EECS C149/249
Line-Maze Solving Robot

Project Group:
Kyle Chiang
Nolan Wagener

Project Mentor:
Garvit Juniwal

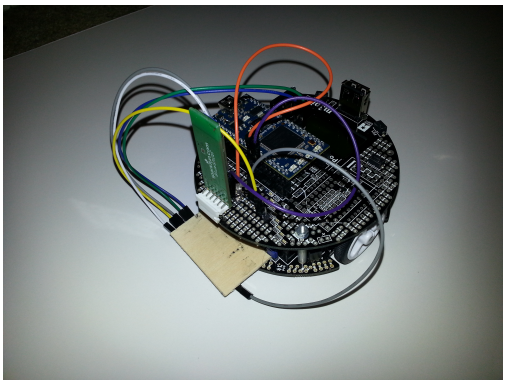
December 20, 2013

Introduction & Problem Definition

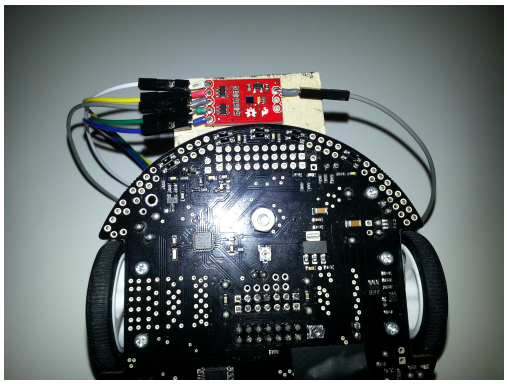
Our project consisted of using a Pololu m3pi robot and Bluetooth communication with a computer to construct the map of a line maze, be able to go to desired destinations optimally, and be able to change paths to these destinations depending on whether "doors" in the maze are open or closed. This system is is meant to simulate an office environment where a robot moves among different rooms to pick up and later deliver things like mail. At first, the robot must learn its environment by exploring the office and generating a map of it. However, certain conditions in the environment may change, such as doors opening and closing, which the robot must be able to replan how it reaches destinations optimally.

Outline of Approach

Tools Used



(a) m3pi robot with extra sensors and hardware



(b) Color sensor and reflectance sensors

We decided to use the Pololu m3pi robot because it is a standard robot among hobbyists that can be used to solve line mazes. The robot came built-in with reflectance sensors that can detect varying levels of brightness. Since our destinations and doors were going to be represented with different colors of tape and had to be identified properly, we added an ADJD color sensor to the bottom of the robot. Both of these sensors are shown in Figure 1b. For Bluetooth communication, we added a BlueSMIRF Bluetooth modem. Finally, we added an mbed microcontroller because it had more processing power and memory than the m3pi's ATmega328 microcontroller. The robot is shown on one of our mazes in Figure 2.

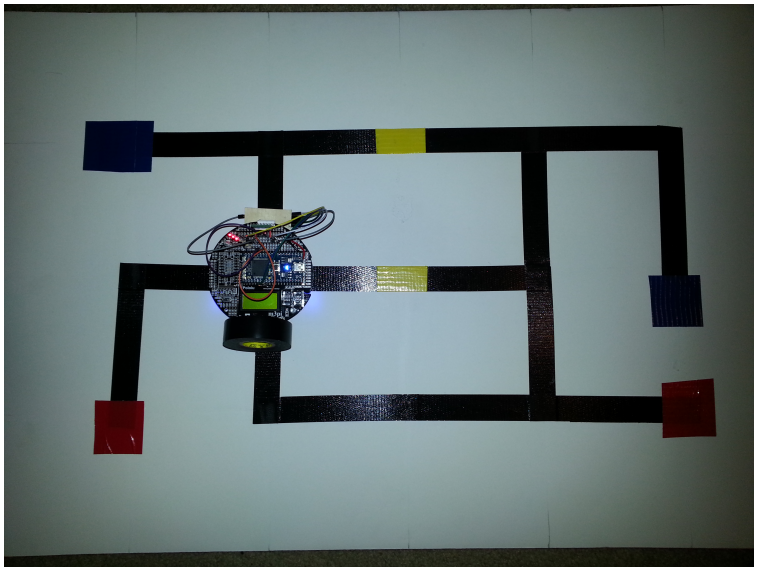


Figure 2: Maze with robot

Similarities to Other Projects

At first glance the problem statement seems very similar to that of a typical maze-solving robot project, where a robot is given a maze, and must navigate and solve the maze several times as quickly as possible. There are many competitions where the goal is to do just that, where nearly all the competitors use some variant of the following algorithm:

1. Traverse the maze by hugging the left wall while maintaining a sequence of actions at every intersection until the robot arrives at the destination.

2. Upon arrival at the destination, reduce the series of actions by eliminating any series of actions that brings the robot back to a previous location, for instance reaching and coming back from a dead end. For a maze without loops, this would result in the shortest path from start to finish.

3. Repeat this path as many times as necessary

However, comparing the standard maze-solving approach to ours, we notice a few key differences in the assumptions that lead to very different solutions.

1. The typical maze solving project assumes that the robot has a fixed start position and destination. In our project, we allow the robot to start at any location and the robot must be able to find its way to multiple destinations.

2. The typical project assumes a maze without loops. However, in our project, not only do we have loops that we make sure the robot doesn't get stuck in, but we also have doors that open and close, resulting in a somewhat "dynamic" maze.

Two Phases of Action

The problem naturally separated itself in to two major phases: mapping and navigation.

- **Mapping**
  Before any path finding can be done, the maze must first be mapped out. During the mapping phase, the computer instructs the robot to continue exploring until a map can be fully drawn. Meanwhile, the robot follows instructions sent by the computer and provides information to the computer regarding what turns can be made at each intersection. The robot also lets the computer know when it has traveled to doors and destinations. All doors are assumed to be open during the mapping phase. By using a computer to construct a map and determine the exploration path, we prevent the robot from getting stuck in a loop. This method also allows us to more efficiently explore the maze.

- **Navigation**
  Once the map of the maze has been constructed, a user can open and close the doors and instruct the robot to go to any of the explored destination points. The computer will then use uniform cost search (UCS) to find the shortest path to the destination. If the robot encounters a closed door, it will stop and the computer will determine the next shortest path to get to the destination.

Multiple Levels of Control

We approached this project with three levels of abstraction, a high level computer for data collection and path planning, an mbed microcontroller to handle maze navigation and line following, and an ATmega328 microcontroller to control the individual actuators. In a real world situation, the computer would be on-board the robot, taking in user inputs from a web interface. However, due to the lack of processors on the robot and a focus on robot navigation instead of web development, we decided to provide this functionality from a laptop.

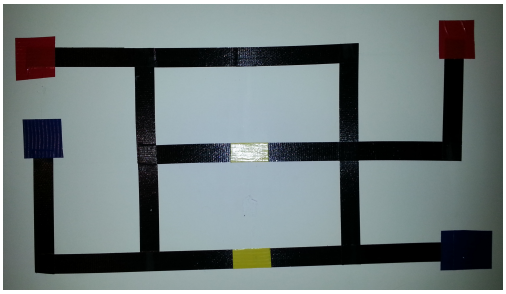
- **Path-Finding and User Input**
  At the highest level, we have a laptop computer to store the explored maze and interact with the user to determine where the robot would go next. The laptop is also used to determine what paths the robot needs to take to explore the maze and how get to each destination using UCS.

- **PID Control**
  On the m3pi robot, we attached an mbed microcontroller to handle the maze navigation and PID control loop for line following. Taking the instructions from the laptop, the mbed would determine how fast the robot needed to travel and how much it needed to turn to follow the line and make the turns at the appropriate intersections.

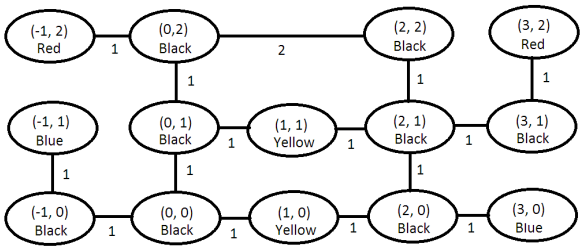
- **Actuator / Sensor Control**
  At the lowest level, the m3pi comes with an ATmega328 microcontroller. We use this microcontroller as a slave that takes the robot velocities and determines how much voltage and current would be needed to be supplied to the actuators to produce the desired behavior.

Algorithms & Formal Models

Mapping the Maze



(a) Picture of a line maze with doors and destinations



(b) Same line maze modeled as an undirected graph

Figure 3: Modeling of line maze

A typical maze that we map and solve is shown in Figure 3a. Because the vertices are spaced evenly and all motion is restricted to north, south, east, and west, we can represent the maze with a graph such as that in Figure 3b. Each vertex is a coordinate and color of the coordinate. For our project, yellow tape represents a door, and blue and red tape represent destinations. Edges represent black tape that connect vertices and the cost of the edge is the length of the tape. Therefore, we can model the exploration using graph algorithms, two key ones of which are Algorithms 1 and 2.

How these algorithms work with the robot can be described as follows: When the robot begins exploring, it first inspects what edges surround its current location. It will then add this vertex to a list of vertices that have edges that need to be traversed. The following process is then repeated until there are no vertices in the list:

1. The computer will have the robot go to the vertex the shortest distance away in the list of vertices with unvisited edges. If the vertex the robot is at currently has unvisited edges, the robot does not move.

2. The computer will have the robot go to a direction that has not been traversed yet.

3. The computer will time how long it takes the robot to reach either a branch, fork, or a tape segment that is not black (meaning it has reached either a door or destination). It will create a vertex at coordinates based on how long it takes the robot to get there.

4. The computer will add edges to the maze it is creating based on the information the robot sends back. If the vertex has edges that need to be visited, this vertex will be added to the list of vertices that must be visited. If at any time there are vertices in the list that don't need to be visited, they'll be removed from it.

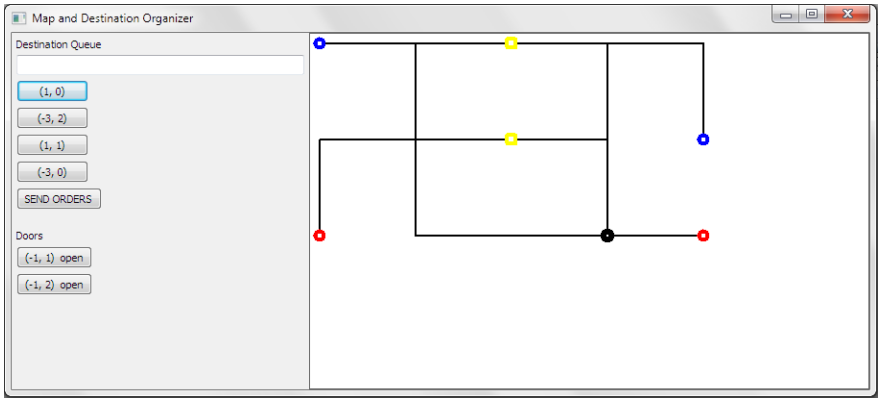
Algorithm 1 Algorithm to explore maze

```
1: procedure ExploreMaze(unknownMaze)
2:     maze ← empty graph
3:     maze.VerticesToVisit ← {}
4:     maze.AddVertexAndEdges(unknownMaze.StartState)
5:     while maze.VerticesToVisit is not empty do
6:         currentState ← maze.NearestVertexToVisit()
7:         unknownMaze.SetCurrentState(currentState)
8:         maze.AddVertexAndEdges(unknownMaze.ExploreNewVertex())
9:     end while
10: end procedure
```

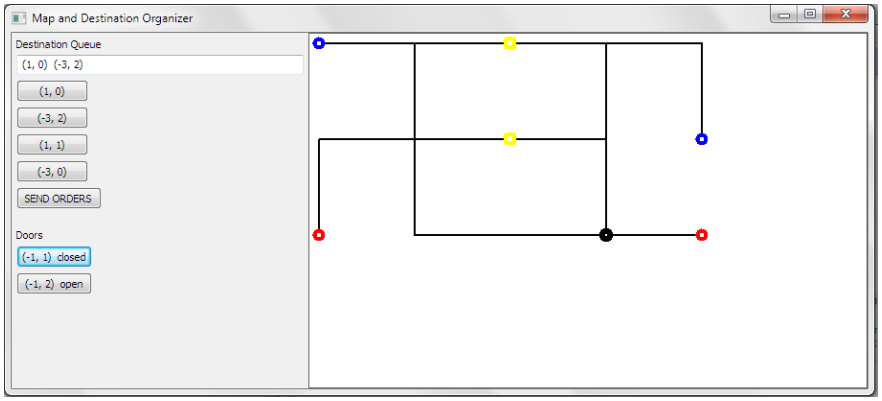
Algorithm 2 Algorithm to add vertex and edges and to update *maze*.VerticesToVisit

```
1: procedure AddVerticesAndEdges(maze, newVertex)
2:     previousVertex ← maze.CurrentState
3:     maze.AddVertex(newVertex)                ▷ also sets the current state to newVertex
4:     maze.VerticesToVisit.Add(maze.CurrentState)
5:     for all direction in [North, East, South, West] do
6:         if newVertex has edge of orientation direction then
7:             vertex ← maze.FirstVertexAfterAction(maze.CurrentState, direction)
8:             if vertex exists and (ManhattanDistance(maze.CurrentState, vertex) = 1 or
   vertex = previousVertex) then
9:                 maze.AddEdge(maze.CurrentState, vertex)
10:                vertex.ChangeVisit(direction, False)
11:                maze.CurrentState.ChangeVisit(direction, False)
12:                if vertex.NoEdgesToVisit() then maze.VerticesToVisit.Remove(vertex)
13:                end if
14:            else
15:                maze.CurrentState.ChangeVisit(direction, True)
16:            end if
17:        end if
18:    end for
19:    if maze.CurrentState.NoEdgesToVisit() then
20:        maze.VerticesToVisit.Remove(maze.CurrentState)
21:    end if
22: end procedure
```

Navigating the Maze



(a) The solved maze



(b) Adding destinations and closing doors

Figure 4: Resultant Map GUI

Once every vertex in the maze has been discovered, a window like that in Figure 4a appears. The black circle indicates the position of the robot when exploration began and is used to help the user match the destination on the map to the coordinates in the GUI. For instance, the upper-left blue vertex has coordinates $(-3, 2)$. The user can then click destination buttons to have the robot visit destinations in a certain order. The user can also close doors, meaning that the robot cannot travel on an edge to the door and exit the door using a different edge. Both of these features are shown in Figure 4b.

Once the user presses the "SEND ORDERS" button, the computer will iterate through each destination in the destination queue, building an optimal list of actions using UCS and sending this list to the robot so that it can reach the destination. However, the computer first assumes that all doors are open since in reality doors are open and closed without the robot's knowledge. If a robot happens upon a door which the user made to be closed, the computer will remove all edges except whichever one the robot used to get to the door (so the robot can still access the maze) and will perform UCS again. The computer will then assume that all doors that were discovered to be closed remain closed until all desired destinations have been visited. Then all doors are assumed to be open again.

Maze Traversal

For the robot, we based the maze traversal code on PID control. Given a series of commands the robot needs to execute:

1. The robot follows the line it is currently on using standard PID control until it detects a vertex. We define a vertex as a point where the reflectance sensors detect a path to the left or right, no path to the front, or colored tape right in front of it.

2. Upon arriving at a vertex, we break out of the PID control loop and stop the robot.

3. If the computer had expected the robot to arrive at this vertex and had already sent a command (e.g. left turn) for the robot to execute at this vertex, the robot would execute the command and resume PID control.

4. If no command was listed, the robot will read from the color sensor and send the computer information regarding the vertex. The robot sends the color of the vertex as well as which directions paths extend from this vertex (front, left, right) and the computer uses this information to decide on the next series of moves for the robot to execute.

Major Technical Challenges

Color Sensor

One huge problem we encountered was that obtaining an accurate reading from the color sensor took about one tenth of a second. We originally wanted to use the color sensor within the PID loop to help us determine when we arrive at a door or intersection and stop the robot. However, with the PID loop running hundreds of times each second, adding in the color sensor would slow the PID loop so much that the robot could no longer consistently follow the lines. We did find that measuring from the reflectance sensors took very little time and wouldn't affect the PID controller's performance if included in the loop. Thus, we only used colored tape that the reflectance sensors would read as equivalent to white, in our case red, blue, and yellow. This way, when the reflectance sensors detects a dead end ahead, we can stop and use the color sensor to detect the color before telling the computer what type of intersection we stopped at.

Bluetooth

Another problem we encountered was that when the robot was far from the laptop, data packets transmitted between the two would occasionally be lost and take extra time to arrive. Because the laptop uses the time between data packets from the robot to determine path lengths during the mapping phase, this would sometimes result in incorrectly drawn maps. As a result, we've had to make sure the laptop is close enough to the robot to prevent many lost packets. In a real world situation, however, this would not be an issue, as the computer would be on board the robot and wireless communication would not be necessary.

Reflectance Sensors

We've had a lot of issues with the reflectance sensors not reading intersections correctly. The robot would sometimes detect a path where there was none, and would occasionally drive off the maze without being able to correct itself and make its way back. Another problem we've had was that it

would sometimes not pick up a path where there was one. If this occurred in the mapping phase, this would sometimes result in unexplored sections of the map. This problem was not particularly difficult to solve. It ended up involving tedious tweaking of reflectance thresholds to make sure the sensors could properly detect intersections in various lighting conditions.

Summary of Results

The robot was able to completely traverse a small maze, and the computer could map out the maze properly. The optimal path code worked both within maze exploration and maze navigation, including when doors were discovered to be closed.

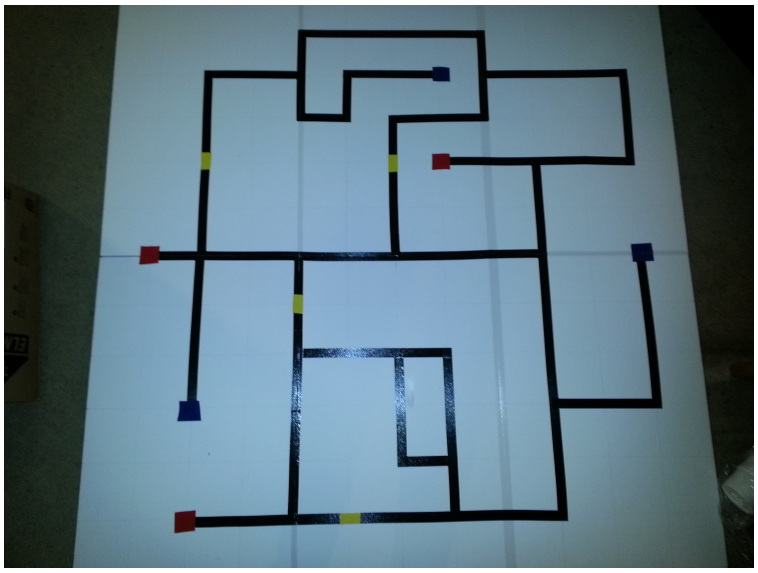


Figure 5: Larger maze

However, the maze exploration could not be done on bigger mazes such as that in Figure 5. Because larger mazes had to be made from multiple posterboards taped together, black tape layed along where posterboards were joined needed to be cut down the middle. This would cause faulty readings to come from the reflectance sensors while the robot drove over the tape, causing the robot to drive off the map. Improvements in the code to measure from the reflectance sensors or restricting maze constructions so that tape doesn't run along gaps in the posterboards would allow the robot to traverse bigger mazes.

Team

Nolan Wagener

Nolan was in charge of the code on the laptop. The code for mapping out and navigating through the maze was his responsibility. Nolan was responsible for creating the GUI for user interaction as well. UCS search code was adapted from work in CS188. Finally, Nolan added code to the laptop and the mbed to handle calibration of and color classification from the color sensor.

Kyle Chiang

Kyle was in charge of the code on the robot. The PID and maze traversal code was his responsibility. Making sure the controller was robust and that the robot could navigate the maze in different lighting conditions was also his responsibility. The low level slave code was provided by Pololu.

Shared Responsibilities

Responsibilities for obtaining and adding hardware onto the m3pi were shared. Creation of the mazes as well as testing all of the code was also done together.

Relation to Class

Concurrency

While only a single process was run on each processor, we did have critical sections where code on the computer and mbed needed to be run in a specific order, such as when a new edge was being explored. The robot would send a message to the computer saying that it was about to explore a new edge. The computer would wait until it received another message, this time signifying that the robot found a new vertex. Using the amount of time between the messages, the robot could infer the length of the edge the robot traversed. The third message from the robot would contain information about the intersection or the color of the vertex.

Modal Behavior

While the individual states in the maze following code were not clearly labeled, the robot transitions between specific states to reliably navigate the maze.

Real-Time Networks

The mbed on the m3pi communicates with the laptop in real time over a Bluetooth connection.

Feedback

Information about serial communication and how to code around them was very useful when we dealt with the Bluetooth communication. As well, labs that focused on the iRobot Create helped to prepare us on how to program the m3pi.

Acknowledgments

We would like to thank the following people:

- Professors Lee and Seshia for giving us the opportunity to work on this project and providing helpful information from lab and lectures that aided us in the completion of the project,
- Skot Croshere for providing us with materials and guidance with how to get started with them,
- Pololu for providing an AVR Library for the ATmega328 slave code,

- the `mbed.org` community, specifically Jon Marsh for PID code for the m3pi and Michael Walker for code to measure the RGB-Clear channels of the S371 color sensor,
- the CS 188 staff for giving us the code framework to be able to find optimal paths in our maze, and
- Garvit Juniwal for offering to be a mentor and providing moral support when no other mentor was available.