Labyrinth

Slam with Dstar Algorithm for solving maze

Jake Krajewski
The Kate Gleason College of Engineering department
RIT
Rochester, USA
jbk1514@rit.edu

Sarah Kordiyak
The Kate Gleason College of Engineering department
RIT
Rochester, USA
sjk1810@rit.edu

Janardhan Chaudhari
The Kate Gleason College of Engineering department
RIT
Rochester, USA
jc6073@rit.edu

***Abstract*—The plan of this project is to program the AmigoBot to navigate and map a maze. SLAM will be used to build the map of the maze as the robot moves around and explores the maze, using an Xtion camera to map the walls. D* will be used as the main path-planner to determine the best route to take. As the robot detects walls via its fake lidar, it will update its map and use D* to recalculate its course.**

Keywords : amigobot, asus xtion pro live, costmap, d* algorithm, gmapping, slam, path-planning, ROS, Rviz.

I. Introduction

Mobile robots have a number of applications, but function best in known environments. The premise of this project is to place a mobile robot in an unknown maze and task it with finding the most efficient way out. The maze will have multiple paths that the robot can take to get to the exit, and it will be expected to find the shortest path.

The proposed solution is to place an Amigobot in a maze with two solutions - one long, twisting path, and a shorter path with fewer turns. The maze walls will be made from cardboard and will not be one solid piece. This is done for easy storage as well as to be able to create new maze arrangements. The robot will first map the maze by exploring it and creating an internal map using simultaneous localization and mapping (SLAM). This will be accomplished with an ASUS Xtion camera mounted on top of the robot as its primary sensor. The robot will wander the entire maze, mapping all of it. From here, the maze will be solved “offline” by taking the created map and feeding it into a program containing a D* path-planning algorithm. In the meantime, the robot will be returned to the starting position of the maze. The path determined by the program will then be uploaded to the Amigobot, which will follow the path out of the maze.

In the past there have been quite a few projects similar to this one. Robots have been tasked with mapping areas and finding their ways through unknown locations. This has been accomplished through various means and algorithms, though A* is the most prominent and popular. A* does have its uses, but D* is more complex and offers higher utility. Additionally, many times the maze is given to the robot, rather than expecting it to explore and map the maze on its own.

These solutions have been shown to be reliable as a way to map unknown areas and find the shortest path out of the unknown area once it is mapped. Some of the projects found discuss the creation of maps using SLAM, while others detail the various possible implementations and uses of the D* algorithm. These papers prove that reliable map creation is possible using SLAM and that D* can be used to find the most efficient path through a known area. Thus, it can be concluded that combining both of these methods will allow for a robot to discover and navigate an unknown area.

The expected results of this solution are that the robot, when placed at a starting location, will drive around, avoid bumping into the walls, and explore the maze until the entire maze is mapped using its Xtion sensor. Once the maze is mapped, the D* program will

use this data to calculate the shortest path, which will be fed to the robot to follow. Using this data, the robot will navigate the shortest path out of the maze.

II. Literature Survey

There are many papers that discuss different ways to go about path planning. They discuss using different mapping methods, different path planning algorithms, and other such things. One such paper discusses how to use a wireless navigation mobile robot system with path planning and trajectory execution. It uses devices such as a visual sensor and a ZigBee wireless communication device. The camera was used within the maze to determine its position within the maze using color markers. The paper discussed using Breadth First Search (BFS) and Depth First Search (DFS) to determine which algorithm was the best to use [1]. It was determined that BFS was more efficient at path planning.

Another paper discusses using multiple robots to map out an unknown maze and then finding the shortest path between two points. The algorithm that the paper discusses using considers both the distance the robot goes as well as the number of turns the robot makes. It was then compared to other known path planning algorithms and results were determined to be closer to the ideal case[2].

There is another paper that discusses using a hierarchical approach for incrementally planning optimal paths. A* is used as the path planning algorithm in this paper and it is modified to handle a map with unknown information[3]. The path is calculated with using different approach strategies rather than just having the objective of reaching its location.

Another approach at path planning was to try to create a new shortest path planning algorithm using Lagrangian relaxation. The algorithm searches for the shortest path while given some constraints. The system does not narrow down the number of paths very well though so after it is run through the code the resulting paths are then run through an existing path planning algorithm to narrow down the number of paths even further[4].

Another paper discusses about using path planning to create a safe and steady path for a needle to be inserted into specific areas inside the body. The idea is to plan the shortest path that has the least number of rotations[5]. This paper mostly discusses how to calculate the forces caused by moving through tissue, but there are a lot of equations shown about how the paths were calculated. This paper concludes with discussing future plans because only simulations were done by the end of it.

Another paper about path planning discusses using D* as the algorithm. There were some issues with using D* though, whenever the robot got too close to an object the algorithm would get stuck deriving a detour path for the robot which is unnecessary and inconvenient for the robot[6]. The result of this caused the authors of the paper to replace the algorithm with another one.

There is a paper that discusses using A* and Dijikstra algorithms as a base and working with MATLAB to calculate path planning. The result of this was an improved A* algorithm where the path generation time is reduced significantly and the performance indexes are superior to the ordinary A* algorithm[7].

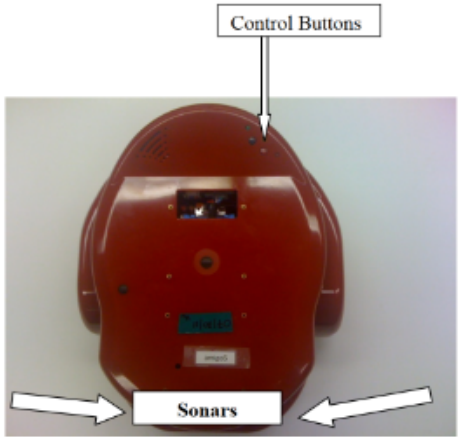
There is another paper that discusses how SLAM methods cannot be used directly to compute path planning for navigation, so it uses the Pose SLAM graph of constraints to calculate a path that is the most reliable to the goal. When this is used the navigation results were shown to improve. This method is used where the mapping of the environment is done while guided but the execution of the path is autonomous. It is stated that in the future the goal is to have it be fully autonomous[8].

Another paper discusses about optimal path planning for navigating a rectangular robot amongst obstacles and weighted regions. The authors use an algorithm based on a higher geometry maze routing algorithm. The algorithm shows to be very efficient when compared to other methods. The calculated paths can also instruct the robot to rotate to fit into narrower paths[9].

There is another paper that discusses modifying the D* algorithm for incremental path finding throughout a flight regime[10]. The concept is for situations in a 3D environment and for autonomous navigation. The paper does not take into account how wind may affect the flight though, it is a concept that may be worked upon to improve on the topic.

III. Implementation

An Amigobot has two different driving wheels and a balance caster as shown in the figure.1. Each drive axle is attached with a high-resolution optical quadrature shaft encoder which is used for sensing the environment, its position and speed for dead-reckoning task. The sonar sensors are mounted on the amigobot, sonar sensor has the range from 10 cm to 3 meters which can provide us the 360 degree sensing of the nearby environment.



• Technical Specs:
Max. Forward/Backward Speed: 1 m/s
Rotation speed: 100 o/ s
Sensors: 1 each side, 4 Forward, 2 rear- Ultrasonic
Encoders: 39000 ticks/wheel revolution
Microcontroller: 44 Mhz Renesas SH2-7144
Comm ports: 2 RS-232
Control Buttons: Red (Reset), Black (Motor test)

Figure.1: Amigobot

An additional sensor was added to amigobot for implementing SLAM is Asus Xtion pro live. The odometry can be obtained by using the xtion pro sensor which will produce 3D images and depth image information. The amigo bot is connected to the system wirelessly and on top the xtion pro is mounted in the middle of the amigobot. In the Rviz for the amigobot, to add the camera to simulation, executed the static tf transform node in the amigobot launch file. The Depth image received from the xtion sensor is collected. The openni 2 is the driver for using xtion sensor to connect it to the system.In figure 2 is the xtion sensor

Figure 2: Asus Xtion Pro live

Figure 3: Amigobot with Xtion

The Depth to laserscan application refers to the application that has access to every bit of the Asus xtion pro depth information, which occurs in a horizontal line of sight. It also exploits the use of laser scan data, which is later used within the application that is the main area for detecting and for the building of the grid map.

SLAM is simultaneous localization and mapping within the robot entails complex computational problems that are focused on dealing for building the map of an unknown environment. It also helps the robot to track its location in that environment. The 3D sensor which gives out the depth information can do slam. The Xtion sensor gives the depth image which is converting it in to laserscan data. laserscan data is usually of 12 format data in real time. This helped us in generating the local grid map with robot's pose in the map. The GMapping slam algorithm is used to solve the SLAM problem. The Gmapping algorithm uses the Rao-Blackwellized particle filter to sort the laser data. In this case we are converting the depth image data to laser data. Using this laser data it takes into account about the recent movements and the observation made by the amigobot. By filtering it the algorithm decreases its chances for uncertainty in the pose and prediction for the robot. Gmapping uses the odometry of the robot to keep the measurements.

Figure 4: Intial GMapping output

After the map was created using GMapping slam algorithm. The Graph representation of the environment which is the collection of nodes and edge weights. This creates a grid for each cell. such grid is known as occupancy grid map. The Gmapping slam algorithm is implemented in Robot operating system(ROS).

There are various ways in which we can map robot trajectories, path planning and path following. The path planning algorithm is used for finding the path for final position. When the robot has all information related to its surroundings and potential obstacles present in the environment. The path planning algorithm using that data it creates the shortest path and obstacle free path for the robot to reach its goal position. The each point is published to the robot to reach its desire goal position. The path will be planned through localization point.To search the shortest path the graph uses the criterion as sum of all edge weights from robot current position to goal position. It calculates the euclidean distance between the two cells, it finds the cost. The path planning algorithm which we are using here is D* path planning algorithm. The D* algorithm is capable of the fast replanning in dynamic environment. It is also well known as Dynamic version of Dijkstra's algorithm or dynamic version of the A* algorithm without heuristic function. The D* algorithm finds the shortest path in graphs in which the weights changes due to time, the occupancy value become higher or lower due obstacles.

The D* algorithm is divided in two parts firstly initial planning and second is replanning. Initial planning is the phase were the robot is not moving and still at the start position and replanning is when the robot detects the node changed with occupancy value due its motion.

Below given figure.4 explains the Dstar algorithm. The Dstar algorithm is implement in MATLAB.

D-STAR

1 **for** $(s_i, s_j) \in I$
2 **for** $\langle s_k, s_\ell \rangle \in I_d - \langle s_i, s_j \rangle$
3 Perform a pairwise sequence comparison to find all positions in s_i which has a neighbor of distance $2d$ in s_k. Let the positions be $P_1 = \{u_1, u_2, ..u_g\}$.
4 Do the same for s_j and s_ℓ and get the list of positions in s_j which is $P_2 = \{v_1, v_2, ..v_h\}$.
5 **if** $P_1 \neq \emptyset$
6 **for all** $u \in P_1$ add s_k into $\mathcal{S}'_{2d}(s_i[u])$
7 **if** $P_2 \neq \emptyset$
8 **for all** $v \in P_2$ add s_ℓ into $\mathcal{S}'_{2d}(s_j[v])$
9 **for** $(u, v) \in P_1 \times P_2$,
10 Add $\langle s_k, s_\ell \rangle$ into $I_{(s_i[u], s_j[v])}$.
11 **for** (u, v) whose $|\mathcal{S}'_{2d}(s_i[u])|, |\mathcal{S}'_{2d}(s_j[v])| \geq k_n$ and $|I_{(s_i[u], s_j[v])}| \geq k_i$.
12 Compute $\mathcal{S}_{2d}(s_i[u])$, $\mathcal{S}_{2d}(s_j[v])$, and $\chi(\mathcal{S}_{2d}(s_i[u]), \mathcal{S}_{2d}(s_j[v]))$
13 Put the (l, d)-star $(\mathcal{S}'_{2d}(s_i[u]), \mathcal{S}'_{2d}(s_j[v]))$ into the sorted list L.

Figure 5: D* algorithm

The amigobot with xtion pro sensor is made to solve the maze, using gmapping slam algorithm to map the maze and then to find the shortest path to reach its goal position the D* algorithm is used.

IV. RESULTS

The gmapping of the maze was mostly effective, but did have a few small issues. There were times where a sensor would misread its surroundings and remove obstacles it had previously seen. There were also instances of the sensors not reading the entirety of an obstacle, meaning it would map small gaps in solid walls.

Figure 6: Map Saved from Robot Gmapping

These less-than-ideal maps would, when run through the D* Matlab script, create erroneous paths - paths that would be the shortest to the goal, but would accomplish this by cutting through the holes recorded in the walls. When a map was built correctly, however, the script

would work exactly as intended. The Xtion also has an effective minimum range of around 40 cm - this means it cannot map obstacles closer than 40 cm to the sensor. This also means that it is very important to set the maximum obstacle detection range below the maximum range of the Xtion - if the Xtion is too close to an obstacle, it publishes points at its maximum range. This will erase any obstacle that it previously saw, but is now too close to to see. It would find the path of the lowest cost starting from the goal and working back to the starting point. This aligns with the article from *IEEE/ASME Transactions on Mechatronics*, as the article also used weighted regions to determine optimal paths with reasonable success[9].

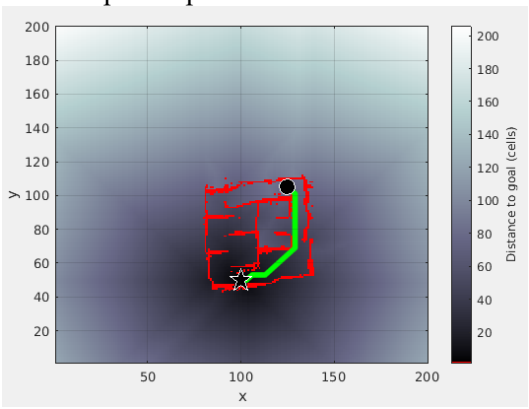


Figure 7: D* Path Planner Results

The script assumes that the amigobot is starting in the center of the map, then feeds the robot the points of the path one point at a time as move_base goal points. Although move_base is technically an A* algorithm, the points of the path are generated using D* via the Matlab script. This is a similar approach to the approach detailed in the Peng, Huang, and Luo paper, as they too used a Matlab-enhanced A* algorithm for mobile robot navigation [7]. Our script, however, went slightly above a few minor improvements to the A* formula. With the first point of the path set as the current move_base goal, the script then loops until the robot's current position matches that of the goal point before feeding it the next point on the path. The move_base navigation, however, is very slow. The robot takes a very long time to move to each point it is given, but it does eventually reach its goal. It is also worth noting that in creating a D* occupancy grid, the script alters the coordinates from the coordinates used by the robot - in a 200x200 pixel map, the robot considers the center to be (0,0). The occupancy grid, however, sees this point as (100,100). Additionally, the grid generator flips all y-axis points; (X,200) becomes (X,1) and vise-versa, (X,199) becomes (X,2), etc. The script accounts for this when publishing the points, but requires the start and finish points to be given in this coordinate system.

V. Future Works

There are several points that could be improved upon in this project. A proper 360-degree lidar sensor with a smaller minimum range would allow for much more reliable mapping in the confined space of the maze. A more reliable SLAM program would also help avoid the broken map issue. Furthermore, using a library other than move_base to move the robot along the D*-generated path might allow the robot to follow the path faster. The D* script could use some tweaks as well - currently, it's necessary for the user to alter the start and goal of the robot by hand and to account for the fact that the script alters the coordinates of the map when it creates the occupancy grid. Additional tinkering with the toolbox used and the script would help to mitigate this strain on the user.

Acknowledgment

Thank you to Dr. Sahin for support this project, which has been successful.

VI. References

[1] Jose, S., & Antony, A. (2016). Mobile robot remote path planning and motion control in a maze environment. *2016 IEEE International Conference on Engineering and Technology (ICETECH)*. doi:10.1109/icetech.2016.7569242

[2] Rahnama, B., Ozdemir, M. C., Kiran, Y., & Elci, A. (2013). Design and Implementation of a Novel Weighted Shortest Path Algorithm for Maze Solving Robots. *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*. doi:10.1109/compsacw.2013.49

[3] Lai, X., Ge, S. S., & Mamun, A. A. (2007). Hierarchical Incremental Path Planning and Situation-Dependent Optimized Dynamic Motion Planning Considering Accelerations. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),37*(6), 1541-1554. doi:10.1109/tsmcb.2007.906577

[4] Feng, G., & Korkmaz, T. (2015). Finding Multi-Constrained Multiple Shortest Paths. *IEEE Transactions on Computers,64*(9), 2559-2572. doi:10.1109/tc.2014.2366762

[5] Wang, J., Li, X., Zheng, J., & Sun, D. (2014). Dynamic Path Planning for Inserting a Steerable Needle Into a Soft Tissue. *IEEE/ASME Transactions on Mechatronics,19*(2), 549-558. doi:10.1109/tmech.2013.2250297

[6] Hsu, C., Chen, Y., Lu, M., & Li, S. (2012). Optimal path planning incorporating global and local search for mobile robots. *The 1st IEEE Global Conference on Consumer Electronics 2012*. doi:10.1109/gcce.2012.6379947

[7] Peng, J., Huang, Y. and Luo, G. (2015). Robot Path Planning Based on Improved A* Algorithm. *Cybernetics and Information Technologies*, 15(2), pp.171-180.

[8] Valencia, R., Morta, M., Andrade-Cetto, J., & Porta, J. M. (2013). Planning Reliable Paths With Pose SLAM. *IEEE Transactions on Robotics,29*(4), 1050-1059. doi:10.1109/tro.2013.2257577

[9] *Optimal Path Planning for Mobile Robot Navigation - IEEE Journals & Magazine*. [online] Available at: https://ieeexplore.ieee.org/document/4598863/ [Accessed 1 May 2019].

[10] *Three dimensional D* algorithm for incremental path planning in uncooperative environment - IEEE Conference Publication*. [online] Available at: https://ieeexplore.ieee.org/document/7566733/ [Accessed 1 May 2019].

[11] The D-STAR algorithm. (2009, April 1). Retrieved from *https://www.researchgate.net/figure/The-D-STAR-algorithm_fig1_6689399*

[12] Lab Manual 1