Multi-Robot Autonomous Exploration and Navigation

Abstract

A Map Graphing and Traversal system for Multi-Robot system using Frontier based search and a Distributed Boundary Based Search Algorithm for Exploration along with Task Allocation system for Map Traversal in ROS2

Introduction

This report henceforth mentions the underlying system of our solution to the PS - *Centralized Intelligence for Dynamic Swarm Intelligence*. The tasks are divided into two parts

1. **Autonomous Navigation for Map Graphing -** Using multiple robots to explore an unknown map of generalized size. Thus, The system is adaptable to N number of bots and unknown areas.
2. **Task Allocation System and Navigation -** Allocating tasks to best-suited bots in the whole fleet depending on various factors.

The combination of the above tasks gives a profound system capable of complete autonomy in exploring as well as navigating any given environment.

Literature Review

Various technologies and algorithms are adapted to best suit our needs-

turtlebot3

We have used turtlebot3 with certain required modifications as our choice of the robot. Turtlebot3 is great for this use case as it provides predefined topics from LiDAR sensors and SLAM Mapping, among others.

nav2

The ROS2 Navigation Stack allows mobile robots to navigate through complex environments to complete user-defined tasks, accommodating various robot kinematics. It leverages sensor and semantic data to build an understanding of the environment, facilitating dynamic path planning, obstacle avoidance, and motor velocity computation with the $A*$ pathfinding algorithm, which ensures optimal route planning. Nav2 uses behaviour trees to customize navigation behaviour.

Nav2's local cost map continuously builds and updates a dynamic representation of the robot's surroundings, supporting task prioritization by adjusting navigation paths based on real-time environmental changes and task urgency. The $costmap_2d$ package generates a layered 2D map of the environment using sensor data and can start with a static map. Layers are combined through plugins and inflated based on the robot's size. Cost-map filters allow adding features like safety zones, speed limits, and preferred lanes, helping robots adjust their behaviour and paths dynamically.

In GPS-free multi-agent swarm navigation and autonomous navigation system, the InflationLayer, ObstacleLayer, and RangeSensorLayer ensure safe navigation by maintaining distance from obstacles, detecting obstacles in real-time, and integrating sensor data for improved path planning, all without relying on GPS.

B-Spline Path Smoothing refines the paths, ensuring smoother and more efficient trajectories for the robots. Multi-robot coordination ensures effective collaboration between robots in shared spaces, optimizing task completion and resource allocation. [5]

$$C(t) = \sum_{i=0}^{r-1} N_{i,r}(t) P_i, \quad 0 \le t \le t_{max} \tag{1}$$

where $N_{i,r}(t)$ represents the base function of $r-1$ order B-spline curve and can be derived via de Boor-Cox recursive formula, as illustrated in Eq.2

$$\begin{cases} N_{i,1}(t) & = \begin{cases} 1, & t_i \le t \le t_{max} \\ 0, & \text{otherwise} \end{cases} \\ N_{i,r}(t) & = \frac{t-t_i}{t_{i+r-1}-t_i} N_{i,r-1}(t) + \frac{t_{i+r}-t}{t_{i+r}-t_{i+1}} N_{i+1,r-1}(t) \end{cases} \tag{2}$$

Implementation

We have to implement the processes in a holistic manner so that they stay modular yet integrated. We have decided to pursue the PS in two parts, the first being exploration and the second being Task navigation.

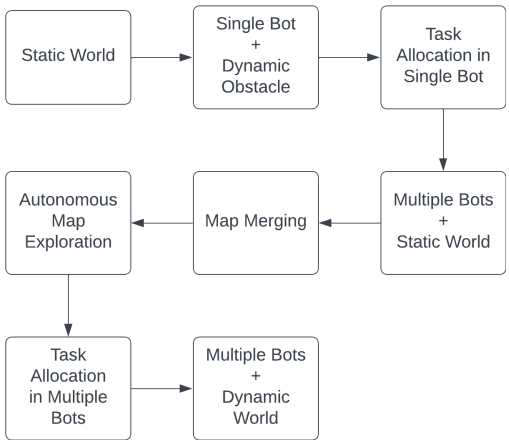

Fig. 1. Our thought process

Exploration

Exploration is further divided into different parts to make it easier to work on it as a team. The parts overall were:

1. Generating Map using SLAM and exploring Manually
2. Exploring autonomously using a single bot
3. Exploring manually using multiple bots
4. Merge maps from multiple bots
5. Extending autonomous exploration to multiple bots

Let's look into it step-by-step.

Generating map using SLAM and exploring manually

SLAM toolbox provides amazing tools that take LiDAR data from the `/LaserScan` topic and generate a map. This requires certain algorithms like **PLICP**, **gMapping** etc.

Manual exploration can be done using `teleop_twist_keyboard`, and the scans served to SLAM along with transformations via the topic `/tf` are combined to map the entire area. *gMapping* and *PLICP* are explained below:

PLICP - Point-to-Line Iterative Closest Point[4]
PLICP is an improvement over the ICP algorithm, which was previously used in mapping the environment. ICP uses a point-to-point metric to map changes from LaserScan from a *LiDAR sensor* to build a map for the environment. ICP iteratively tries to find trans-rotation(represented by $R(\theta), t$) to fit a point set p_i to a reference S^{ref} and minimizing the result

$$\min_{q} \sum_{i} ||p_i \oplus q - \Pi\{S^{ref}, p_i \oplus q\}||^2$$

$\Pi\{S^{ref}, p\} \rightarrow$ Euclidean Projection of p on S^{ref}

PLICP improves this by using a point-to-line approach instead of a point-to-point approach.

$$\min_{q_{k+1}} \sum_{i} \left(n_i^T \left[p_i \oplus q - \Pi\{S^{ref}, p_i \oplus q\} \right] \right)^2$$

$n_i^T \rightarrow$ Transpose of normal of closest line on reference line to a given point

gMapping[6]
gMapping is a highly efficient Rao-Blackwellized particle filer that helps learn grid maps from laser range data. [3]. The algorithm has been ported to support ROS2 for working with the topic `/scan` of the message type `LaserScan` and applies the required transformation. The map formed is published on the topic `/map`

Exploring autonomously using a single bot

Exploring manually is how most robotics-based solutions ensure most of the area is mapped. But in the case of robots left to explore the area and autonomously map the area, we have a couple of different prominent solutions. Each of the solutions has its pros and cons. We mainly focused on and tested **RRT*** and **Frontier Based Search**.

We settled on using Frontier-based exploration due to its a lower requirement and more prominent autonomy, which is favoured. But to understand why it is our choice, we must first understand the algorithms.

Let's explore how both of them works:

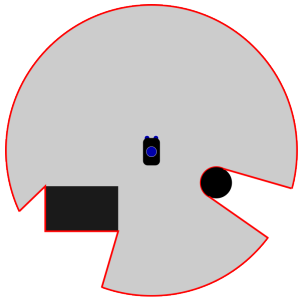

Fig. 2. Frontier Representation. (In Red) Frontiers, (In Grey) Known Area, (In Black) Obstacles, (In Center) Bot

Algorithm 1 Frontier-Based Algorithm

while !map_covered && frontier $\geq$ Predefined Cluster Size **do**
  $M \leftarrow$ current map
  $R \leftarrow$ robot position
  $F \leftarrow$ Set of frontier points
  $F_c \leftarrow$ Distance of Frontier Clusters
  for each F_c **do**
    $F_{c_g} \leftarrow$ Find Cluster Center
  end for
  $F_{target} \leftarrow$ min(Traversable_Distance(F_{c_g}, R))
end while

Frontier Based Exploration

There are several algorithms that can help in achieving this in the ANS approach. We have used an approach that is crucial to exploring unexplored areas containing obstacles, namely **Frontier-Based Exploration**. *Frontiers* are areas/boundaries between explored and unexplored sections

Fig. 2 represents how a SLAM model generates frontiers in the presence of obstacles.

The frontier algorithm works on the process of evaluating a grid with the frontier target point in the centre and navigating the bot towards the target point and updating the frontiers[7] The Frontier-Based Exploration (FBE) algorithm enables autonomous robots to efficiently explore unknown environments by identifying and navigating to frontiers, which are boundaries between known and unknown areas. The environment is represented as a grid, where cells are marked as free, occupied, or unknown, and sensors provide data to update this grid. The algorithm detects frontier points where free space meets unknown areas, then plans the robot's path to the nearest frontier using algorithms like A* or Dijkstra, while avoiding obstacles. As the robot explores, it updates the map and performs a 360-degree sweep at each frontier to gather new data. The process continues until all frontiers are explored, signaling the completion of the mapping task. This approach ensures efficient exploration by focusing on the boundaries, reduces redundant exploration and provides comprehensive coverage.

RRT*

We used a rapidly exploring Random Tree Star algorithm for exploration, path planning and optimization, particularly in environments where the goal is to navigate through unknown or partially known areas. The bot searches for the path by

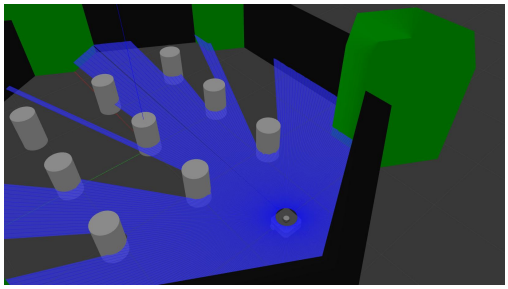

Fig. 3. Actual SLAM Projections in Default Environment

extending an exploring tree from the root node until it touches the target node, and then it improves the path quality by sampling more nodes. As this RRT* is an asymptotically optimal method, it provides convergent solutions that can converge to optimal as the number of samples tends to infinity. [5]

Algorithm 2 RRT*

Input
  n_{start}
  n_{end}
Output
  $T = (V, E)$
while LoopCondition() **do**
  $n_{rand} \leftarrow$ Sample()
  $n_{nearest} \leftarrow$ Nearest(V, n_{rand})
  $n_{new} \leftarrow$ Steer$(n_{nearest}, n_{rand})$
  if CollisionFree$(n_{nearest}, n_{new})$ **then**
    $V \leftarrow V \cup \{n_{new}\}$
    $E \leftarrow E \cup \{(n_{new}, n_{nearest})\}$
    $N_{near} \leftarrow$ Near(V, n_{new})
    $n_{parent} \leftarrow$ ChooseBestParent(N_{near}, n_{new})
    if (**then**n_{parent} ! = null
      $E \leftarrow E \cup \{(n_{new}, n_{parent})\} - \{(n_{new}, n_{nearest})\}$
    end if
    $E \leftarrow$ Rewire(E, N_{near}, n_{new})
  end if
end while
return $T = (V, E)$

Our main concern with RRT* was its requirement of cost-map, which, although works for local autonomy, starts to show ill effects on global autonomy

So, mainly, FBS provides a better autonomous exploring system than RRT* and at lower input requirements.

Exploring manually using multiple bots

Multiple bots can be launched simultaneously in a gazebo environment. All bots would individually have SLAM integrations to produce the local map using the aforementioned algorithms. Each bot needs to be defined with a separate namespace to ensure no conflicts in the ros2 communication environment. This also allows us to control the bots individually to explore the environment.

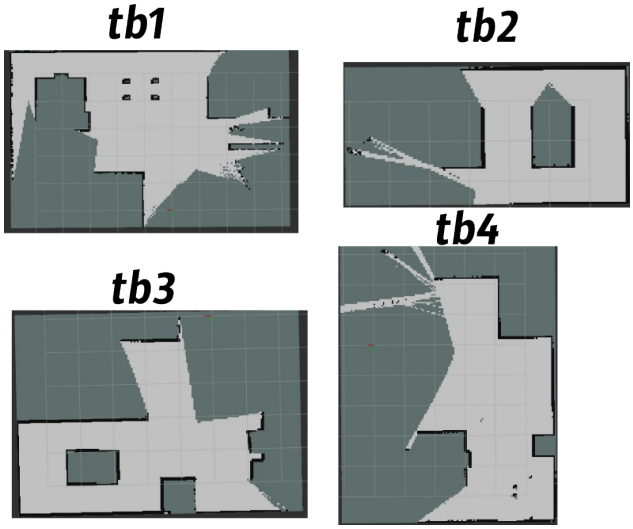

Fig. 4. This image shows local maps from 4 turtle bot instances

Fig. 5. This image shows the merged map of the above shown four local maps

Merge maps from multiple bots

Map merging is an important process in multi-robot SLAM(Simultaneous Localization and Mapping), enabling the construction of global maps by merging the local maps built by individual robots to provide a better view of the environment. We define *ros*2 nodes, which allow multiple robots to send their occupancy grid maps, which are then merged into a single map. The number of robots is specified, and then we launch two files: $RViz2$ for visualizing the merged map and merge map for merging the maps. When multiple bots construct their own map, it becomes crucial to merge these maps correctly. This is extremely important as the flaws in merged maps can propagate to the final map and can cause problems later in navigating the environment.

From the various maps present, we have used Occupancy Grid Maps.

Occupancy Grid Maps[8]

These maps divide the space into grids, where each cell reflects the probability of being occupied(0.9-1), free(0-0.1), or unknown(Intermediate probability values). For merging

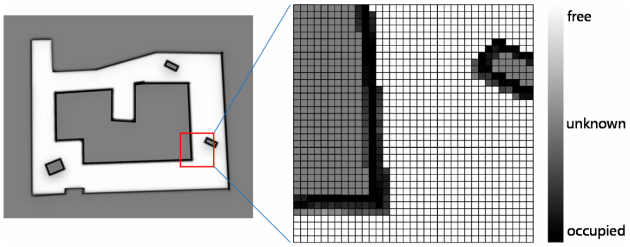

Fig. 6. An example of Occupancy Grid Maps

occupancy Grid Maps, we use Probability-Based Methods in which each cell represents the probability of being occupied by an obstacle. For map merging, we combine the probability values for corresponding grid cells from different robots, accounting for their observations and trajectories. If the initial poses are known, then the posterior probability of the global map and the trajectories of all robots based on their measurements (z), controls (u), and initial positions (x_0) are estimated. The full joint probability distribution :

$$p(x_1, x_2, m|z_1, u_1, x_0, z_2, u_2) = p(m|x_1, z_1, x_2, z_2) \cdot p(x_1|z_1, u_1, x_0) \cdot p(x_2|z_2, u_2, x_0) \tag{3}$$

$x_i \rightarrow$ Trajectory of robot i
$z_i \rightarrow$ Measurements of robot i
$u_i \rightarrow$ Control input of robot i
$m \rightarrow$ Global Map
For multiple robots:

$$P(occ_{x,y}) = \frac{odds_{x,y}}{1 + odds_{x,y}} \tag{4}$$

$$odds_{x,y} = \prod_{i=1}^{n} odds^i_{x,y} \tag{5}$$

$$odds^i_{x,y} = \frac{P(occ^i_{x,y})}{1 - P(oc^i_{x,y})} \tag{6}$$

Now, when initial poses are not known, the relative pose δs must be estimated:

$$p(x_1, x_2, m|z_1, u_1, x_0, z_2, u_2, \delta s) = p(m|x_1, z_1, x_2, z_2) \cdot p(x_1|z_1, u_1, x_0) \cdot p(x_2|z_2, u_2, \delta s) \tag{7}$$

$\delta s \rightarrow$ Relative pose of Robot 2 w.r.t. Robot 1 when they first meet.

For objective function based on overlapping, we optimize the maps that maximize or minimize this function to find the best transformation that aligns the maps, represented as matrices, where each cell stores an occupancy probability. The rigid transformation (including rotation and translation) between two maps is given by:

$$T_{tx,ty,\theta(x,y)} = \begin{bmatrix} \cos\theta & -\sin\theta & tx \\ \sin\theta & \cos\theta & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{8}$$

$\theta \rightarrow$ Rotation angle b/w two maps
$tx, ty \rightarrow$ Longitudinal and lateral translation b/w two maps
$[x, y, 1]^T \rightarrow$ Homogeneous coordinate representation of any point $(x, y) \in$ map(m) We can even approach this problem with the help of map overlapping. So for map overlapping between two maps $m1$ and $m2$:

$$\omega(m_1, m2) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} Eq(m1[i, j], m2[i, j]) \tag{9}$$

where, $Eq(a, b) = \begin{cases} 1, & a = b \\ 0, & \text{otherwise} \end{cases}$
For optimal transformation we need max$[\omega(m1, T_{x,y,\theta}(m2))]$

For objective function F_c based on occupancy, we measure consistency between maps m_1 and m_2:

$$F_C(m_A, p_{BA} \oplus m_B) = \sum_{i=1}^{n} [m_A(p_{BA} \oplus o_{B(i)}) \mid p_{BA} \oplus o_{B(i)} \text{is occupied in } m_A] \tag{10}$$

Task Allocation

Various methods can be used to perform task allocation in a ROS2 environment. The most famous and advanced one available in the market is either

1. Open-RMF
2. Custom Solution

Open-RMF

Open-RMF is made by OSRF[2]. Open-RMF is a fleet management and Traffic management system that can be suited for various types of environments and robot types.

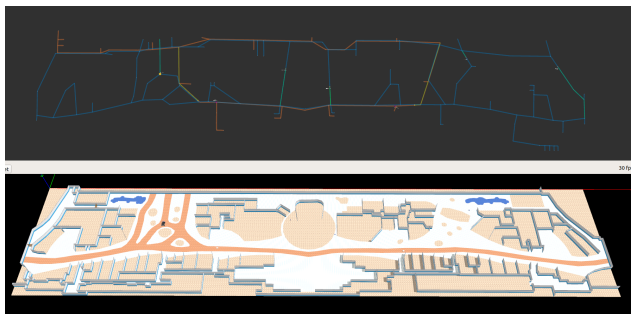

Fig. 7. Demonstration of Airport Pathways generated by Open-RMF

Custom Solution

A custom solution involves handling the tasks independently and managing the bot's location and priorities. This can be tedious but can prove to be very useful in specific conditions like ours.

Choosing between the two

We have gone for a *Custom Solution* instead of using the comprehensive solution provided by Open-RMF because of the reasons specified below:

- **Requirement of predefined pats in Open-RMF -** Open-RMF requires a predefined path for bots to follow as it is also a traffic serializing solution. However, this can harm our motive because of the bot's freedom to move among static and dynamic maps.
- **Auction-based solution -** Auction-based solution as used by Open-RMF can be beneficial but is counteractive in our

solution as it creates a complex solution that is less suitable for non-competing fleets. Our bots are working towards a common goal instead of competing for the goal as provided by the central server.

We will further elaborate on our custom solution's capability.

Custom Solution Specifications

Our custom solution uses algorithmic computing and an SQLite3 database to support an efficient yet simple ticketing/tasking solution. We have also used the actions `/{namespace}/compute_path_to_pose` and `/{namespace}/navigate_to_pose`

Navigate To Pose

The bot begins by initiating the navigation task and setting the goal. It then uses the selected planner to compute a path to the goal. If the planner fails, it retries a specified number of times. If the retries are exhausted without success, the system transitions to a system-level recovery phase. If the path is successfully computed, the bot follows the path using the selected controller. It continues along this path until navigation is successfully completed. A new goal can be issued once the navigation task is finished, and the process restarts.

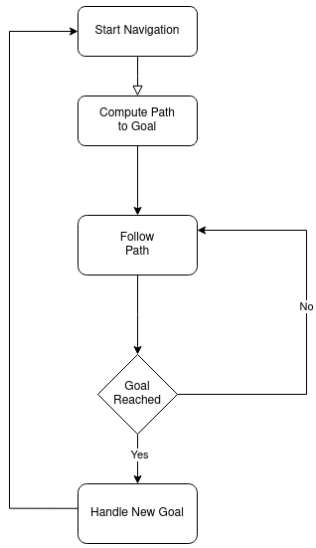

Fig. 8. Flowchart demonstrating Navigation to a Goal

Compute Path To Pose

It is an action server designed to calculate the robot's path from a given initial point to a specified endpoint. It utilizes algorithms such as Grid-Based Path Planning to compute the most efficient route while ensuring obstacle avoidance.

Algorithm 3 Compute Path To Pose

Input

start_pose — Optional start position (PoseStamped)

goal_pose — Target position (PoseStamped)

planner_id — ID of the planner plugin (e.g., "GridBased")

server_name — Name of the action server

server_timeout — Timeout for server response (default: 10ms)

Output

path — Computed path (nav_msgs::msg::Path)

error_code_id — Error code indicating success or failure (uint16)

Use `start_pose` if provided; otherwise, use the robot's current position.
Set `goal_pose` as the target position.
Select the planner plugin using `planner_id`.
Connect to the `server_name` action server.
Send action requests with `start_pose`, `goal_pose`, and `planner_id`.
if Response Received **then**
    Retrieve and return the computed path.
    Set `error_code_id` to `SUCCESS`.
else
    Handle failure and set `error_code_id` to the appropriate error.
end if

SQLite3

SQLite3 database is used to store pending and completed tasks along with Available bots ready to take on pending tasks.

The database schema for all three is defined below:

1. AvailableBots:

```
id INTEGER PRIMARY KEY AUTOINCREMENT,
namespace TEXT NOT NULL
```

2. Completed

```
id INTEGER PRIMARY KEY,
x_coordinate FLOAT NOT NULL,
y_coordinate FLOAT NOT NULL,
bot TEXT NOT NULL,
timestamp_completed TIMESTAMP NOT NULL,
timestamp_added TIMESTAMP NOT NULL,
timestamp_started TIMESTAMP NOT NULL,
bot_intial_pose TEXT NOT NULL
```

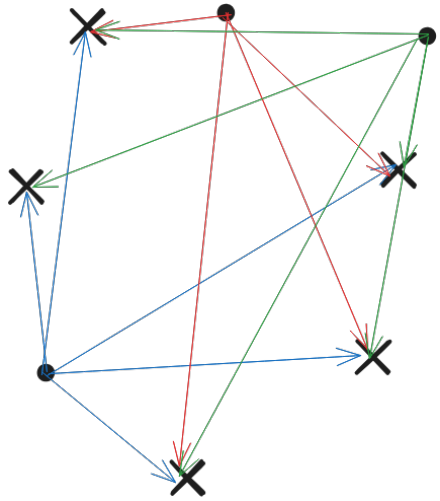

Fig. 9. Graph demonstrating connections to build cost matrix for Hungarian Algorithm. Dots represent bots, cross represents targets

3. Pending

```
id INTEGER PRIMARY KEY AUTOINCREMENT,
x_coordinate FLOAT NOT NULL,
y_coordinate FLOAT NOT NULL,
timestamp_added TIMESTAMP NOT NULL,
```

The `id` in the Completed Table is made sure to be the same as `id` in the Pending table by reusing the same id after the bot completes the task.

Hungarian Algorithm

We wanted a solution that made sure that we travel a combined minimal distance from the starting to ending position of bots to be most efficient. More specifically, **Traversable Distance** should be minimized. The traversable distance for each bot and the goal is calculated using the *nav2* action `/compute_path_to_pose`. Minimizing traversable distance is very simple when we only have one target. We just need to compare the distances and pick the lowest values among the distances.

This quickly gets complicated when we have multiple targets with multiple bots. Our solution has incorporated a graph-based algorithm that gets the path for minimal traversable distance. We need to find the distance of every bot from every target and make a cost matrix from there. The connections can be seen in Fig. 9

The Hungarian algorithm can also be applied using Bipartite-Graphs [1]

Hungarian algorithm in its base form only works when the cost matrix is symmetric i.e.

$$\text{no. of bots} == \text{no. of targets}$$

We have adjusted for this using dummy values with weight equal to either 0 or ∞

$$\text{weight} = \begin{cases} 0, & \text{no. of bots} > \text{no. of targets} \\ \infty, & \text{no. of bots} < \text{no. of targets} \end{cases}$$

Note: ∞ is approximated for by using a large enough value(10^6)

Scalability

The scalability of autonomous navigation systems is one of its most notable strengths. The system is designed to handle multiple robots, enabling them to work together to explore and map larger environments more efficiently. Each robot operates autonomously; it can share sensor data and dynamically adjust its path in response to the actions of other robots, minimizing redundancy in exploration. As the system can handle static and dynamic obstacles, it is adaptable to various real-world environments, from cluttered indoor spaces to larger outdoor terrains. The system can work with any number of robots, making it flexible for exploring and mapping environments of different sizes.

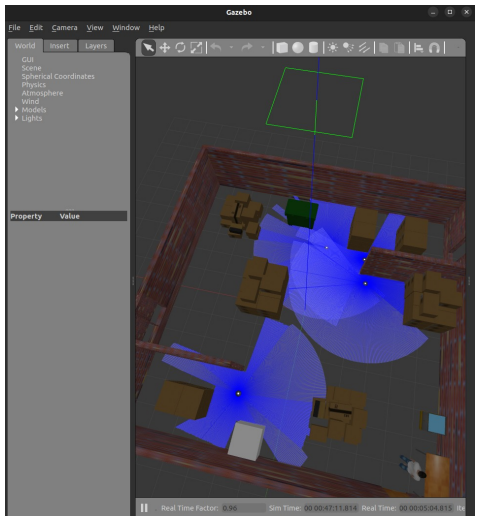

Fig. 10. Scalability using four bots

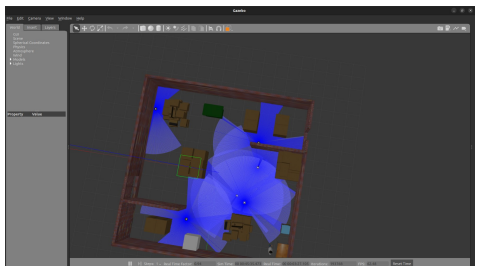

Fig. 11. Scalability using six bots

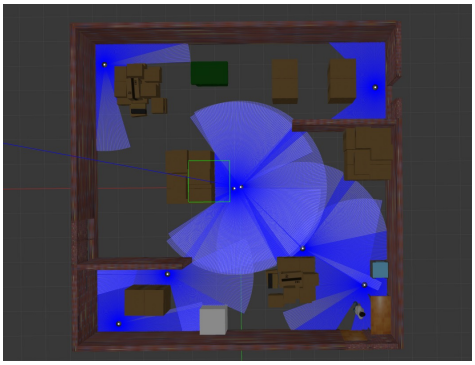

Fig. 12. Scalability using eight bots

Future Prospects

The system could use advanced machine learning to learn from past actions and plan better paths. Robots could collaborate to explore large areas efficiently, like warehouses or rescue sites. Adding sensors like thermal cameras could improve obstacle detection and adaptability. The system could also serve specific needs like farm mapping or guiding vehicles without GPS. With real-time updates and cloud processing, it would become smarter, faster, and more accurate over time.

Conclusion

The autonomous navigation system leverages a combination of ROS (Robot Operating System), hardware sensors, and a frontier-based exploration (FBE) algorithm to autonomously explore and map unknown environments. The robot uses LIDAR, and the FBE algorithm identifies frontier points, or boundaries between known and unknown areas, to guide the robot's exploration. It plans the optimal path to these frontiers using algorithms like A*, avoiding obstacles along the way, and continuously updates the occupancy grid as it explores. Multiple robots can collaborate to explore the environment more efficiently by sharing data and avoiding redundant exploration. Dynamic obstacles are handled by real-time adjustments to the robot's path. Once all frontiers are explored, the environment is fully mapped, marking the completion of the task.

Exploration for a single robot system in our context considers uncertainty reduction in an unknown environment performed by a single agent. It involves the system making decisions about where it should go to acquire information and learn about the environment. Multi-robot exploration extends the principle of individual exploration by enabling multiple robots to collaborate and coordinate their actions. This requires them to assess how their collective behaviors contribute to acquiring information and improving the shared understanding of the environment. The interplay between coordination and distribution is crucial, as robots must plan and execute their actions in a way that optimizes the overall system's efficiency in mapping and exploration.

References

1. Hungarian Maximum Matching Algorithm — Brilliant Math & Science Wiki — brilliant.org. `https://brilliant.org/wiki/hungarian-matching/`. [Accessed 05-12-2024].
2. OpenRMF - An Open Source Risk Management Framework tool — openrmf.io. `https://www.openrmf.io/`. [Accessed 05-12-2024].
3. Giorgio Grisetti; Cyrill Stachniss; Wolfram Burgard. Gmapping.
4. Andrea Censi. An icp variant using a point-to-line metric. In *2008 IEEE International Conference on Robotics and Automation*, pages 19–25, 2008.
5. Haobo Feng, Qiao Hu, Zhenyi Zhao, and Xinglong Feng. Smooth path planning under maximum curvature constraints for autonomous underwater vehicles based on rapidly-exploring random tree star with b-spline curves. *Engineering Applications of Artificial Intelligence*, 133:108583, 2024.
6. Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.
7. Erkan Uslu, Furkan Çakmak, Muhammet Balcılar, Attila Akıncı, M. Fatih Amasyalı, and Sırma Yavuz. Implementation of frontier-based exploration algorithm for an autonomous robot. In *2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*, pages 1–7, 2015.
8. Shuien Yu, Chunyun Fu, Amirali K. Gostar, and Minghui Hu. A review on map-merging methods for typical map types in multiple-ground-robot slam solutions. *Sensors*, 20(23), 2020.