Maze Escaper

Ganesh Rohit Nirogi

Abstract–Autonomous movement within an unknown area and complex environment like mazes requires the robots to carry out investigations and Maze solving plays an imperative role in the area of robotics and in the development of optimal algorithms for autonomous robots to explore and navigate. It can be challenging for navigating multiple robots deployed in an unknown environment filled with obstacles .This project aims at finding the shortest path for each autonomous robot to reach its destination in an environment, for the robots to move towards their goal. These robots have to move safely ensuring collision freeness and liveness. We sense the nearby surroundings and apply A* search to find an optimal path to the goal state of the robot.

I. INTRODUCTION

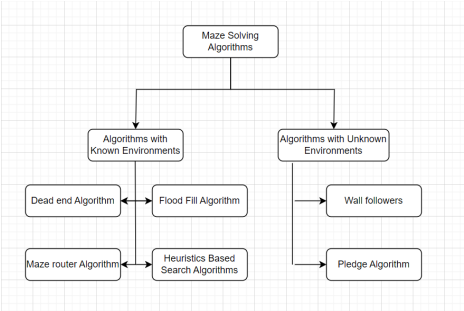
Autonomous robots navigation play a crucial role in diverse applications, including warehouse robots, self-driving vehicles and terrain exploration. The key mission of an autonomous maze solving robot is to reach a target location by navigating through a maze area. Autonomous robots have a tremendous role and they can be the best option for various and specific tasks. It is important for the robots to corroborate in order to complete complicated tasks in a complex environment. We incorporate the use of a camera based system in order to achieve this functionality. There are several steps involved in the making of this approach, first we capture images of the maze which includes the calibration and image processing. After that, we incorporate the use of graph theory algorithm such as the breadth first search, depth first search, best first search and A* in order to find the shortest path. Further this path will be converted from pixel coordinates stored in the form of occupancy grid which will be converted to real world coordinates by applying a camera calibration process. The constructed path is translated to a guiding path to be sent to the robot. The guiding path consists of a group of sequence instructions as translation distance in cm and movement directions.

II. RELATED WORK

The autonomous maze solving robotic systems can be categorized into two categories, camera-based systems and sensor-based systems. The image-processing-based solving robot systems require capturing the entire image of the maze area using a digital camera in order to analyze and process the maze's paths, determine the possible paths between the source and destination points, and identify shortest path possible. And sensor-based maze solving robotic systems can be classified into two sub-categories: rangefinder and line-follower. The former employs rangefinder technologies (such as infrared and ultrasonic sensors) to measure the distance between the solver robot and the facing wall, whereas the latter employs color (light intensity) sensors to follow the line drawn on the floor in order to explore the correct path to the destination points and for the algorithms used for the maze solving problems, they can also be categorized into two categories, known environment algorithms and unknown environment algorithms. The known environment algorithms have prior knowledge about the maze area while the unknown environment algorithms solve the maze with no prior knowledge of the maze area.

There are a number of algorithms that can be used based on the knowledge about the environment. A Considerable amount of research has been done in

both the fields. For algorithms with known environments.



- Dead end Algorithm: The dead-end algorithm fills/closes all dead ends, leaving just the right pathways empty. By using this technique, the solver robot first locates all of the maze's dead ends before filling in the route from each dead end to the first junction. This method examines the maze in an empty sequence, identifies the dead ends, and then blocks the concurrent routes.
- Maze routing Algorithm: The method for maze routing determines the best route between any two spots in the maze. Only known mazes with a recognized and stored maze area will this algorithm function. The maze routing method can determine the quickest route between the source and destination sites by investigating all potential routes.
- Flood-Fill Algorithm: Flood-fill algorithm assigns a distance value between any point (intersection) and the center-point (destination). The flood-fill algorithm floods the maze when the robot reaches a new node
- Heuristics search algorithm:The heuristic search algorithm is based on the concept of 'greedy best first' search, which is like the breadth-first search. The heuristic search algorithm explores multiple paths in parallel, and the best-first focuses on paths which are closest to the goal. The distance from the goal serves as a heuristic to direct the search. The major drawback for the heuristic algorithm is its space complexity, since it stores all generated nodes in memory.

For algorithms with unknown environment where there is no prior knowledge,

- Wall-follower algorithm: The most popular method for solving mazes is called the wall follower, and its primary principle is to follow walls inside the maze. The solver robot goes around the maze area while keeping an eye on the right or left wall until it discovers the exit. The wall follower has two potential rules: the left-hand rule and the right-hand rule. Depending on the rule selected, the turning precedence will either be to the left or the right.
- Pledge algorithm:The Pledge algorithm approach integrates the wall follower algorithm. When there are obstacles and the walls aren't connected, the Pledge algorithm is used to navigate the maze. The robot keeps the obstruction either to its left or right, and it maintains track of all turns using a counter, where a turn to the right increases the number and a turn to the left reduces it. The solver robot starts on its trajectory for the right-hand side algorithm until the counter reaches zero, indicating that it has passed through the obstruction.The solver robot always starts by turning to the left, then for each subsequent movement it applies the following logic: if there is no obstacle on the right, it moves right; if there is an obstacle on the right but no obstacle in front, it moves forward; and if there are obstacles on the right and in front, it turns to the left. This process will be carried out by the solver robot until the counter hits zero once more.

III. PROBLEM STATEMENT

Consider a 3-d environment $\boldsymbol{\mathcal{W}} \subseteq \mathrm{R}^3$ populated by n identical disc-shaped robots indexed $1, \ldots, n$. These robots has radius r and can only move on the ground. $\boldsymbol{\mathcal{W}}$ has a plain ground which any two points are of the same height.Give a 2-d representation of an $x - y$ coordinate system to

$\boldsymbol{W}$ to better represent this environment. Width is considered as length in x axis, while length is considered as height in y axis. A rectangle maze M with width w and length l is within this environment. The four vertices of this maze is represented as $[0, 0]$, $[w, 0]$, $[w, l]$, $[0, l]$ respectively. In the maze there are o obstacles. Robots are not allowed to go through any obstacle k $[ox_k, oy_k]$. The space in M that are not occupied by any obstacles form channels for robots to pass. The robots should always stay in the maze M.

We have n start positions respectively for n robots $[x_1, y_1]$, ..., $[x_n, y_n]$ and n target positions respectively for n robots $[a_1, b_1]$, ..., $[a_n, b_n]$. More precisely, robot i is first placed at $[x_i, y_i]$ where $0 < x_i < w \; and \; 0 < y_i < l$ and is within a channel path. The goal for robot i is to find itself to a target position $[a_i, b_i]$ where $0 < a_i < w \; and \; 0 < b_i < l$. $t \in \{0, ..., T\}$ where T is the time when the last robot gets to its target position. Let $[x_i(t), y_i(t)]$ indicates the position of robot i at time t. We assume that the position of obstacles does not prevent any robots from getting to its target position.

Our objective is to implement an algorithm that can control these n robots to go from the start position to its target position with the following equation satisfied.

- Robots are not allowed to step on any obstacles or go outside the maze M.

$\forall t \forall i \forall k$: $[ox_k, oy_k] \in O$, $x_i(t) \neq ox_k \; or \; y_i(t) \neq oy_k$

- All the robots should finally get to their target position.
  $\forall i \exists t$: $x_i(t) = a_i(t) \; and \; y_i(t) = b_i(t)$
- Robots should never collide.
  $\forall t \forall i, j \; i \neq j$: $|[x_i(t), y_i(t)] - [x_j(t), y_j(t)]| > 2r$

IV. Approach And Implementation

In this section we will introduce our approach to design and implement the navigating algorithm for the multi-robots system. Our approach relies on the assumption that the position of obstacles in the maze does not prevent any robots from getting to its target position. We will first introduce the method we used to get the 2D mathematical representation of the maze world. Then we will introduce how we apply the A-star algorithm to our approach. Finally we will introduce a control rule that controls the robots to follow the assigned path.

A. Build a Maze World

We use the Building editor to create the maze worlds according to our requirements. We can import a floor plan using a png or jpg or we can create walls manually such that the maze is a rectangular maze with obstacles in it in the form of walls. Further this floor plan is saved as a sdf file. Create similar robot model files for different maze worlds. Create world files and include the maze model created using the uri tag specifying its path. Export the models path by including it in the package.xml file.

B. Convert to 2D Mathematical Representation

We Use gazebo_ros_2Dmap_plugin to generate the overlook of a maze. gazebo_ros_2Dmap_plugin is a gazebo simulator plugin to automatically generate a 2D occupancy map from the simulated world at a given certain height. [1] After applying this plugin to the maze world we created, we will get an image of the maze world. The image will appear as shown in Figure 1. This image is not a well-formed one, so we need to use OpenCV to convert the overlook to a binary image where we can see it as a graph and apply the path-finding algorithm. OpenCV will do the work to binarize the image to pure black and white as shown in Figure 2. Now we can read this image file into python. In the python image representation, this image will be converted into a 2-d list, where each

element in this 2-d list is of integer type, either 0 or 1, respectively representing the color of the corresponding pixels. Then we can make use of this 0-1 2D list to represent which pixels are obstacles or are able to let robots to step on. In the next part, we will apply the A-start algorithm to this 2D list.

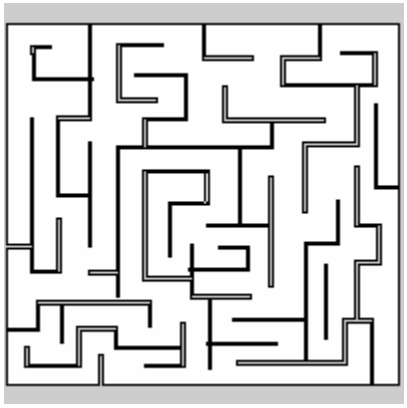


Figure 1. The image generated by gazebo_ros_2Dmap_plugin

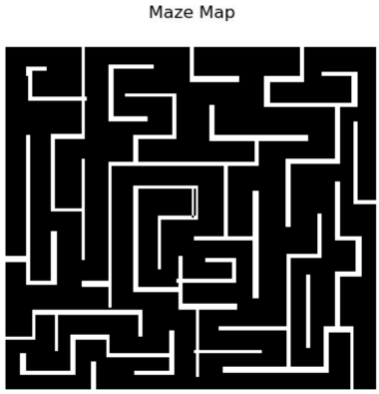


Figure 2. The image outputted after applying OpenCV

C. Compute Paths Using A-star Algorithm

The problem we want to solve in this subsection is to get the paths for each robot that will lead it to get to its target position. The input is a 2-d list whose elements are either 0 or 1, where 0 represents that the robots are able to step on it, while 1 represents the obstacles. The output is a set of paths, each path corresponds to a robot. All the paths in the set satisfies the condition that the start of the path is the start point of the corresponding robot, and the end of the path is the target point of the corresponding robot. Before we can apply the A-star algorithm to the 2-d list, we have to do some pre-work to the list. Since the robot's odometry is represented by the center of the robot, considering all the robots have a radius r, we need to bolden the obstacle representation in order not to get the robots to collide with the walls.Bolden element"1" by radius r can avoid collision with walls as shown in figure 3.

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & .. \\ 0 & 1 & 0 & 0 & 0 & .. \\ 0 & 1 & 0 & 0 & 0 & .. \\ 0 & 1 & 1 & 1 & 1 & .. \\ 0 & 0 & 0 & 0 & 0 & .. \\ 0 & 0 & 0 & 0 & 0 & .. \\ .. & .. & .. & .. & .. & .. \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & .. \\ 1 & 1 & 1 & 0 & 0 & .. \\ 1 & 1 & 1 & 1 & 1 & .. \\ 1 & 1 & 1 & 1 & 1 & .. \\ 1 & 1 & 1 & 1 & 1 & .. \\ 0 & 0 & 0 & 0 & 0 & .. \\ .. & .. & .. & .. & .. & .. \end{pmatrix}$$

Figure 3. Bolden "1" in the 2-d list

The process of implementing the A-start algorithm is as stated in the Pseudocode 1.

Algorithm 1: A-star algorithms

```
Result: Return the path from start position to target position
openList ← empty list of nodes;
closeList ← empty list of nodes;
openList.add(startNode);
while openList is not empty do
    cur←the node with the least f value;
    openList.remove(cur);
    closeList.add(cur);
    if currentNode is goal then
        end; backtrack to get path;
    end
    children of cur ← the adjacent nodes;
    while not till the end of cur.children do
        currentNode ← current child;
        if child is in the closedList then
            continue;
        end
        child.g = currentNode.g + distance between child and current;
        child.h = distance from child to end;
        child.f = child.g + child.h;
        if child.position is in the openList's nodes positions then
            if if the child.g is higher than the openList node's g then
                continue;
            end
        end
        openList.add(child);
    end
end
```

Pseudocode 1. A-star Algorithm

The output of each path is a list of points. The robots are expected to follow these points one by one and finally get to their target points.

D. Control Rules

Ideally, the robots will exactly follow the points given and get to their target points. However, the deviation of the movement of a robot prevents us from simply doing that. The rotation and the moving forward action will both cause some deviation to the robots' movement, which may lead to not getting the robot to the target point or even let the robot collide with the wall. Thus, we propose this control rule to control the robot only moving within a certain deviation range and can guarantee that the robot can finally get to the series of target points assigned to it.

Define a float threshold parameter **β**, indicating the maximum acceptable angle α between the orientation of the robot and the vector from the position of the robot to the target point. During the movement of the robots, we need to closely observe the value of α. See figure 4. The robots are controlled to continue to move forward only when $\alpha < \beta$. When $\alpha \geqq \beta$. we need to calculate the expected movement for the robot again to let it rotate towards the target position.

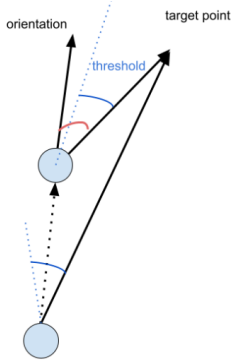


Figure 4. Control Rule

The pseudocode for the control rule is shown in pseudocode 2.

Algorithm 1: Control rule

```
rate ← rospy.Rate(FREQUENCY);
while rospy is not shut down do
    if α ≥ β then
        calculate rotation angle θ;
        moveForward(distance to target position);
        rotate(θ);
    else
        publish twist msg according to d;
    end
    rate.sleep();
end
```

Pseudocode 2. Control Rule

V. EXPERIMENT

A. Experiment Setup

First, a solvable maze in the gazebo world is created, we created two three worlds with different obstacles as shown in Figure 5.

- Empty World
- Simple Maze
- Complex Maze

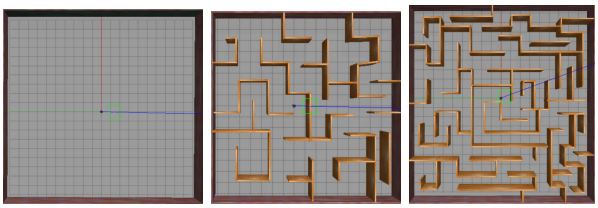


Figure 5. Empty World, Simple Maze, Complex Maze

Then we test the world with two robots. The start point of robot 1 is [9.3, 8], the end point of robot 1 is [-9.5, -7.5]. The start point of robot 2 is [-5, 5], the end point of robot 2 is [9.3, -8]. The two robots are expected to move from their start point to their assigned destination.

B. Evaluation

A full video of running test on the most complex maze can be found here https://drive.google.com/file/d/1WhypIJw_o-mRmAtYtCOOO0ZRL6OGOusc/view?resourcekey. Two robots start as Figure 6,

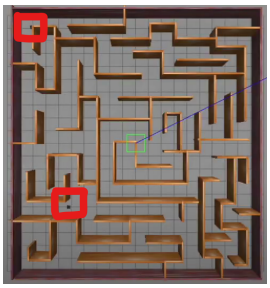


Figure 6. Robots are at their starting positions

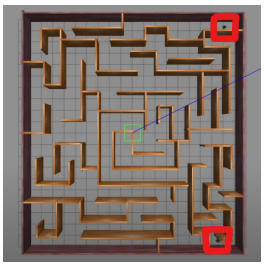


Figure 7. All the robots finally get to their destination.

The result should be that our approach finally leads all the robots to its destination under the assumption that the maze is solvable and there exists a path for each robot to get to its target point. However since the robot is small regarding the whole maze and we didn't assign a quick speed to the robots, two robots take a while to reach their destination. Also, the tuning of the threshold parameter **β** also affects the efficiency of the approach. **β** can not be assigned a too large value, otherwise it won't have a good effect on letting the robot to follow its target point. If we assign a really small value to **β**, then the robots do follow the assign target point, but the efficiency of the movement of robot will decrease. The robot needs to frequently adjust its orientation to stay within the threshold range. Thus the appropriate choice for **β** is another key point.

VI. Conclusion

The approach for solving the multi-robot maze was successfully implemented by using the combination of image processing and heuristics search algorithm to find the optimal path for the robots to take to reach its final position. It is observed that the robots will eventually reach their final destination and also avoid obstacles and other robots thereby achieve liveness and collision freeness. The approach also works efficiently for a higher number of robots in the environment.

For future work, we may consider finding the best range for the threshold parameter according to the shape or the complexity of the maze, and may find an approach to improve the approach's efficiency.

References

[1]GitHub-marinaKollmitz/gazebo_ros_2Dmap_plugin: Gazebo simulator plugin to automatically generate a 2D occupancy map from the simulated world at a given height.(n.d.).GitHub.https://github.com/marinaKollmitz/gazebo_ros_2Dmap_plugin

[2] Alamri, Shatha, et al. "Autonomous maze solving robotics: Algorithms and systems." International Journal of Mechanical Engineering and Robotics Research 10.12 (2021).

[3] Aqel, Mohammad OA, et al. "Intelligent maze solving robot based on image processing and graph theory algorithms." 2017 International Conference on Promising Electronic Technologies (ICPET). IEEE, 2017.

[4] Kumar, Rahul, et al. "Maze solving robot with automated obstacle avoidance." Procedia Computer Science 105 (2017): 57-61.

[5] Covaci, Rares, Gabriel Harja, and Ioan Nascu. "Autonomous Maze Solving Robot." 2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR). IEEE, 2020.

[6] Hidayatullah, Ahmad Syarif, Agung Nugroho Jati, and Casi Setianingsih. "Realization of depth first search algorithm on line maze solver robot." 2017 International Conference on Control, Electronics, Renewable Energy and Communications (ICCREC). IEEE, 2017.

[7] Del Rosario, Jay Robert B., et al. "Modelling and characterization of a maze-solving mobile robot using wall follower algorithm." Applied Mechanics and Materials. Vol. 446. Trans Tech Publications Ltd, 2014.