Control of Mobile Robots
CDA 4621
Spring 18
Lab 3
Conner Wulf, David Ton
Localization and Mapping

Submitted Files

Mapping48.ino - Main function that contains logic for robot to navigate every cell in the give maze, map them, then find the shortest path.

MyServos.cpp - definition of MyServos functions that set Speeds to left and right servos (in IPS, RPS and in terms of v and w), and that calibrate servos speeds.

MyServos.h - declaration of servos functions, including the left and right motors

MyEncoders.cpp - definition of MyEncoders functions, determining wheel revolution count, and

wheel speed.

MyEncoders.h - contains declaration of functions for encoder setup

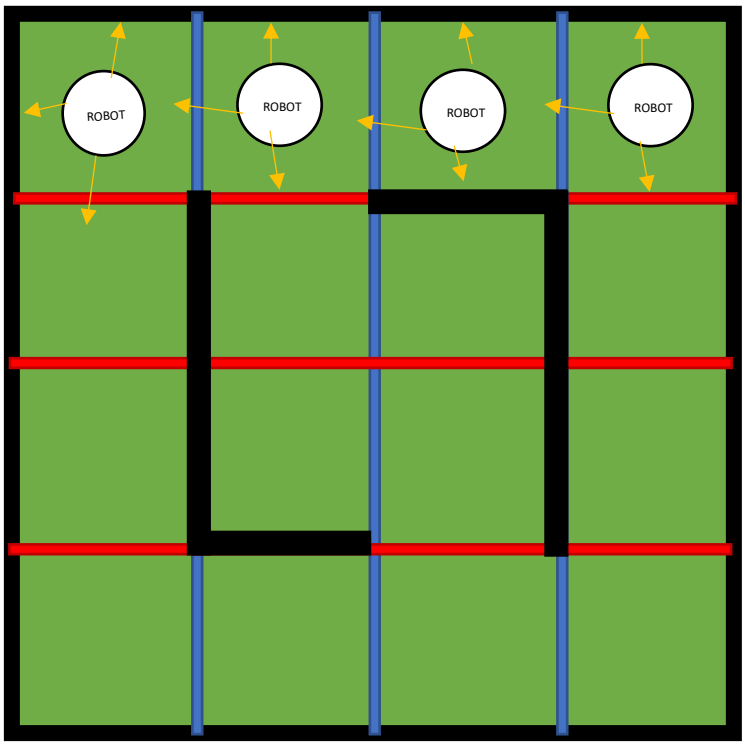
MySharpSensor.cpp – Definition of MySharpSensor functions to convert readings from sensors into inches and determine the combined distance.

MySharpSensor.h – contains declarations of functions for Long and Short sensor setup

ColorSensor.h – Contains declarations of functions for ColorSensor.h

ColorSensor.cpp - Definitions of ColorSensor functions to convert readings into rgb values.

Maze Mapping



For the maze mapping algorithm, we implemented the search algorithm DFS with backtracking. We used depth first search because this search algorithm guarantees that every cell in the maze will be visited. After the color and speeds are calibrated, the user picks the starting orientation, starting location and then hits select. Once the select button is started the robot starts travelling through the maze. As it reaches each new sell it adds it to a stack. After it reaches a cell it marks it as visited and continues. The robot chooses which direction to go sequentially starting at look forward, then left, and then right. If the robot gets to a point where it cannot visit any more unvisited cells by moving in one direction it starts to backtrack using the stack. The robot will continue to backtrack until it reaches a point to where it is able to go to an unvisited cell. Once all the cells have been visited the robot does the required output logic specified in the lab requirements. During the entire mapping program, the correct color is flashed on the LCD corresponding to the color.

Maze Mapping Problems

1) One problem we had when implementing the Maze Mapping program was getting the program to output the walls correctly. After the robot traversed the maze correctly it would give us false positives, or 'Ghost Walls' meaning that it read a wall was there when there was no wall

Solution: In order to fix this bug, we implemented two different methods. The first method was implemented during the part of the program where the robot was moving to a different cell. As the robot was moving it a different cell and crossed a line it set the value that corresponded to whether or not a wall was there to open, since there couldn't be a wall there because the robot crossed that section.

The second method we implemented was a helper function that was executed after the mapping was complete. The helper function compared wall values that are shared between a cell. If a cell shared a wall with another cell and the first cell read that there was not a wall in that shared position, but the second cell did, both cells were set to open. This helped take care a lot of the problems with ghost walls.

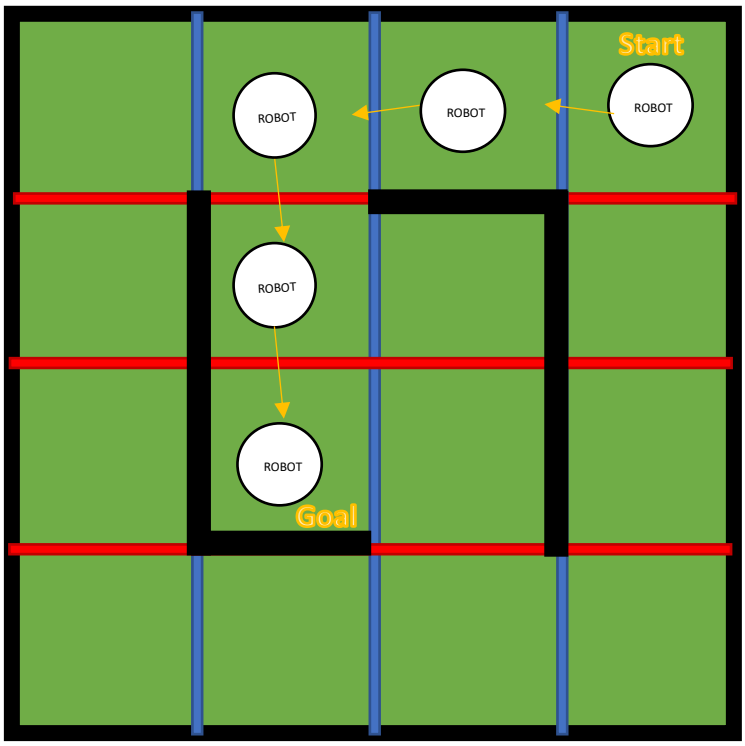
2) Another problem we found was trying to keep the robot straight while it was navigating the cells. The robot would get too close to a wall and either stop or contribute to the ghost wall problem.

Solution: To counteract this problem we used some of the techniques from lab 2 and incorporated wall following. We found that checking to make sure the robot was roughly in the middle of the cell helped keep our results accurate.

3) A third problem we had was in our backtracking logic during the DFS search algorithm. After the robot hit a spot where it needed to backtrack, it would spin around and start the backtracking, however when it reached a cell it already visited, it kept pushing that cell back onto the navigation stack, messing up our mapping.

Solution: To solve this problem we created a Boolean value that determined if the robot was in backtracking mode. If the robot was backtracking, it traversed the maze backwards using the stack but did not push anything to the stack. The only time the program could push anything to the stack was when backtracking was equal to False.

Path Planning



For the path planning program, we implemented the BFS search algorithm. The program uses the map created from the first part of the lab and starts at the location specified by the user. Once the search algorithm starts it checks every direction in the cell to see if a neighbor exists. The program determines if a neighbor exists if there is no wall in that direction. If a neighbor does exist, the program sets the cell you are in as the predecessor to the neighbor it found and then adds the neighbor to a queue. The search ends once the goal is found. After that, the program uses the array of predecessor to traverse the shortest path and then moves cell to cell, stopping after it reaches the goal.

Path Planning Problems:

1) A problem we had with the path planning part of the lab was with having enough storage to run the program. We found out that the Arduino Uno becomes unstable when the percentage of global variables used is over 75%. Our program memory also hit over 95% by the time we finished.

Solution: We spent a large amount of time refactoring our code. Found that the LCD print function took up a lot of memory, so we tried to keep any printing short and sweet. We also changed all of our int to bytes and found that helped a lot. The last thing we did was delete all the 'serial.begin' lines of code and that did the trick.

2) A second problem we had with the path planning part of the project was when the robot was traversing the shortest path. The robot would get about halfway through the path and then just stop.
   Solution: In debugging this problem we started printing out the states to the LCD like we printed in mapping method during path planning. This ended up fixing the problem, we don't really know why, but it did…

3) A third problem we had was trying to get multiple different path planning to happen one after another after the mapping. The most I could get to happen was 2 path plans and then the robot would freeze.
   Solution: We did not find a solution to this problem and settled with the robot being able to do one path plan per navigation.

Video: https://youtu.be/FIXB98nw8Mk

Conclusion:

We had a lot of fun with this lab. It really helped us understand how different search algorithms we learned in different classes work in a real-life situation. We learned how hard it is to get all of the components working together, correctly. Most of our problems came from sensor readings not updating fast enough and those sensors being inconsistent. This lab challenged us the most out of all the labs we have had. This lab was harder because we ended up not being able to use our lab 3 code and had to rewrite everything. It feels really satisfying that everything is working a hundred percent correctly.