Neil Khera

Final Report

16-811

10 December 2021

Maze-solving through gradient descent on WiFi signal intensity

Motivation:

In recent years there has been increased robotic development in the field of search and rescue in environments that have the possibility of being too dangerous for human beings to traverse safely. One inspiration for me was being around people in the Robotics Institute who were members of Team Discovery in the DARPA subterranean challenge. The DARPA Subterranean (SubT) Challenge aimed to develop innovative technologies that would augment operations underground. The purpose of the challenge was to explore new approaches to rapidly map, navigate, search, and exploit complex underground environments, including human-made tunnel systems, urban underground, and natural cave networks.

What I found to be an interesting challenge was being able to create robots capable of exploration in GPS denied environments. Specifically, I wished to explore how we can effectively recover the machines when they are required to return to a specific base of operations, perhaps when they are running out of energy, or if they must bring back data to be analyzed, or simply when the mission comes to an end.

Typical techniques of employing this Return-to-Base procedure would involve some sort of map generation through the course of the exploration and keeping track of the robot’s path through it, and then using the localized position to backtrack to the home position (or other similar techniques). However, I considered the possibility wherein a system issue causes some sort of reset of the robot and consequent loss of data such that the robot loses its localization and ability to backtrack to base.

Another inspiration also came from my participation in CMU’s MoonRanger project. We intend to create the first fully autonomous planetary exploration rover. One of MoonRanger’s purposes will be to conduct a multi kilometer trek away from the lander in search for Permanently Shaded regions on the Moon’s south pole, and then quantify the level of water content in the form of ice in the lunar regolith. MoonRanger is a microrover weighing in at a mere 18 kilograms. As such it lacks the heavy and power-hungry comms systems that can communicate directly to Earth. As such MoonRanger must transmit its data products first to the lander and which then transmit it to Earth (and vice versa). This rover-to-lander communication takes place through regular 2.4GHz WiFi which is specced to have a range of about 80m. However, something like a 5 km trek outside of comm range with an odometry system with 2% accuracy will yield a displacement of 200m on return (5km + 5km).

Thus, once again this becomes a Return-to-Base problem. I was inspired to use the WiFi signal as effectively a beacon that guides the robot. Using the strength of the WiFi signal (which can be detected much further than 80m even if a connection cannot be established), we can guide the robot to the base position.

The Algorithm:

As I am only beginning my research, I wanted to create a demonstration that would serve as a reasonable proof of concept, be visually engaging, and most importantly be a fun final project for 16-811: Math Fundamentals for Robotics. I thought to reduce the problem down to something simpler. The first simplification was to do path planning in 2D. This a common simplification in the world of path-planning and many algorithms such as A* are taught in two dimensions in the beginning. Further, I noticed the maze-like quality of underground cave systems. Thus, I came up with the idea to create a 2D maze solver that utilizes the intensity of a simulated Wireless emitter to guide the robot's movements using gradient descent path planning.

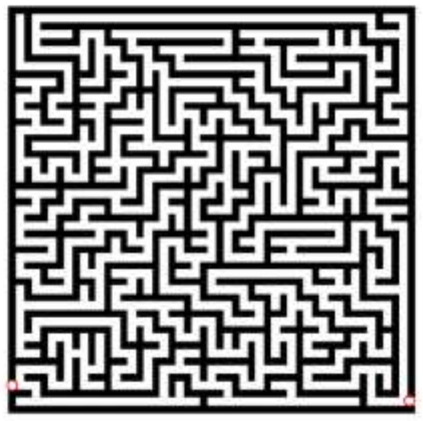


Figure 1: The kind of problem we are attempting to solve

The algorithm proceeds in two steps. The first steps involve setting up the environment, and the second is determining the path to the goal. These are described in detail as follows

Step 1: Setting up the Environment.

Before getting around to solving a maze, we need to find a way to represent the maze in a programmatical manner, as well as representing the wireless signal intensity as a field that flows through the maze starting at the goal location and one that we may be able to perform gradient descent over. My intention was to create a maze that discretizes a continuous environment – thus we may have a scale, for example 1 pixel equals 1 meter, such that when we perform path generation, the robot can be represent as a single pixel and thus have a footprint of 1 square meter. While on a 2D maze this does not matter as all pathways are the same size, it may be useful in the future where some pathways are too narrow for the robot to traverse. We ignore this issue for the time being.

Notice that such mazes can be divided up into square "cells" of equal size with walls on at most 3 of the 4 edges. A visual aid is provided in Fig. 2 to describe this more easily. We describe these edges with the cardinal directions North, South, East, and West. Note that when solving the maze, the robot may only move in one of these 4 directions.

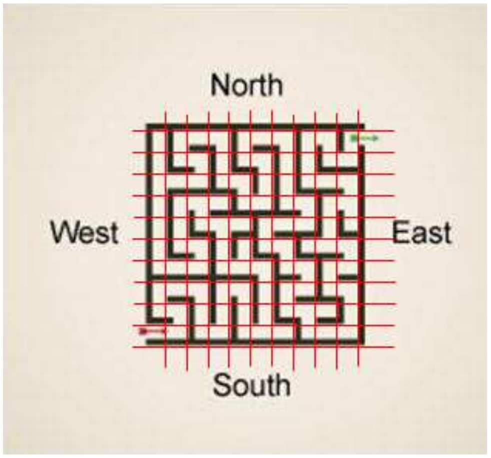


Figure 2: Dividing the map up into cells.

Thus, we define the maze using an NxN array of a Boolean 4-tuple. Each element of this array represents the cell at position [i][j] and the value True or False of this array represents whether there is a wall at the North, East, South, and West sides. Finally, specific cells are chosen as the starting and ending positions for the robot.

We assert that no cell will have walls on all four sides, although so long as a path from the starting position to the final position exists (i.e., these are not in locked-in cells), our algorithm would work even with such locked-in cells, however such cells don't offer any exploration value and as such we ignore them.

Next, we can choose how large (in terms of pixels) we wish for each cell to be. We choose a value of 11 pixels. This gives us a 9-pixel x 9-pixel (and based on our scale, a 9 meter x 9 meter) area in each cell for the rover to traverse (allowing one pixel on each side for walls). By serializing our maze in this manner, we can create any number and size of mazes as well as read and write information about large maps in simple text files.

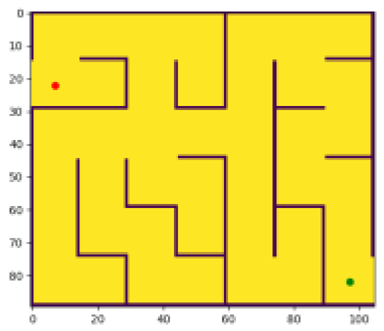


Figure 3: An example map generated through a python script with the starting position (red dot) and ending position (green dot) marked. Visualization through Matplotlib

Next in our environment generation we attempt to plot the intensity of the Wi-Fi signal. I decided that the best way to accomplish this would be through a Breadth-First Search (BFS) starting at the ending position of the maze with a staring high intensity value (say 1000), and then successively reducing the intensity by 1 at each iteration of the search. If a cell is encountered that represents a wall, or one that has already been visited, the search terminated. We let the search proceed until all cells in the maze have been visited (or are walls).

Note that the generation of this environment requires a few assumptions that likely would not apply in real life. Firstly, as described earlier, the maze is a rather basic approximation of a search-and-rescue operation and that we ignore the existence of paths too narrow to traverse. We also assume that there are no loops in the maze since this would break our signal intensity generation process. We also assume that the walls of the maze are perfect reflectors and that there is no noise in the signal. Lastly, for simplicity we assume that the signal falls linearly with distance, as opposed to falling by the inverse square law.

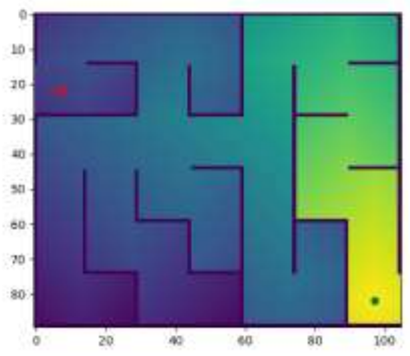


Figure 4: Signal intensity populated on the same map. Yellow corresponds to high values; purple corresponds to low values.

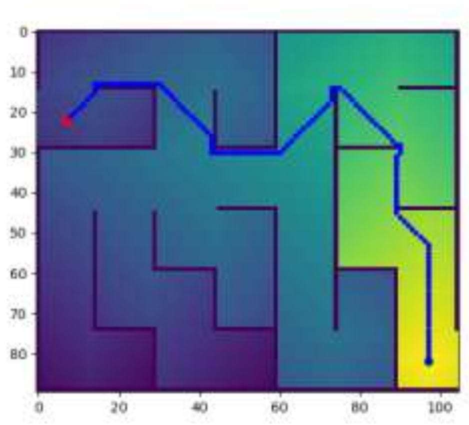
Step 2: Path generation through gradient descent

The process of path-generation is blind – i.e., the robot is unaware of what the map looks like and how the signal intensity looks within it. As such path generation cannot be conducted for the entire map at once. We must thus determine how the robot moves instantaneously by determining the gradient of the signal intensity at the current *(x, y)* position.

Since we can only measure the value of the intensity at the current position of the robot, we proceed by taking measurements I_1 at *(x + c, y)* and I_2 at *(x – c, y)* for some constant *c* distance in the *x*-axis and then calculating *dI/dx* = $((I_2 – I) + (I – I1)) / 2c$. (For our implementation we take c to be 1 pixel). Any positions that neighbor walls use only one of the two measurements. Similar measurements are performed in the *y*-axis. Thus, we now have the gradients *dI/dx* and *dI/dy*. We have the robot move in the direction vector defined by the negative of these values. The reason we perform a negation is since we wish to move towards a more positive value of *I.* As such we wish to perform gradient descent on the value of *-I.*

Measurements are taken as such every few meters (aka pixels), until we reach within a threshold value of I, effectively at a threshold distance away from our ending position.

This is a relatively simple algorithm but works very well. Note that since we are always proceeding to a higher value of I, we will always end up at the global maxima of *I*, aka the global minima of -*I,* which will always be at the ending position. An assumption is that the process of taking those multiple measurements around *(x, y)* provide a reasonable change in Intensity that can be measure with confidence, and that no navigation errors occur in the span between two successive measurements. In any case, the result we get is something like this.



Future Work and Conclusion

I think an interesting avenue of research could include how to measure signal intensity gradients on-the-fly without having to pause and do the 4-point maneuver each time. Perhaps through means of using change of Intensity over multiple sensors on the same robot, or through some interesting S-shaped path generation, such that we can get at least 3 Intensity points that can be using to generate the approximate intensity gradient of the region the robot is in.

Furthermore, it would be crucial to expand the process for 3-dimensional environments, and explore its viability based on the distance the signal may be detected versus the energy required to power the signal emitter. Also, trials in real life locations would be essential to determine performance since real environments are not perfect reflectors and introduce large amounts of noise. In any case, I think that this demonstration, while simple, was able to display a viable technique for allowing robots to autonomously home in on a location. An implementation of the mature version of this program may fly on the first fully autonomous lunar rover, MoonRanger, in late 2023.