Leo Explore: Reconnaissance Robot

EECE 5550 Mobile Robotics Final Project Report

Bhanu Prasad AJ
Northeastern University
M.S. in Robotics

Atharva Jamsandekar
Northeastern University
M.S. in Robotics

Jiatong Wu
Northeastern University
M.S. in Robotics

Abhishek Uddaraju
Northeastern University
M.S. in Robotics

Yijian Huang
Northeastern University
M.S. in Robotics

I. Introduction and Motivation

The Reconnaissance Bot represents a significant advancement in autonomous navigation systems, particularly in the context of exploring unknown environments. Leveraging a sophisticated Navigation stack built on frontier exploration principles, this package demonstrates remarkable capabilities in traversing uncharted territories while efficiently mapping the surroundings. By seamlessly integrating with Simultaneous Localization and Mapping (SLAM) algorithms, the Reconnaissance Bot obtains crucial spatial information essential for effective exploration.

Leveraging a well-designed Navigation stack divided into three modules, it showcases unparalleled capabilities in traversing uncharted territories while efficiently mapping the surroundings. The frontier server, employing a K-means clustering algorithm, identifies frontier points for exploration, guiding the robot towards unexplored regions. The Global Planner, utilizing an adaptation of the A-star algorithm, generates optimal paths through the environment, enabling efficient exploration and mapping. Complementing these modules, the Local Planner incorporates a Model Predictive Path Integral approach, ensuring agile and obstacle-aware motion planning in complex environments.

The motivation behind the development of the Reconnaissance Bot stems from the pressing need for autonomous systems capable of safely and efficiently navigating unfamiliar terrain. In scenarios such as surveying unsafe disaster-stricken environments or conducting reconnaissance missions in military-active regions, traditional exploration methods are often impractical or hazardous for human operators. Therefore, the deployment of robotic platforms equipped with advanced navigation capabilities becomes paramount in mitigating risks and gathering essential data.

The successful testing of the Reconnaissance Bot on the TurtleBot3 Burger platform underscores its practical viability and adaptability to real-world scenarios. As a result, this package holds immense promise in revolutionizing the way autonomous systems navigate and explore unknown environments, thereby contributing to enhanced safety, efficiency, and effectiveness across diverse fields of application.

II. Proposed Solution

A. Architecture

As a reconnaissance robot, Leo must possess the ability to locate itself at all times, generate a map of its surroundings, autonomously identify unexplored areas, estimate the next goal to optimally explore the environment and navigate its way to reach the goal efficiently and effectively. For this to work, Leo needs a proper system of control flow to make sure the operations listed above are executed smoothly and seamlessly.

Since the scope of this project limits itself to the planning aspect of the navigation stack, the task of Localization and Mapping the environment is delegated by simply using the existing turtlebot SLAM packages and hence will not be discussed in detail.

In the solution proposed, Leo employs a top-down control flow, where the Global Planner and Local Planner are implemented as nodes while the Frontier exploration is implemented as a service. The Frontier exploration server is responsible for generating goals to explore the map, the global planner node generates a path that connects Leo's current position with a goal. The local planner

will command Leo's velocity to reach the required pose.

Fig.1: A snap of the ROS node graph

The process starts at the global planner node, which sends a request to the frontier server to receive a new goal. The global planner processes it and publishes the immediate next target pose to the local planner. The global planner constantly refreshes the path and does not request for another goal until the current goal is reached or is determined to be unreachable. Not unlike the global planner, the local planner receives the next target pose and latches on to it until the target is reached. Once the target pose is reached, the local planner latches on to the latest next target pose.

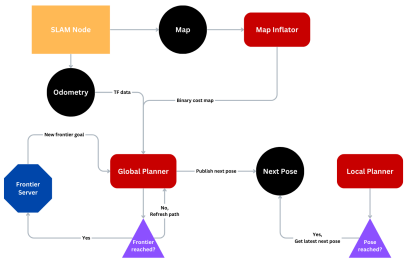

Fig.2: The software system architecture

In order to make the operation of switching to a new exploration goal seamless, the global planner sends a request to the frontier exploration server when Leo is a certain distance away from the current exploration goal. This is done so that by the time Leo reaches the current exploration goal, the path to the goal produced by the frontier server is already generated by the global planner and is published to the local planner.

B. Frontier Server

The exploration algorithm used by the exploration/navigation stack of Leo is directly derived from the one proposed by [1]. This algorithm was chosen over the other popular frontier exploration algorithms due to its feasibility and simplicity .

The method suggested in [1] involves 2 simple steps to determine the next goal in order to explore more of the surroundings.

(1) Selecting frontier candidates: The basis for selecting points on the frontier to be candidates is that the point should promise a chance to increase the information that the robot has about its surroundings while also being reachable.

(2) Clustering the candidates: The clustering is done based on distance-based connected component method. The centroids of the clusters are determined and a centroid is chosen through some selection criteria and becomes the next exploration goal.

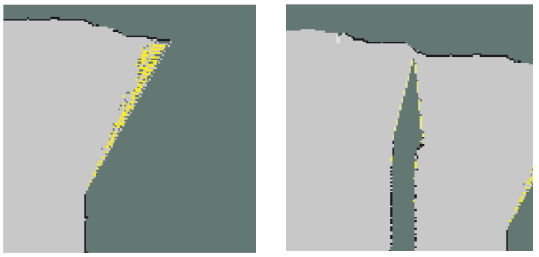

Fig.3: Examples of frontier points from [1]

The exploration algorithm used by Leo also shares the first step with the method proposed in [1], which is to select eligible candidates from all the frontier points. This is done by checking the 5x5 neighborhood of each cell. If the neighborhood contain any occupied region, the cell is rejected. The cell is eligible to be a candidate if the ratio between the free cells and unknown cells are between a set threshold.

The next step is to cluster the candidates. However, in contrast to the distance based connected component clustering methods proposed in [1], Leo's exploration algorithm makes use of K-Means clustering algorithm.

K-Means clustering algorithm partitions candidates into a predetermined number of clusters, each represented by a centroid calculated as the mean position of all candidates within that cluster. The process involves initializing random centroids for each cluster, assigning each candidate to the nearest centroid based on Euclidean distance, recalculating centroids as the mean of assigned candidates, and iterating these steps until the centroids stabilize. This ensures candidates within each cluster are more similar to each other than those in different clusters.

Once the clusters and their respective centroids are obtained, each cluster is scored based on the proximity and the mass of the cluster. The proximity is defined by the Euclidean distance between Leo's current position and the position of the centroid. The mass of the cluster is given simply by the number of candidates belonging to the cluster. The score function is a weighted sum of mass and the inverse of proximity. The cluster with the highest score is selected.

One of the limitations of the method proposed in [1] is that the derived exploration

goal is not guaranteed to be reachable. The paper suggests circumventing this limitation by trying to resolve this at the global and local planner layers.

Instead of resolving this at the planner level, Leo uses a different approach to resolve this issue, instead of setting the centroid of the selected cluster to be the next exploration goal, a random candidate belonging to the selected cluster is chosen to be the next.

C. Global Planner

Leo's global planner uses the A-Star algorithm. The decision to go with A-star over other path-planning algorithms can be explained using the simplicity and effectiveness of the algorithm.

A-Star (A*) algorithm is widely used in path planning and graph search problems. It is the heuristic search algorithm that finds the optimal path from the starting point to the endpoint by combining a heuristic function and a cost function. The key components of the A-Star algorithm are the cost function and the heuristic function. The cost function (g) represents the actual path cost from the starting point to the current node, while the heuristic function (h) estimates the minimum path cost from the current node to the goal. A-Star uses a total cost function, f = g + h, to evaluate the path's quality and to find the optimal path in the shortest time.

In this project, we use distance as the cost function (g). As for the heuristic function, it must meet specific conditions to be acceptable, ensuring that the estimate does not exceed the actual cost while also guiding the search effectively. We use Euclidean distance as the heuristic function (h) for the A-Star algorithm applied to our robot.

In a practical program, the A-Star algorithm is as follows. We use an open set and a closed set to track the search process. The open set contains nodes to be processed, while the closed set contains nodes that have been processed. Once all nodes are processed, the closed set is reversed to form a path (a list of positions) for the local planner. The steps of the algorithm are as follows:

1) Initialization

Add the start node to the open set. Set the cost function (g) to 0 and the heuristic function (h) to the estimated cost to the target node.

2) Choose the Optimal Node

Calculate the sum of the cost function (g) and the heuristic function (h) for the surrounding grid cells in all directions (up, down, left, right, and diagonals). If the new f-value is less than the current value, choose the neighboring node with the lowest total cost function as the next node to move to.

3) Check the Target Node

If the current node is the target node, the optimal path has been found.

4) Calculate the Path

Output the path nodes in order to a list, then publish it to ROS.

Leo's global planner node sends a request to the frontier service server and receives the next exploration goal as the response. The global planner listens to the \tf publisher and gets the current position of Leo with respect to the map frame. The current position and the exploration goal position is fed into the A* algorithm to obtain the shortest path between them.

This path is constantly refreshed to make sure the path is still relevant and to account for noisy sensor readings. As mentioned previously, the global planner requests for the next exploration goal once Leo is within a certain distance of the current exploration goal. This pipeline enables a smooth, almost zero latency transition from the current exploration goal to the next.

D. Local Planner

In this project, we designed and implemented a local planner using the Model Predictive Path Integral (MPPI) control algorithm. The local planner's primary objective is to generate safe and feasible trajectories for a robotic system within its immediate environment. In our implementation of the MPPI control algorithm, we followed several key steps to generate optimal trajectories for our robotic system. These steps can be summarized as follows:

1) Initialization

We start by initializing the system's state, including the robot's position, orientation, and velocity. Additionally, we define the target or goal state that the robot aims to reach.

2) Rollout Generation

Using the current state and the MPPI algorithm, we generate multiple trajectory rollouts. Each rollout represents a potential path that the robot could follow from its current state to the goal state.

3) Rollout Evaluation

For each generated rollout, we evaluate a cost function that captures various criteria such as path length, smoothness, and proximity to obstacles. This step helps us assess the quality of each trajectory candidate.

4) Weighted Sampling

The MPPI algorithm employs weighted sampling to select trajectories based on their cost. Costs are assigned based on the rollout intersection with an obstacle and the distance from the goal position.

5) Weighted Sum of perturbations

At this crucial step, the selected trajectories undergo a weighted sum of perturbations. This involves adding weighted perturbations to the nominal trajectory.

Our implementation utilized specific parameters to control generation. Specifically, we accounted for the following variances of perturbations: a linear velocity variance of 0.05 m/s and an angular velocity variance of 0.3 rad/s. We set the number of rollouts to 100 and the number of steps to 20, ensuring a comprehensive exploration of possible trajectories while maintaining computational efficiency.

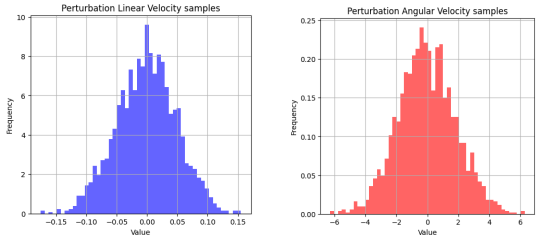

Fig.4: Plot of perturbations in velocity samples

Additionally, we defined the nominal velocity as 0.8 m/s and allowed for a nominal angular velocity range between 0 and π radians/s, facilitating smooth and agile motion planning.

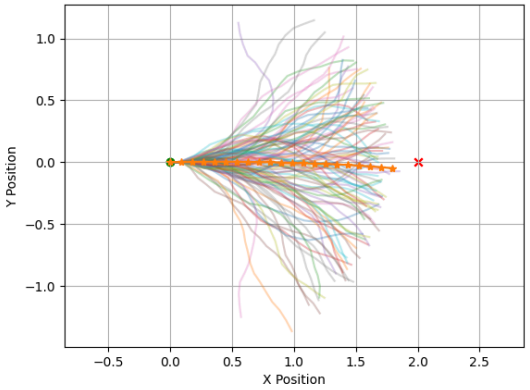

Fig.5: Plot of paths obtained through propagation

The selected perturbations can turn the robot around when there is an obstacle ahead of the robot only with a single iteration of MPPI this helps us attain a real-world operation frequency of 12 Hz. The rollouts are shown in the figure above this is the final updated path. Overall, our local planner implementation using MPPI control with the specified parameters and operating frequency showcases its capability to generate smooth, collision-free trajectories in real-time scenarios.

III. Results

The testing of the package has been bimodal, i.e., both simulation and Real-world testing setup. The bimodal setup benefited the iterative improvement of the package during development. The testing was carried out using the Gmapping algorithm in the Turtlebot3 SLAM package[2].

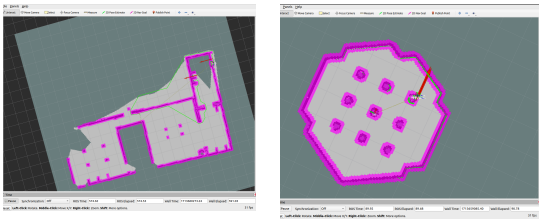

Fig.6: Output of the exploration and navigation stack

A. Simulation

Two simulation environments were set up in the Gazebo simulator, a part of the Turtlebot3 Gazebo package[3]:

1) Turtlebot3 House: A hexagon-shaped grid that visually resembles a turtle.
2) Turtlebot3 World: A Real house-like environment.

The results of the efficacy of the environment were calculated in terms of two parameters: 1) Time to complete and 2) Area of the environment mapped. The final ratio of area mapped to the time taken is calculated and shown for both environments (Fig x). The calculated parameter is called the Rate of Exploration (m^2/s). The results show that the rate of exploration is empirically inversely proportional to the 1.5th power of the Area of the setup.

B. Real-World Setup

Two real-world environments were set up on the Northeastern University campus:

1) 4x4 Maze World:
2) Hurtig Hall Corridor:

The demonstration videos show the efficacy of Leo Explore in Real World Environment, refer to Section VI.

IV. Conclusion

In conclusion, this research marks a significant milestone in the development of autonomous navigation systems through the creation of the Reconnaissance Bot. While our exploration has demonstrated its efficacy in navigating unknown environments and integrating seamlessly with the TurtleBot3 platform, several critical areas for extension have been identified. Chief among these is the imperative to address scalability concerns inherent to the global planner, map size management, and frontier calculation mechanisms.

Efforts towards scalability must prioritize the optimization of the global planner to accommodate larger environments without compromising efficiency or accuracy. This entails exploring advanced algorithms or techniques capable of efficiently generating optimal paths across expansive terrain. Concurrently, strategies for managing map size must be devised to ensure the system remains lightweight and responsive, even when operating in environments with extensive spatial coverage.

Addressing these scalability challenges will be pivotal in unlocking the full potential of the Reconnaissance Bot for applications spanning disaster response to military reconnaissance. Through continued research and development efforts in these areas, we can propel autonomous navigation technology towards greater versatility, adaptability, and real-world impact.

V. Work Distribution Diagram

Work	Participation
Hardware Set Up	Bhanu Prasad AJ Jiatong Wu Yijian Huang Abhishek Uddaraju Atharva Jamsandekar
Software Set Up & Camera Calibration	Bhanu Prasad AJ Jiatong Wu Yijian Huang Abhishek Uddaraju Atharva Jamsandekar
Frontier Exploration	Bhanu Prasad AJ Yijian Huang
Global Planner	Bhanu Prasad AJ Jiatong Wu Yijian Huang
Local Planner	Abhishek Uddaraju Atharva Jamsandekar
April Tag	Abhishek Uddaraju Atharva Jamsandekar

VI. Video Link

https://youtu.be/fyIsYrZ6wks

VII. References

[1] Erkan Uslu, Furkan Çakmak, Muhammet Balcılar, Attila Akinci, Mehmet Fatih Amasyalı, and S. Yavuz, “Implementation of frontier-based exploration algorithm for an autonomous robot,” Sep. 2015, doi: https://doi.org/10.1109/inista.2015.7276723

[2] “turtlebot3_slam - ROS Wiki,” wiki.ros.org. http://wiki.ros.org/turtlebot3_slam

[3] “turtlebot3_gazebo - ROS Wiki,” *wiki.ros.org*. http://wiki.ros.org/turtlebot3_gazebo

[4] Williams, Grady, Andrew Aldrich, and Evangelos Theodorou. "Model predictive path integral control using covariance variable importance sampling." arXiv preprint arXiv:1509.01149 (2015).

Environment	**Area Covered (in m^2)**	**Time (in s)**	**Rate of Exploration (in m^2/s)**
House	21.88	74s	0.30
World	83.82	560s	0.15