Group 53 - Odometry
Alessandro Commodari 260636932
Asher Wright 260559393

Lab 2: “Odometry” Report

Data

ECSE 211 Lab 2 Data Step 3 (Without Light Sensor Correction):

	Robot Odometer Distance (cm)		Measured Distance (cm)	
Run #	**X**	**Y**	**X**	**Y**
1	-0.17	-.07	1.1	2.0
2	-0.45	-0.37	1.50	1.70
3	-0.27	-0.30	0.40	-0.1
4	-0.65	-0.4	3.5	3.7
5	-0.44	-0.55	4.3	3.5
6	-0.66	0.08	3.3	4.1
7	-0.22	-0.41	0.3	0.4
8	-0.29	-0.31	0.6	0.1
9	-0.95	-0.44	-0.5	-0.1
10	-0.19	0.12	0.4	0.7

ECSE 211 Lab 2 Data Step 5 (With Light Sensor Correction):

	Robot Odometer Distance (cm)		Measured Distance (cm)	
Run #	**X**	**Y**	**X**	**Y**
1	.76	.05	.6	.3
2	0.70	-1.11	0.6	0.40
3	-0.29	0.58	0.3	-0.20
4	-0.13	.75	1.0	0.80
5	0.45	.25	0.80	0.50
6	0.46	.00	0.80	0.30
7	0.51	.34	0.90	0.50
8	-1.01	1.27	0.10	1.40
9	0.11	.01	0.80	0.20
10	.5	.74	1.0	0.8

Data Analysis

a)
The formula for standard deviation is (taken from Wikipedia):

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}.$$

To calculate the standard deviation in this case, we used Microsoft Excel.

NOTE: There was conflicting information on of what we were supposed to calculate the standard deviation. We think that it is of the Error in X and the Error in Y. In case it was also supposed to be just the x and y readings, those are included too.

Standard Deviation without correction:

	Odometer X	Odometer Y	Measured X	Measured Y	Error X	Error Y
STDEV (cm)	0.255	0.229	1.630	1.660	1.700	1.649

Standard Deviation with correction:

	Odometer X	Odometer Y	Measured X	Measured Y	Error X	Error Y
STDEV (cm)	0.548	0.636	0.296	0.429	0.430	0.552

Assuming that we want to discuss the standard deviations of the errors of X and Y:

	Pre-Correction X	Pre-Correction Y	Post-Correction X	Post-Correction Y
STDEV (cm)	1.700	1.649	0.430	0.552

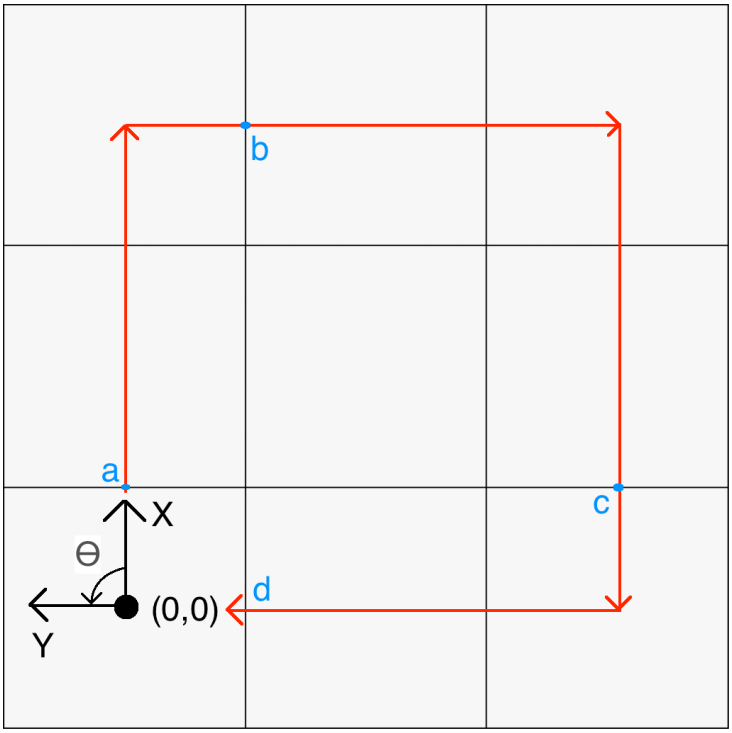
The standard deviation of the error between X and Y significantly decreased when the correction was introduced. It went from 1.7 to 0.43 in the X, and from 1.65 to 0.55 in the Y. This is roughly 1/3 in the X and the Y.

Before the correction, the error of X and Y was completely related to how good of a square the robot was making, and how accurate the wheel radius and track values were. If the robot made a perfect square (with perfect values), then the robot's odometer and the actual position would be (0,0). However, since the square was not perfect (and sometimes bad, due to slipping or wheels shifting and track increasing), the robot's odometer was not accurate. Whereas it thought it was traveling in one square, it may have been traveling in another (due to track/wheel radius/slipping), and the readings would be off.

After the correction, the error of X and Y was no longer completely related to how good of a square it was making. Instead, the robot's position was corrected whenever it hit a black line, providing it with real-world information. This information theoretically meant that, no matter how poor of a square the robot made, it would always know where it was, and display correct readings on its odometer. This is why the error lowered so significantly. The reason the error is still not zero is because the robot can still slip and/or have poor values for radius/track, and these can result in error after a correction has been made (before the next correction or at the end).

b)

To clearly answer this question, it is important to understand how the Cartesian plane was defined, and the direction in which it travels.



X is oriented "up", in the direction the robot first travels, and Y is oriented "left", 90 degrees counterclockwise to X.

With this coordinate system, the error in the Y position is expected to be smaller. This is, simply put, because the Y position is updated last.

Points "a" and "b" show the first updating of the X-coordinate and Y-coordinate respectively.

Points “c” and “d” show the final updating of the X-coordinate and Y-coordinate respectively.

In-between “c” and “d”, the X and Y coordinates are accumulating some error (which could be the result of many factors). However, once the robot hits point “d”, the Y-coordinate is “fixed”, but the X-coordinate is not. Thus, it is expected that the error in the y position is smaller.

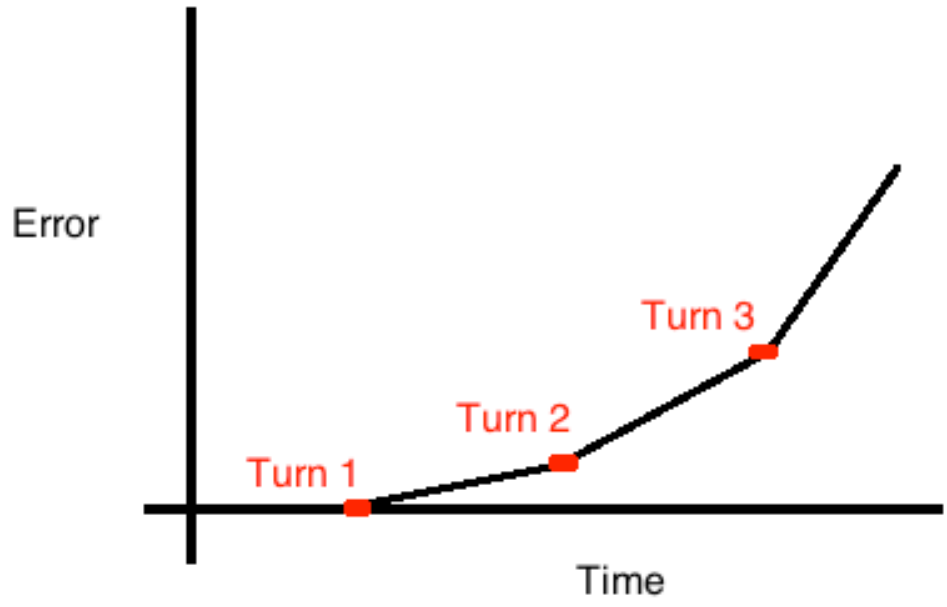
Observations and Conclusions

The error is not tolerable for larger distances and it will continue to grow for larger and larger distances as it compounds on itself. Even after having fine-tuned the robot to rotate as close as possible to 90 degrees the robot will always either over or under rotate by a certain amount. This means that the larger the distance it has to travel after rotating the greater the angle deviation will be. Also this error gets compounded with every rotation it takes leading to even more significant errors ultimately meaning that this odometer (without correction) is intolerable for larger distances.

The growth of the error is dependent on the path of the robot. Assuming that **the robot’s path is a square**, the error will grow “kind-of linearly” (see below). To explain, if, on a turn, the robot overturns an amount, theta, then **the error will grow in a linear piecewise fashion**. This is because, after each turn, the angle that the robot is off will increase (theta, 2*theta, 3*theta, etc.).

Before the first turn, there is expected that there is no error. After the first turn, it is expected that the error in X and Y will grow linearly, as the angle is now off by theta. After the second turn, the error in X and Y will grow linearly, but **double** the amount they were before, as now the angle is off by 2*theta. For the n^{th} turn of the robot, the error in X and Y will increase linearly, but n-times the amount as initially, since the angle is off by n*theta.

The graph of the error in X and Y relative to time will look something like this (for a square).



Although the error is “linear” between turns, the slope of the line increases with every turn (since the error is “compounded”). The overall error is more like an exponential, or a parabola.

This is even just assuming overturning, and not taking into account slipping. The longer the distance the higher the chances of external factors also affecting the robot are. Without any correction, these factors will add up, and the error will grow with respect to travel distance.

Further Improvements

a)

There are a few ways one can reduce the slip of the robot’s wheels. Using software only, the best way is to lower the acceleration of the motors of the robot. You can also adjust the speed (lower speed generally will mean less slipping), but the acceleration is the most important property to change. If the acceleration of the motors is much lower, then there will not be as much slip.

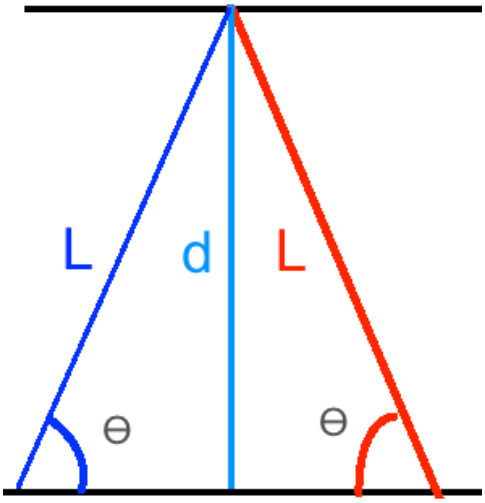
b) i)

With two light sensors, it is not too difficult to correct the angle reported by the odometer. The best way to do this would be to put a light sensor in front of each wheel. Then, you can correct the angle based on the difference in time of when the two sensors read the same black

line. There will be no difference in time if the robot is heading perfectly perpendicular to the black line, as the left and right sensors will hit at the same time. Thus, the odometer would know the exact angle to report (90*k, k integer). If the right sensor hits first, then the robot is counterclockwise from the perpendicular to the black line. If the left sensor hits first, then the robot is clockwise from the perpendicular to the black line. To correct the odometer angle in one of these cases,
This would not necessarily require the robot to stop.

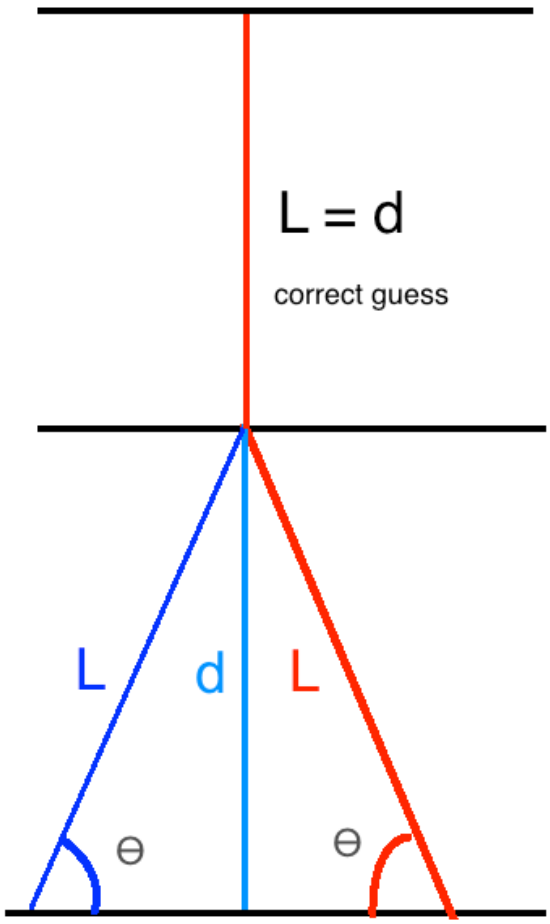
ii)

With one light sensor, it is more difficult to correct the angle, theta. One solution involves recording the amount of distance it takes the robot to travel from one black line to the next. Then, the robot can compare distance it thinks it has travelled in the X (X points perpendicular to black lines) to how long the blocks are. The robot can then determine theta, the angle that the robot is off from its heading. One problem with this is that the robot will not know whether or not theta is clockwise or counterclockwise.



Here, the red line is the actual path of the robot. The blue line is what the sensor *thought* the robot was taking. After comparing the distance from the first black line to the second (L), the sensor realized that it was different from expected (d). Thus, the sensor determined that it was off by: **theta = arcsin(d/L)**. However, the sensor does not know if the robot has taken the purple path or the red path.

A way to solve this issue is to make an educated guess, and keep track of it. The next time the robot hits a black line, if it is even more off than it was, the robot knows it guessed wrong. Here, it must double-correct, correcting x,y, and theta for the last block as if it had corrected theta initially. If, when it hits the black line, the distance is correct then the robot knows it has guessed correctly, and does not need to adjust. This can also be applied if the robot is about to take a turn.



In this case, the robot (correctly) guesses that it was on the red path. Thus, it corrects its odometer reading accordingly. After that, it compares the distance it took to reach the next black line to the expected distance, and found that they were equal. Thus, it knows it made the correct guess. Had it not, at this point, it would have to double-correct itself (and adjust the incorrect x, y, theta values).