ECSE 211
Design Principles and Methods
Fall 2019

Lab 2 Report:
Odometry

Group 21
Culha, Dafne (260785524)
Udupa, Suhas (260867826)

Section 1: Design Evaluation

In this laboratory, we were required to implement an odometer to accurately determine the position of a robot driving in a square pattern. We then had to implement correction to the odometer readings using a grid on the floor and a light sensor. Using the color sensor, the robot was able to analyse patterns on the ground and through the adjustments of its motor rotations, the robot was able to move around its environment.

Hardware Design:

First, we had to build a robot capable of driving in a square pattern. We used our design from the first lab as a starting point, since it was already built, and then proceeded with some minor modifications. Our robot consisted of two motors to which we connected wheels, one color sensor, one EV3 LEGO brick and various connecting pieces. To add more stability, we also added a caster ball in the back of the robot. The sensor was placed at the front of the robot, close to the brick, to minimize movement during rotations. We adjusted the positioning of the sensor to be as close as possible to the center of the robot (at the midpoint between the two front wheels) without changing the wheel placement in order for the sensor readings to more accurately represent the position of the middle of the wheel axis of the robot. It is also attached in such a way that it is as near as possible to the ground (approx. 0.5 cm) facing downward. This placement was designed in order to minimize noise and to assure that the sensor correctly detects the black line.

One issue we encountered during our testing was that the ball in the back would sometimes drag on the floor, thus creating unnecessary friction and preventing our robot from performing the sharp 90-degree turns. To solve this problem, we tilted the processor brick to the front and hence shifted the weight onto the wheels to relieve the pressure from the ball and allow the wheels to slip less. By placing more weight on the wheels, the pressure was increased on the wheels and this significantly enhanced traction. Our robot's final design is shown in Figure 1.

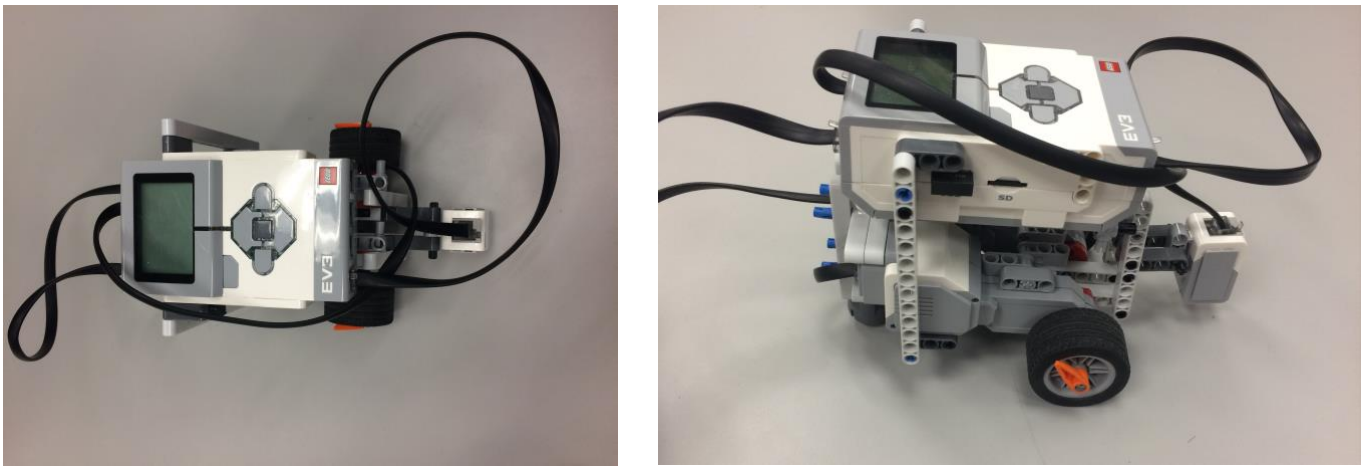

Figure 1: Hardware Design of the Robot

Culha, Dafne (260785524)
Udupa, Suhas (260867826)

Software Design:

After designing our robot, we began to work on the software component of the lab.
To implement the odometer, we used the sample code provided for this laboratory and the example given in the lecture slides. In essence, we modified the odometer class to calculate the x and y positions of the robot based on the displacement calculated from the TachoCount of both wheels (which is the number of turns both wheels make). The robot's heading on the other hand is calculated from the difference in displacement of the left wheel and right wheel, as seen in the snippet of code in Figure 2 below.

```
while (true) {
  updateStart = System.currentTimeMillis(); // record start time

  leftMotorTachoCount = leftMotor.getTachoCount(); // get current tachocount for both wheels
  rightMotorTachoCount = rightMotor.getTachoCount();

  leftDist = (WHEEL_RAD * TO_RAD * (leftMotorTachoCount - lastTachoL));
  rightDist = (WHEEL_RAD * TO_RAD * (rightMotorTachoCount - lastTachoR));

  lastTachoL = leftMotorTachoCount; // save for next iteration
  lastTachoR = rightMotorTachoCount;

  deltaD = (leftDist + rightDist) * 0.5;
  deltaT = ((leftDist - rightDist) / TRACK) * 90 / Math.PI;

  theta += deltaT;

  dx = deltaD * Math.sin(theta * TO_RAD);
  dy = deltaD * Math.cos(theta * TO_RAD);

  odo.update(dx, dy, deltaT); // Update odometer values with new calculated values
```

Figure 2: Snippet of Code from the Odometer Class

We then conducted a test to adjust the wheelbase constant (*TRACK*) of our robot in order for the robot to make a proper 360-degree turn. We realized that if the wheelbase value was too large, the robot would overspin (beyond 360 degrees) and if the value was too small, the robot would underspin (below 360 degrees). From our tests, we settled on the value of 9.56 cm for the wheelbase value, which gave the most accurate 360-degree turn for our robot. Moving on, the Odometer Correction uses light sensors to detect and keep track of the number of black lines of the square grid. It also uses known values of the distance of each tile to reduce the error in the odometer readings. We decided to use the "red" mode of the sensor. We then set up a sample provider in the code to retrieve the data from the sensor. To check whether the robot crossed a black line, we checked for a difference in color ratio (i.e. black line to wood floor). By setting an appropriate threshold of the values, the color sensor was able detect the blackline.

Correction is made whenever the sensor detects a blackline. Knowing the length/width of the square tiles and the number of tiles (number of black lines the color sensor reads) in the square traversal, we were able to set up the software such that when the robot encountered a line, it would correct the internal value of its position. We used two counters (one for x and one for y) to keep track of which line the robot is at and to hence correct the robot's position accordingly.

Culha, Dafne (260785524)
Udupa, Suhas (260867826)

When a blackline is detected, then based on the angle of the robot, we decide which value should be corrected and which counter should be incremented/decremented. If the angle is around 0 or 180 degrees (robot is moving along the y-axis), then the y-value is corrected, and the y counter is incremented if the robot is moving up and decremented if the robot is moving down. On the other hand, if the angle is around 90 or 270 degrees (robot is moving along the x-axis), then the x-value is corrected, and the x-counter is incremented if the robot is moving right and decremented if it is moving left. The software design is represented in Figure 3.

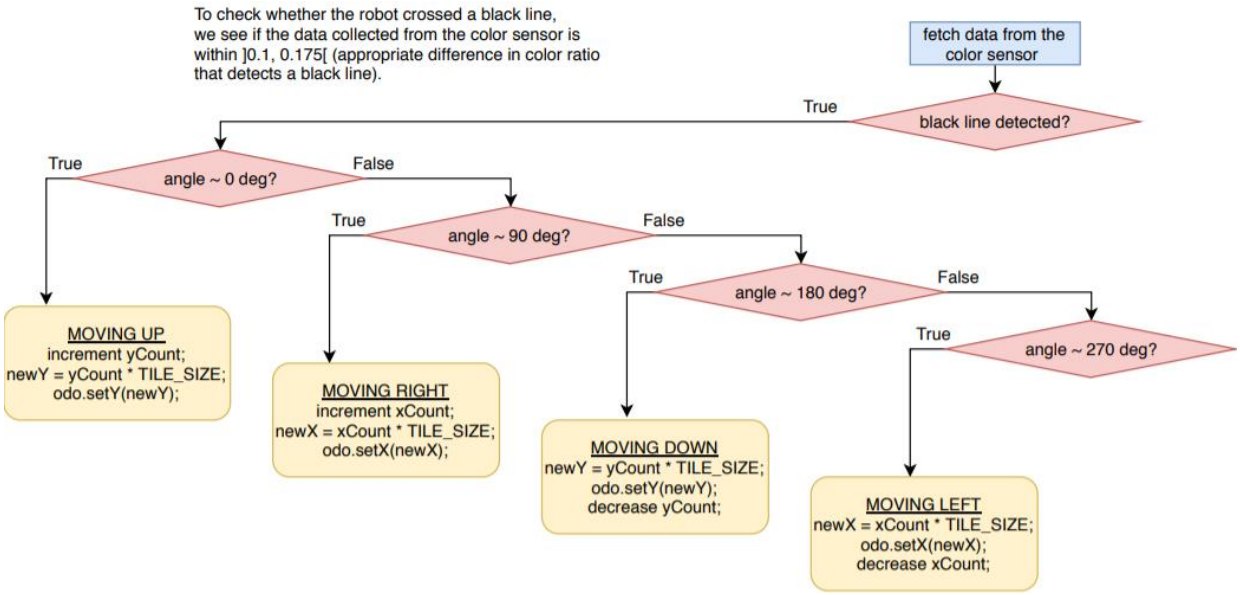

Figure 3: Flowchart Explaining Odometry Correction

Section 2: Test Data

The first table, as seen below, presents the distance displayed by the odometer (X, Y) as well as the actual measured distance of the robot's position (X_f, Y_f) for 10 independent trials in a 3x3 square *without* using odometry correction.

Table 1: Data from Odometer Test

Trial Number	X (cm)	Y (cm)	X_f (cm)	Y_f (cm)
1	0,65	0,70	2,00	0,90
2	-0,53	0,81	0,80	2,20
3	0,79	1,31	0,80	4,00
4	1,78	1,60	-2,00	0,90
5	0,53	1,40	3,00	2,30
6	0,90	0,76	1,00	4,00
7	-1,20	-1,28	0,40	-1,30
8	0,46	0,68	2,00	1,00
9	0,44	-0,43	1,00	2,00
10	0,53	0,81	0,70	3,00

Culha, Dafne (260785524)
Udupa, Suhas (260867826)

The second table presents the distance displayed by the odometer (X, Y) as well as the actual measured distance of the robot's position (X_f, Y_f) for 10 independent trials in a 3x3 square *with* the use of odometry correction.

Table 2: Data from Odometer Correction Test Presenting X, Y, X_f, Y_f

Trial Number	X (cm)	Y (cm)	X_f (cm)	Y_f (cm)
1	14,93	14,83	16,20	17,00
2	15,71	14,05	17,80	13,20
3	17,10	14,08	19,30	12,70
4	17,14	14,05	15,40	13,40
5	16,10	15,38	14,30	16,70
6	17,17	15,14	13,00	18,50
7	15,46	15,20	12,00	18,00
8	16,02	15,20	17,00	13,00
9	15,93	16,83	16,00	14,00
10	15,46	14,83	16,00	17,00

Section 3: Test Analysis

The Euclidean error distance ε of the position was calculated for every trial using the following equation (1). The Euclidean error is a good method of determining the order of the error, as it takes into account the error in X and in Y. In the following equation, X is the value displayed by the odometer and X_f is the measured value.

$$\varepsilon = \sqrt{\left(X - X_f\right)^2 + \left(Y - Y_f\right)^2} \quad (1)$$

The Euclidean error can be seen in Table 3 for the odometer test without correction.

Table 3: Euclidean Error from Odometer Test

Trial Number	ε
1	1,36
2	1,93
3	2,69
4	3,85
5	2,63
6	3,24
7	1,60
8	1,57
9	2,49
10	2,20

Culha, Dafne (260785524)
Udupa, Suhas (260867826)

The Euclidean error can be seen in Table 4 for the odometer test with correction.

Table 4: Euclidean Error from Odometer Correction Test

Trial Number	ε
1	3,00
2	2,00
3	2,59
4	1,86
5	2,23
6	5,35
7	4,45
8	2,41
9	2,83
10	2,23

The mean μ of the error in X and Y were calculated using the absolute value of the errors as seen in equation (2). This was done as the error in the odometer's measure would sometimes be negative as well. The mean of the Euclidean error was calculated using the same equation, although it only comprised positive values. In the following equation, Z_i is the error for each trial and N is the number of trials.

$$\mu = \frac{1}{N}\sum_{i=1}^{N}|Z_i| \tag{2}$$

The standard deviation σ of the errors in X and Y as well as the Euclidean error were calculated using equation (3). The standard deviation formula gives the average deviation of each error in comparison to the mean of all errors. This gives us an idea of how large the variation in error is. In the following equation, Z_i is the error for each trial, N is the number of trials and μ is the mean.

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(Z_i - \mu)^2} \tag{3}$$

Mean and standard deviation values for X, Y, and ε are presented in Table 5 for the odometer test and in Table 6 for the odometer correction test.

Table 5: Standard Deviations and Mean Values of X, Y and ε in Odometer Test

	Standard Deviation (cm)	Mean (cm)
X	0,80	0,44
Y	0,87	0,64
ε	0,79	2,36

Culha, Dafne (260785524)
Udupa, Suhas (260867826)

Table 6: Standard Deviations and Mean Values of X, Y and ε in Odometer Correction Test

	Standard Deviation (cm)	Mean (cm)
X	0,79	16,10
Y	0,84	14,96
ε	1,12	2,87

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

The standard deviation represents the accuracy of the test values to their respective mean. In both tests, the robot aims to finish where it started. However, the means and standard deviations of odometer test with correction and without correction are very different and this makes sense due to using a different coordinate system for both tests. They consider different points to be the origin and this causes the drastic change between standard deviations. In the test without correction, origin is considered to be the center of the robot and the values displayed are with respect to where the robot starts moving. Hence, a mean of X:0,44cm and Y:0,64cm describes an average offset from where it began relative to its exact starting position. However, in the odometer test with correction, the origin is considered to be the bottom left corner of the grid. As the robot started moving from the middle of the tile, mean values of X:16,10 cm and Y:14,96 make sense as the robot aims to finish as close as possible to the starting position.

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

We would expect error in the x direction to be smaller than that of y. Our test data also supports this claim since our robot had more error in the y-direction than it did in along the x-axis (see Table 2). While driving in a square, our robot last passes through a grid line along the x-axis (in the 270-degree angle) thus the x position is corrected until the very end of the path. However, the y position is last corrected only 3 lines prior which causes more error in y coordinate. Also, our robot performs the final turn after correcting the y-axis. However, the turns in general are never exactly perfect and this tends to make our robot deviate even more and cause more error in the y-axis.

Section 4: Observations and Conclusions

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

The error would not be tolerable for large distances. The robot doesn't receive any information on its current position without the correction and the errors aren't corrected. Thus, any error made by the odometer on the estimation of its position would accumulate and more errors would keep building up. To illustrate, an error made by the first lap would be

used as the starting point for the second lap and the error would keep increasing if the robot travelled 5 times a 3-by-3 grid.

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

Yes, the absolute error of the odometer would grow linearly with distance. For example, if the odometer consistently has x cm of error with each lap, an error of x cm would be added to the total error after every lap completed. This would give a linear error growth over distance and the robot would be off by x cm off after 1st lap, 2x cm off after 2nd lap, 3x cm off after 3rd lap and so on.

Section 5: Further Improvements

We were able to produce a successful implementation of odometer as required for this lab. However, our robot had its flaws.

Propose a means of reducing the slip of the robot's wheels using software.

The issue of slipping could have been solved by simply reducing the acceleration of the robot since lower acceleration would result in a lower force being applied to the floor which would result in less slipping. This would lead to a smoother motion of the robot and could be achieved by reducing the speed of the robot, which would also reduce the acceleration needed.

Propose a means of correcting the angle reported by the odometer using software when the robot has two light sensors.

To fix the direction of the robot, we could install the sensors in front of each wheel to track the position of the wheels and calculate an angle correction for the robot. To do this, we would start a timer whenever a wheel crosses a black line before the other wheel and record the time needed for the second wheel to cross the same line. Finally, we would calculate the angle by using simple trigonometry rules since we would know the distance (as a function of time and speed of the wheels) and the distance between the wheels.

Propose a means of correcting the angle reported by the odometer using software when the robot has only one light sensor.

We can calculate the time it should have taken the robot to get to the next black line (T) since we already know the speed and the length of a tile. Then, we would record the time our robot actually takes to get to the black line (t). We would calculate the angle correction simply by using trigonometry rules and find the angle correction to be arccos(T/t). Finally, we would add or subtract this angle from 0, 90, 180 or 270 depending on the current direction the robot is heading.