Rose-Hulman Institute of Technology

ECE 425 - Mobile Robotics

Devon Adair & Hunter LaMantia

Robot: Moravec

Navigation Competencies & Wireless Communication

2/17/2019

**Abstract**

In this project, a mobile robot was given algorithms to plan paths through a given map, find itself in a map, and create a map using its sensors. The project was successful, with the robot completing all of its tasks. This was done by creating movement algorithms for all three goals, as well a function that converts the four by four occupancy grids and topological maps into nine by nine occupancy grids.

The robot suffered from a few sources of error. The greatest source of error was odometry error. The robot has a bad wheel that frequently slips, causing all movement not based on a controller to be inaccurate. Even in the cases where the movement was controlled, the controller would sometimes struggle to keep up with the odometry error. Additionally, the robot’s sensors have a tendency to give inconsistent data, but this was fixed by taking many readings at a time, averaging them, and putting them through a calibration equation.

**Table of Contents**

Abstract ...…………………………………………………………………………………………1

Table of Contents …….........………………………………………………………………………2

I. Objective ……...………………………………………………………………………………...3

II. Theory ………………………………………………………………………………………….3

III. Methods ……………………………………………………………………………………….4

IV. Results ………………………………………………………………………………………...7

V. Conclusion ……………………………………………………………………………………..8

References ……………………………………………………………………………………..….9

Appendix ………………………………………………………………………………………...10

**I. Objective**

The purpose of this project is to develop a program that allows our robot to travel a four square by four square environment using path planning, localization, and mapping. The robot can be driven automatically or with manual wireless inputs. In both cases, the robot wirelessly sends map data back to a transceiver connected to a laptop, where the map data is displayed. The main movement algorithm utilized by the robot functions by using wall following code from a previous lab. The robot is also given a series of directions to go in at T junctions, allowing it to make turns.

**II. Theory**

Wireless communication is required for all of our main objectives, so it is very important to have a full understanding of it. The wireless transceivers we use are able to send each other information, but an individual one cannot send and receive at the same time. They must be set to the same channel and have the same pipes. A pipe can be designated as a reading pipe or a writing pipe, so each transceiver needs one of each. According to Dejan (2017), once the transceivers are set up, they can communicate when within 100 feet of each other, so for our purpose we don’t need to worry about the robot getting too far from the laptop. However, we do need to ensure that we are set to a different channel than everyone else in the class to avoid interference.

We interact with occupancy grids and topological map all throughout the project, so we need to know what the differences between them are and how they work. According to Lakemeyer and Nebel (2003), an occupancy grid is a map based on the world as its own entity, whereas a topological map is more concerned with the robot’s point of view of the world. What this means is that an occupancy grid considers each grid square of a map as its own entity, which is either filled or not filled. A topological map considers each grid square to either have obstacles to each of its sides or it does not. The difference is subtle, but important to understand to succeed in this project.

Path planning requires the entire map to be known ahead of time, as well as the starting and ending points. We already have and understand algorithms for basic movement through hallways and around corners, but planning out a specific path is more difficult. To simplify our algorithms, we convert both map types into nine by nine occupancy grids, removing the need to make a separate algorithm for both map types. Once this is done, the path planning is essentially a much simpler version Galceran and Carreras’s (2013) path planning concept, with the major difference being that we do not need to touch every grid square.

Mapping should be as simple as using our movement algorithms to travel around the map until every grid square has been touched. Our greatest concern is echoed by Drumheller (1987), who warns that odometry error can skew a map beyond recognition. To combat the odometry error, we will simply have to be very careful about fine-tuning our movement algorithm. He also had to fight sensor error, but our environment is much simpler than his, so it should not be a major problem.

**III. Methods**

The first algorithm made was a simple movement algorithm meant to supplement the eventual path planning algorithm. The movement algorithm is passed a string containing the letters S, T, L, R, and F. The robot increments through the string and its behavior changes based on what letter it is currently on. S tells the robot to start its hallway following, and always appears at the beginning of the passed string. T tells the robot to stop moving once it reaches its goal and is always placed at the end of the string. L, R, and F are all behaviors for when the robot senses a place where it could potentially turn. L tells the robot to turn left, R tells it to turn right, and F tells it to keep moving forward.

We opted to give the path planning algorithm a second helper function to make maps easier to parse. Instead of writing path planning functions for the robot to navigate four by four occupancy grids and four by four topological maps, we converted both into nine by nine occupancy grids, where the give extra rows and columns represent the spaces between cells of the real grid and are counted as full if walls exist there. This allowed us to make only one path planning function for both types of input. The nine by nine is initialized as a blank map, as shown in Figure 1.

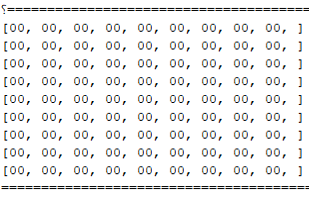


Figure 1: Blank occupancy grid

The occupancy grid is then filled in based on the contents of the initial four by four map. Spacing with obstacles are marked with 99s as shown in Figure 2.

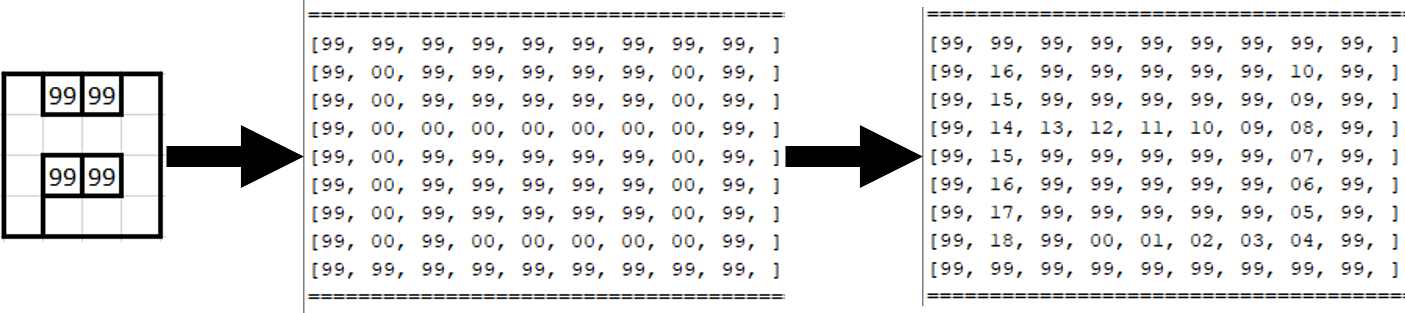


Figure 2: Map type conversion

The path planning algorithm itself takes the starting coordinates and target coordinates of the robot as inputs, and also accesses the nine by nine occupancy grid created by the aforementioned helper function. The algorithm assigns a value of zero to the location of the target coordinates and increments the value up for the surrounding map squares. This results in each square having a number equal to its distance from the target square, as seen in Figure 3.

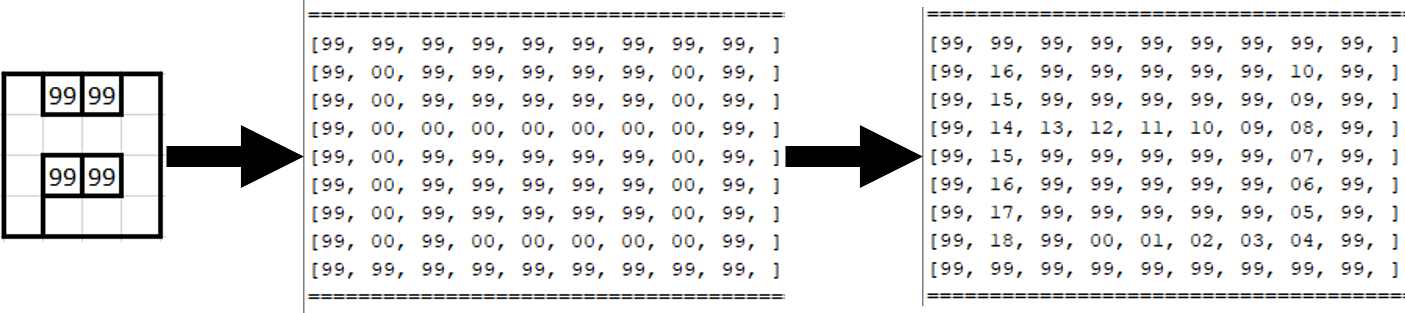


Figure 3: Path planning in the occupancy grid

The algorithm also takes walls into account, making the numbers snake around them. The algorithm then creates a list of instructions in the form of a string that the movement algorithm and read and passes it.

The localization algorithm is similar to the path planning algorithm in that it uses the nine by nine occupancy grid to navigate, except it does not know where on the map it begins. We set up a basic movement algorithm for localization that uses the sensors to determine which adjacent squares are empty, and then it chooses one to travel to. Every time the robot travels to a new square, it updates how far from its starting position it has gone and uses its sensors to scan the area around it. After scanning the area, it interprets its current location as a topological map square and adds it to an array. It then checks the map for possible combinations of squares it could have traveled based on the topological values. Once it has been narrowed down to one path, the robot knows where it is and can back calculate where it began based on its tracked displacement. The robot then sends its current coordinates to the path planning algorithm and it travels to its target. This process can also be done by driving the robot manual with commands from the laptop and transceiver. In either case, the robot also sends its current, beginning, and ending coordinates to the transceiver using bi-directional communication.

The mapping algorithm, much like the localization algorithm, works with both manual driving and automatic driving. It is essentially the opposite of the localization algorithm, in that the robot knows its starting position, but does not know anything about the world around it. We set up another movement algorithm similar to the one made for localization, except we make sure that the robot eventually enters and scans every map square that it detects as open. It starts out with an occupancy grid marked as completely full, and whenever it enters a new map square and scans it, it changes the relevant squares to be empty. Once the map is completely explored, all empty map squares have been marked as empty. Once again, the completed data is then used by the path planning algorithm to make the robot go to its target location.

*1. Were there any issues with the wireless communication? How could you resolve them? If at all.*

We had a lot of problems where the robot would fail to send maps to the transceiver, or it would only send brief parts of maps. We eventually found that us sending commands to the robot was interrupting the robot sending us maps, so we resolved the problem by only having the robot send one row of the map at a time and adding delays to the sending/receiving code.

*2. What does the state machine, subsumption architecture, flowchart, or pseudocode look like for the path planning, localization, and mapping? (It should be in the appendix of the report).*

(State machine and pseudocode in Appendix)

*3. How would you implement SLAM on the CEENBot given what you have learned about navigation competencies after completing the final project? If you research solutions, make sure you cite and list references in APA or MLA format.*

SLAM would be simple to implement using our current code. The robot

*4. What was the strategy for implementing the wavefront algorithm?*

(Pseudocode shown in Appendix)

*5. Were there any points during the navigation when the robot got stuck? If so, how did you extract the robot from that situation?*

For some time, we had a problem where the robot would encounter a dead end and continuously move back and forth into and out of it. This was found to be an edge case in our movement algorithms, so we set up special behavior for avoiding dead ends once they are discovered.

*6. How long did it take for the robot to move from the start position to the goal?*

The robot moves relatively slowly in its movement, because we wanted it to take its time. We have found in the past that moving quickly leads to more odometry error, which we were already dealing with and did not want to make worse. Because of this, it took a while for the robot to get from place to place.

*7. What type of algorithm did you use to selection the most optimal or efficient path?*

(Path planning pseudocode shown in Appendix)

*8. How did you represent the robot’s start and goal position at run time?*

On the second map in Figure 3 you can see numbers representing the path from start to finish. The goal position is represented as 0, and the starting position is just whatever number that grid square’s value happens to be. In the cases where it was necessary to make the starting position explicitly clear we placed an large, arbitrary number (great than 20 and less than 99) on the starting position.

*9. Do you have any recommendations for improving that robot’s navigation or wavefront algorithm?*

Our wavefront algorithm already gives the ideal path, but the code itself is very long and not module. To improve the algorithm, we would split the function up into many smaller functions, making it easier to interpret and troubleshoot the code, if necessary.

*10. How did you use the serial monitor and bi-directional wireless communication to represent the map?*

The robot wirelessly sends the laptop one row of the map at a time, and the laptop prints it. We send one line at a time because the sending always got interrupted when we tried sending the whole map at once, and also because it made the code simpler.

*11. What type of map did you create and why?*

We used a nine by nine occupancy grid because it allowed us to only need to make one type of behavior for each function, rather than one for a four by four occupancy grid and one for the four by four topographical map.

*12. What was key in the integration of the localization, mapping, and path planning?*

Our base movement and map conversion algorithms were key to all of our goals. Localization, mapping, and path planning all used both, so we had to ensure that they worked perfectly.

**IV. Results**

In general, our robot worked very well. Our path planning algorithm has no problems whatsoever. Our old hallway following behavior is sufficient to get the robot from one place to another, and our wavefront does a great job at telling the robot what to do at intersections. However, we did have a lot of problems leading up to the current design. Our original design was not robust at all and was incapable of recognizing when it had to go straight through intersections.

Our localization algorithm also works with very few problems. Initially, we had a problem where the robot would get stuck going back and forth repeatedly when it reached a dead end. We easily fixed that by making the robot recognize when a dead end was reached and preventing it from returning to it. We also had a problem where our code that worked for an occupancy grid did not work for topological maps with thin walls. This was because the robot was able to disregard walls in the occupancy grid, because there was no way for the robot to end up on the other side of one. However, the thin walls had occupiable space on both sides, so we had to teach the robot to recognize wall and not think that it went through them. The only remaining problem is odometry error, which we have done our best to minimize by getting the robot’s wheels fixed and recalibrated.

The mapping algorithm has the same odometry error as the localization. Additionally, we have found a few edge cases involving dead ends that we resolved. We believe that we have fixed all of the edge cases, but we can’t be certain. The biggest error we resolved was a problem with bi-directional transceiver communication. We found that when we sent commands to the robot from the laptop, it would prevent the laptop from sending the map back. This timing issue was resolved by skipping the manual inputs and going straight to automatic mapping. This was not ideal, but it was the only solution we could find.

**V. Conclusion**

We successfully completed all goals and requirements for this project. The biggest thing that we learned is how to deal with the inevitable error that comes with moving a mobile robot. They are not the perfect machines that we expected them to be, and they need some sort of feedback control to be truly useful. Even a task as simple as driving forward comes with enough odometry error to make the robot crash into a wall. Besides odometry error, sensor error is also a very big problem with mobile robots. Thankfully, we learned that both problems can be dealt with using controllers and sensor calibration.

Given the chance to do this project again, we would completely change how our movement algorithm works. It currently works by following a list of instructions of what to do at junctions, but we would rather have made it follow a list of instructions to move grid square by grid square. That would have made the wavefront algorithm much more simple to generate, which was one of the most difficult parts of the project. Additionally, we feel that with our current knowledge we could have expanded the project to include hall following behavior in our localization and mapping, which we deemed too difficult to do this time. Our problem was that we weren’t sure how to make the behavior track which grid square the robot is in, but that would be simple for us now. We also could have split our code into smaller functions. Our wavefront function is currently very long, to the point where reading through it is very difficult.

References

Dejan. (2017). Arduino Wireless Communication – NRF24L01 Tutorial. Retrieved from https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/

Lakemeyer, Gerhard, Nebel, Bernhard. (2003). Exploring Artificial Intelligence in the New Millennium. San Francisco: Morgan Kaufmann Publishers.

Galceran, Enric, & Cerreras, Marc. (2013). A survey on coverage path planning for robotics. Elsevier, 61(12). Retrieved from  
https://www.sciencedirect.com/science/article/pii/S092188901300167X

Drumheller, Michael. (1987). Mobile Robot Localization Using Sonar. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, 9(2). Retrieved from  
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4767907

**Appendix**

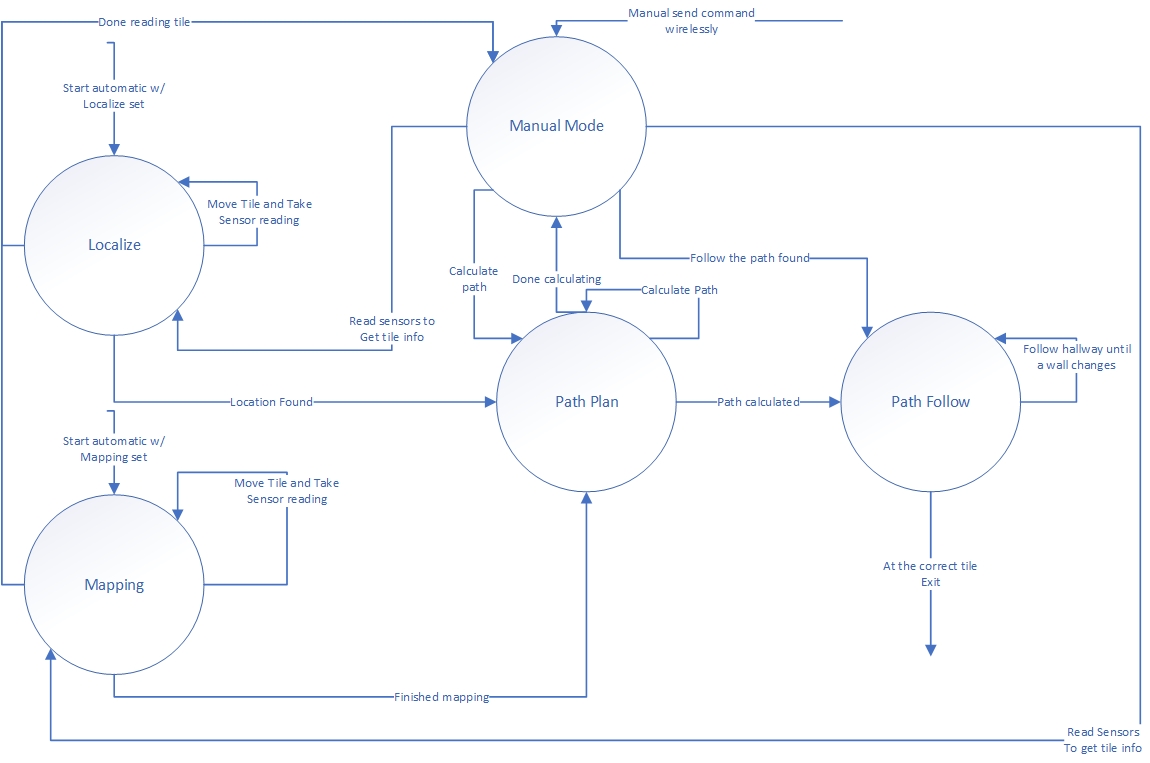


Figure 4: State diagram

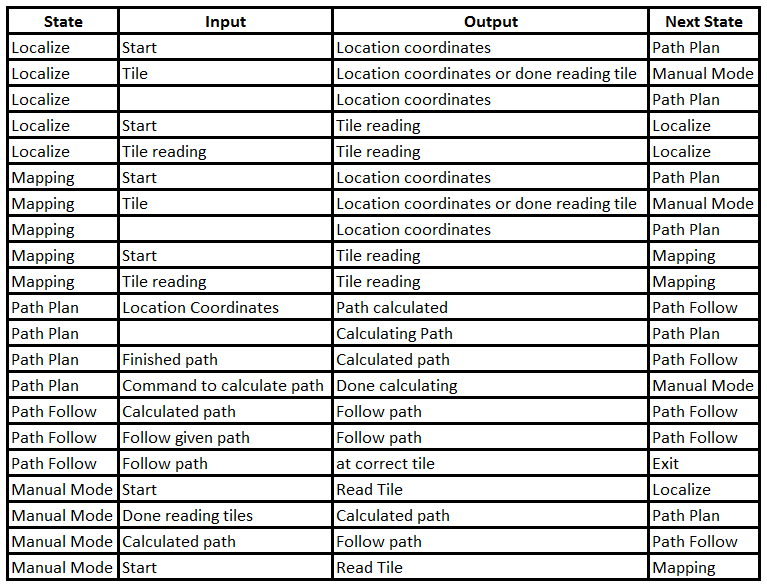


Figure 5: State transition table

**Wavefront Pseudocode:**

Mark north, south, east, west, tile around goal as one step

Mark goal as a high number not 99

While tile not marked:

               If tile value is zero and not an obstacle:

                              Set current step value as the current tiles value +1

                              Bounds check row is less than 9:

                                             If the row + 1 is a zero and not the goal:

                                                            Set row+1 == current step value

                              Bounds check row is greater than 0:

                                             If the row +-1 is a zero and not the goal:

                                                            Set row-1 == current step value

                              Bounds check column is less than 9:

                                             If the column + 1 is a zero and not the goal:

                                                            Set column+1 == current step value

                              Bounds check column is greater than 0:

                                             If the column - 1 is a zero and not the goal:

                                                            Set column-1 == current step value

               Increment col

               If column equals 8:

                              Loop across double:

                                             Check if tile value is zero and the row and column aren’t the goal:

                                                            Set flag that there is remaining tile

                              Check flag for remaining tiles:

                                             Done if false

                              Else reset the flag

               Increment row

**Path Planning Pseudocode:**

While the goal isn’t found:

               If current tile is goal:

                              exit

               set options array to

               south

               north

               east

               west

               for (i = 0; i < 4; i++):

                              if option[i] is less than other option

                                             set the index of smallest option

               switch on smallest index:

                              case south:

                                             set correct direction character

                                             set direction as south

                              case north:

                                             set correct direction character

                                             set direction as north

                              case east:

                                             set correct direction character

                                             set direction as east

                              case west:

                                             set correct direction character

                                             set direction as west

               set previous step as the smallest index