robot End X	Robot End Y	Robot End Theta	Euclidean Error Distance
1	-12.7	-10.9	-11.4	-9.2	-11	-8.3	359.76	0.98488578
2	-13	-11.3	-11.6	-8.3	-11.44	-8.76	359.76	0.487031826
3	-12.4	-10	-13.7	-7.9	-13.65	-10.5	359.76	2.600480725
4	-14	-10.8	-17.5	-4.4	-17.38	-9.25	359.88	4.851484309
5	-12.9	-12.1	-12.5	-9.6	-12.02	-10.56	359.76	1.073312629
6	-14.9	-10.9	-19.4	-5.1	-19.2	-10.78	359.64	5.683520036
7	-12.6	-11.5	-10.6	-11.5	-10.3	-8.64	359.76	2.875691221
8	-12.2	-11.4	-11.3	-10.2	-10.92	-7.89	359.76	2.341046774
9	-11.7	-11.5	-12.7	-10.8	-12.36	-10.99	359.76	0.389486842
10	-12.6	-12	-12.2	-12.4	-12.06	-11.24	359.76	1.168417734

3. Test Analysis

Without Correction		
	Mean	Standard Deviation
Robot X	0.2327	0.319674
Robot Y	-0.179	0.242554
Error	3.319918286	1.852012

With Correction		
	Mean	Standard Deviation
Robot X	-13.033	2.950322205
Robot Y	-9.691	1.24840743
Error	2.24554	1.819758539

1. The mean error is quite improved between the designs with and without error. The improved perception of the robot's position comes from the code written in odometry correction. However, the standard deviations are still similar, this is due to no code being changed in the SquareDriver class and no physical changes were made to the model that could improve performance. Functionally, the robot's knowledge of its position had been improved, but not its ability to perform a perfect square.
2. The way we coded our robot is that on the final stretch X is being corrected more frequently than Y. A X/Y value is being corrected for only if the robot is crossing a line across one of the axes. The order of these corrections go Positive Y, Positive X, Negative Y, Negative X. Since the error in the X direction has been corrected the most recently, we expect it to be smaller than the Y error.

4. Observations

A.) The error observed without correction was intolerable for larger distances. When we tried to test our robot by making it travel 5 times by the 3 by 3 grid distance, it fell off the board on its third lap. There are several factors that lead to the odometer being intolerable for larger distances traveled. These include the amount that the robot's wheels slip against the floor, a lack of motor synchronization, and sudden starts and stops that all build together to throw the robot off its trajectory. Moreover, there are errors resulting from the assumptions that the wheels are equidistant from the robot's centre, are parallel, and have a constant radius because the wheel axles can flex, changing their distance to the centre, and the rubber on tires can stretch or contract, changing the radius too.The assumption that the robot is capable of sampling the wheel motors' tachometers quickly enough, and that the wheels' motion between samples is at a velocity that is constant, will also contribute to the overall error.

B.) Overtime, errors will accumulate, causing a robot to be strongly off course at larger distances.We believe these errors will accumulate linearly with distance because as observed in our findings, our standard deviation was consistent across both trials, which leads us to believe that the increasing error would increase at a rate similar to that of our standard deviation.

5. Further Improvements

A.) Propose a means of reducing the slip of the robot's wheels using software.
When the robot crosses a line and discovers it is no longer following the the desired axis line outside of a band of tolerance, it can be presumed that slipping caused this event to occur. So when the robot discovers it is no longer following the correct angle, a proportional controller can increase the speed of the motor that has slipped to correct. Once corrected for, the robot would return to the square driver.

B 1.) If a robot has two sensors, they can both be placed at the front right near the wheels of the robot. At an ideal 90 degree angle, both sensors would cross a line at the same time. When one sensor detects a line, the robot records the distance covered until the second sensor detects the line. With knowledge of the distance between the two sensors and the distance travelled before the second, simple trigonometry shown if Figure 5 can be used to find the error. The following equation can be used to find the true angle off a perfect perpendicular travel across lines. The resulting theta would be added or subtracted from the current angle.

$$Theta = Tan^{-1}(Recorded\ Distance\ between\ reports \mid Known\ Distance\ between\ sensors)$$

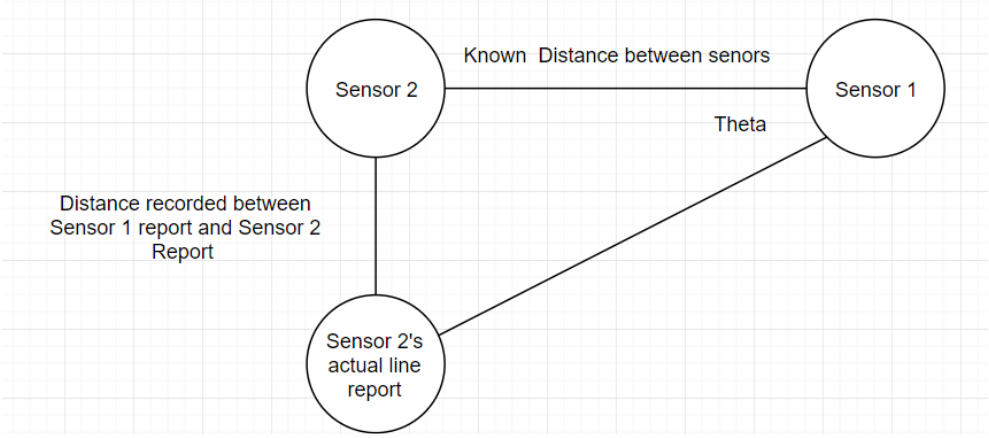


Figure 5.

B. 2.)In the case where there is only one sensor, when the robot approaches a corner of the square the robot should remember the distance from the last line it crossed. If the robot was following a perfect square the distance from the X axis crossing line should be equivalent to the distance to the Y axis crossing line. The angle can be corrected for in the manner displayed in Figure 6. The following equation would be used to find the theta:

$$Theta = cos^{-1}(Expected\ Distance / Measured\ Distance)$$

The resulting theta would be added or removed depending on whether the measured distance was larger or lower than the expected distance.

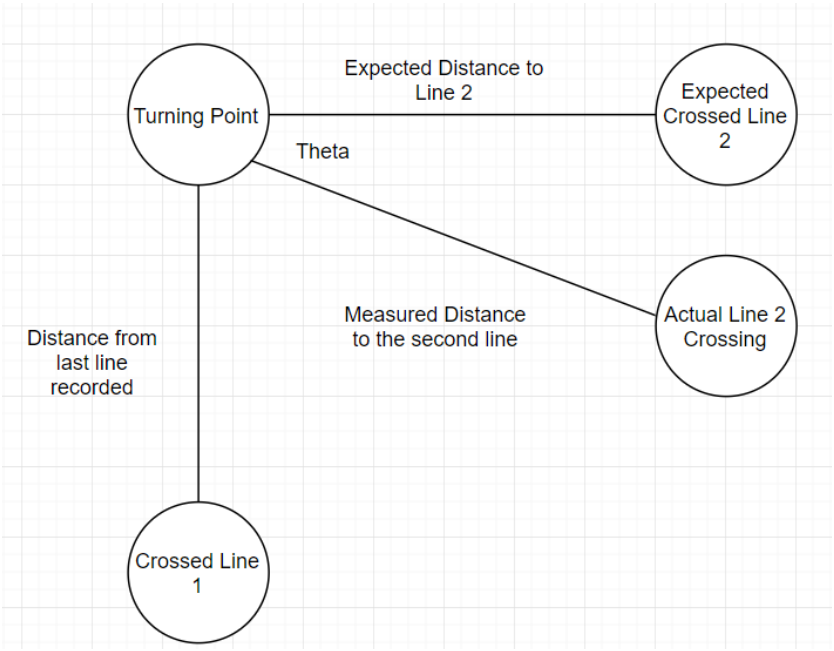


Figure 6.