ECSE 211 Lab 2 Report Group 33

Authors: Aleksas Murauskas, 260718389 and Talaal Mazhar Shafi, 260865281

1. Design Evaluation

The following section will discuss the hardware and software design of the navigation robot. After our success with the previous lab's wide track we kept with same core design. The larger track required more rotations of the wheels to accomplish the same turn in comparison to a smaller track, this made over or under turns more visible to the human eye. The measured track length found was 17.6cm but after testing Lab 2's Square Driver code to check track measurements, the measured value was found to be to small as the the robot would not complete the desired 90 degree turn. Trial and error found the true track value to be about 18.15cm. This robot model was able to complete an obstacle-free track with ease. A Ultrasonic was attached to the robot's forward crossbar to detect any obstructions in front of the vehicle[Visible best in Figure 1]. We placed the sensor below the crossbar to lower the center of mass a small amount, as it was a little too high in the previous lab resulting in the motors and wheels being pushed out. This was mostly remedied with a crossbar in the previous lab [Visible best in figure 2], but a small amount of lean still remained. After the sensor was attached low, the lean on the wheels became negligible. When the preliminary tests occurred with obstacle avoidance, the robot was able to avoid objects quite well at first. However, the wide track became an issue when passing by a wall at a angle as the sensor did not detect a corner of the wall as it approached causing a collisions. Afterwards, we made a change to our obstacle avoidance thread to make sure the robot took into account the large track when avoiding objects. Our finalized hardware design is visible in figures 1, 2, and 3

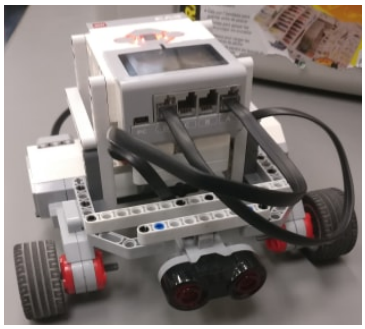


Figure 1

Figure 2

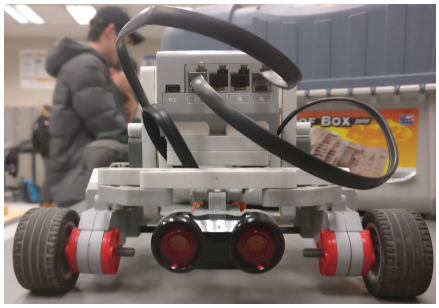
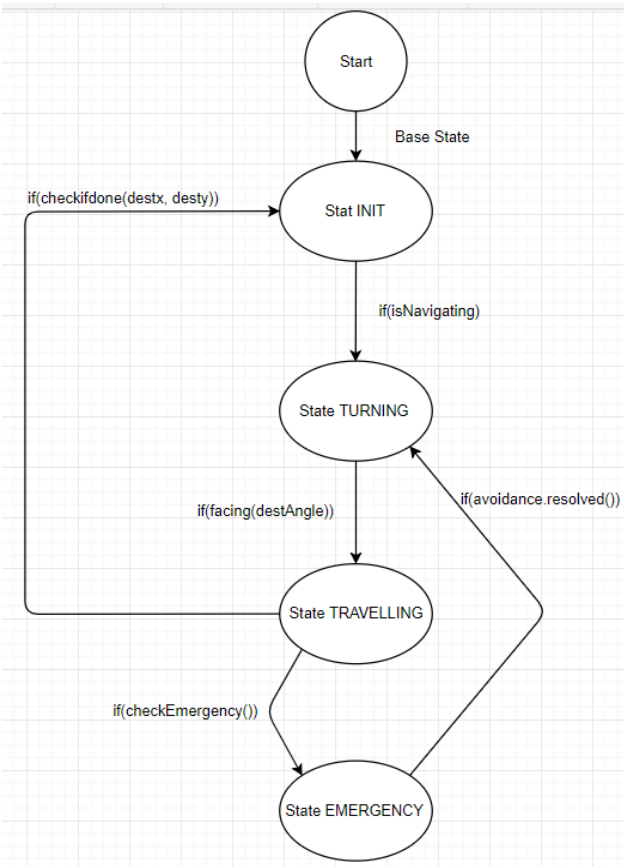


Figure 3

Our software was based upon the tutorial code with the Main and Obstacle Avoidance classes facing the largest amount of augmentation. The key piece to the Navigator class is its State Machine[Figure 4.]. There are four key states: INIT, TURNING, TRAVELLING, and EMERGENCY. INIT represents the robot's base state, in this state the robot does not not move. If the Travelto() method is called where the isNavigating boolean is set to true, the machine will enter it's next state TURNING. Within TURNING, The robot turns to the desired destination angle following the minimal angle until the method facing(destAngle) returns true. Once that happens the machine enters the state TRAVELLING. Within this state the robot is actively rolling itself to the destination. If the Ultrasonic sensor detects an obstacle checkEmergency() returns true, resulting in a state change into EMERGENCY. Within EMERGENCY the Obstacle Avoidance Thread starts and avoids the obstacle, once the obstacle has been avoided the resolved() method returns true and the machine returns to TURNING. Once the robot has reached its destination, the robot leaves TRAVELLING and returns to INIT state, awaiting another waypoint to be entered.

Figure 4.



The key the to this was having several threads run simultaneously. The Main class controlled all but one of the threads. First, the odometer thread is created to record the robots x and y position as well as it's directional angle. Second, the Ultrasonic Poller Thread is created to check for an obstacle in the robot's path. Both of the above threads are passed when creating the Navigation thread, which controls the state machine previously defined and the robot's movement to waypoints. The Final Thread created in Main is the LCD Info thread, which displays the odometer data so that a user can see the robot's x,y, and theta values on the screen. The code includes one other thread not introduced in main. When the robot is TRAVELLING and the Ultrasonic thread reports and emergency, and Obstacle Avoidance thread is created. This thread maneuvers the robot around a obstacle, then once it has been avoided, terminates and sets Navigation's state machine to TURNING returning control of the motors.

Our Obstacle Avoidance Thread operates by first setting a safe boolean to false when it is initialized. The navigator is passed to the thread so Obstacle Avoidance can take control of the wheels and issue movement commands. Within its run method, the robot first comes to a stops as it computes whether to take a left turn or right turn around the objects. The robot's current x

position, y position, and theta facing are obtained from the navigator's odometer. Using its current cardinal directional facing [North, East, South, West] and it's position on the board, a left or right turn is chosen through a series of if statements. The logic used is visible in Figure 5. For example, If the robot is facing north [along the positive y] and in the right most part of the board, a right turn would result in the robot falling off the board entirely. To avoid this a second if statement is used using its x coordinate or y coordinate for North/South or East/West respectively. This Northern facing robot, if its x coordinate is too large[coded to be 45 cm along the axis] to make a comfortable right turn round the obstacle, a left turn is made instead. In the case of a right turn the method avoidRight() is called and a mirrored method avoidLeft() is called in the event of a needed left turn. Both operate by making a half large square around the obstacle and travels quite a bit past the object to ensure the robot does not get its wide frame caught on the corner of the obstacle. After, avoidLeft() or avoidRight() are called the log is updated to notify the avoidance success and returns the control of the navigation class in the state of TURNING.

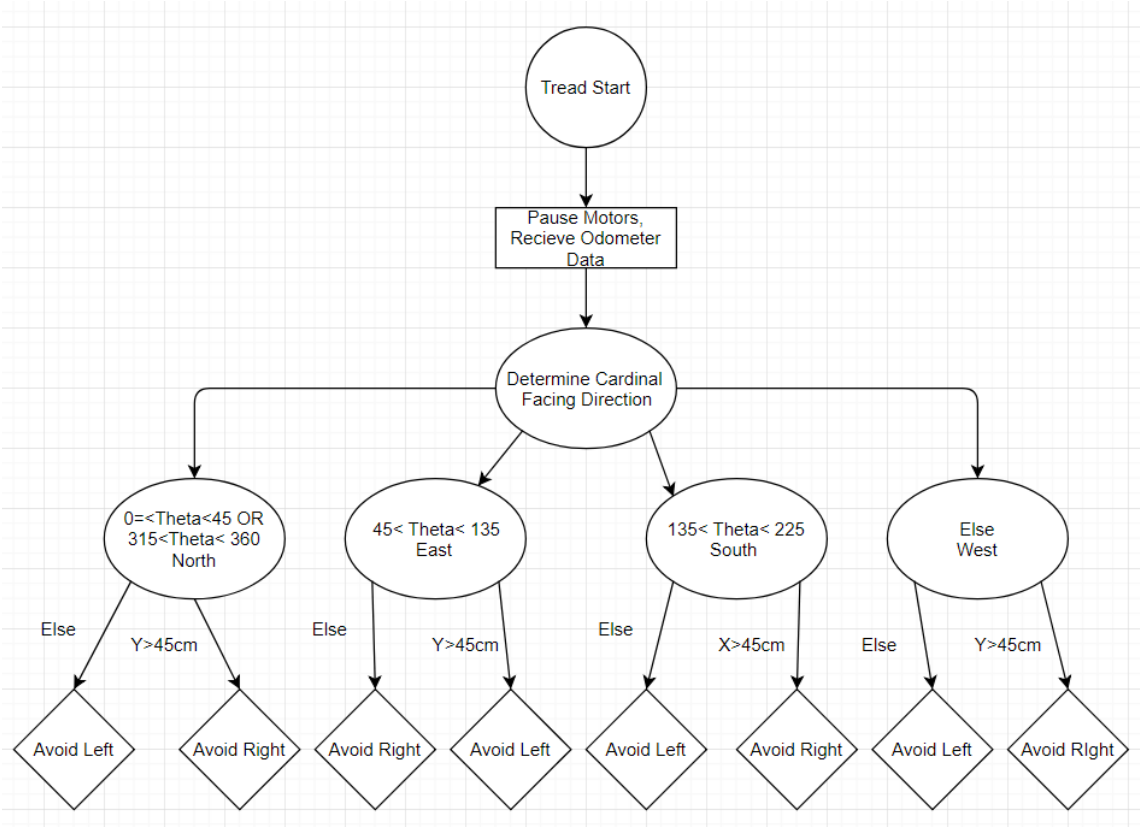


Figure 5.

2. Test data

The following table was gathered from 10 trials of the navigation robot's traversal of the points (2,1), (1,1), (1,2), (2,0).

Trials	Hand End X	Hand End Y	Robot End X	Robot End Y	Error
1	55.5	-3.5	60.7	0.5	6.56049
2	58	-4.1	59.8	-0.4	4.11461
3	57.1	-3	59.7	0.6	4.44072
4	59.1	-2.1	60.5	0.2	2.69258
5	54.5	-5.5	60.2	-0.6	7.51665
6	59.4	4.2	60	0	4.24264
7	57.5	-2.3	59.6	0.6	3.5805
8	58.2	-2.1	60.3	0.5	3.34215
9	57.2	-4.8	60.1	-0.2	5.43783
10	57.3	-3.5	61	0.4	5.37587

3. Test Analysis

Equations Used:

$$Euclidean\ Error = \epsilon = \sqrt{(Robot\ End\ X - Hand\ End\ X)^2 + (Robot\ End\ Y - Hand\ End\ Y)^2}$$

Sample Trial 1:

$$\epsilon_1 = \sqrt{(60.7 - 55.5)^2 + (0.5 - -3.5)^2} = \sqrt{(5.2)^2 + (4.0)^2} = \sqrt{43.04} = 6.56049$$

$$Error\ Mean = \bar{\epsilon} = \sum_{i=1}^{10} \frac{\epsilon_i}{10} = 4.7304$$

$$Standard\ Deviation\ of\ Error = \sqrt{\frac{\sum_{i=1}^{10} |\epsilon_i - \bar{\epsilon}|^2}{10}} = 1.49565$$

Error Calculations	
Mean	4.7304
Standard Deviation	1.49565

4. Observations and Conclusions

Although the navigator produces a small amount error due to using integers rather than floats, and because our code has been set up to allow the robot to be one degree off the destination angle and one cm of the destination from where it is programmed to go, these errors are small compared to those of the odometer. As mentioned in our previous lab report, the odometer produces a significant amount of error for large distances due to several factors. These include the amount that the robot's wheels slip against the floor, a lack of motor synchronization, and sudden starts and stops that all build together to throw the robot off its trajectory. Moreover, there are errors resulting from the assumptions that the wheels are equidistant from the robot's centre, are parallel, and have a constant radius because the wheel axles can flex, changing their distance to the centre, and the rubber on tires can stretch or contract, changing the radius too. Together, these errors accumulate to what has been shown in our trials.

The navigation controller is quite accurate in moving the robot to its destination. As stated previously, our code allows for a one degree and one cm distance error from the point to which the robot is intended to go, because having it get to the exact point on the board would lead to a large amount of oscillation to orient itself permanently, further contributing to the odometer error. In addition, the navigation controller works in integers rather than floats, also decreasing its precision. As our table shows, the robot is usually within 4.7 centimeters off its true position.

From our observations, we noted that our robot had about 4 oscillations on average for each point travelled, and the majority of these oscillations occurred when the robot was very close to its intended destination point. Because we decreased the allowed error in how closely the robot needs to reach its destination point, there were more oscillations in our tests than in the base code.

Increasing the speed of our robot would affect the navigator's accuracy by making it less accurate. Every thread has it own update cycle, and the odometer is updating several times each to determine its location. The more distance covered in between updates results in more error in the odometer. Increasing the speed would cause then path the robot takes to be covered more quickly, which would give the odometer less updates in total and more distance in between updates, therefore causing more errors within the odometer's readings. The increased inertia due to the higher speed could result in more wheel slippage which the odometer does not account for, increasing error.

5. Further Improvements

In terms of software, we could improve our obstacle avoidance thread to avoid the wall with less total displacement. As error increases with distance travelled linearly, an avoidance method with minimal distance travelled will reduce total error. improve the robot's awareness of where it thinks it is, thus improving the overall error. For a hardware improvement, we can add a light sensor that allows us to use the aforementioned odometry correction. This would correct the odometer understanding of the robot's position, as proved in our last lab,where mean error dropped from 4.3cm to 2.4 this would improve our error mean significantly.