Grid Navigation Report

Team Members: Trevor Davenport, Phuoc Nguyen, Khoa Tran, Corey Short

Brief Description: The Grid Navigation Project include five different milestones. As of this writing, our report is currently for Milestone 4, where we implemented the Shortest Path distance

Project Repository Location: https://github.com/IEOR140-T5/Project2NXT/

Code: in our Project3 directory is a `src` directory which contains all the code. We break the code down to different packages. `experimental` indicates that the code there is used for data-collecting purposes. `robot` has all the robot functionalities code needed for all milestones. Finally, we have a separate package for every different milestone to keep our test code clean

Javadoc: can be found in the `doc` directory.

Software Design:

- **Sources of Programming Ideas**: we use the while loop to keep calling the method trackLine() to determine the center of the line and to steer at a desired turn rate in order to keep on track. Beside, we count the number of black marker that the robot has passed, the travel a small distance in order for the robot for not detect the black market anymore. The robot will stop or turn when it reaches a specific numbers of black market detections.
- **Class Responsibilities**: The class Tracker take responsiblity for the calibration setup and basic control of the robot. For Milestone 2, the only method we implement is go() which is detailed above.

Most Interesting/Challenging Parts: We found that the most interesting portion of the project was implementing the light sensors. The light sensors, in our opinion, were the most intriguing aspect of the project. After figuring out how to correctly calibrate their values, the coding and algorithms came next. We found that developing the algorithm and implementing code was the most difficult portion of this project.

Milestone 1 Report

In this milestone, we tried to calculate the center of the line, the turn rate to make the robot not losing its track. For the main part, the robot will trace the line oval track 4 times, turn back and do more 4 rounds. For the extra credit part, it will trace the oval track but in a way of 8.

Milestone 2 Report

In Milestone 2, our group used a method named go(). This method begins by calibrating the tracker, and then runs our algorithm designed by Peter. Next, the method is seperated into four seperate for loops (2 rounds of left and right turns and next uses coordinate mappings). Based on our light sensor data,we deduced that if the left or right values ever read less than -10, we know that the rover has come across the black marking. We than pilot the rover, put it to sleep, and as our iterations for each for loop are odd, the robot will turn, sleep, and then continue to trackLine(). This same basic algorithm is implemented throughout our go() method.

Subtasks:

- **Data and Algorithms**: Our algorithm for computing the turnrate involved using a method provided (CLDistance) to determine the distance between each sensor and how far off the desired path our rover was. Next, after calculating this distance, we implemented the `steer` built-in method from lejos with a control variable set to 1.02. The next algorithm we developed for Milestone 2 goes as follows:

1. Loop Necessary Iterations
2. Determine if the Sensors have come in contact with a black marker
3. If true, travel X distance and Sleep Y ms.
4. Next, Determine if the current iteration is odd.
5. If true, Sleep Y ms, TurnRobot X direction, and then Sleep Y again.
6. Else, continue to trackLine().

- **Task Sequence and Repetition**: After running multiple tests with the oval track, we were able to determine correct values while observing how our rover reacted to changes in code.

Milestone 3 Report

We found Milestone 3 to be the most difficult part of the project. For some reason, our group kept getting

some OutOfBoundsExceptions. After some meticulous skimming of our code we found the error. The error involved our _heading variable becoming less than 0, i.e. -1.

This caused our Array to index into negative numbers causing the above exception. We chose to represent points by using two seperate two element arrays (_destination and _position). Next, our _heading variable is used to determine the direction the robot is facing. The entire workload of this milestone is completed in the method toDestination().

After overriding the equals method to determine if two points have the exact same coordinates, we implemented our algorithm to navigate to a specificed set of coordinates. Part of our algorithm is done in the method newHeading(). newHeading returns a direction between [0 .. 3]. We chose to represent our coordinates as numbers between [0..3] inclusive. Once newHeading returns the direction, we implement the correct turnRate.

Milestone 4 Report (Shotest Path)

Distance to block: We measured and chose 25 as the threshold for the distance to the closest block. If the value is below this threshold, we mark the node where the block is at as blocked, and recalculate the distance.

Task Analysis: Our ShortestPath module first instantiates a grid of size 6x8 as in the last milestone. We then calibrate the tracker, and ask the users to enter the desired destination.

From the starting point, we calculate the shortest path to the destination. Along the way, if any node is detected by the ultrasonic sensor, we mark it as blocked, and recalculate the distance to the destination without using that blocked node.

If the destination is a blocked node, we set the destination to the current location, so that the robot won't move at all. We also print out to the screen the locations of the blocked nodes, so that users can have a better idea of where all the blocked nodes are.

While moving, the robot always looks for the best destination to turn into according to its pre-calculated shortest path distance, which we wrote in the homework. We also update the heading to the current turn direction, and we have a function to handle the cases where the heading falls out of range, i.e. [-2, 2].

Difficulty: We were having some problems with our tracker not staying on a straight line. Professor Glassey suggested a fix by setting the acceleration to 200, and our robot performs much better since

then.