Project 2 Milestone 3
9/13/2012
Nate Bailey - Co-Program Designer, Program Coder, Co-Report Writer (15 hrs spent on project)
Raymond Ma - Co-Program Designer, Mechanical Designer, Co-Report Writer (15 hrs spent on project)

Java code is on GitHub: https://github.com/ieor140-team4/Project2-good

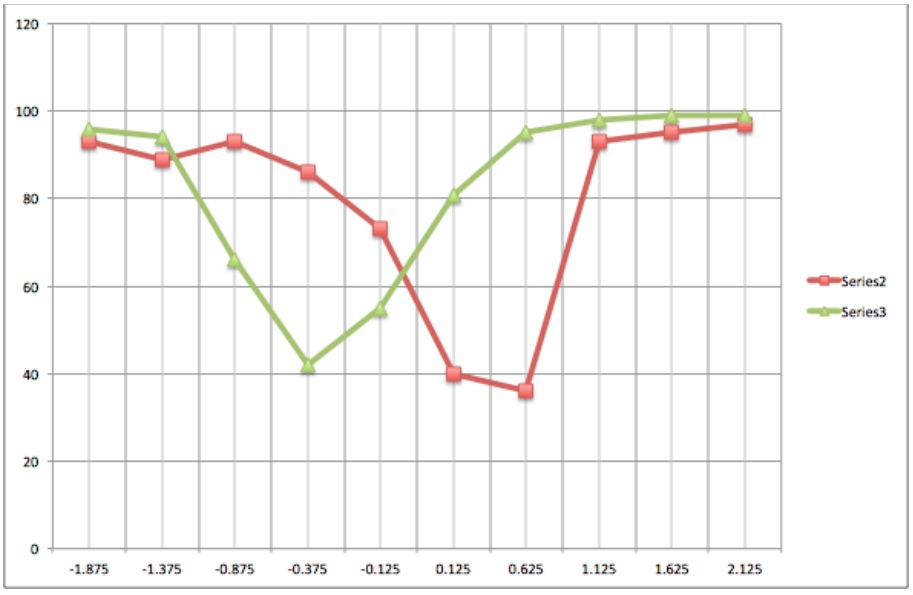
Performance Specifications
For Milestone 1,

For Milestone 2, we received full credit for the part and there was no extra credit. Our program successfully obtained measurements of the light readings from our sensors, used those measurements to tell us something about where our robot was compared to the center line - in other words, measure our error, applied control theory and feed the error into a scheme that led our robot back to the center line, found the black square that marks the halfway points of the oval and stopped tracking, and implemented this in a loop that allowed our robot to perform four loops, turn around, four more loops, etc.

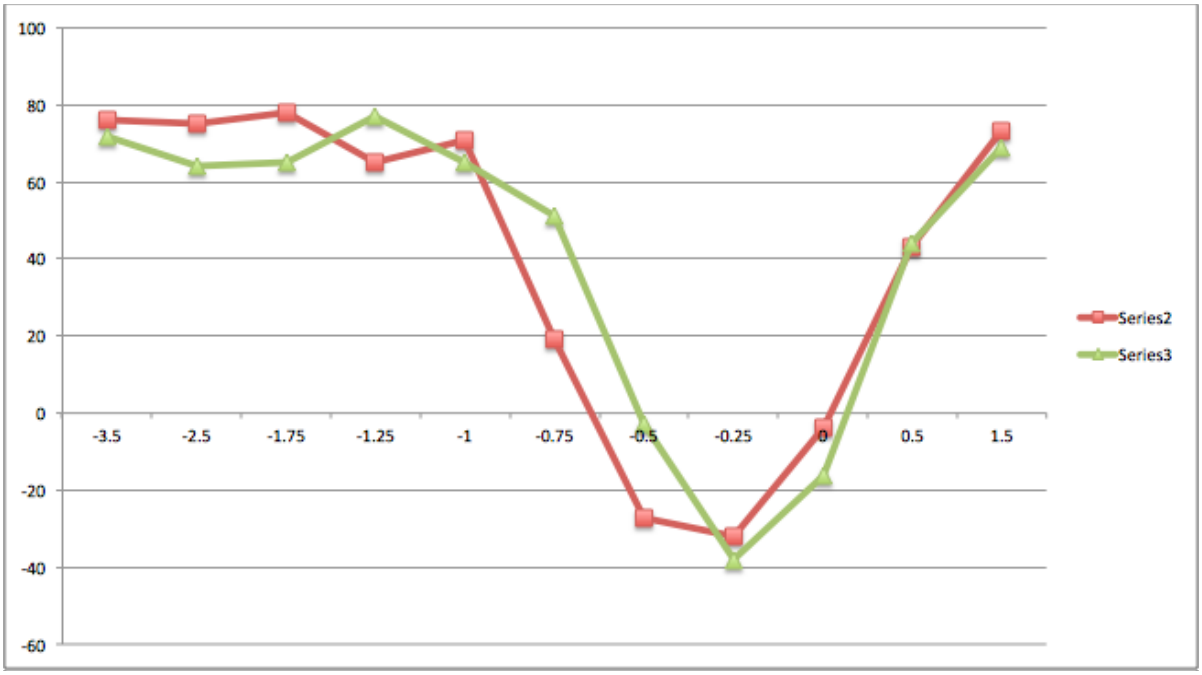
For Milestone 3, we received full credit for the part and there was no extra credit. Our robot successfully navigated a grid when given grid coordinates and remembered its positioning and direction on the grid during execution.

Hardware Design
The hardware for this project was pretty much entirely dictated by specifications given to us. The robot used two wheels mounted on a main axle, with a third wheel mounted for stability in the back. It also had two light sensors mounted on the front, facing down towards the ground. These sensors were used to keep track of the line that we wanted to follow. Keeping track of the distance between these sensors and the wheel axle was important as there was a delay between detecting a stopping point and having the robot position itself over that stopping point.

Experimental Work

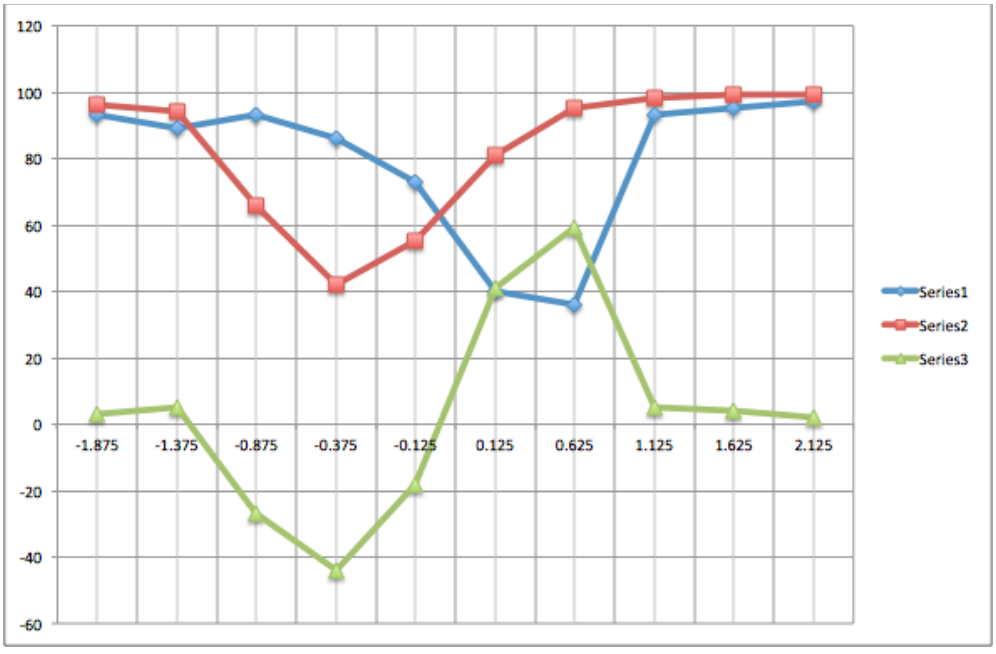


In order to see how the light sensors behaved at different distances from the center line, we used some of the code supplied to us in order to perform tests of the light sensors. The above graph shows light sensor readings as a function of the distance that the center of the light sensors was from the center of the line. The x-axis shows this distance while the y-axis shows readings from the leftmost light sensor (red) and from the rightmost light sensor (green).



In addition to that data, we also tried to find out how our light sensors were able to find the black square which marked the halfway points around the track. In the above graph, the x-axis represents the distance that the light sensors were from the center of the black square, while the y-axis represents light sensor readings from the left and right light sensors (same colors as above).

The first set of data actually did not turn out to be that useful for us, as we ended up doing more guesswork to find a good gain constant as opposed to using some kind of a function that translated the difference between the two light readings into an estimate of the distance from the center line. However, the second set of data proved very useful as we were able to use it to obtain accurate thresholds that told us when our robot was on the black square and when it had gone past it.



This is the graph that shows how our robot steers towards the center line. The value for green is calculated by taking the left eye measurement and subtracting it from the right eye. The closer the points of the left and right eye measurements are, the closer to zero the green line is. Our robot multiplies the green line with a gain constant of 1 and then passes this information into the differential pilot steer method.

Problem Analysis & Software Design

For this code, we had three main classes we needed to use: a tracker class, a grid navigator class, and a button inputter class. Each class had different responsibilities at different levels of abstraction.
Tracker tasks:

- -Track a line
  - -Take in data from the light sensors
  - -Use the data from the light sensors to steer the robot.
  - -Steer the robot so that it stays on the line.
  - -After detecting a black square, move the robot onto the black square
- -Rotate and track heading
  - -Keep an internal track of what the heading of the robot is
  - -To turn, compare the angle that we want to turn to and the current heading, then rotate through that angle

In trackLine(), the tracker class observes the difference between the right eye and the left eye, and multiplies this difference by a gain constant of 1, then feeds it into DifferentialPilot.steer() in order to guide the robot. As the robot strays from the line, the difference should increase which produces a sharper turn for the robot to come back to the line.
The tracker also has some other auxiliary functions. rotateTo() rotates the robot to a new heading and updates the internally stored heading to make sure the robot remains up to date. It also optionally will move the robot so that it is over the black square to provide a seamless rotation. crossBlack() lets the robot cross over a black square without interrupting the flow of movement, which is nice but maybe not technically necessary.

Grid Navigator tasks:

- -Move from position to position
  - -Keep an internal track of what position the robot is at
  - -Determine how to get from the current position to the new position
  - -Move from here to there and update the internal position (including rotations)

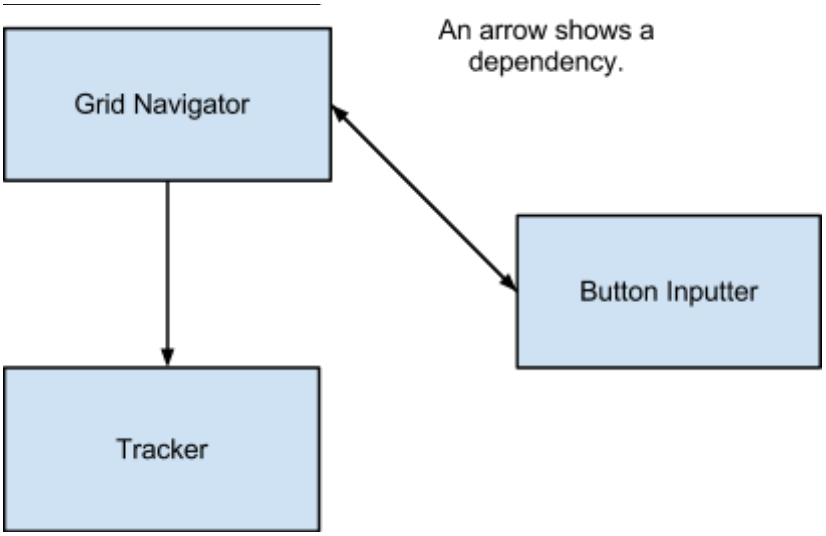
The Grid Navigator controls a tracker and also a point that represents the position the robot is at right now. When given a new point to move to, it first figures out what direction in x it needs to move (positive, negative, or not at all), then rotates to that direction and moves to the new x coordinate. It then does the same for y. It is given the new point by the button inputter class in the Milestone3.main() method.

Button Inputter tasks:

- -Display a menu on the screen that allows the user to:
  - -Input x
  - -Input y
  - -Save
- -Respond to user input when the user changes something

The button inputter displays a menu and allows user input to specify an x coordinate and a y coordinate to move to. It then gives this point to the grid navigator to navigate the robot towards. It also gets information from the grid navigator about where the robot is at the moment so that it can display the current position to the user.

Thus, the flow diagram looks like:



Interesting/Challenging/Difficult

In this project, there were many interesting, challenging, and difficult things we encountered when writing this Project. The most interesting part of this project was designing a robot that could successfully trace the line by differentiating between the blue, white, and black colors. The most challenging part of the project was coding the GridNavigator class, because it needed to take the point given by the ButtonInputter and convert it into useful directional information for the robot. The most difficult part of this project was designing a way for the robot to detect the black dot and continue to move off of it until the wheels were right in line with the dot.