Yuhang Zhang [260787441] Group 7
Mohamed Youssef Bouaouina [260765511]

Lab 2 Report - Odometry
ECSE 211: Design Principles and Methods

Design Evaluation

As it can be seen in figure 1, our workflow consisted essentially of 3 stages: setting up the hardware, designing the software, and conducting tests. In this first section, we will mostly cover the challenges we faced and the decisions we made during the first two stages.

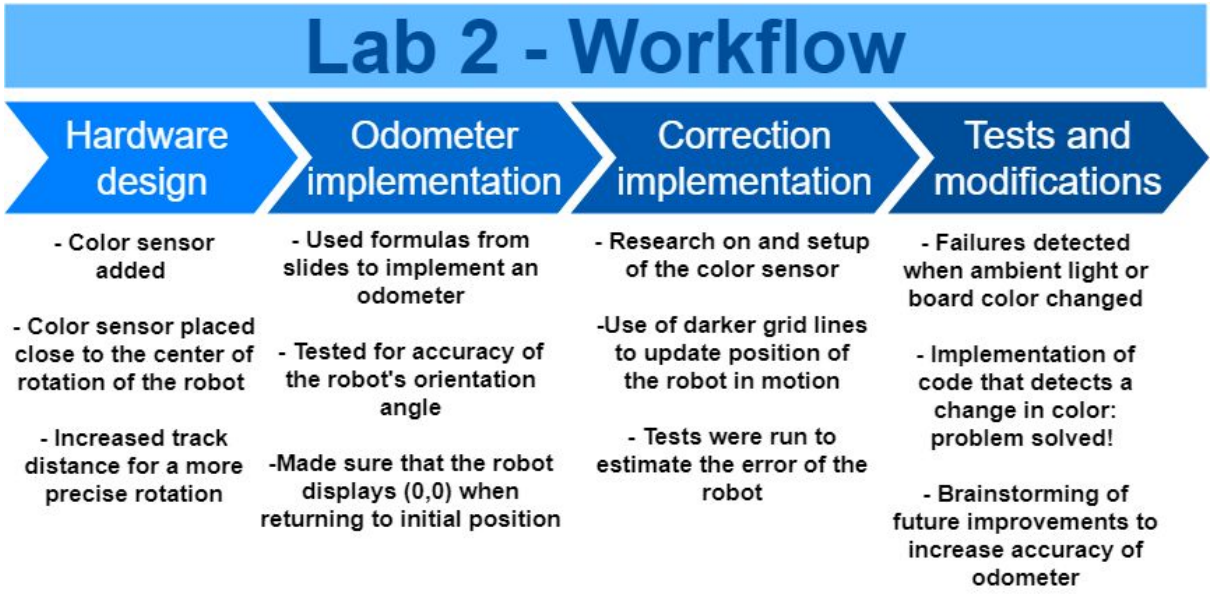

Fig. 1: workflow during lab 2

a) Setting up the hardware :

In contrast to the first lab, where our emphasis was on maneuverability and a light weight, we chose for this lab to assemble a more robust robot that would not slip easily on the surface of the board. Indeed, we used more parts in this lab than in the previous one to ensure more stability for the robot.

As it can be seen in figure 2, we also increased the track, i.e. the distance between the wheels, as such an improvement will result in a more precise rotation. We have also placed the sensor near the center of rotation of the robot (midpoint of the wheels axis). Indeed, as mentioned in the odometry tutorial, the sensor needs to be close to the center of rotation of the robot to reduce the error caused by the assumption that we have perfect knowledge of the heading of the robot

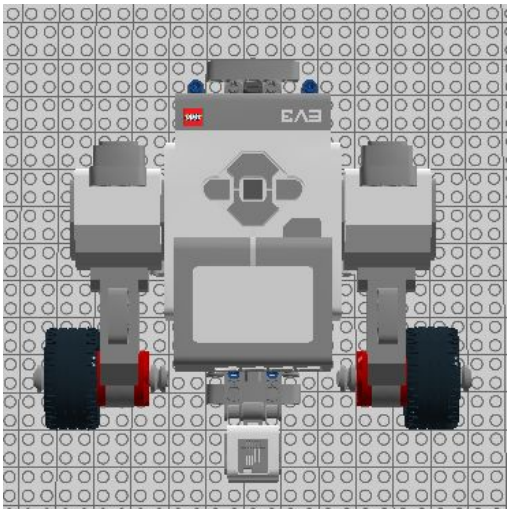

Fig. 2: top view of the robot

b) Software design:

As for the software design of the lab 2, six Java classes were provided, two of which implemented the Runnable interface. Please refer to figure 3 for the class diagram of the program. The main class is Lab2 which calls the other threads and determines the behaviour of the robot depending on the users' input on the brick. The Odometer class consists of the logic behind odometry, hence performing the calculation based on the wheel rotation and radius to determine X and Y coordinated as well as theta, the class OdometerCorrection entails the correction based on the X and Y coordinates. The class OdometerData provides functions for creating and updating the positional information. Moreover, the display class provides printing content that establish a user interface on the robot's LCD screen. Class OdometerException passes the system exception. The SquareDriver is the method that drives the robot in a square trajectory.

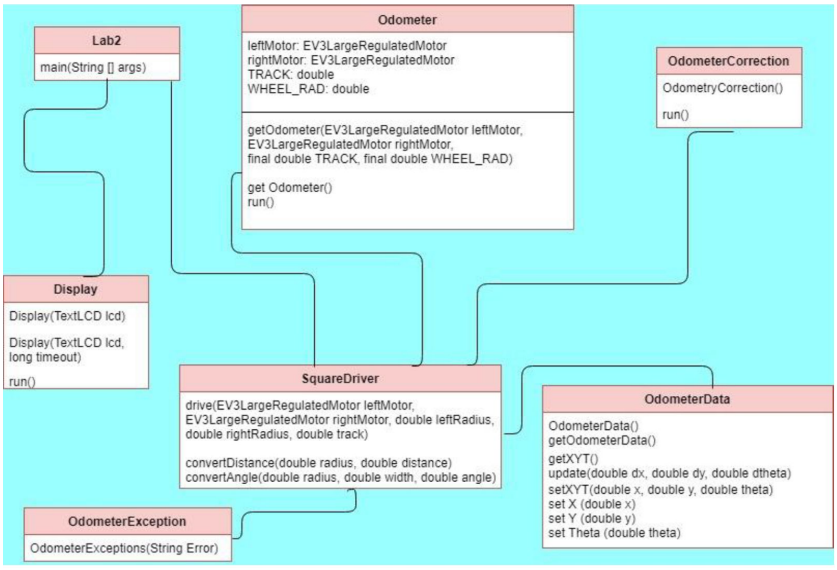

Figure 3: hierarchy of classes functionality

The first main task in this lab was to implement the odometer in the appropriate class. Before starting to code, we researched the concept of odometry and used the provided lecture slides to understand it. Using the provided material, including the odometry into the code was a straightforward task.

Working on the OdometryCorrection class was more challenging. From the lectures and thanks to the TAs' advice, we knew how to use counters to correct the displayed x and y position of the robot when the grid lines of the board are detected. However, the detection of the line itself posed an issue. We first had to go through the leJOS API documentation to choose which mode to use for the light sensor. We opted to use the RGB mode which provided us with access to the measured intensity of the reflected red light. We set a specific value of intensity to detect the black lines on the "yellow-board" in the large lab. We were later notified that the robot should also be able to run on the blue board of the small lab. Therefore, to adapt to this new restriction, we recorded the light intensity range of the tiles and lines in both labs respectively and hardcoded the measured values into our program.

However, during our first demo, the TA suggested to turn off the lights of the room to show us that our robot does not work appropriately as our measurements also depend on ambient light. Such an issue was corrected by implementing a comparison intensity detection: the correction function is triggered when the light intensity reported by the sensor is significantly lower than the initially recorded intensity. With this idea in mind, we created a static variable outside the execution function: startColorId. This variable records the initial color intensity when the program starts to execute i.e. the color intensity of the center of the tile. If the absolute difference between the sampled intensity and startColorId is greater than 25 i.e. a darker line was crossed, the block of code in charge of correction the position is run.

Test Data

a) Odometer test:

Trial	*Starting position*	*X (cm)*	*Y (cm)*	X_F*(cm)*	Y_F*(cm)*	ϵ *(cm)(Euclidean distance error)*
1	(0,0)	0.82	-0.34	1.8	-2.2	2.10
2	(0,0)	0.55	-0.10	1.5	-3.1	3.14
3	(0,0)	0.07	-1.02	1.0	-2.4	2.47
4	(0,0)	0.17	0.97	1.2	1.5	1.15
5	(0,0)	0.78	-1.04	1.9	-1.7	1.30
6	(0,0)	0.98	0.00	2.1	-0.6	1.27
7	(0,0)	1.67	-0.23	1.3	-1.9	1.71
8	(0,0)	0.76	-0.98	1.2	-2.5	1.58
9	(0,0)	0.52	0.12	1.3	-1.4	1.50
10	(0,0)	0.66	1.01	1.2	-3.2	2.26

b) Odometer correction test:

Trial	*X (cm)*	*Y (cm)*	*X_F(cm)*	*Y_F(cm)*	*ϵ (Euclidean distance error)*	*Notes*
1	-13.87	-20.12	-12.3	-16.8	3.67	
2	-15.21	-22.30	-14.1	-19.8	2.73	
3	-14.04	-14.98	-13.1	-12.9	2.67	
4	-12.91	-20.31	-13.2	-18.3	2.03	
5	-16.32	-20.77	-15.6	-17.9	2.95	
6	-16.91	-22.72	-14.0	-20.9	3.43	
7	-13.83	-22.89	-12.2	-19.7	3.58	
8	-13.51	-21.77	-11.2	-16.5	8.57	**Exception: Low Battery Level**
9	-15.12	-23.62	-13.4	-19.2	8.58	**Exception: Low Battery Level**
10	-15.61	-21.32	-13.5	-19.3	2.92	**Battery recharged**

Test Analysis

We use the following formulae to measure the required means and standard deviations:

Mean $\bar{x}$:

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

Where,
x_i is the i^{th} sample point
and n holds the number of sample points

Standard deviation σ:

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

Where,
x_i is the i^{th} sample point,
$\bar{x}$ is the mean,
and n holds the number of sample points.

Odometer tests	*Without Correction*	*With Correction (2 exceptions eliminated)*
X mean (cm)	2.64	-14.83
Y mean (cm)	-1.75	-21.08
X standard deviation (cm)	0.37	5.01
Y standard deviation (cm)	-0.82	4.67
ϵ *mean (cm)*	2.85	2.11
ϵ *standard deviation (cm)*	2.91	1.15

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?

Without correction, in terms of mean of X and Y, the robot's position is approximately 2 cm away from starting position. Their standard deviation are within 1 cm. Moreover, the mean of ϵ without correction is around 2.85 cm and the standard deviation is 2.91 cm.

It can be seen that, with correction, the mean and standard deviation ϵ are much smaller compared to the values we obtained from without the correction. The reason for this variation could be when we perform the tests without correction, the robot does not measure the actual distance travelled, and throughout the whole path, the error as the robot is traveling will accumulate without being corrected.

As for the monitor, the program assumes the robot draws a perfect 3x3 square. However, the actual slip and error in rotation lead to an imperfect path. Therefore, the final readings on the display only represent an ideal result and is different from the actual position. Without correction, the number that is displayed on the screen is significantly dependent on the pre-calculated value in the program, where the robot runs in a perfect square trajectory. However, the robot could actually deviate from where it started but the values displayed on the screen would still be the same because of the assumption that the robot actually comes back to where its initial position was.

In the program where the correction is implemented, it continuously corrects the position on the screen; therefore, the error can be controlled and is relatively smaller in comparison to the results without correction. This also leads to the test results become more stable with correction.

When running the tests with correction, it detects and counts the number of squares the robot has crossed, and processes correction every time it passes the black line. The coordinates appearing on the screen are reset and adjusted according to the current travelling direction and the line that the robot crosses. Therefore, with correction, we receive data that is more accurate to the actual current position.

Despite, the robot being aware that it is not returning to its initial position without correction, it does not accurately know its final position. The mean and standard deviation of X and Y in the odometer correction test are irrelevant as the initial position of the robot is arbitrary and affects its final position.

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

When using the program that utilizes correction, it is expected that the error in the X direction will be smaller. It can be seen that, when the robot crosses the last black line and returns to the initial position, the x coordinate will be reset to 0, (where the final correction happens), and, the final reading will be displaced from the last black line which is more accurate than the Y direction. Indeed, the final Y reading is fixed after the third corner of the square i.e. (the third 90 degrees' rotation) and then the robots travels three tiles without correction the Y position.

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 we observed in the odometer without correction is not tolerable for large distances because the final reading on the display of the robot is not representative of the actual final position of the robot despite having gone back to its' initial position. The readings on the display would not change even if the robot significantly deviates from its path and would reach a different position. If the robot travels 5 times the 3-by-3 grid's distance, then the chances of the robot deviating from its' path is much larger, as the larger the distance the robot travels the greater the error. The error between the value shown on the screen and the actual position will accumulate quickly.

If no correction is implemented, more issues might appear and make the error even less tolerable for larger distances. Indeed, as mentioned in the Odometry tutorial in MyCourses, the wheels' axles may flex and modify their parallelism and the track distance. Additionally, due to the continuous use, the rubber tire may stretch and contract, modifying the used wheel radius. Finally, we have also noticed in previous tests that, when battery levels drop (as they would when the robot runs for a lengthy period of time), errors become more significant.

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

We expect the odometer's error to grow linearly with respect to travel distance.

The error is a result of difference in the linear movement between the two wheels and as scaling goes linearly as there is a fixed turning error for every turn and these accumulate as the robot turns. Therefore, if the robot were to travel several laps, it will have turned several times and will have had significant error due to the accumulation of the error at each turn. Based on the assumption that the error is fixed for every turn, then the error will add up in a linear fashion.

Further improvements

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

Slip of the robot's wheels is partly due to the acceleration of the robot when it is changes its velocity to make a turn. This problem can be addressed by adjusting the speed of rotation to decrease the problematic acceleration.

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

If the robot were to have two light sensors, placing each of the sensors on opposite corners of the front of the robot would be an effective way to detect an error in the angle and correct it. Indeed, every time the robot crosses a line perpendicularly, the sensors should detect the line at the same time. However, if that is not the case, there will be a time difference between two sensors detecting the line, and the correction to fix the orientation can be calculated.

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

Similarly to the previous improvement, we suggest to place the sensor on a corner at the front of the robot. If the robot is travelling in a straight line, we can use our knowledge of the tile size and the velocity of the robot to determine when the next line should be detected. If the line is detected too soon or too late, we can conclude that the robot deviated off of its original orientation and we can use that time difference to determine the factor by which to correct the robot's angle.