**Lab 2: Odometry**

ECSE 211: Design Principle &Methods

Group 27

Student Name: Roger Zhang

Student ID: 2608051430

Student Name: Boyang Ma

Student ID: 260674928

Faculty of Engineering   
Department of Electrical, Computer and Software Engineering  
McGill University  
September 27st 2018

**Design Evaluation:**

**Workflow Overview**

The overall workflow of the odometry laboratory includes three main steps: hardware redesign, software design and data tuning. The robot had to be reconstructed to meet the requirements of this particular lab and be flexible for future labs. Moreover, there were two major classes that had to be completed: odometer and odometry correction. According to the lab instructions, the odometry calculations were implemented first and followed by the odometry correction based on the data collected by the additional red mode light sensor. Last but not least, data tuning was essential to the performance of the robot. In order to accomplish the goal, a set of parameters must be carefully selected due to many external and internal uncertainties from hardware components such as the precision of the sensor and the quality of the motors.

**Figure 1: Detailed Workflow**

**Hardware Design**

The general structure of the robot was kept the same as the wall following laboratory. In order to decrease the distance between the two wheels, the brick was relocated above the motors. The supporting system of the brick was exactly the same as lab 1 (Figure 1). The two motors were kept parallel to each other in order to make the robot move linearly.

However, the ultrasonic sensor was replaced with the light sensor. The location of the sensor and the distance between the sensor and the ground was essential to the measurement updates and the accuracy of the collected samples. The light sensor should be as close to the ground as possible. Therefore, it was placed one centimeter above the ground to collect better samples. Our design was to measure the final coordinates from the center of the wheels to the X and Y axis. As a result, the light sensor was located at the front of the robot, approximately 4.5 cm away from the center of the wheels (Figure 2).

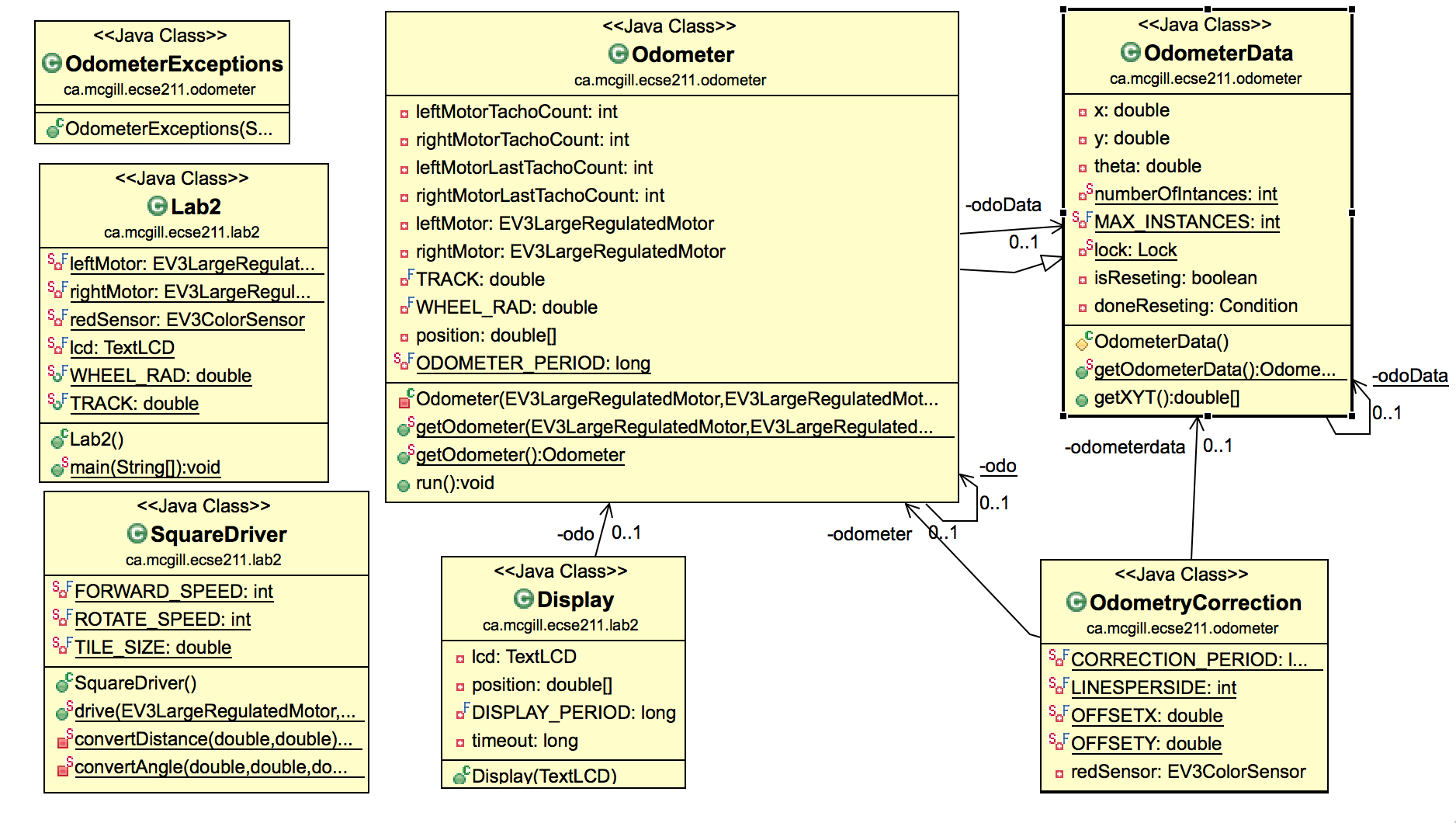
For the back weight supporting system, the rolling sphere was place right under the end of the brick instead of externally at the far back to decrease the length of the robot for more flexible movement during turns (Figure 3).

## **Software Design**

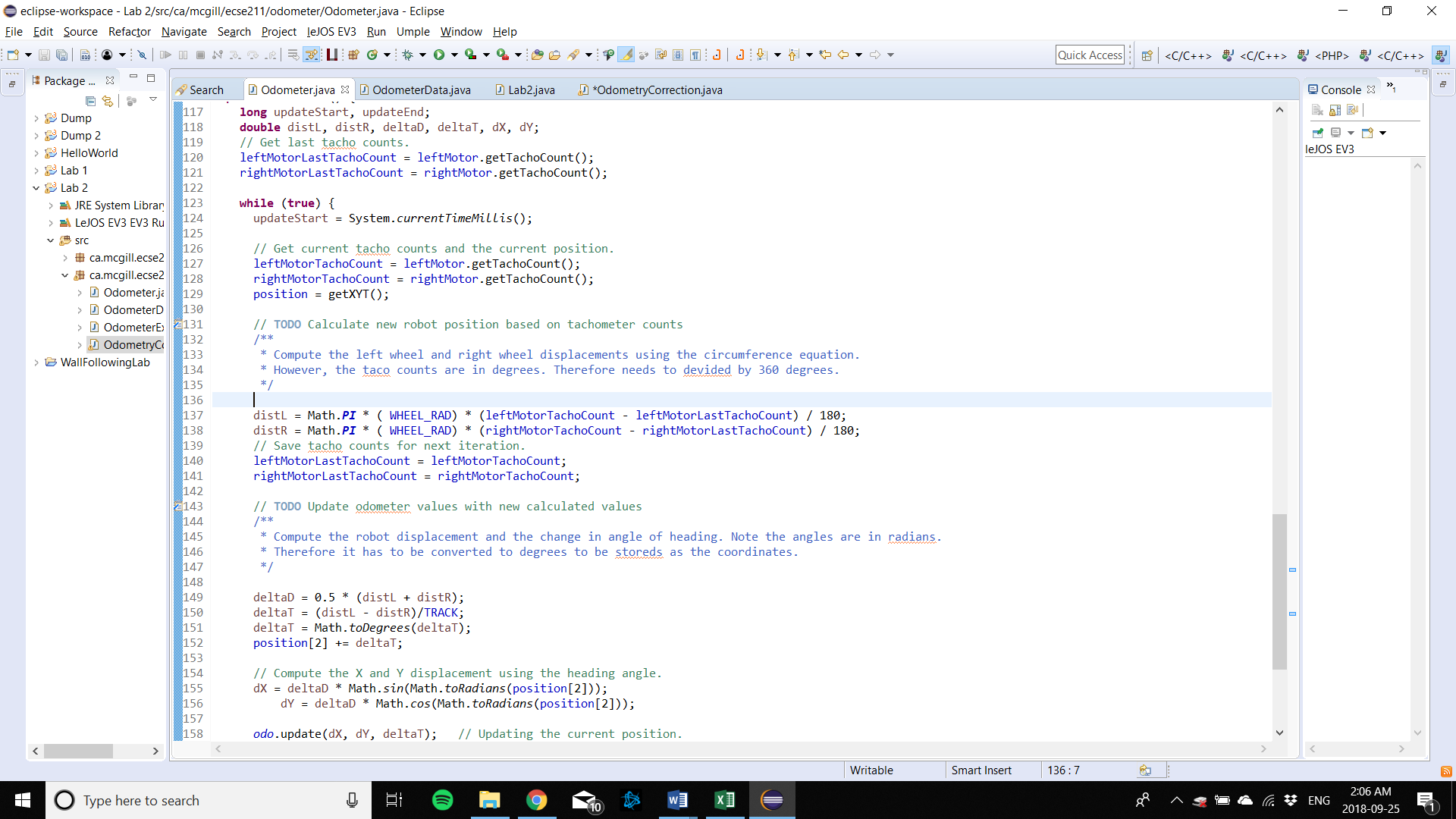
The software design for this lab implements 3 different modes for the robot: float, odometer without correction and odometer with correction. The float mod stops the motors without breaking and is used to verify how the coordinates and angle of heading update by turning the wheels manually. The odometer without correction mode mainly relies on an Odometer class and a Square Driver class. It makes the robot drive in a square-shaped trajectory and displays the coordinates of the robot, assuming it started at (0, 0). The odometer with correction mode is similar to the odometer mode but also implement an Odometer Correction class, which uses lines on the ground to correct the coordinated updated by the Odometer class.

The Square Driver class makes the robot drive in a square of size 3 tiles by 3 tiles. It simply makes the robot go forward for the length of 3 tiles and then rotates it 90 degrees. This procedure is repeated four times and at the end of it, the robot would have ideally returned to where it started. The job of the Odometer class is to compute and update the displacement and the angle of heading of the robot. By using the tacho counts of the wheels, it is possible to calculate the distance traveled by each wheel since the last period. These two distances can then be used to compute the angle of heading. This angle can finally be used with some trigonometry to figure out the X and Y displacement of the robot (Figure 2).

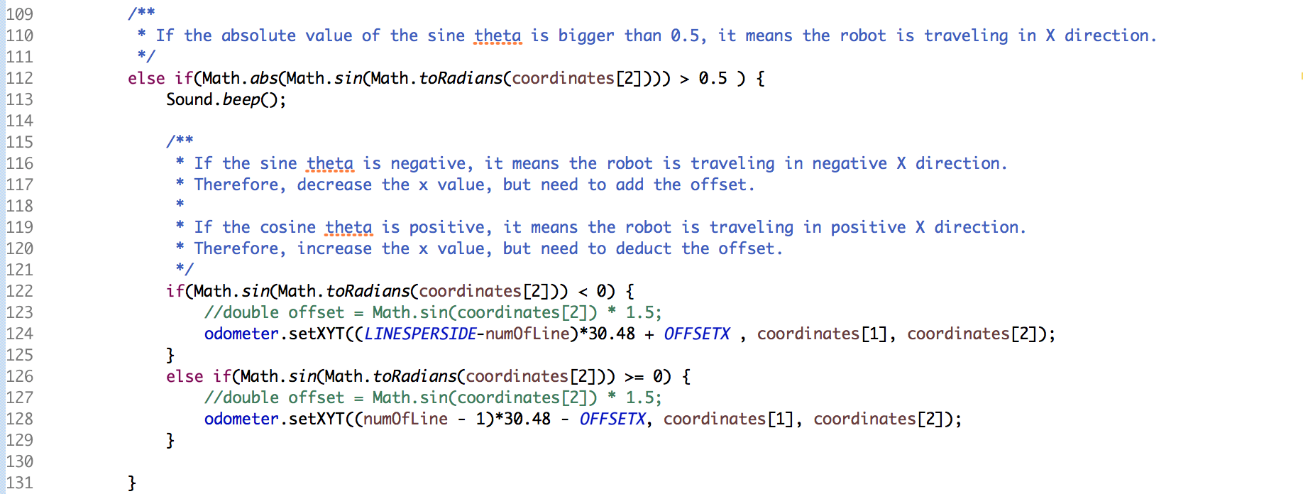
The Odometer Correction class uses readings from a color sensor to detect the lines from the tiles on the ground. These lines allow it to situate the robot and correct the coordinates computed by the Odometer class. Because the starting coordinates and angle of heading are known, it is possible to calculate and correct the coordinates of the robot by using some trigonometry and by keeping track of the number of lines the light sensor has detected (Figure 6 & 7). The direction of the robot is determined by the following rule: if the cosine of the angle is bigger than 0.5, then it means the robot is heading in the Y direction (Figure 7). Oppositely, if the sine of the angle is bigger than 0.5, it means the robot is heading in the X direction (Figure 6).



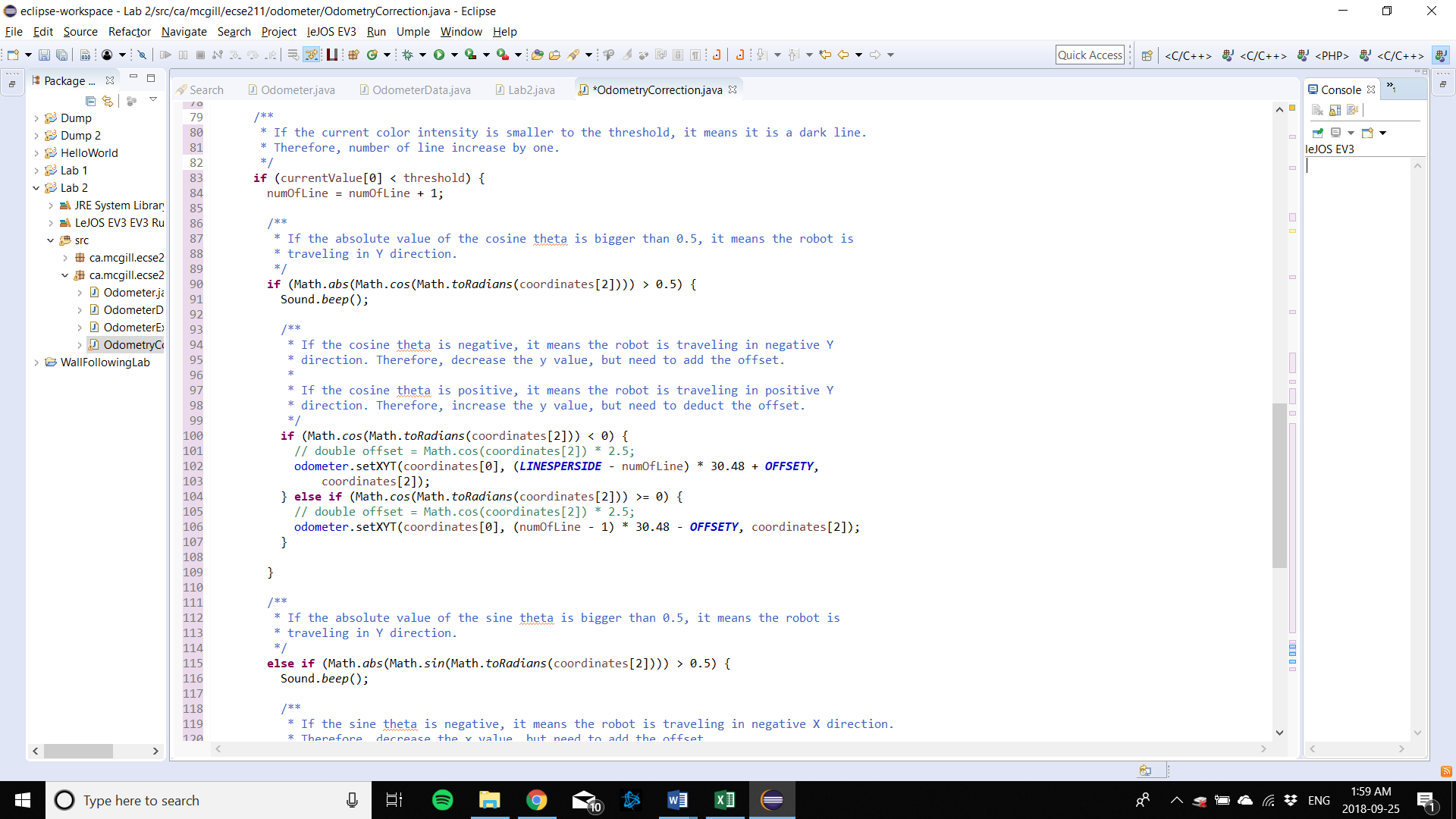
**Figure 4: UML Class Diagram (Using ObjectAID UML tool in Eclipse).**



**Figure 5: Computations for displacement and angle of heading**

****

**Figure 6: Odometer correction for the X value**



**Figure 7: Odometer correction for the Y value**

**Tuning Process**

In order to optimize the detection precision of the dark lines on the wood board and since the color intensity samples collected by the sensor are in percentage, red mode of the light sensor was chosen. The threshold of the color intensity was easy to find, since there is a distinguishable color difference between the black lines and the color of the wood board.

Due to the uncertainties of real-life measurements, another important parameter that needed to be tuned was the wheel base (TRACK) of the robot. Whether the robot goes straight and turns at a perfect 90 degrees depends on the value of the wheel base value, because it is directly related to the robot heading angle calculation. If the robot is turning more than 90 degrees, the TRACK number must be decreased. Oppositely, if the robot is turning less than 90 degrees, the TRACK number must be increased.

**Test Data:**

The robot with the odometer with and without correction were tested according to the following steps:

1. Place the robot at (0,0) if no correction is applied. If correction is applied, starting point is arbitrary.
2. Run the robot in a 3-by-3 tile square using its respective mode.
3. Measure the resulting final position with respect to the origin.
4. Note the reported X and Y values by the odometer

5. Measure the actual position XF and XY of the robot with respect to (0,0)

**Odometer test (Without Correction)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trials | XF | XF | X | Y | Euclidean error (𝛜) |
| 1 | 4.4 | -3.2 | 0.252 | 0.07 | 5.281931844 |
| 2 | 4.2 | -3.8 | 0.381 | -0.43 | 5.093295691 |
| 3 | 4.5 | -3.5 | 0.959 | -1.36 | 4.137424440 |
| 4 | 6.2 | -3.4 | 1.301 | -0.62 | 5.632814661 |
| 5 | 5.6 | -4.0 | -0.390 | -0.72 | 6.829238611 |
| 6 | 4.6 | -2.6 | 0.212 | -0.58 | 4.830625632 |
| 7 | 5.7 | -2.0 | 1.649 | 0.05 | 4.554246590 |
| 8 | 7.1 | -2.5 | 0.607 | -0.99 | 6.666269497 |
| 9 | 5.3 | -3.2 | 1.505 | -0.03 | 4.422095092 |
| 10 | 3.3 | -2.7 | 0.207 | -0.84 | 3.609189521 |

**Table 1: Real world coordinates (XF, YF), displayed coordinates (X, Y) and the Euclidean error of the odometer without correction**

**Odometer correction test:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trials | XF | YF | X | Y | Euclidean error (𝛜) |
| 1 | -12.1 | -14.5 | -11.07 | -13.7 | 1.304185570 |
| 2 | -8.80 | -10.3 | -7.400 | -6.80 | 3.769615365 |
| 3 | -10.1 | -14.2 | -11.30 | -14.7 | 1.300000000 |
| 4 | -15.1 | -13.1 | -14.70 | -13.35 | 0.471699057 |
| 5 | -15.8 | -14.5 | -14.70 | -13.35 | 1.591383046 |
| 6 | -13.5 | -13.5 | -13.26 | -13.47 | 0.241867732 |
| 7 | -15.1 | -12.4 | -15.04 | -12.40 | 0.060000000 |
| 8 | -15.3 | -14.6 | -15.04 | -14.41 | 0.322024844 |
| 9 | -9.50 | -10.7 | -9.070 | -10.35 | 0.554436651 |
| 10 | -19.1 | -16.8 | -18.21 | -16.35 | 0.997296345 |

**Table 2: Real world coordinates (XF, YF), displayed coordinates (X, Y) and the Euclidean error of the odometer with correction test**

**Test Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | X | Y | e |
| Mean | 0.6683 | -0.6349 | 5.1057130 |
| Standard Deviation | 0.662258 | 0.4473156 | 1.0402568 |

**Table 3: Mean and standard deviation of the displayed coordinates and the Euclidean error distance of the odometer without correction test**

|  |  |  |  |
| --- | --- | --- | --- |
|  | X | Y | e |
| Mean | -12.979 | -12.888 | 1.0612510 |
| Standard Deviation | 3.244316 | 2.6405715 | 1.0832649 |

**Table 4– Mean and standard deviation of the displayed coordinates and the Euclidean error distance of the odometer without correction test**

Considering the robot started at (0, 0) for the odometer tests without correction and at (-15, -15) for the odometer tests with correction, it seems, looking at the means, that the odometer without correction is more precise by around 1.5 cm. However, the final coordinates displayed during the tests without correction only relies on the number of rotations the wheels made during the test. Therefore, as long as the robot does not encounter any obstacle during its run, it will most likely think it has returned close to its starting point at the end of its trajectory. Yet, the Euclidean error distance (ε) attests to the contrary. It shows that on average, the robot is around 5 cm away from the origin. Therefore, the means of the coordinates are close to (0, 0) because the robot thinks it returned there, when it, in truth, is most likely around 5 cm off.

On the other hand, although the robot using the odometer with correction is on average around 2.1 cm away from the starting point, the ε is around 4 cm smaller. This shows that even though the robot using the odometer with correction does not end up at its starting point, it is better at knowing its real-world end location. This shows that it does a good job a correcting its coordinates during the tests.

The variation of the standard deviations can also be explained via these results. Because the robot often falsely thinks it has returned to the origin when using the odometer without correction, the end coordinates will always be similar from test to test and therefore the standard deviation will be small. The standard deviation for the odometer with correction is larger because it is better at knowing its real-world end coordinates. It, therefore, records a lot of different end coordinates because they actually do vary a lot from one test to another.

Given the design of the odometer with correction, the error in the X direction is expected to be smaller than the error in the Y direction. Indeed, the X or Y coordinates can only be corrected one at a time. For example, when the robot is travelling on the Y-axis, only the Y value is being corrected. Therefore, since the X value is the last coordinate to get corrected before the end of the tests, it is expected to have a smaller error then the Y direction. This expectation was observed while computing the Euclidean error distances. The error in the X direction is slightly smaller, on average, then in the Y direction. This difference would be more obvious for bigger squares.

**Observations and Conclusions:**

The error ε observed for the odometer with no correction would not be tolerable for larger distance. The error starts growing as soon as the robot stop going perfectly straight. Even if the robot is well tuned to turn as close to 90 degrees as possible, the robot will always either under or over rotate by a certain degree due to many uncertainties from the surroundings. Therefore, the longer the robot drives around, the larger the error is going to grow. If the robot travelled 5 times the 3-by-3 grid, the error would be at least 5 times as big.

The error might seem to grow linearly, but only between changes of directions. The growth of the error is directly associated with the path of the robot. For the square path like the requirements in this particular lab, even though there is a small error in the angle of heading at the beginning the error will grow linearly through one side of the square, then the error will grow bigger and bigger after every turn as long as the overturns exist. For example, before the first turn, it is expected that there is no error. After the first turn, the error of X and Y should increase linearly, as the angle now is already off by a delta theta. After the second turn, the delta theta will be at least twice as big as before. Therefore, for each straight section, the error would grow linearly, but the overall error growth graph would be something similar to an exponential function formed by four pieces of linear functions. Furthermore, unpredictable causes of error such as slipping at the turns add up the longer the test goes on and contribute to the error growth even more.

**Further Improvements:**

**Software improvement for reducing the slipping of the robot’s wheels:**

* One possible method to reduce the slipping of the robot wheels would be implement a method that increase the speed of the wheel linearly with a relatively small slope when the motor starts. In this way, the wheels would increase its speed gradually instead of the sudden acceleration at the beginning.
* Another possible way to reduce the slipping problem would be to decrease the speed of the wheels throughout its movement to minimize the effect of slipping during turns.

**Software improvement for angle correction reported by the odometer:**

* **The robot has two light sensors.**

If there are two light sensors, both sensors can be put at the front of the robot. One on the right side and the other one on the left side, but they must be located on the same line. In this way, it would be possible to get the time of the two sensors when they detect a black line. If there is a time difference between the time that the two sensors detect the black line, the robot is not heading straight. The angle can then be adjusted based on which sensor is slower. If the right sensor is slower, the angle must be increased on the odometer. If the left sensor is slower, the angle must be decreased.

* **The robot has only one light sensor.**

If there was only one light sensor, it would be harder to correct the angle of heading. However, one possible way to do so would be to measure the time it takes the robot to move from one black line to the next and compare with the calculated time that the robot supposed to use to go across one block base on the block length (30.48 cm) and the speed of the wheels. If it takes longer for the robot to detect the next black line, it means the robot is not going straight. However, the robot will not know which way it should correct, whether to increase the angle or decrease the angle. The robot would have to take an educated guess, but the impact would only be seen when the robot detects the third black line. If it takes even longer for the robot to detect the third line, it means the first guess was wrong, then a double correction must be done in the opposite way. If the time is what the robot is supposed to take, then it means the first guess was correct and there is no need to correct the angle.