Mobile Robot Path Planning and Obstacle Avoidance

MEM380 – Robotics I, Final Project

A. Alspach, S. Mason

ABSTRACT

Given only a map in advance with no obstacles but static walls, an objective of defining optimal paths to multiple goals while avoiding unforeseen obstacles and changing walls was to be met. Techniques using the laser range finder and Inertial Navigation Sensor for localization were employed to map the static walls and this data was descretized into an occupancy matrix. A* was used to plan the shortest path from one point to the next. While the path is being travelled, the laser range finder is used to detect any obstacles not already accounted for in the occupancy matrix and A* is re-planned if these obstacles interfere with the robot's current trajectory. This method works well but due to re-planning via A* often, takes time to perform effectively. Future work will include a dynamic method of path re-planning more suitable for 'on-the-fly' navigation.

Table of Contents

Introduction:

Path planning for mobile robots is an old problem that is still being looked at today. There are many methods to approach this problem such as bug algorithms, Breadth First Search, Depth First Search, A Star (A*), Dikstra,'s Algorithm, D Star (D*), Potential Fields and others. As in all engineering problems, the solution is dictated by both the constraints on the problem and what is trying to be optimized. One can optimize speed, distance, battery life, or any other variable.

One outlet for testing path-planning code is USARSim, which contains both the physics and variable error associated with mobile robots. Through software like USARSim programmers are able to develop and test path-planning techniques in a controlled environment. In actuality, there are far more problems with sensors and communications than can be simulated.

Problem Statement:

Given a global map, the goal of this project is program a differential drive robot to travel to a selected list of goal locations. Along the way, there will be obstacles whose location and sizes are unknown; however, general obstacle geometry is known. In addition to unknown obstacles, there are doors that may open and close on a set frequency or stay open entirely. USARSim is use to simulate this situation, utilizing the Unreal Tournament 2004 Physics and Environment Platform, which is controlled through a MATLAB interface. The robot used has access to a Inertial Navigation System (INS), Global Positioning Satellites (GPS), wheel encoders, ultrasonic sensors, laser range finder, and contact sensors. Success is measured by not contacting walls, doors, or obstacles, traversing to all of the given goal locations, using fewer sensors, and completing the travel under a 10 min time stipulation.

Methodology:

Given the problem and its constraints, the INS, and laser range finder were the sensors of choice. The INS error was eliminated (through editing of a USAR configuration file) to give a perfect representation of our robot's location and orientation (pose) in the USAR map space. Within our navigation controller, the INS served as global localization and the data obtained was compared to planned positions and orientations to determine an optimized path.

Localization and Mapping: Global knowledge was necessary to intelligently plan a route minimizing distance traveled between start and goal positions. A function 'laserPlot' was developed using the laser range finder and INS to see walls within the map and plot their positions (as 180 points per scan) relative to the robot's pose. Inputs to this function are the robot name, left and right wheel drive commands, and the number of iterations at specified drive command. To relate velocity and angular velocity to our left wheel speed, right wheel speed, wheel radius robot length the following equations were used.

$$\dot{\phi}_L = \frac{1}{r}(v_f - \frac{l}{2}\dot{\theta})$$

$$\dot{\phi}_R = \frac{1}{r}(v_f + \frac{l}{2}\dot{\theta})$$

Let: r – wheel radius
l – axle length
$\dot{\phi}_R$ – right wheel speed
$\dot{\phi}_L$ – left wheel speed

The function output (MAPdata.txt) is a text file list of points (in USAR map coordinates) of obstacle perimeters.

A second function ‘desMap’ was developed to utilize this data to descretize the map into an occupancy matrix of a given resolution (e.g. a resolution of 0.25 would yield 16 matrix cells per square USAR meter). The inputs for ‘desMap’ are the list of obstacle points and a resolution. This function outputs an occupancy matrix with ones in the places of obstacles and zeroes covering the free space.

The last step of map preparation is obstacle padding. A function ‘PadMap’ was written to add a ring of ones around any matrix cell that equals one in the occupancy matrix. The inputs to this function are the occupancy matrix and number of rings of padding. The effect of this function is rings of ones around all of the walls and/or obstacles in the occupancy map. This padded map is output as a secondary occupancy matrix with ones in place of obstacles and in place of the newly added padding. The padded map will be used for path planning to keep the robot at least one-half its width from obstacles.

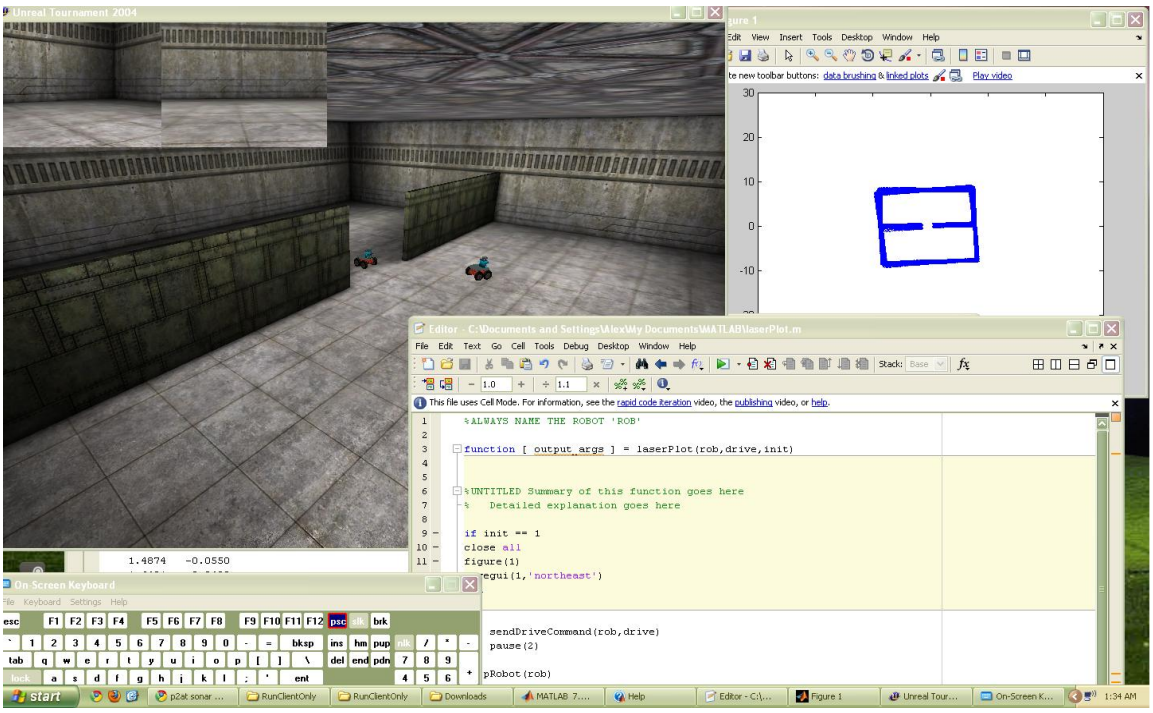

Figure 1: Screen shot of laser mapping being performed on map with resulting plot output.

Path Planning: A* is used on the descretized, padded map to find the shortest path between the start position and the goal position. This path is output by the A* function as a list of matrix cell coordinates which are the converted to map coordinates. This list of map coordinates is sent to the robot controller as a series of start and goal points. The start position is determined by the robots current

position (INS) and the goal is pulled from the list starting at the second point. The proportional controller (velocity ~ distance from goal, angular velocity ~ difference between robot orientation and angle of vector to goal) brings the robot to within a radius (<0.4m) of its goal then sets the start position to the INS reading and the goal to the next goal on the A* map path points list. The controller then brings the robot to the next goal. This list is iterated through until the last goal position is reached. The robot never stops during transitions because the last drive command is maintained until the next drive command is calculated and sent to the robot. As a list of goals are provided for the robot to reach during a specified amount of time, when the first goal is hit, the robot stops, uses A* to plan its trajectory to the next goal and repeats the same process specified above.

As the robot travels from its start to its goal position, it is constantly checking, using the laser range finder, for elements within the free space not represented in the padded occupancy matrix. If those elements it finds are not adjacent to an obstacle it is aware of, if will add it to the map. This check is performed as a filter to avoid mapping random elements round near walls and wall vertices. As these points are found, they are padded within the occupancy matrix to be avoided in the A* planning to the next goal. Also, in the chance that an obstacle is encountered in the way of an A* path, its position will be promptly added to the occupancy matrix, padded, and an A* path will be re-planned with this obstacle taken into account. These obstacles are realized when the position of obstacle being added to the occupancy matrix is within 0.5m of an A* planned matrix path point.

Velocity and Angular Velocity Control: Using the list generated from A* the robot takes each point in the list and uses it as a sub goal along the way. The robot travels from its current position, given by the INS sensor, to the next point in the A* path using a proportional control method. Because proportional control methods move slower when goal positions and orientations are closer to the current position and orientation, a scale factor was multiplied by the calculated wheel speeds to create faster movement.

Moving Doors: To handle the case of the moving doors first all possible door center locations were determined from our global map. This list of locations was then divided into two separate lists of Horizontal Doors and Vertical Doors. This helped determine how each door would be approached. A buffer zone surrounding the doors was made using a length and depth value. When the robot entered the buffer zone it would execute the following command:

1. Wait for door to close
2. Wait for door to open
3. Drive through the open door

Because certain doors would be left completely open, the robot also tracked time when waiting for the door to close. If the time exceeded the total frequency of an opening and closing cycle, the door position would be removed from the list and the robot would continue its path. Because the robot is equipped with a laser range finder, the reading perpendicular to the doors orientation would be used as the basis of determining whether the door was open or closed.

Simulation Results:

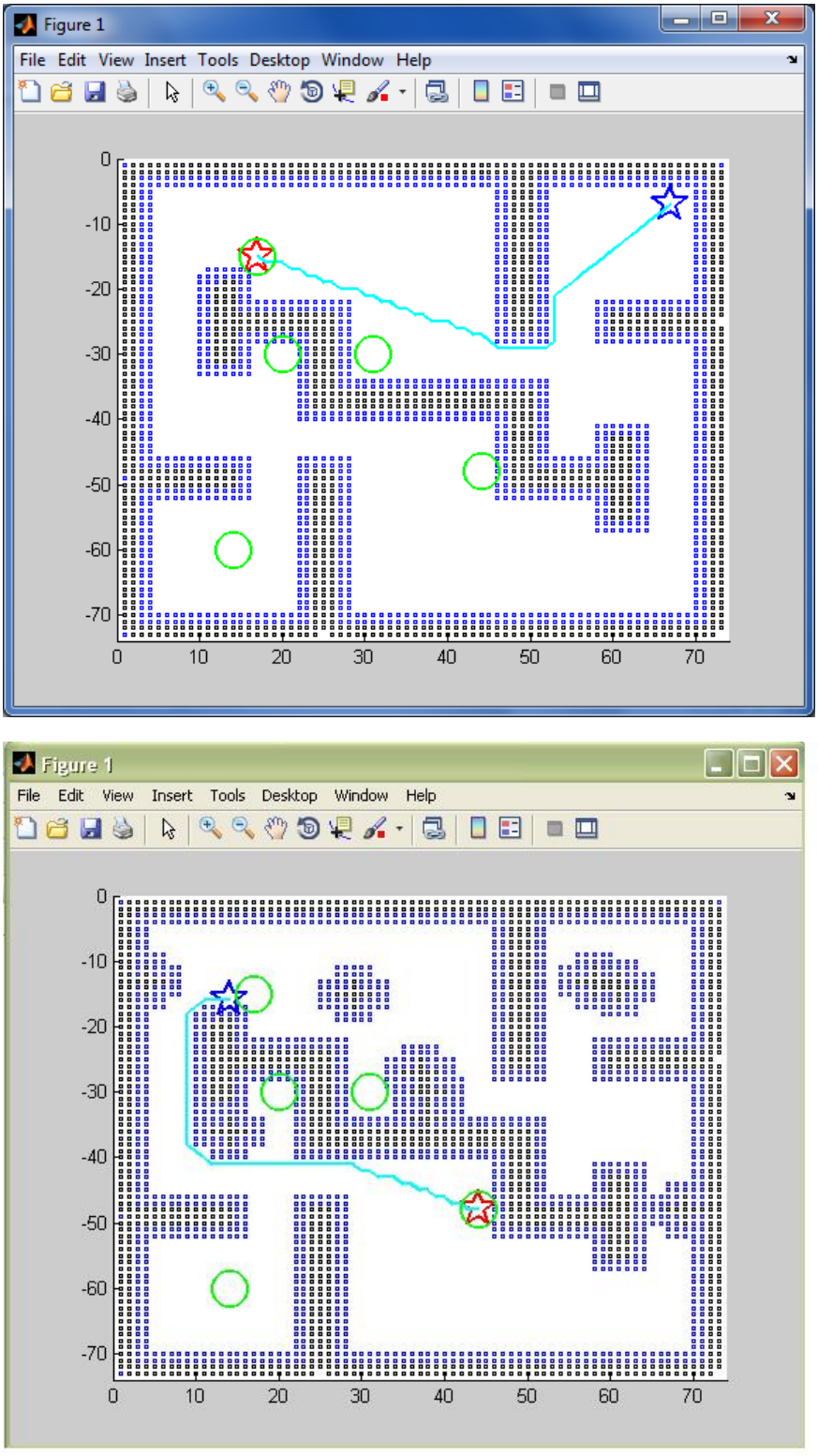

Figure 2: Screen shots of A Path Planning.*

The first screenshot of Figure 2 shows the visualization of the occupancy matrix when the robot first starts. The black squares represent obstacles, the blue squares represent padding, the green circles represent goals, and the blue line represents the calculated A* path. The second screen shot of Figure 2 shows the occupancy matrix at goal 2. It can be seen that new barrels and walls unknown by the robot have been added to the occupancy matrix.

Lessons Learned:

1. A matrix coordinate system is intuitively [row,column]. When translated into a Cartesian form when plotting, this becomes [columns, -rows] or [x,y] with the origin at the top left. When translating this to USAR coordinates, we must scale the values, account for the resolution of the matrix and any possibly odd orientation. If more visual methods (like sketching) of planning the translation functions from map to matrix, matrix to map, and their implementation in code were invoked, less discrepancy would have been realized when running the robot in the maps and plotting obstacles seen by the laser range finder.

2. Plotting data during each iteration has proven to very computationally intensive. When plotting for visual interpretation of what is going on in the program, that computation of the program slows down dramatically, greatly affecting the motion of the robot and its ability to move from goal to goal on the path.

3. We were forced to optimize our code to avoid large lag in our program between system input and output. Lag in the system causes discrepancies between what the program thinks the robot is doing and what it is actually doing.

4. Many iterations of the code and supporting functions were produced during the course of the project. It was found that posting separate folders of working code at each point in the process drastically helps organization. If mistakes are made, one can simply revert back to the last saved compilation.

5. Major inefficiencies in coding can be overcome by searching the MATLAB tool-box for pre-existing functions to replace ones hand written.

Conclusion:

While our mapping and descretization methods and A* path planning and travelling is comprehensive, not enough attention was given to consistent obstacle addition techniques. Poor methods of USAR coordinates and occupancy matrix coordinates led to many issues when attempting to add a new obstacle to the map for future A* planning. When running the code for the final

demonstrations the robot encountered an issue where an obstacle was mapped at [0,0] on the map despite no obstacle actually existing in that location. Also because of the way the code maps barrels, they show up offset on the map.

Future work include:

1. Research into methods of scanning our field of view more often to find obstacles earlier and refining map to matrix and matrix to map functions to allow for consistent syntax and therefore more productive debugging in the path planning algorithm.
2. Mapping barrels from their center location and not the first point that is seen.
3. Fixing glitches with door avoidance and mapping obstacles that do not exist.
4. Re-structure functions for consistency with how coordinates are referenced.