ARK TASK-2 REPORT

Abstract:

This report presents the implementation of the Probabilistic Roadmap (PRM) algorithm in Python for path planning in a 2D environment represented by an image map. The algorithm is applied to find a collision-free path between a start and endpoint, considering both "Easy" and "Hard" scenarios. The implementation involves identifying obstacles in the map, generating nodes in free space, connecting nodes to form a roadmap, and utilizing the A* algorithm to find one of the shortest paths in the roadmap. The program visualizes the algorithm's implementation on the original image map, showcasing the generated nodes, edges, and the shortest path found.

In the second part, the implementation is extended to a dynamic environment using the autonavsim2D library. A custom planner integrating PRM and A* algorithms is developed to function within the Pygame environment provided by autonavsim2D. The program allows users to define the robot's starting and goal configurations interactively and finds a collision-free path between them. Despite facing challenges in A* implementation for hard start points and limitations in robot path-following behavior, the project demonstrates a comprehensive understanding of path planning algorithms in complex environments.

INTRODUCTION AND PROBLEM STATEMENT:

TASK-2(PART 1)-

Develop a Python program that implements the PRM algorithm for this 2D image map maze.png representing the environment. You need to run your algorithm for both Start Easy and Start Hard

The program should:

1). Read the image map provided and identify obstacles. – Implement the PRM algorithm to find a path between the start point and the endpoint.

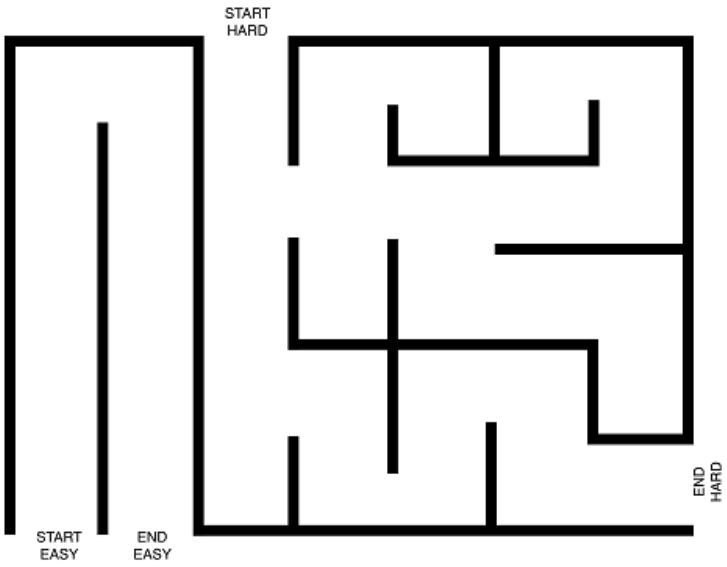
2). Visualize the implementation on the original image map.

PART2-

• Utilize the autonavsim2D library to create a more dynamic environment.

• Adapt the PRM implementation to function within the Pygame environment provided by autonavsim2D.

• The program should: – Allow the user to define the robot's starting and goal configurations within the Pygame environment. – Utilize the PRM algorithm to find a collision-free path between the start and endpoints.



This is the maze that was given as a problem and we need to implement the PRM algorithm in it and then find the shortest paths for both the easy and hard points.

2). MY APPROACH:

- Firstly I defined a function for identifying all the obstacles and the free space in the image.
- Then I defined a function for storing the valid indices for free space in the image given by considering some boundary conditions as well and then generated nodes at those places. No nodes to be generated can be decided by the user.
- Then I used kdtree to find indices and distances of the neighbor nodes of a particular node. In this function, I made use of another collision function to check that there is no collision between the current node and its neighbors. Take care that I used the the num_neighbors in the function by adding it by 1 as the first nearest neighbors will be the node itself.

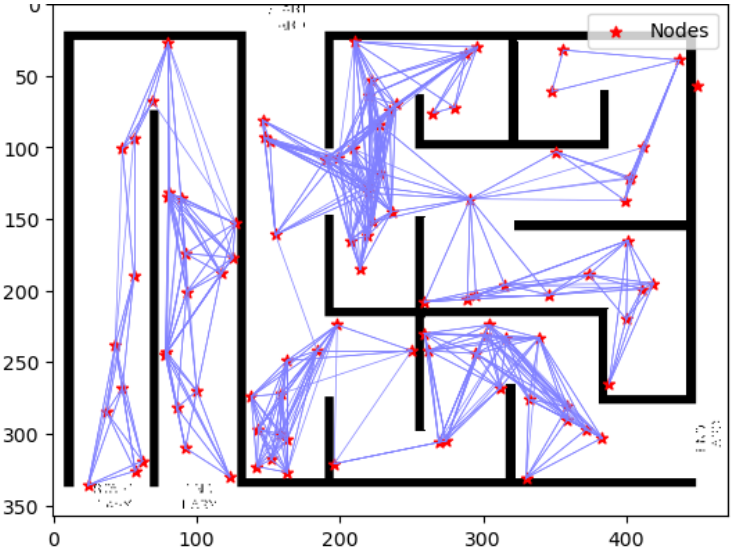
- In this code I defined the starting and the ending points by visualization and then found the nearest nodes to these points for planning the path.

A* IMPLEMENTATION FOR FINDING ONE OF THE SHORTEST PATHS FOUNDED BY PRM:

- I defined a **heuristic function** for calculating the distance between the two nodes.
- I have defined a **priority queue** that will give priority to the node with less priority value so I will decide the priority value based on the cost value at each of the nodes.

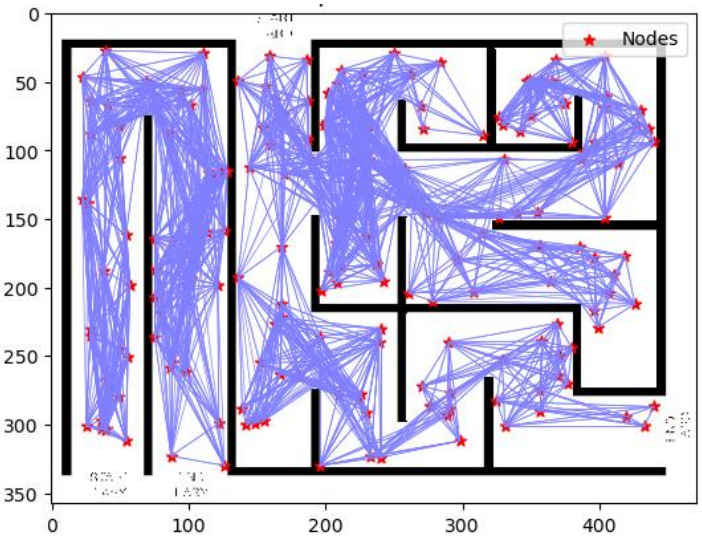
- In A* implementation I have used two dictionaries came_from(Used for storing the index of the parent node at each node) and cost_so_far(For storing the cost value at a particular node from its parent node).
- Then afterward I used loops to calculate the cost function at each of the nodes and set the priority of the nodes in the path based on the value of cost at each node.
- Further for visualization purposes I have displayed all the nodes and their feasible paths and afterward the shortest path implemented by A*.

CONNECTIONS OF THE NODE BY CHANGING THE NO OF NODES AND THEIR NEIGHBORS:



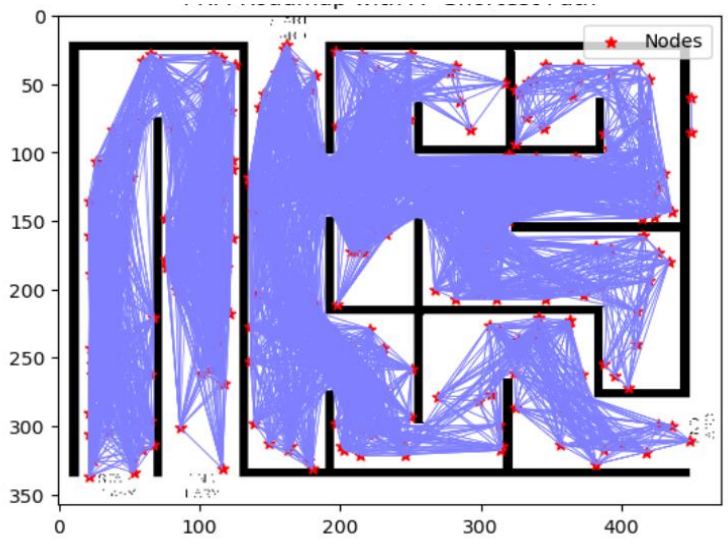
NUM_NODES=**100**

NUM OF MAX NEIGHBORS FOR A NODE =**20**



NUM_NODES=**200**

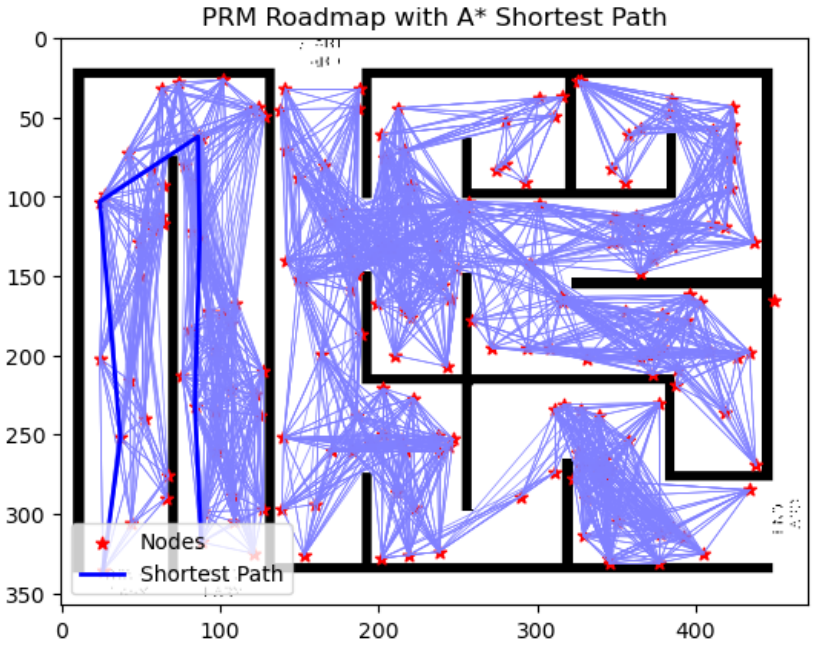
NUM OF MAX NEIGHBORS FOR A NODE =**50**



NUM_NODES=**300**

NUM OF MAX NEIGHBORS FOR A NODE =**100**

SHORTEST PATH FOUND FOR EASY START POINT BY A*:



NO_OF NODES=200

NUM OF MAX NEIGHBORS FOR A NODE =50

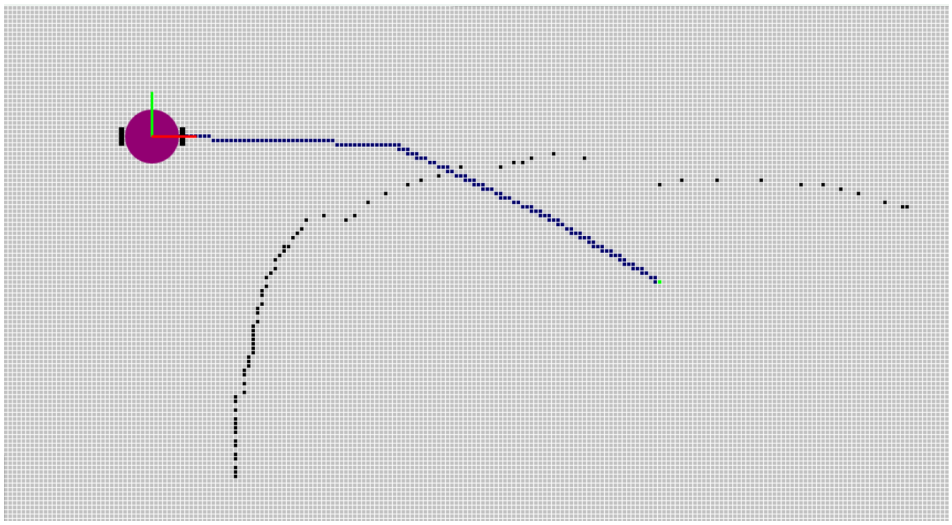
PART_02:

- Here in part 2 I have utilized the autonavsim2D library to create a more dynamic environment.

- I have defined my own custom planner which uses the PRM and the A* and it works in the same way as told above.
- I have used custom_planner as defined above, custom_motion_planner as default and window as (amr).

```
nav = AutoNavSim2D(
custom_planner=custom_planner,
custom_motion_planner='default',
window='amr'
)
nav.run()
```

ROBOT PLANNING THE PATH FROM START POSITIION TO THE END:



CONCLUSION AND DIFFICULTIES FACED:

Doing this was very challenging and interesting. I learned various new things such as PRM implementation then using A* to find the shortest paths.

I faced difficulty while implementing A* in the PRM as it wasn't easy for the algorithm to find the path every time for the hard start point.

One **limitation** is there in my algorithm in second part that my robot is not following the path generated instead it is always going to east(An infinite loop)

kindly restart the kernel and clear the output if the second part doesn't run.

REFERENCES:

1).https://www.youtube.com/watch?v=ujvgJzgZ9xg&t=462s

I used to this video to understand the concept of PRM and implement the code.

2). https://www.redblobgames.com/pathfinding/a-star/implementation.html

SUBMITTED BY- AYUSH GOEL

ROLL NO-23EX10008