MacroMouse: A Maze Solving Robot Final Report

Gagan Awhad, Kris Marose, Xi Qiao, Carl Magnuson
{awhad001, mar0057, qiaox019, magnu213}@umn.edu
Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455

***Abstract*—This report is based around the robots involved in a MicroMouse competition, that is, a robot which is able to search and solve a maze. This project will focus on the design of such a robot, including the advantages and disadvantages of different robot designs. It will also touch on methods for solving a maze, specifically focusing on a real-time search, where the robot searches an unknown maze until it finds a goal. The project also includes localization and mapping of the maze in real time.**

I. Introduction

Research in robotic maze solving problems has been highly active for the past few decades in both robotics and artificial intelligence. The general objective for this kind of problem is to use a mobile robot to navigate an unknown or semi-known environment and to accomplish some specific tasks. Generally, a common task is to navigate from the starting point, to the goal, and sometimes back to the starting point again.

One interesting robotics event related to the maze solving problem is the MicroMouse robotics competition, which was first held in 1979 in New York, and since then various competitions similar to this have started around the world. Small, self-contained robots are used in the competition which attempt to map, then solve and navigate a previously unseen maze. This project isn't necessarily attempting to recreate a robot that would be used in this competition, but is attempting to address a similar problem - robots mapping and solving mazes.

II. Related Work

The robotic mapping problem is one of the fundamental problems in the area of mobile robotics. Much research work has been done on it in the past several decades [1]. Robotic mapping systems have been implemented using many different kinds of the sensors. Some systems use a video camera or a stereo camera and follow a vision based approach [2]. Other systems are based on active range finder sensors, such as laser scanners, sonar [3], and infrared sensors. There are other systems based on the sensor fusion approach [4]. The mapping algorithms used can be classified as deterministic based approach or probabilistic based approach which includes those using methods of Bayesian Probability. Based on different task environments, one may apply indoor mapping algorithms verses outdoor mapping algorithms, as well as statistical algorithms verses dynamic algorithms [5].

As the robot is also keeping track of its position, this project is in fact also related to SLAM (Simultaneous Localization and Mapping). SLAM is a technique in which a robot builds a map of an unknown environment while keeping track of its position relative to the map it is creating [6]. The primary difficulties involved in SLAM however is the representation of the virtual world versus real world. Any inaccuracies related to error from the robot's sensing or odometry propagates into the virtual world and results in an inaccurate map of the real world [7]. This project is a subset of SLAM, where rather than a full scale version, it only deals with discrete dimensions of cells of the maze. This provides some ability to correct for the inaccuracies of the robotic system as each cell of a map has pre-set discrete dimensions that accurately map to the real world (where the real world is a maze that is).

Specifically relating to mazes, there are quite a number of algorithms which have been studied and can be implemented to search for a goal in a maze. Each different algorithm has its advantages and disadvantages, generally trading algorithm run time for solution optimization. Simple approaches toward maze solving which also appear somewhat naive would be wall following or random walking. However, given the general randomness of a path in a maze, these simple algorithms are actually not that terrible of an approach. Unfortunately they will almost never provide an optimal solution. There are other more comprehensive approaches to finding paths in a

maze, however they are usually subject to taking more tome to solve the problem.

The most traditional approach is a simple depth first search, however despite guaranteeing a solution if it exists, it will not necessarily be the shortest solution [8]. The most popular maze searching algorithm that is most commonly used in MicroMouse competitions is Bellman Flooding [9]. This method involves starting from the goal position in a maze, and computing the distance to reach that position as you 'flood' outward from the goal. This however requires that the entire maze already be known, which is not the case for this project. It could however be applied to quickly find a solution after the robot is finished mapping the maze.

III. Problem Description

The goal of this project is to design and implement a robotic system which attempts to explore an unknown maze, constructs a partial map (or the whole map) for the maze , and accomplishes specific maze solving tasks. The most common of tasks would be that of navigation from one point to another. In this, the robot would be given a pair of coordinates which indicate the starting point and the objective point (or goal), respectively. The robot is then asked to find an optimized (or a non optimized) path to connect these two points.

Along with the starting point and the objective point, the robot can also be provided with some information about the dimensions of the maze. For simplicity, the maze will be considered to be broken down into individual cells of standard unit dimensions. Each of these cells will have 0 to 4 walls, and the maze is formed by the interconnection of these cells. There will generally be at least one path connecting the starting point to the objective point, however in the case that there were none, the robot would be able to show that there is no path leading to the goal. Dividing the maze into cells implies that the a wall or the absence of a wall, with unit dimension can be placed only at discrete intervals while building the maze.

The proposed task can be broken down into two subproblems. One of them is the robotic mapping problem. Since the maze environment is fairly unknown to the robot, (the robot only has a few assumptions about the environment configuration), it needs to explore the maze and record the map.

Furthermore, in order to record the map information, an appropriate approach to describe and store the virtual map needs to be chosen. There are many techniques which attempt to accurately record a representation of the real world map [1]. To construct a consistent map, it is necessary to have integration of the observations from different views of the maze and data association will be required. Employing motion estimation techniques will also aid in reducing errors caused by noise in data from the robot's sensors.

The other subproblem is the robotic path planning problem. The objective in this part is to provide the optimized routing choice according to the knowledge of the map. In this process the robot needs to search an applicable path to navigate the maze. There are two strategies for the task at hand. One is to build the map first, then search for the path having the complete information of the map. This can provide the optimal solution in terms of distance, but requires the robot to run through the entire maze to complete the task. Another way is trying to find the path while the robot is exploring the maze. In this case the problem will be an on-line path planning problem. The latter method was implemented in this project, but there was not time to implement the former - making a complete map of the maze and then finding an optimal route between the initial and goal positions. Although, given the current setup, the project is fully able to implement this complete mapping feature.

IV. Methods

The original main choices for a robot were between the Pioneer, iRobot, and Explorer; but it has been determined that the Explorer will suit this projects needs best considering the circumstances. The Pioneer may be much easier to program than the Explorer, however it was determined that the sheer size of the Pioneer would not fit the specifications of this project. The iRobot on the other hand is easier to program and would not present a size issue, however availability and time issues eliminated its candidacy for the project. Therefore it was initially determined that the Explorer robot, despite the difficulties that may arise in programming, would be the best fit to the project.

The first focus of our work was to familiarize ourselves with the Explorer robot, set up the build environment and implement and test simple motion and sensing. There were a number of difficulties however in getting a working build environment such as changed and undocumented access passwords to online data sources and dead links to important files. After creating a working build environment the problem was to test simple motion and sensing. Motion was already written into the Explorer Player driver, however there was no existing code to read values from the two infrared sensors mounted on the front of the robot. Reading this data would be essential to localization and decision planning for mapping the maze.

We were able to implement the necessary sensing functions in the Explorer Player driver and wrote code for the Roboaudiostix board which handles the sensor I/O to communicate the sensor data to the Player driver over the i2c bus. However in testing the sensing code we were receiving nonsense data from the sensors and expect there is some fault in the code running on the Roboaudiostix. Due to time constraints of the project and continuing difficulties in reading sensor data we made the decision to stop work on the Explorer sensing and use a fully functional and well documented robotic platform - the Pioneer robot.

The decision to move to the Pioneer platform changed many of the physical implementation details. It would not be possible to build a very large physical maze due to the size needed for each cell to fit the Pioneer plus some padding room to allow for minor drift and imprecision. The Pioneer does however have the advantage of having a 180 degree laser sensor which is very precise. This became essential in the localization and motion problem because as we would find out, its odometry is very poor. We also lost some of the direct hardware control we would have had with the Explorer in moving to the Pioneer which incorporates its own smoothing and trajectory to commands its Player driver receives. There was fortunately a much smaller learning curve to working with the Pioneer as we were already familiar with it from a previous assignment and because it is a popular commercial product with many resources for help and information.

V. Results

A. Simulation Results

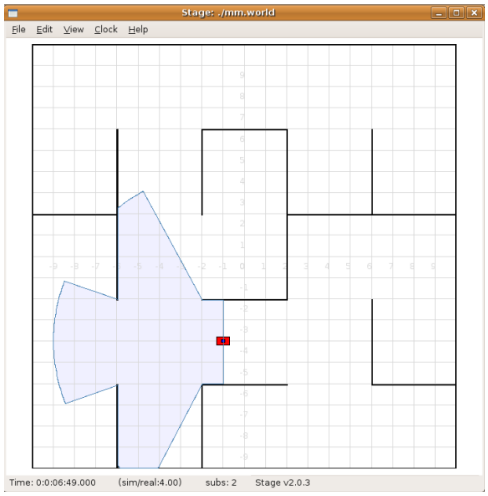

Fig. 1. Simulation Run

Localization in the simulation was done in a simple manner. As the robot was aware of its initial position and orientation, and motion primitives were in place to do simple rotations and single cell movement, all that is required for localization is to keep an updated (x,y) location and an orientation. Because of the motion primitives it was very easy to keep this position data up to date. The motion functions were assumed to correctly move the robot between cells without drift so that the calculated position at all times matched the robots' actual position. In the case of the simulation this assumption held true, and much work was done in the real world trials to keep this assumption true.

Since the maze is well defined and we have strong assumptions of the walls, it's less difficult to maintain a map than mapping in an unstructured environment. We use a matrix to record the map, and it is updated by the sensing function on each step. We didn't consider the case of inconsistencies in the map, which is a common issue in a dynamic environment - where the sensing data at different views could be in disagreement due to moving objects or the errors from the sensor. The robot always moves forward to one of the four directions, North, East, South, West, so we only need to be concerned about the walls on these directions.

We use the data from the laser sensor to detect the range of the front, right and left side walls. The average ranges are calculated to reduce the noise of the sensor. If the range is approximately half the size of one cell, a wall on that direction is declared. The threshold is chosen carefully, because if the robot is drifts far away from one side of a wall, the wall would be recorded incorrectly and searching may fail as well. One method of error correction that the sensing does is remapping a cell each time the robot enters it. This method can correct previous errors but also could introduce new ones. A voting or averaging algorithm could be useful in the case of multiple readings in disagreement, or the robot could simply re-sense a cell if it is noticed that a new reading is in conflict with a previous one.

Fig. 2. Incorrect Wall Sensing

A simple function for motion was used which took a North/South/East/West position as well as the robots current orientation and moved it to the desired position. In this way all of the other functions in the maze solving program could assume a maze of discrete segments and not have to account for the continuous world in which the robot operated. The goTo Player function was used in the simulation to accomplish the task of movement, and the motion function updated global position and orientation variables used by the search functions in the program. The advantage of the goTo function was that all the motion primitives involved in moving the robot were taken care of by this function. This greatly simplified one area of programming the robot in the simulation, but it was later found to not port to the Pioneer like we expected, which is discussed in greater detail further on in this report.

As a maze is discrete and its dimensions also well defined, the simplest virtual representation of it is a two dimensional array of individual cell structures. The program for this project was written in C++, an object oriented programming language, so we were able to take advantage of this and write classes for each cell as well as the maze itself, keeping all cell and maze mutators and other functions specific to each class. This provided for an environment which was much easier to envision, which in turn made the remainder of the programming a less difficult task. More specifically, this higher level abstract of the maze allowed for a much easier process of designing the search algorithm.

The main structures used in the program were a simple position structure, as well as a cell and maze class. The position structure simply contained x and y coordinate information for referencing a position in the maze. As well it also maintained an absolute orientation direction of North, South, East, or West; relating to the direction in which an instance of the position structure was pointing. Each cell class contained information on what the known status of its four walls were (unknown, not present, or present). The cell class also provided for retrieval and mutating of each of these values. Finally, the maze class contained all relevant information to the maze itself, including width, height, starting position, goal position, and the two dimensional array of cells - the virtual representation of the maze.

In each cell, the robot has to choose from the possible directions one that will lead it to the next state. The choice can be made based on priority. The priority is given by what is known as the search algorithm. The simplest search algorithm is the random search, in which the priority is given randomly. Though for certain situations this can be useful and even more successful than other search algorithms (there is always a chance that most optimum path is chosen randomly) there exists a high possibility of the robot getting caught up in infinite loops.

There are many existing algorithms to find the shortest path form the initial position to the final position, for example the Bellman flooding or the A* search. However they can be used only when the whole maze is known to the algorithm a prior to searching it, so they do not apply directly to this project.

The algorithm that we extensively used was a modified version of the online Depth First Search. It was a complete search and therefore it would find a goal if one existed or search the whole maze if it did not. It was also modified to be 'online' so that it avoided shifting search between the nodes at different branches of the trees. Also, when confronted between a choice of two directions for the next move moving between two cells it choses the one that is closest to the goal using the manhattan distance as a heuristic.

B. Experimental Results

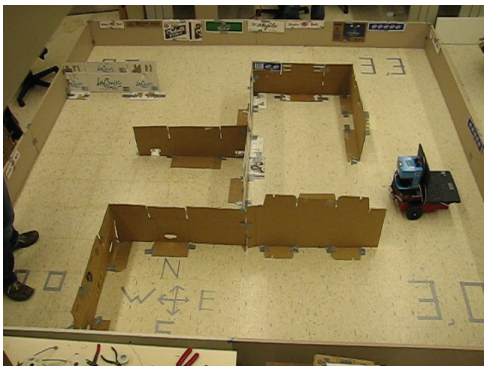

Fig. 3. Real World Run

In order to implement the project into the real world we had to build a maze of a size that was big enough for the robot as well as complex enough to demonstrate the maze solving capabilities. The maze was 12ft. x 12ft. with each cell having the 3ft x 3ft dimensions, which produced a 4x4 maze. It was built using the cardboard from boxes used for transporting goods, as well as some 6ft wooden boards running around the edge of the maze. All walls were approximately 1ft tall, which was enough for the SICK laser range finder on the Pioneer to detect the walls.

The main problem we discovered when trying to implement the project on the Pioneer was that the goTo Player function could not be implemented onto the Pioneer. This function highly simplified the simulation by allowing for telling the robot to go to given coordinates.

Since this did not work on the Pioneer, we had to instead manually design our own functions for the motion of the robot.

Since the maze cells are discrete in size and since there are only four possible directions the robot can face, this simplified the movement functions, but did not necessarily make it a simple task to program. As the robot will only be facing north, south, east, or west; a simple turn function to the left or right that rotates the robot 90 degrees provided for all the turns the robot made in the maze. Similarly, a forward motion function was designed that moved the robot forward one single cell length.

Unfortunately, on account of the inaccuracies of the odometry in the robot, simply relying on preset timed movements was not enough to accurately navigate the robot through the maze. As the odometry is inaccurate and unreliable, although the Pioneer thinks it is going straight or turning for a set number of degrees, in reality there is some error. This error accumulates and in the case of forward motion, the robot tends to drift to one direction or the other. In order to account for this error, we programmed the robot to adjust the angle of its forward motion depending on its relative position to a nearby wall. Essentially, it relies on the accurate and more reliable laser range finder to gauge distances which provides for more precise movement.

All of the motion corrections were implemented in the forward motion function. First, so that the robot accurately moves single cell distances, the robot uses its distance to the next cell in front of it to determine when to stop its forward motion. Basically the robot needs to stop when it is centered in the next cell, so when the range reading from directly ahead of it reaches half a cell (or a modulus of half a cell), it will know that it has traveled far enough forward. As it actually turns out, the SICK laser on the Pioneer is located slightly forward on the robot, so the values dictating when it must stop needed to be decreased a bit so that it moved slightly further forward. We have also included some other tweaks to forward motion such as slowing down after it has crossed a threshold so that it does not overshoot its destination.

The other correction implemented into the forward motion function was designed to account for the drift that accumulated on account of odometry error. The idea is simple - as the robot drifts to either the left or right, turn slightly to head back towards being aligned with the center of the cell. This was done by looking to the left and right of the robot with the laser and checking if the robot was less than half a cell away (including some marginal room for error). It can be noted that the robot is not always guaranteed to have a wall next to it, but more often than not there is at least one wall for some of its forward motion that it will be able to base this correction off of, so in the long run, it will help the robot stay more on track. As it turns out in the end these corrections did indeed help the robot more accurately navigate the maze.

We also found that the Pioneer seems to be a bit particular depending on the path it is taking through the maze. That is to say, it did not always accurately run through the maze if you simply arbitrarily place it at a different starting point with a different goal. This of course was not a problem in the simulation, so the theory behind the project was repeatable, but error involved in the odometry for the Pioneer did not always allow for repeatability. The settings for rotation and forward motion instead had to be tweaked slightly for the path it was currently trying to solve.

One interesting and beneficial consequence of the maze structure and backtracking that takes place in a search is it's ability to be somewhat self correcting for drift. As the robot explored one section of the maze and returned to the position it entered that section from, the drift accumulated exploring the section in one direction was minimized by drift which took place as the robot backtracked out of the section. We found that this is not a solution to the problem of accumulated drift, but that it did lesson the impact of cumulative drift in certain circumstances.

VI. Future Work

There are many possible directions from which future work could proceed from this project. While we were able to successfully solve a maze in simulation and in the real world, there are many improvements that could be made in order to correct for drift, be able to solve a larger or more complex maze, or utilize multiple robots to solve a maze in parallel.

Fig. 4. Drift Example

The drift accumulated throughout the experiments was by far the biggest factor keeping the robot from solving the maze in any given trial. One possible improvement to compensate for the drift could be an improved rotation primitive. Such a function could act similar to the previous one in that it will rotate as near to 90 degrees as possible, but would improve upon it by using laser

readings in order to position itself more precisely to 90 degrees. By making small rotations and observing the left and right wall distances the function could look for a local minimum in the wall distance on both sides, which should be the situation in which the robot is aligned perfectly parallel to its side walls. In this way very accurate 90 degree turns should be possible, even in the case where the robot has undershot or overshot forward motion, or has a slightly incorrect orientation prior to the rotation.

Another motion improvement could take place in the forward motion function. By determining the number of cells to either side of the robot using the laser, the forward movement could be compensated with a small angle to keep the distance on either side of ratios of cell sizes. We implemented a simple version of this which would apply compensation when the robot is following a wall to either side, but a version of the function as described above may have superior results. Given slow enough velocities such a method should be able to keep the robot nearly perfectly aligned in the center of a maze cell. Implementing this in addition to the improved turn function described above is expected to solve much of the drift error and localization issues which we struggled with.

Implementing better search algorithms could be a major advantage especially in situation where time is a major cost. The easiest way to increase the data available to the search algorithms can be done by scanning the walls of the cells that the robot may not have visited yet lie in the range scanning range of the SICK lasers. This can save us the time of having the robot explore some cells, and allows it to make better informed decisions during the course of the search. The more information that can be plotted by the robot, the more effective the searching can be done.

The search algorithm can also be enhanced by making it enough to detect and avoid getting into areas which can by guarantee not have the goal. One such situation is when the algorithm tries to get the robot to search into an area which is surrounded by all four sides by scanned cells and the goal lies outside it. By that we can know that there is no path that escapes out of this area to the goal and thus can be avoided. Given that the robot knows the dimensions of the maze, a segmentation algorithm could be very effective - avoiding many possible routes by attempting to segment off the goal location from the majority of the maze recursively.

Apart from that, though the use of SICK lasers has been highly beneficial, using a different and better-suited-for-the-task mechanical design than that of the Pioneer can definately be looked into. Research can also be done on using motors that would be have a higher accuracy and are still capable of high velocities. One of the possibilities would be using the Explorer instead of the Pioneer.

After implementing sensing on an Explorer, it could be an exciting platform for maze solving problems. Due to its size it would be simpler to implement and test large maze structures then with a Pioneer robot. Because of the smaller mass of the Pioneer, it could make a high velocity search of a maze structure more feasible without risking damage to the robot or the environment from accidental collisions. Due to the onboard wireless communication capabilities, the Explorer could be well suited for multi-agent maze search problems as well.

One very interesting prospect for research would be decreasing the cell size in the maze and relaxing some of the other experimental constraints until this problem is similar to common SLAM problems being looked at. Theoretically speaking, as the cell size reaches zero, the problem opens up into full-scale SLAM. The same or similar techniques as we used for the maze solving could then be applied to the SLAM problem to see what their effectiveness is versus standard SLAM techniques.

VII. Conclusions

From our research we have learned some key lessons. The transition from a simulated model to real world testing is not a trivial one, and exposes many factors which were not or could not be accounted for in the simulation. The hard constraints of the real world mean that what is theoretically possible and shown in a simulation may not be necessarily feasible to implement in practice.

The biggest challenge in our transition from simulation to the real world was of that of accurate movement and localization. Due to the constraints of the Pioneer robot, we solved the maze at a fairly low speed and did not do more advanced scanning of cells to look ahead multiple positions as was possible in the simulation.

The localization issue was the biggest factor we had to deal with once we moved to the physical world. If this problem were solved we could have much easier motion primitives and drift compensation, have more precise and useful sensing and solved the maze at a higher speed with essentially no risk of a collision. This project attempted to minimize the difficulties of localization by leveraging the discrete structure of the maze, something which did simplify the problem, but did not eliminate the problem of localization.

References

[1] S. Thrun, “Robotic mapping: A survey,” in *Exploring Artificial Intelligence in the New Millenium*, G. Lakemeyer and B. Nebel, Eds. Morgan Kaufmann, 2002, to appear.

[2] D. Murray and C. Jennings, "Stereo vision based mapping and navigation for mobile robots," in *Proceedings of the 1997 IEEE Conference on Robotics and Automation*, 1997, pp. 1694–1699.
[3] S. Thrun and A. Bucken, "Learning maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, pp. 21–71, 1998.
[4] S. Thrun, A. Bucken, W. Burgard, D. Fox, T. Frohlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt, *Map learning and high-speed navigation in RHINO*. Cambridge, MA, USA: MIT Press, 1998.
[5] J. del R. Millan, *The Handbook of Brain Theory and Neural Networks*, 2nd ed. MIT Press, 2002.
[6] H. Durrant-Whyte and T. Bailey, "Simultaneous localisation and mapping (slam): Part i the essential algorithms," *Robotics and Automation Magazine*, June 2006.
[7] M. Dissanayake, P.Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localisation and map building (slam) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, June 2001.
[8] C. S. Ananian and G. Humphreys, "Theseus: A maze-solving robot," May 1997, independent Work Presented to the Department of Electrical Engineering at Princeton University.
[9] (2008, December) Maze solving. Online. [Online]. Available: http://www.lboro.ac.uk/departments/el/robotics/Maze%5FSolver.html