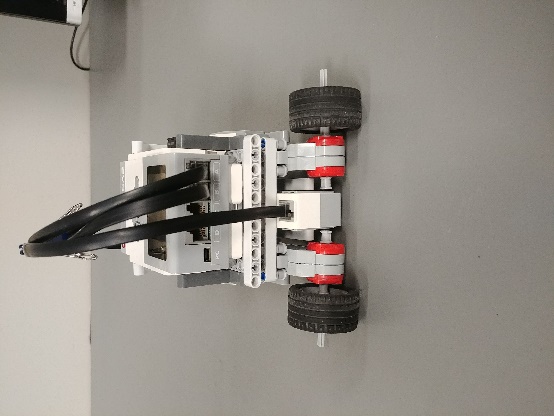
**Section 1-Design evaluation:**

The goal of this lab was 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 using a grid on the floor and a light sensor. First, we had to build a robot capable of driving in a square pattern. We used our design of the first lab as starting point, since it was already built. It was a simple three-wheel chassis with the brick on top. This design had a lot of trouble going straight so improvements had to be made. The motors were changed because the left motor wasn’t going at the same speed as the right one, making the robot deviate from its square trajectory. We also added arms around the wheels to prevent the wheel axle from bending. The third wheel was moved even more to the back of the robot, to remove weight from it. This wheel tended to get stuck and prevent our robot from turning well. This improvement also added some weight to the front wheels, allowing them to slip less. Even after all these improvements, the robot was still not going in a square trajectory (does not go straight, does not turn 90 degrees), but the trajectory was better and probably good enough for our lab. We also had to implement a light sensor for the second part of the lab, which was applying correction to our odometer readings. The light sensor was simply fixed between the two front wheels, 0.5 centimeter above the ground facing downward.

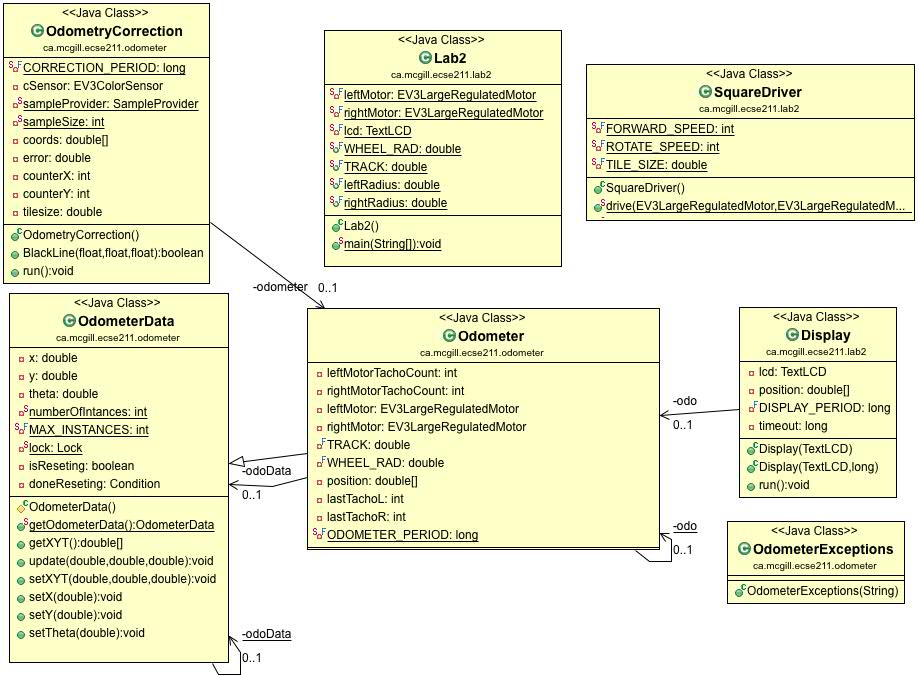


*Figure 1: improvements on hardware design*

Software Design:

Using the sample code provided for this lab, and the example given in the lecture notes, we modified the odometer class to calculate the x, y positions based on the displacement calculated from the TachoCount of both wheels. The angle the robot is facing is calculated from the difference of displacements in left wheel and right wheel.

To implement the OdometryCorrection, we used a color sensor to detect the blacklines on the board. Using the Tools on the brick, we found out that the RGB mode gives the largest contrast from black line to wood floor. By setting the threshold of the RGB values, the color sensor can detect the blackline. Correction is made whenever the sensor detects a blackline. If a blackline is detected, then check the angle that the robot is facing to decide which value should be corrected. If the angle is around 0 or 180 degrees, then the y value should be corrected. We used two counters to keep track of which line the robot is at.



*Figure 2: Class Diagram*

**Section 2- Test data:**

Odometer test:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test** | **Battery voltage (V)** | **Real X (cm)** | **Real Y (cm)** | **Real theta (degree)\*** | **Odometer X (cm)** | **Odometer**  **Y (cm)** | **Odometer theta (degree)** | **Error or Euclidean distance (cm)** | **Error theta (degree)** |
| 1 | 7.8 | -5.8 | 3.85 | 10 | -5.57 | 5.61 | 9.6012 | 1.77 | 0.3988 |
| 2 | 7.8 | -4.8 | 5.05 | 11 | -4.82 | 5.683 | 9.6018 | 0.63 | 1.3982 |
| 3 | 7.8 | -2.8 | 2.65 | 7 | -5.37 | 5.79 | 9.6015 | 4.06 | 2.6015 |
| 4 | 7.9 | -1.7 | 0.75 | 5 | -5.3 | 5.19 | 9.6084 | 5.72 | 4.6084 |
| 5 | 7.7 | -2.8 | 2.75 | 6 | -4.7 | 5.72 | 9.3307 | 3.57 | 3.3307 |
| 6 | 7.8 | 1.6 | 0.25 | 8 | -4.81 | 5.272 | 9.1060 | 8.14 | 1.1060 |
| 7 | 7.8 | 3.2 | 1.25 | 8 | -4.11 | 5.113 | 9.3300 | 8.27 | 1.3300 |
| 8 | 7.8 | 4.4 | -1.75 | 9 | -4.73 | 5.152 | 9.2400 | 11.4 | 0.2400 |
| 9 | 7.8 | -1 | -0.25 | 7 | -3.16 | 3.181 | 6.0100 | 4.05 | 0.9900 |
| 10 | 7.8 | -1 | 1.55 | 5 | -4.77 | 5.181 | 0.4746 | 5.23 | 4.5254 |

\*Theta was measured from the Y-axis, clockwise

The results shown above show that the robot was roughly able to do a square trajectory. The error in position and theta was non-negligible but since we only had to one run in the lab, we considered the result to be acceptable. The software and hardware stayed pretty much the same across the tests. The only thing that changed was the wheel base a bit.

Odometer correction test:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test** | **Battery**  **voltage (V)** | **Real X (cm)** | **Real Y (cm)** | **Real theta (degree)\*** | **Odometer X (cm)** | **Odometer**  **Y (cm)** | **Odometer theta (degree)** | **Error or Euclidean distance (cm)** | **Error theta (degree)** | **Comments** |
| 1 | 7.8 | -11.5 | -11.5 | -9 | 20.43 | -0.16 | 0.920 | 33.883258 | 9.92 | Missed a line |
| 2 | 7.6 | -10.3 | -16.2 | 15 | -6.07 | 2.91 | 2.400 | 19.573533 | 12.6 | None |
| 3 | 7.8 | -8.0 | -15.0 | 220 | -5.70 | -6.51 | 0.920 | 8.7960275 | 19.08 | Added arms around wheels |
| 4 | 7.8 | -11.5 | -12.0 | -5 | -9.74 | -9.26 | -0.140 | 3.2565626 | 4.86 | Improved arms |
| 5 | 7.8 | -14.0 | -12.0 | 15 | -9.88 | -9.67 | 0.270 | 4.73147165 | 14.73 | Different surface |
| 6 | 7.7 | -16.0 | -7.0 | -7 | -14.4 | -8.97 | 0.480 | 2.53789283 | 7.48 | 3rd wheel moved |
| 7 | 7.7 | -7.6 | -16.5 | -14 | -6.23 | -8.56 | 0.013 | 8.057326 | 14.013 | None |
| 8 | 7.7 | -18.3 | -8.8 | -5 | -18.02 | -11.52 | -0.002 | 2.7343738 | 4.998 | Acc. changed to 300 |
| 9 | 7.7 | -16.5 | -4.8 | -16 | -15.42 | -7.54 | 0.699 | 2.94516553 | 16.699 | None |
| 10 | 7.8 | -15.5 | -7.5 | -8 | -14.28 | -8.88 | 0.620 | 1.84195548 | 8.62 | None |

\*theta was measure from the Y axis, clock wise.

These results come from tests that were taken progressively during the lab. As the data shows, the first iterations of our software and hardware were far from being perfect. The light sensor was not working reliably and missed lines. But by setting the sensor to the right mode (RGB worked best for us), and compiling averages values for wood and black lines, we made some improvements. We determined the threshold to detect a line by trial and error, and after that we reached a sufficient accuracy for the lab. From the start, the robot could not go in a straight line. This posed a big problem and so we tried different motors with little success. We added arms around the wheels and changed the position of the 3rd wheel. This improved the trajectory of the robot and reduce the odometer error. The acceleration was also reduced to allow the robot to slip less. After tuning the values for the wheel base and wheel radius, we were able to get the position error under 5cm and theta error under 15 a couple of times in a row. We considered these values good enough for the lab. We were never able to get consistently errors under 2 for the position simply because even after all the improvements (software and hardware) the robot was still not going straight. As some of the results show, the Y value was always off by quite a lot. This is simply because, on the last stretch of the trajectory, the robot was only correcting is X position, leaving the Y untouched. Since the robot was deviating (not going parallel to the X axis) and the Y value not corrected, the reported odometer value was off. If we had more time we could have tried swapping the motors, changed the cables and the ports used or even tried another EV3 brick.

Odometer with correction, ten best runs:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test** | **Battery**  **voltage (V)** | **Real X (cm)** | **Real Y (cm)** | **Real theta (degree)\*** | **Odometer X (cm)** | **Odometer**  **Y (cm)** | **Odometer theta (degree)** | **Error or Euclidean distance (cm)** | **Error theta (degree)** |
| 1 | 7.6 | -4.0 | -13.5 | 5 | -2.15 | -9 | 0.056 | 4.86543934 | 4.944 |
| 2 | 7.8 | -13.5 | -6.8 | -7 | -12.38 | -5.18 | 0.477 | 1.96946693 | 7.477 |
| 3 | 7.6 | -19.2 | -7.5 | 3 | -18.14 | -10.76 | 0.140 | 3.42800233 | 2.86 |
| 4 | 7.8 | -11.5 | -12.0 | -5 | -9.74 | -9.26 | -0.140 | 3.2565626 | 4.86 |
| 5 | 7.8 | -14.0 | -12.0 | 15 | -9.88 | -9.67 | 0.270 | 4.73147165 | 14.73 |
| 6 | 7.7 | -16.0 | -7.0 | -7 | -14.4 | -8.97 | 0.480 | 2.53789283 | 7.48 |
| 7 | 7.6 | -14.5 | -11.0 | -2 | -14.28 | -8.88 | -0.450 | 1.21938509 | 1.55 |
| 8 | 7.7 | -18.3 | -8.8 | -5 | -18.02 | -11.52 | -0.002 | 2.7343738 | 4.998 |
| 9 | 7.7 | -16.5 | -4.8 | -16 | -15.42 | -7.54 | 0.699 | 2.94516553 | 16.699 |
| 10 | 7.8 | -15.5 | -7.5 | -8 | -14.28 | -8.88 | 0.620 | 1.84195548 | 8.62 |

**Section 3-Test analysis:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Odometer without correction | Real X | Real Y | Real Theta | Odometer X | Odometer Y | Odometer Theta | Euclidean Error |
| Mean | -1.07 | 1.61 | 7.6 | -4.734 | 5.194 | 8.19 | 5.284 |
| Standard Deviation | 3.30 | 2.02 | 2.01108 | 0.69 | 0.76 | 2.92 | 3.242736 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Odometer with correction\* | Real X | Real Y | Real Theta | Odometer X | Odometer Y | Odometer Theta | Euclidean Error |
| Mean | -14.3 | -9.09 | -2.7 | -12.869 | -8.966 | 0.215 | 2.952972 |
| Standard Deviation | 4.267187 | 2.851686 | 8.551153 | 4.725939 | 1.722616 | 0.363076 | 1.181596 |

\*We used the 10 best runs for this table.

1. 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 mean and the standard deviation for the Euclidean error distance both decreased when going from no correction to correction. This improvement in accuracy and reliability is due to the correction of the odometer. Since the robot doesn’t do a perfect square, the odometer will always be off by a bit. But by correcting it this can be reduced to a minimum. Using these result, we can say that the odometer with correction is a better design than with no correction.

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

Yes, we do expect a smaller error in both the X direction and Y direction comparing to the design without correction, because we set a fixed coordinate system with the grid lines. Both X and Y readings on the odometer are corrected based on the blacklines on the board. To be more specific, assuming the robot is always moving straight, we set the values of Y when the robot is facing 0 and 180 degrees, and we set the values of X when the robot is facing 90 and 270 degrees.

Comparing the preciseness of x and y readings, we expected the error in X to be smaller. This is because, for the last stretch of the square path, only the X is corrected. The value of Y is left untouched and so any deviation cannot be corrected. The data we gathered confirms this.

**Section 4-Observations and conclusions:**

The error observed for odometer without correction is probably not tolerable if you want the robot to accurately know where it is after a long distance. The square path we test the odometer on was only a total distance of about 3.66 meters and we often saw errors of more than 0.1m. This represent about 2.7% of error. We can expect the error to grow linearly with the distance traveled. Since the error is cause by the robot deviating from the trajectory at a certain rate, we can probably assume that the robot deviates an X amount per distance traveled. This correspond to a linear relationship.

**Section 5-Further improvements:**

To reduce the slipping of the wheels, there are two software solutions. First, we can decrease the overall speed of the robot (forward and turning speed). If the robot is going slower, logically it should slip less. Second, the acceleration of both motors can be reduced. Making the change of speeds smoother can reduce the chances of the robot sliding.

The angle theta reported by the odometer could be corrected by having two light sensors positioned on each side of the robot. Having two different light sensors would allow to detect one line two times. We could then design an algorithm that uses the time elapsed between the two recordings of the same line. If the two sensors measure the line at the same time, it means the robot is parallel to the lines and is going straight. If the left sensor sees the line first and the second sensor sees it one second later, then the robot is not going straight. It means the robot is heading towards the right a bit. Using the time difference and the velocity of the robot, theta could be corrected.

If we were to use only one sensor, we could use some rotating movement to determine the real theta of the robot. One possible way of doing it could be that every 3 lines the robot crosses for example, we make the robot stop and turn 360 degrees. Knowing the rotation speed, the radius of rotation and the time elapsed between the sensor seeing the line twice, we could probably determine the orientation of the robot accurately enough. This calculated value could then be set as the new theta and the robot could continue moving.