**Rose-Hulman**

16

**Mapping and localization**

**ECE 425**

**Matthew Schack and Peter Heath**

**Abstract:**

The purpose of this project was to get Data to: map a maze and localize. There was also a stretch goal of combining the two into a SLAM method. Data was tested in 6’ x 6’ maze segmented into 18” x 18” squares making it a 4 square x 4 square maze. Data had to traverse the maze with different amounts of starting data ranging from having just a map to having only the starting orientation. We were able to implement all goals of this project including the stretch goal of creating a SLAM method.

**Table of Contents:**

Abstract 1

Objective 3

Theory 4

Methods 5

Results 7

Conclusions and Recommendations 8

Appendix 9

1. **Objective:**

The objective of this project was to implement mapping and localization methods with the Arduino robot. Mapping required use of wavefront propagation as well as topological mapping. Localization required feature identification. These objectives were tested on a maze to see if Data could perform what we wanted it to do. There was also a stretch goal of getting localization and mapping to work at the same time, but because of memory issues it is not required.

1. **Theory**

Map making – To make a map knowing only a location we used a recursive algorithm that looked for an unmapped square and drove towards it. Data also used a topological map that gave each square a value depending on the squares around it, this made localization a lot easier because all the data about the surrounding cells was given in every cell.

Localization – To find where the robot was in a defined map we recorded the topological positions Data visited and compared them to the well-known map. Eventually (usually after 2 iterations) there is only one possible spot for Data to be.

SLAM – To both localize while making a map Data just assumed it started at any coordinate (I chose (2, 2)) and continued to make a map like normal. However, when Data encountered an opening when it should have been at an outside wall it shifted the entire map to correct herself. If there is a path that runs from top to bottom and a path that runs from the left to the right once the final map is complete Data will also be localized because Data will have hit every boundary and shifted the map until it was correct. This is a safe assumption because a 4 x 4 map must have a path that runs from top to bottom and from left to right, otherwise it would be a smaller map than 4 x 4.

1. **Methods**

Map Making – As mentioned before the map making algorithm recursively looked for a non-mapped spot and drove in that direction. The actual algorithm is shown graphically in Appendix A. The algorithm first looked if the coordinates passed in were out of bounds, a wall, or a square the algorithm already looked at, if it was then the algorithm returned false. Otherwise, the algorithm looked at every possible movement Data could make in that square. If there were any spots that were unvisited Data set a global variable to that direction and returned true. In the case where there was more than 1 direction that was unvisited Data set the global variable to the direction that moved Data closer to an outside wall. This is because since we needed to map the entire maze we wanted Data to move towards dead ends first. The squares closer to outside walls were more likely to be dead ends due to the fact there were less places to move, so Data moved in that direction first. If there was not unexplored direction next to Data then the algorithm recursively called itself in every direction around Data. The algorithm looked for the directions that returned true and set the global variable to that value and returned true. If no direction returned true then the algorithm returned false. If the first call to the algorithm returned false, then the map was completely mapped.

Localization – The localization method worked by finding every square in the map that matches the current square Data is at. It then back traces the path Data took to get that square and sees if the recorded values match the map values. It repeats this until it finds only one possible position that matches every step which would be Data’s current location. A graphical representation is shown in Appendix A.

SLAM – SLAM was implemented by adding an extra twist to map making. SLAM assumed that Data started at an arbitrary point and preceded to map make like usual. If Data ever got an opening where it thought there should be an outside border Data shifted the map and its position in the map to compensate. Once, the map had been completed Data would have localized because of all of the map shifts.

1. **Results**

We were really happy with how Data performed. Data was able to map a 4 x 4 maze, localize in the maze, and localize while mapping in the maze. This means that Data accomplished every goal we set out to do. It is important to know that while Data was able to mentally do the 4 x 4 maze it required a lot of manual help to accurately move within the maze. A future goal could be to somehow incorporate more sensors and try to perform this lab with little to no human interaction.

1. **Conclusions and Recommendations**

The purpose of this lab was to have Data perform map making and localization. In that regard, Data definitely performed the purpose of the lab, however when Data was navigating the course it was not 100% autonomous. In fact, Data required a human to correct its movement nearly every square it moved. This shows that for robotics having a robot that can analyze data is only half the battle. The robot also needs to be able to move and accurately know where it is in the world for that data to make any sense. While Data did can analyze data really well, it cannot move accurately. To fix this we would recommend adding on sensors such as motor encoders and a better compass, so Data can know its orientation and how far it has moved. These additions combined with sensor feedback already on Data would enable it to move much more accurately.

**Appendix A:**

No

Get start and end points

Poll Sensors

Update map

Recursively find next direction

Move and update position

Are there unmapped points?

Yes

Plan a path

Execute path

Figure 1. Map making code flow chart

Yes

Set the directions farther away from the border to false

No

Return false

We have completed the map

Return false

No

Input an x, y point and an array of visited points

Are points out of bounds, previously visited, or in a wall?

Add current point to visited array

Is there more than one unmapped direction?

Set next the next movement to the unmapped direction

Check each direction and see if it is unmapped

Yes

Call the function on every direction

Are there unmapped directions?

Yes

No

Are there unmapped directions?

No

Yes

Return true

Figure 2. Recursive algorithm for finding next direction to go for map making

Yes

Get end point

Poll Sensors

Update world map

Find next direction

Move and update position

Does the world map match exactly one point?

No

Set the location and plan a path

Execute path

Figure 3. Localization code flow chart

Yes

No

Initialize

Poll Sensors

Update map

Recursively find next direction

Move and update position

Are there unmapped points?

Yes

Plan a path

Execute path

Shift map and location

Is there an opening where there should be an outer border?

No

Figure 4. SLAM code flow chart