ECSE 211 Design Principles and Methods

Lab 2 Odometry

Group 53
Spencer Handfield
260805699
Yi Zhu
260716006

1. Design evaluation

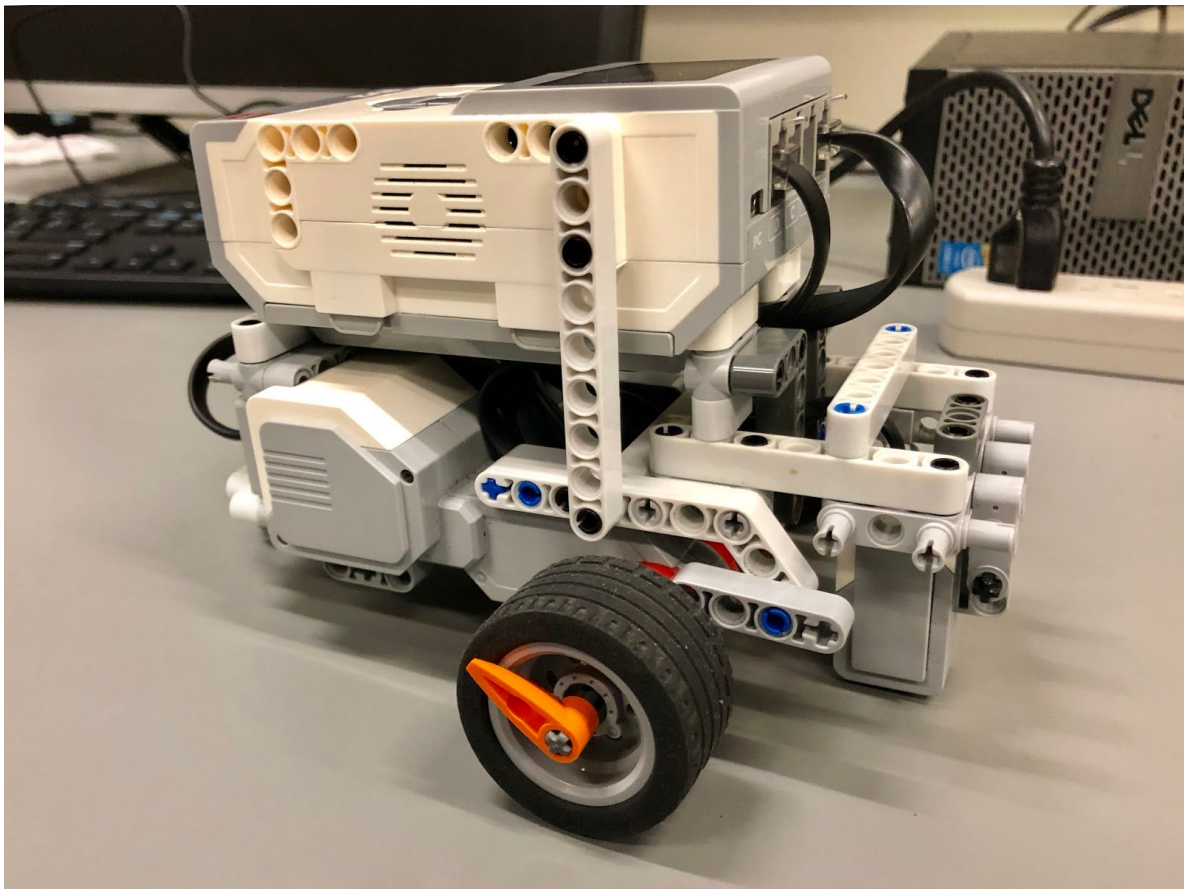

Figure 1

The hardware design of the robot has remained largely unchanged from the first lab as it proved to be a winning formula previously. The robot is still the 2 motors parallel to the brick which is closely propped up above it so as to reduce the width of the robot as much as possible. The only major change was simply extending out 2 more braces from the brick and the wheels support, slightly in front of the robot (previously where the ultrasonic sensor had been located) at a short distance once again to minimize the size of the robot. The sensor was placed in the center of the robot (to reduce central offset for future measurements), directly in front (to reflect precisely what the robot is traversing) and low to the ground (so as to minimize any possible interference). Varying heights were tested to minimal difference, thus the closest to the ground was decided on to eliminate any possible unobserved complications and make the safest decision. It was connected by ethernet cable. Upon completing the

hardware design, we took to writing the software, then tested for hardware flaws, making any necessary adjustments and with a finalized robot iteration, began testing to optimize its interaction with the software

The software came in two parts, firstly the simple odometer. As a means of convention, we always placed our robot on the bottom left corner of the of the test platform with the wall to its left and henceforth set that going forward (with wall to left) increased Y and going right would increase X (similar to a simple graph with XY-axis. Following what was presented in class, the odometer was mainly a large mathematical computation used to track the movement of the robot. It's process was, with the known circumference of the tire, knowing how many turns it had done in radians, converting that into an average travelled distance by the robot and finally using that data the calculate the angle at which it displaced and consequently the the individual x and y movement of the robot. This process began with acquiring the tacho value the robot read from its real world movement. Based on the degree to which the wheel turned, we calculate the displacement of both wheels, averaging them to get a net distance displaced by the entire machine (at this point the tacho count is reset to prepare for the next reading). The angle at which the robot is heading is calculated using those individual wheel movement readings and their possible difference with respect to the overall distance travelled and their proportional contribution, adjusting their respective coordinates. Lastly, with the average distance travelled and the individual wheel displacements, using simple trigonometry,the angle at which such is done is determined and thus, the instantaneous displacement of each axis was calculated. The angle was converted to degrees and all readings were passed to the screen. All the values read into these operations were either measured by hand or acquired directly by reading the robot's movements.

In terms of the OdometryCorrection. The premise is that the tiles on the ground are of a known and exact distance apart. Therefore, upon detecting a line and the subsequent one after that, the robot in theory, and under perfect circumstances, should know it has moved precisely the measured length of the tile. This need arises from any unseen and minor deviations from reality that previous method of measurement could incur (wheels slippage, hardware/software limitations) and thus corrects the readings. With that concept in mind, the software was designed as a simple counter which would keep track of the amount of lines crossed and updated the relevant coordinates as needed. Given that we know in these cases precisely how many tiles will be crossed

during a loop we set up our code to act as a sort of checkpoint counter. Upon crossing the first line, based on instruction, the Y coordinate was set to 0 to reflect the corner of the first tile as the origin. Upon crossing the next line (as detected by the light sensor) the Y coordinate would be updated to reflect the precise distance travelled (1 * TILESIZE). We then could extend that idea to all tiles. The added complication was that on the 4th, 7th and 10th line scan, the robot would be travelling in a different direction following its 90 degree turn (the original odometer handled maintaining overall correct reading but this adjustment meant our checkpoints had to actually correct the proper value). On top of this, the counter was not cumulative as it would be max 3 tiles in a single direction, therefore there were 3 checkpoints per direction based on the tile count and the direction (theta) of the robot. The order of these corrections also changed since when returning, the first checkpoint was multiple tiles away, but decreasing in value versus the first checkpoint leaving the origin which increased by 1 tile and continued to do so. The light sensor would be able to measure these checkpoint since it returned numeric values based on the color scanned and with only light brown wood and black lines, a large if statement that the black line appropriate value represented was the condition to enter the correction process. On top of the counter measuring which tile, an added theta condition was inserted to make doubly sure it was going in the right direction in the real world as well as in theory (acquired from the simple odometer). Lastly, the sensor was offset from the actual center of the robot which represented the origin in the odometer test, but not the correction test and thus, by measuring by hand we could create a sensorOffset variable to adjust the on screen readings to (since the sensor and the center of robot crossed the line at different times). A correction for this value was inserted in each checkpoint to reflect any minor deviation in direction, according to theta, and more accurately placing the actual center of the robot dynamically.

The flow chart of working mechanism is shown in figure 2.

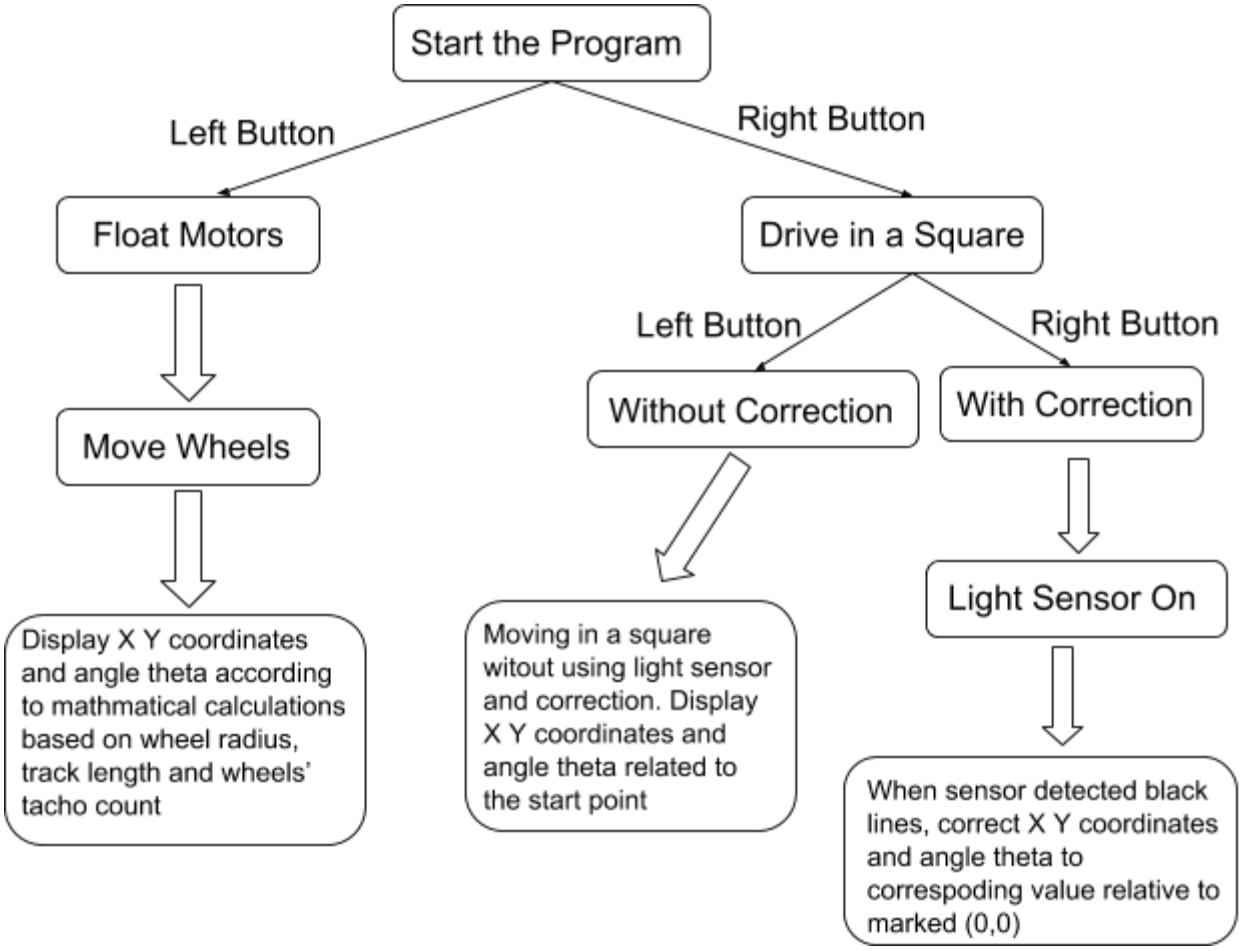
Start the Program
Left Button
Right Button
Float Motors
Drive in a Square
Move Wheels
Left Button
Right Button
Without Correction
With Correction
Light Sensor On
Display X Y coordinates and angle theta according to mathmatical calculations based on wheel radius, track length and wheels' tacho count
Moving in a square witout using light sensor and correction. Display X Y coordinates and angle theta related to the start point
When sensor detected black lines, correct X Y coordinates and angle theta to correspoding value relative to marked (0,0)

Figure 2

2.Test data

2.1 Odometer test

Odometry without correction (cm)					
	Odometer (display)		Actual (measured)		Euclidean error
Trial	x	y	x	y	e
1	-0.46	-0.83	-0.30	7.00	7.83
2	-0.62	-0.63	0.73	6.00	6.77
3	-0.83	-0.34	-3.30	8.80	9.47
4	-0.17	-0.01	-2.20	6.60	6.91
5	-0.49	-0.33	-2.80	6.10	6.83
6	-0.89	-0.04	-0.70	8.10	8.14
7	-0.25	-0.06	-2.30	3.40	4.02
8	0.32	-0.03	3.80	3.73	5.13
9	-0.51	0.17	-1.15	5.69	5.56
10	-0.33	-0.82	-1.30	3.90	4.82
Mean	-0.42	-0.22	-0.95	5.93	6.55
Standard Deviation	0.33	0.34	1.96	1.73	1.59

Chart 1

Odometry with correction (cm)					
	Odometer (display)		Actual (measured)		Euclidean error
Trial	x	y	x	y	e
1	-13.20	-9.42	-13.50	-9.80	0.48
2	-19.02	-8.21	-19.00	-9.00	0.79
3	-15.70	-8.25	-16.10	-9.70	1.50
4	-9.89	-3.22	-11.02	-5.00	2.10
5	-12.66	-12.36	-13.30	-15.60	3.30
6	-11.65	-8.38	-11.60	-13.40	5.02
7	-10.49	-9.48	-10.50	-11.50	2.02
8	-11.58	-5.47	-11.75	-9.80	4.33
9	-12.55	-9.97	-13.10	-14.00	4.07
10	-13.52	-12.30	-14.00	-11.10	1.29
Mean	-12.97	-8.31	-13.32	-10.87	2.62
Standard Deviation	2.53	2.65	2.43	2.84	1.50

Chart 2

Mean:

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

Standard Deviation:

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

Euclidean error:

$$\epsilon = \sqrt{(X - X_F)^2 + (Y - Y_F)^2}$$

3. Test analysis

3.1 How do the mean and standard deviation change between the design with and without?What causes this variation and what does it mean for the designs?

Interestingly enough the mean and standard deviation showcase trends not only between each of the systems but also within each of them as well. The first and most notable observable is actually, the one and only similarity both represent. The standard deviation for the euclidean error of both methods is near identical at 1.5. This means that while the systems may not have been perfect, they were both equally, consistently inaccurate to their own faults. This shows the consistency of the software to hardware relationship and real world effects on such. Yet the distinctions illustrate something much more interesting. Importantly, and almost the point of this exercise, is that the odometer with correction has a significantly lower euclidean error mean, meaning that it reflects much more accurately its current and real world position, its error mean is 2.62 compared to the 6.55 of the one without correction. This trend is clearly observable when we compare the means of the individual X and Y values, particularly the Y of the odometer without error is significantly more off than any other data collected in the entire experiment. The X coordinate fares better yet, overall the X and Y of the system with correction must be accomplishing their job as they demonstrate much more similar values between actual and displayed distances.

However, this does not take into account the standard deviation which demonstrates an entirely different trend. The standard deviation for all the measurements of the odometer without correction are vastly smaller than those of the odometer with correction. In other words, while the odometer with correction more properly displays where exactly it is, that location is much more deviated than the consistency of the one without. This turns into a similar question as to accuracy versus precision. The causes of this can be

vast, most easily explained is the accuracy (low euclidian error of the system with correction) which more appropriately translates the robot's location thanks to precise, predetermined distances as well as the axis reset allowing a measurement over a small distance (the last turn for each) and not allowing errors to pile up and compound in a faulty loop. The one without correction has no way of adjusting for any real world hardware to software issues, not to mention the real possibility of wheel slippage and other minor imperfections of the sort exist and won't be solved via the light sensor, thus explaining why it is less accurate as the errors continuously cumulate. Strangely the precision displayed through the standard deviation runs counter to these ideas as the robot without correction would more precisely run identical routes (albeit displaying less appropriate results) whereas as the one with correction would accurately display location but not return precisely and compactly each time.

However, most importantly, this does counterintuitively demonstrate the potential superiority of one of the designs, the one with correction. As mentioned previously, real world issues exist through wheel slippage, imperfect software hardware relationships, offset starting times for the wheels and so on, but at the end of its journey, with correction has taken into account much more effectively where exactly the robot situates itself. While we discussed the larger standard deviation, this could simply arise from the fact that the robot itself, with all its real world limitations, is not accurate itself in its return, but at least the design with correction more accurately reflects that (and critically its actual location) than the one without who may output more consistent results, but they are consistently more inaccurate as it repeats errors without anyway of solving them. Therefore the system with correction more effectively accounts for the imprecision of the robot with great accuracy, and could lead to the conclusion that the robot itself accurately returns, but is not precise in doing such.

3.2 Given the design which uses correction, do you expect the error in the X direction or in the Y direction be smaller?

We entirely expected that the error in the Y direction to be significantly more pronounced than that of the X coordinate and this is reflected by the

collected data in chart 2. The difference between the mean of the X hovers near the 0.5 mark yet that of the Y is practically 5 times such with one of around 2.5. This was expected due to the nature of the correction. In all our trials, there was always a minimum amount of drift away from the ideal path that the robot undertook when using the method with correction. It would either pull to a side due to slippage or not turn enough or too much at the corners when attempting to correct via modifying the real world variables in the code, thus slightly falsifying the values it would provide. Regardless of all the software and hardware tweaks we attempted. However, the correction always compensated by the end of the loop with accurate translation of the traversed tile distances, except for during one critical moment. The Y coordinate receives its final correction via the light sensor right before the 3rd turn. Once it passes the Y-axis into the negatives before returning towards the origin along the X one, it receives one final correction of it's Y: 0 and then continues the short distance (last side of the square loops) to the Y appropriate S location as set during the final lgiht adjustment and small last measurement via the default odometry system. However, as mentioned previously, the robot often drifted off the perfectly straight, 90 degree turn path and since this final correction only occured on the 3rd turn, it had the entirety of one side of the square to still traverse along the X axis before arriving at S where the Y coordinate, due to the faulty path, would be skewed by the necessary and default odometry system and not have a chance to correct, thus leading to always an accurate X assessment after its own set of final crucial correction, with an almost certain slightly wrong Y measurement thanks to the problematic drift measurements.

4. Observations and conclusions

4.1 Is the error you observed in the odometer, when there is no correction, tolerable for larger distance? What happens if the robot travels 5 times the 3-by-3 grid's distance?

It is tolerable for running only one loop. But since there is no correction in this moving mechanism, error grows as the distance grows. After travelling 5 times 3-by-3 grid's distance, our robot cannot even move back to the original grid.

4.2 Do you expect the odometer's error to grow linearly with respect to travel distance? Why?

Yes. The biggest error occurs at the starting point of the motor, not only the first starting point but also after turning. The lefter motor will start a little bit before the right motor due code execution. Since after each turning the error will occur once and there is no correction during moving, we expect the odometer's error to grow linearly with respect to travel distance. This then consequently affects the trajectory of the robot down a side of the square and the subsequent turn and so on, each time increasing the error and building further upon such with the constant amount of error, representing linear growth.

5. Further improvements

5.1 Means of reducing the slip of the robot's wheels using software

Firstly, by not using an immediate full speed start up since the jolt into full speed from the motor might cause additional slippage, a gradual build up to full movement speed on the tires instead of all at once would prove more effective as the literal 0 to Motor.setSpeed(FORWARD_SPEED) is too drastic and creates too much torque for the small tires to handle. To further solve this problem, we could us methods to synchronize both left and right motors before the robot starts moving straight. ie. use Motor.startSynchronization() for a specific motor to synchronize or use Motor.syschronizeWith(RegularMotor[] syncList) to synchronize one motor with other motors. Thus there would be no theta offset of its initial trajectory to begin a compounding error throughtout the drive

5.2 Propose a means of correcting the angle reported by the odometer using software when:

5.2.1 The robot has two light sensors

Place two light sensors in one line at the front of the robot. First record the distance between two light sensors. After starting up, record the time difference of two light sensors detecting the black line. Then multiply time difference by the speed of the robot. According to the geometry, the angle differences from center line and robot actual direction will be equal to theta showing in figure 3. It can be calculated by using arctan functions with distance robot moved and half of the sensor spacing distance.

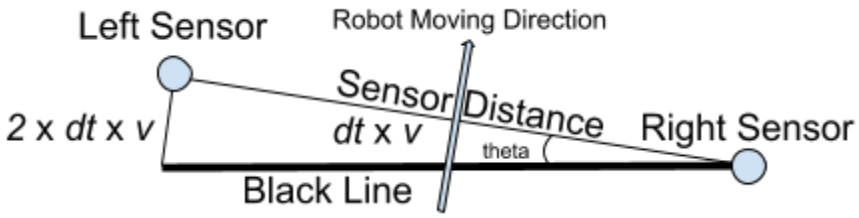

Figure 3

5.2.2 The robot has only one light sensor

First record the distance between two black lines. After starting up, record the time difference of the light sensor detecting two black lines. Then multiply time difference by the speed of the robot. The angle differences will be the theta showing in figure 4, calculated by using acos functions with grid length and distance robot moved.

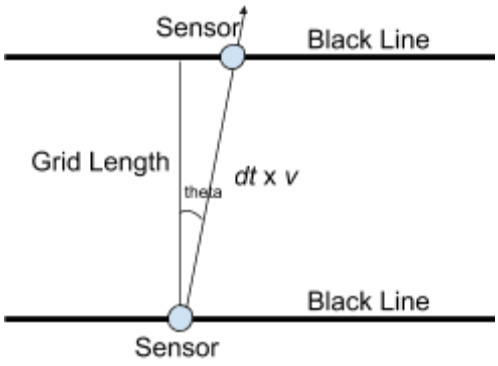

Figure 4