Low Prep problem statement

Centralized Intelligence for Dynamic Swarm Navigation

Final Evaluation Report Team

62

Contents

1 Executive Summary

Overview of the Project

This project is focused on developing a multi-robot system capable of performing complex tasks in dynamic, large-scale environments. We have utilized turtlebot4 and the ROS2 framework to ensure that the system remains modular, scalable, and efficient. The primary goal is to achieve seamless integration of simulation environments, multi-robot coordination, and advanced algorithms for exploration, mapping, and task execution.

By eliminating the reliance on external markers and GPS-based navigation, this system seeks to enhance both efficiency and scalability across diverse environments. Additionally, the robots will continuously update and store a shared map/database of environmental changes, enabling them to learn and adapt as they navigate.

2 Methodology

2.1 Multi-Bot Spawning

The Multi-Bot[2] package facilitates the seamless deployment of multiple robots within a single simulation environment[3]. Key features include:

Namespace Management: Each robot operates within its own namespace (e.g., /robot1/cmd-vel and /robot2/cmd-vel), ensuring isolation of topics, services, and parameters. This design prevents conflicts, enabling precise control and independent operation of each robot.

Modular Design: The package's modular design allows the reuse of the same node configurations across multiple robots, simplifying both debugging and scaling processes.

RViz Integration: Separate RViz instances are launched for each robot, providing real-time visualization of maps and sensor data.This configuration is very useful for jobs that include mapping and navigation.

By combining namespaces with RViz visualization, this component ensures that each robot maintains its own data stream while avoiding interference. These features lay a robust foundation for multi-robot coordination.

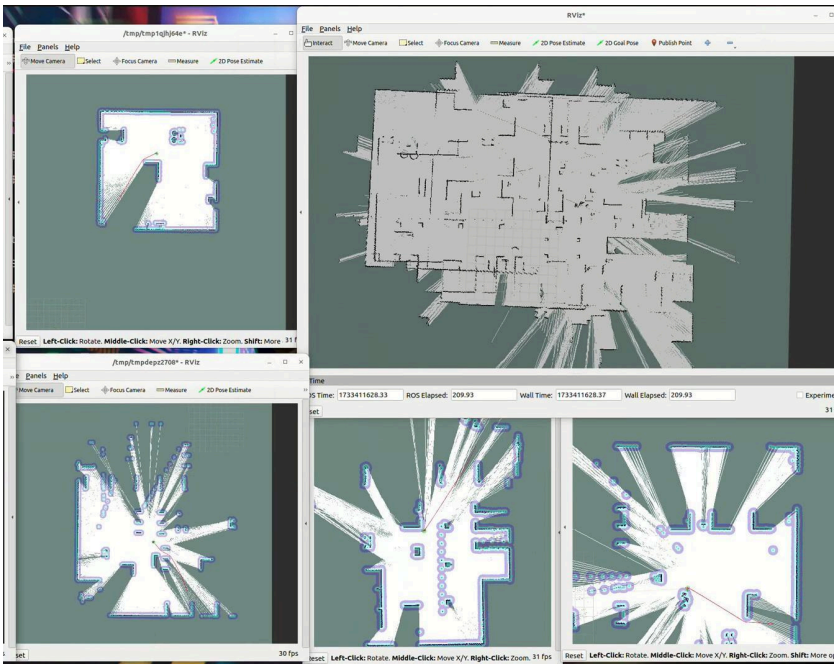

Figure 1: Running the Map Merge for 6 robots mapping different sections simultaneously.

2.2 Map Merging and Integration

Merging Pipeline:

- **Features detection:** The algorithm uses feature extraction techniques to identify distinctive features within the maps (or grids). This is done using cv::detail:: computeImageFeatures,which detects corner-based and edge-based features.
- **Features Matching:** After detecting features in each map (or grid), the algorithm uses a feature matching algorithm to find corresponding features between pairs of maps using cv::detail::AffineBestOf2Neares which finds the best matches between maps. This involves identifying points in one map (or grids) that correspond to points in another map, based on their feature descriptors.
- **Estimating Transformations:** The core step in the map merging process is estimating the transformations (or relative poses) between the grids using the matched feature points.
- **Bundle Adjustment:** After the initial estimation of the transformations, the algorithm performs bundle adjustment (cv::detail::BundleAdjusterAffinePartial), which is a refinement step that optimizes the transforms to minimize errors and improve accuracy. This step ensures the best possible alignment of all the maps (grids) in a global reference frame.

The map_merge package leverages ROS 2 to facilitate multi-robot map merging. Two launch files, map_merge_server.launch and map merge.launch, handle the integration process.

Code Overview:

- map_merge_server.launch:
  - Subscribes to individual robot map topics, dynamically configuring up to two maps for merg- ing.
  - Uses the nav2 map server and nav2 lifecycle manager nodes to manage map inputs.
  - Implements group actions to handle separate namespaces for each robot, ensuring isolated configurations.
  - Enables RViz visualization by launching the rviz2 node with a preconfigured layout.
  - Configurations, such as known initial poses and simulation time, are declared as launch argu- ments for flexibility.
- map_merge.launch:
  - Streamlines the map merging process by focusing on the map merge node.
  - Utilizes remappings for tf and tf static topics to ensure compatibility with the robot state publisher.
  - Handles essential configurations like simulation time and known robot poses, defined in the params.yaml file.

This process significantly enhances localization and navigation capabilities, particularly in large-scale or complex environments, by ensuring that all robots share a unified and up-to-date understanding of the workspace.

2.3 Autonomous Exploration

The Explorer package drives autonomous exploration by directing robots to frontiers—the boundaries between explored and unexplored areas.

Key Features: Central to this process is the concept of *frontiers*, which are defined as the boundaries between already explored areas and unexplored regions. These frontiers are crucial as they represent the most valuable regions for exploration, offering new sensor data to expand the map. The package identifies frontier regions using the robot's position and sensor data (e.g., LiDAR or depth cameras) and dynamically assigns Nav2 goals to guide robots toward these areas. This system continuously updates the frontiers as robots explore, ensuring real-time adaptation to changing environments and maximizing exploration coverage.

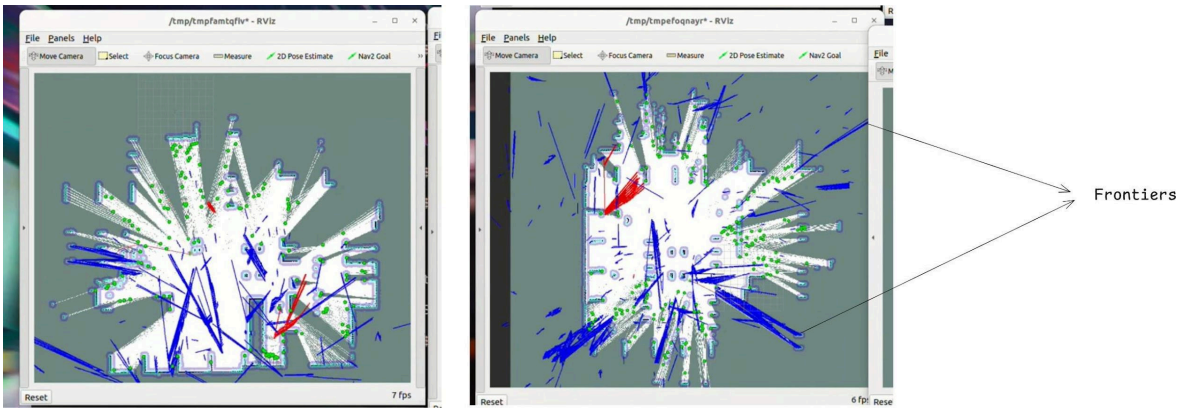

Figure 2: Frontier Search Running for each robots and finding out possible unexplored areas

2.3.1 FrontierSearch Module

The FrontierSearch[4] module utilizes costmap data to identify frontier cells. Key functions include:

- **Search and Discovery:**
  std::vector<Frontier> FrontierSearch::searchFrom(geometry msgs::msg:
  :Point position)
  The robot's position is used to search the environment, using a breadth-first search (BFS) to identify the closest free cells and new frontier cells (unexplored and adjacent to free space).

- **Building and Evaluating Frontiers:**
  Frontier FrontierSearch::buildNewFrontier(unsigned int initial cell, unsigned int reference, std::vector<bool> and frontier flag)
  Once a new frontier is identified, the system forms a new frontier by adding neighboring unexplored cells. It then calculates the frontier's size and centroid to prioritize the most accessible and valuable regions. The frontierCost function evaluates the frontier's value based on proximity and size, ensuring the robot moves toward the most informative areas for map expansion.

By dynamically updating frontiers, this system optimizes exploration, reducing redundancy and maximising efficiency in large or dynamic environments

References

[1] Matt Deitke et al. *Molmo and PixMo: Open Weights and Open Data for State-of-the-Art Multimodal Models*. 2024. arXiv: 2409.17146 [cs.CV]. url: https://arxiv.org/abs/2409.17146.

[2] Robo Friends. *https://medium.com/@arshad.mehmood/a-guide-to-multi-robot-navigation-utilizing-turtlebot4-and-nav2-cd24f96d19c6*. Accessed: 2024-12-05. 2024. url: https://github.com/NU-IDEAS-Lab/patrolling_sim.

[3] Robo Friends. *https://medium.com/@arshad.mehmood/efficient-deployment-and-operation-of-multiple-turtlebot4-robots-in-gazebos-f72f6a364620*. Accessed: 2024-12-05. 2024. url: https://github.com/NU-IDEAS-Lab/patrolling_sim.

[4] Robo Friends. *m-explore-ros2*. Accessed: 2024-12-05. 2024. url: https : / / github . com / robo - friends/m-explore-ros2.

[5] Robo Friends. *m-explore-ros2*. Accessed: 2024-12-05. 2024. url: https://github.com/NU-IDEAS-Lab/patrolling_sim.

[6] Woosuk Kwon et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention". In: *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*. 2023.

[7] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV]. url: https://arxiv.org/abs/2103.00020.

[8] An Yang et al. *Qwen2 Technical Report*. 2024. arXiv: 2407.10671 [cs.CL]. url: https://arxiv.org/abs/2407.10671.