Frontier-Based Exploration Algorithm using 2D LiDAR: Hybrid A* Path Planning & DWA Controller Approach

Ahmed Alghfeli Hassan Alhosani Reem Almheiri

I. Abstract

This paper presents a comprehensive system for autonomous navigation in unknown environments, integrating a frontier-based exploration strategy, a Hybrid A* planner, and a Dynamic Window Approach controller. The proposed system aims to efficiently explore and navigate from a start position to a goal position, while ensuring obstacle avoidance and optimal path planning. The frontier-based exploration strategy focuses on identifying frontiers at the boundary between free and unknown spaces, allowing the prioritization of exploration efforts in information-rich regions. The Hybrid A* planner combines grid-based and graph-based approaches, providing optimized paths that consider the Turtlebot's dynamic constraints. The Dynamic Window Approach controller dynamically adjusts velocity commands based on maximum velocity, goal direction, and obstacle avoidance, enabling safe movement. The system was implemented and tested in the Stonefish simulator, with results demonstrating robust performance in autonomous exploration and navigation tasks.

II. Introduction

In the field of robotics, effective path planning and exploration strategies are fundamental for autonomous robot navigation. This paper presents a comprehensive proposal for a hands-on project that aims to develop an autonomous robot capable of efficiently navigating an environment from a start position to a goal position. The proposed project involves the integration of three key components: a frontier-based exploration strategy, a Hybrid A* planner, and a Dynamic Window Approach (DWA) controller.

III. Methodology

In this paper, the methodology section presents the approach and techniques used for the development of the exploration algorithm. The algorithm employs a frontier-based exploration strategy integrated with the Hybrid A* path planning algorithm and DWA controller.

A. Frontier Detection and Selection

Autonomous exploration is a fundamental task for robots operating in an unknown environment. The ability to efficiently explore is the goal by which the *Frontier-based Algorithm* approach was selected. This algorithm introduces a methodology that leverages frontier detection and navigation to enable the Turtlebot to explore unmapped areas and create a comprehensive map.

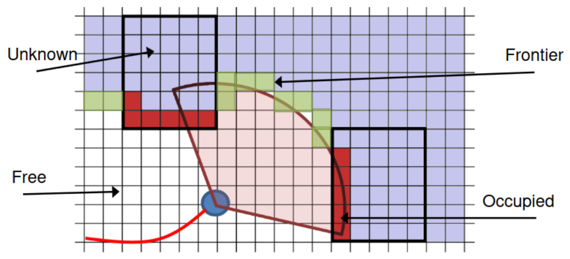

Fig. 1. Graphical Representation of Frontier-based Algorithm

The main objective of this method is to generate a nearly complete map within a reasonable timeframe. To achieve this, the concept of *frontiers* is introduced. Frontiers are regions located at the boundary between known free space and unknown areas. By focusing on frontiers, the Turtlebot can prioritize its exploration efforts on areas that provide the most information gain.

In the project application, a gridmap was utilized to represent the map. Each grid cell stores the probability of occupancy, which allows the Turtlebot to classify cells into either free, unknown, or occupied categories. Additionally, unknown spaces were assumed occupied for the purpose of safe navigation. As soon as frontiers are detected, they are then grouped into clusters and assigned a centroid.

The frontiers were detected using a brute-force method, wherein every pixel is identified as first being free and then checking if it is adjacent to an unknown cell. The pixel position is declared free if it has a value of 0, then, in the four directions being up, down, left, and right, the adjacent cell position is evaluated. If the cell has a value of -1 corresponding to unknown, then the free cell identified previously is added to the frontiers list.

After all the frontiers are detected, the next step is to cluster them. The clustering method selected was the Density-Based Spatial Clustering of Applications with Noise, or DBSCAN for short. DBSCAN is a clustering algorithm capable of identifying clusters of arbitrary shapes in spatial data without requiring prior knowledge of the number of clusters. It operates based on the concept of density, grouping data points that are closely located and have a sufficient number of neighbors within a specified radius. This makes it ideal in identifying the frontier clusters since they form adjacent neighbors.

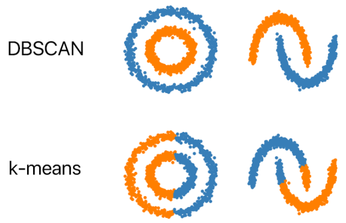

Fig. 2. Comparison between DBSCAN and K-Means

In the code, an instance of the DBSCAN algorithm is created using the DBSCAN function from the SciPy library. The epsilon parameter is set to 3, which determines the maximum distance between two samples for them to be considered neighbors. A labeled array is then created which assigns labels for each frontier cell based on its density and proximity to other cells. The label -1 indicates noise or outliers, and therefore, unique labels excluding -1 are extracted and the clusters are created.

When the clusters are ready, centroids are then assigned to them. This is done by averaging the coordinates of the points in each cluster. Then, the point closest to the calculated centroid is selected as the representative centroid for each cluster. Finally, the centroid is selected according to two criteria: information gain and Euclidean distance. The information gain gives less cost to larger clusters, while Euclidean distance gives less cost to closer centroids. The joint of the two costs determines the next point the Turtlebot visits.

B. *Hybrid A* Algorithm for Path Planning*

Efficiently planning paths in complex environments, particularly when prioritizing drivable solutions, is a critical requirement. Such is the reason for selecting the planning approach called *Hybrid A**, which combines the strengths of both grid-based and graph-based approaches. By integrating the discrete nature of grid-based methods with the continuous search space of graph-based methods, Hybrid A* offers a versatile solution for path planning in diverse scenarios.

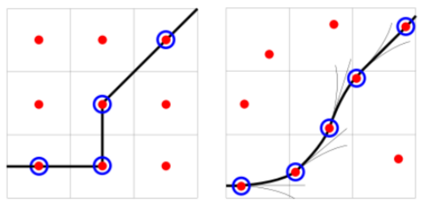

Fig. 3. Left: A*. Right: Hybrid A*

In traditional grid-based approaches, the environment is represented as a gridmap, and the Turtlebot's motion is limited to moving between adjacent cells. However, Hybrid A* addresses these limitations by incorporating a graph-based representation of the environment, called a lattice. A lattice consists of a set of motion primitives, which are predefined, short trajectories that the Turtlebot can follow. These motions guarantee a continuous and drivable solution compared to the grid-based approach.

$$x = vcos(\theta)dt \tag{1}$$

$$y = vsin(\theta)dt \tag{2}$$

$$\theta = \omega dt \tag{3}$$

Eq. 1 through 3 describe the motion of the Turtlebot used to construct the lattice of the Hybrid A*, where both the velocity, v, and angular velocity, ω, are assumed constant. Eq. 1 and 2 represents the change in the x- and y-coordinate respectively, while Eq. 3 represents the change in the Turtlebot's orientation angle over time. In this application, the velocity and angular velocity are taken as the maximum that the Turtlebot can achieve.

The algorithm starts by initializing an open set, which stores the nodes to be explored in a priority queue. The start position is added to the open set. While the open set is not empty, the algorithm continues to explore nodes. It selects the node with the highest priority from the open set as the current node. It then generates a lattice and a list of neighboring positions based on the current position. Neighbors that result in collision or are outside the map are discarded.

For each neighboring position, the algorithm calculates a tentative cost to reach that position and updates the path if a shorter path is found. These costs are defined as the g-score and the f-score. The g-score represents the cumulative cost of reaching a particular node from the start node. When a new path is discovered that reaches a node with a lower cost than its previous value, the g-score is updated with the new lower cost.

The f-score represents the estimated total cost from the start node to the goal node passing through a particular node. It is the sum of the g-score and the estimated heuristic cost from the current node to the goal node. In this application, the heuristic used is the Euclidean distance, which measures the straight-line distance between two points in a two-dimensional space. Together, the g-score and f-score selects the most promising node to explore next.

Finally, the neighboring positions are added to the open set for further exploration. If the goal position is reached, the algorithm reconstructs the path from the start position to the goal position and returns it. The path is reconstructed by iterating through the nodes in a dictionary describing where each node came from, starting from the current node and tracing back to the start node. It collects the trajectory indices in reverse order, representing the sequence of trajectories taken to reach each node.

However, if the goal is not found within a set amount of iterations, an empty path is returned indicating that there is no viable path to the goal. In some cases, the starting position of the Turtlebot resulted in no solutions being found due to the fact that the Turtlebot is facing an obstacle. To tackle this issue, the Hybrid A* tests each path from four different starting headings, each 90° apart. Additionally, a goal threshold is added such that if the goal is within a certain distance away from the final position, then a path is returned.

C. Dynamic Window Approach Controller

The Dynamic Window Approach is a versatile control method that serves as a local planner and facilitates obstacle avoidance. This approach allows the Turtlebot to dynamically adjust it's velocities within a window of feasible motion, enabling them to navigate complex environments while avoiding obstacles and reaching desired goals. By evaluating and selecting optimal velocity commands based on the robot's current state and environment, the DWA algorithm ensures safe and efficient motion.

It operates in a reactive manner, continuously updating velocity commands based on sensor feedback and the surroundings. The DWA uses three key criteria to select the best velocity command for the robot. The first criterion is the maximum velocity, which determines the highest permissible speed for the Turtlebot based on its dynamics constraints. By considering the maximum velocity, the DWA ensures that the robot operates within safe speed limits.

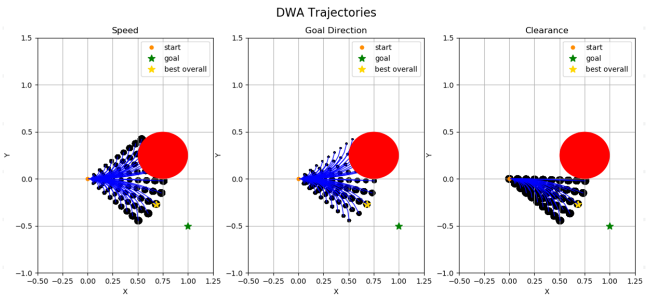

Fig. 4. DWA Key Criteria in Velocity Command Selection

The second criterion is the goal direction. The DWA evaluates the robot's heading towards the goal and compares it to the desired goal direction. By measuring the angular difference between the current heading and the goal direction, the DWA can prioritize commands that align the robot's motion towards the desired goal orientation. This allows the robot to navigate efficiently towards its destination.

The third criterion is clearance or obstacle avoidance. The DWA considers the surrounding environment and assesses the clearance or proximity of obstacles. By analyzing the sensor data, the DWA avoids selecting velocity commands that lead to collisions or unsafe trajectories. This ensures that the robot can navigate while maintaining a safe distance from obstacles.

In the code, the algorithm iterates over a range of linear and angular velocities, simulating the motion of the Turtlebot for each velocity combination. Next, the algorithm computes costs for speed, goal direction, and collision avoidance based on the simulated trajectories. The algorithm then scales the costs and combines them to find the best command, considering the weights of the different costs. The command with the maximum weighted value is selected as the best linear and angular velocities.

IV. Results & Discussion

In this section, we present the results and discussion of our implemented system that integrates a frontier-based exploration strategy, a Hybrid A* planner, and a Dynamic Window Approach controller for autonomous exploration and path planning. The implementation was conducted using the Stonefish simulator, and the results were visualized using RViz to analyze the occupancy map constructed using the Octomap server.

Our integrated system demonstrated robust performance in autonomous exploration and path planning tasks. The frontier-based exploration strategy effectively guided the Turtlebot towards unexplored regions, allowing it to cover a larger portion of the environment while minimizing redundant exploration. By identifying and navigating towards frontiers, the Turtlebot efficiently explored accesssible regions with the highest information gain first.

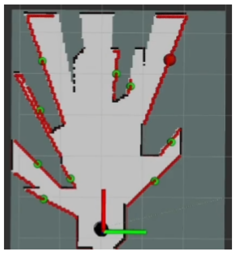

Fig. 5. RViz Visualization: Red is Frontiers, Green is Centroids

The Hybrid A* planner played a crucial role in generating optimized paths. By considering the dynamic constraints of the Turtlebot and utilizing both grid-based and graph-based algorithms, the planner successfully planned paths from the current position to the identified frontiers. It effectively explored the state space and provided paths that ensured efficient and obstacle-free navigation.

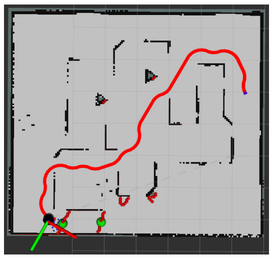

Fig. 6. RViz Visualization: Red Path is Hybrid A*

The integration of the DWA controller further enhanced the Turtlebot's navigation capabilities. By dynamically adjusting the velocity and angular velocity based on factors such as maximum velocity, goal direction, and clearance from obstacles, the DWA controller guaranteed smooth motion along the planned paths. It responded to changes in the environment and avoided collisions, resulting in agile robot movement.

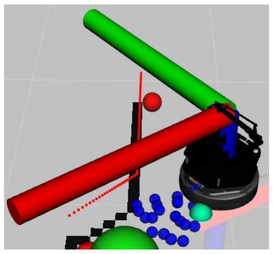

Fig. 7. RViz Visualization: Blue is Commands, Cyan is Best

Throughout the implementation, the Stonefish simulator provided a realistic environment for testing and validating our algorithms. It allowed us to assess the system's performance in various scenarios and evaluate its ability to navigate and explore different types of environments. RViz served as a powerful visualization tool, enabling us to observe and analyze the constructed occupancy map using the Octomap server. The visualization of the occupancy map provided valuable insights into the Turtlebot's perception of the environment and its ability to represent spaces correctly.

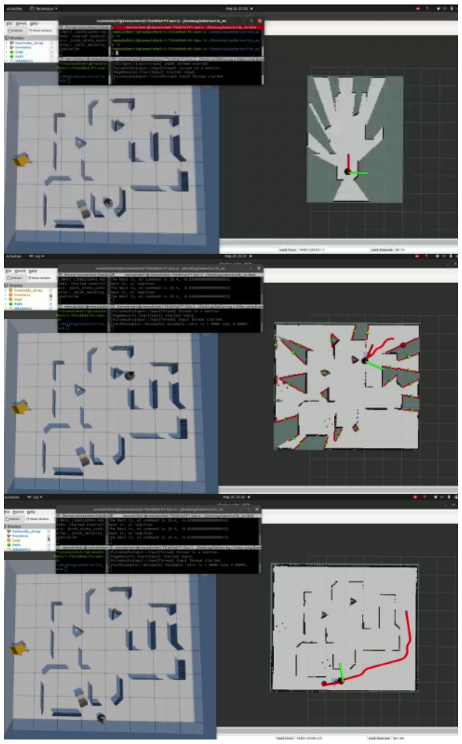

Fig. 8. Left: Stonefish. Right: RViz

Overall, our integrated system showcased robust autonomous exploration and path planning capabilities. The combination of the frontier-based exploration strategy, Hybrid A* planner, and DWA controller resulted in improved coverage of the environment, optimized path planning, and reliable obstacle avoidance.

V. Conclusion

In conclusion, our implemented system successfully integrated a frontier-based exploration strategy, a Hybrid A* planner, and a Dynamic Window Approach controller to enable autonomous navigation in unknown environments. The system demonstrated robust performance in efficiently exploring unknown regions, generating optimized paths, and ensuring obstacle avoidance. Through the use of the Stonefish simulator and RViz visualization, we validated the effectiveness of our approach and observed the Turtlebot's ability to construct an occupancy map.