Centralized Intelligence for Dynamic Swarm Navigation

Inter-IIT Tech Meet 13.0 - *BharatForge*
Team B*

December, 2024

The Problem Statement

The problem focuses on designing a Singular Brain for a robot swarm tasked with performing optimized path planning in highly dynamic environments. The challenges involve developing and training a multi-agent system capable of navigating in environments where:

- **No GPS** is available.
- **Frequent and unpredictable changes occur**, such as moving obstacles or rearranged objects.

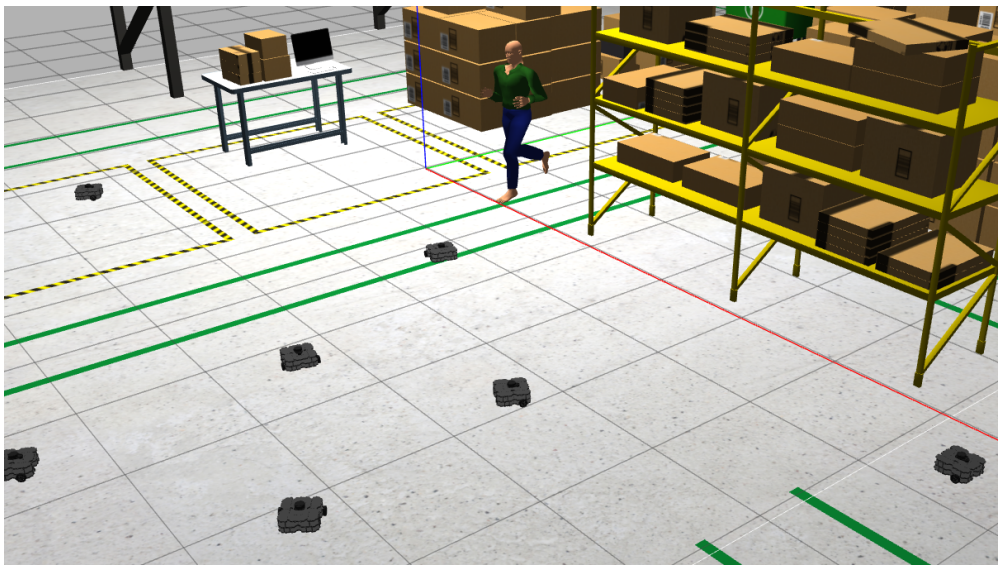
The robot swarm must collaboratively plan the optimal path to accomplish a set of tasks while avoiding collisions and delays. Efficiency is critical, requiring:

- Dynamic **ranking of navigation tasks** based on travel time.
- **Memory persistence**, where robots label and store dynamic changes in the environment in a shared, continuously updating database.

The ideal solution should be:

- **Scalable**, maintaining efficiency even in new, larger environments with additional robots.
- **Adaptable**, enabling the swarm to autonomously explore any environment and create a dynamic database/map of object locations without manual intervention.

The ultimate objective is to develop a software solution capable of ensuring optimal swarm performance, even under highly dynamic and complex conditions.



Our Solution

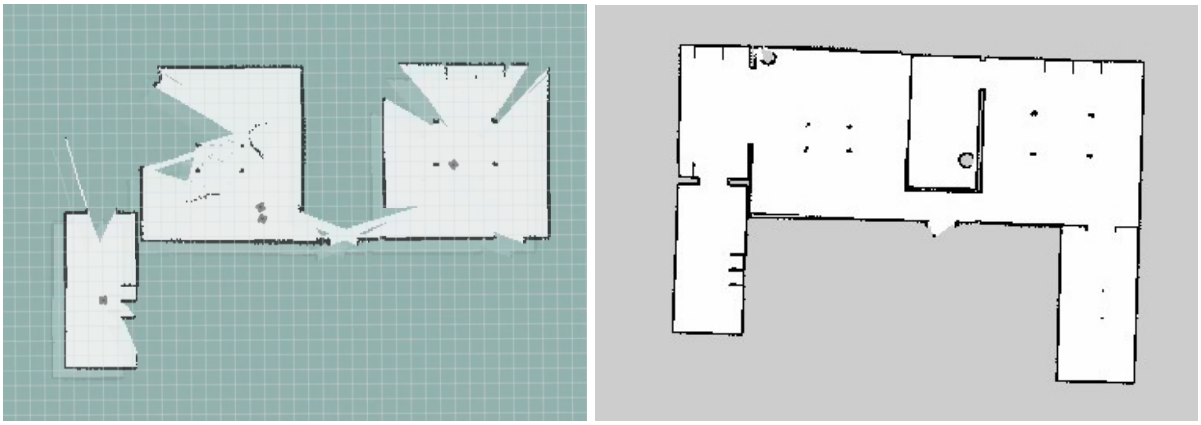
Our solution is a complete package (Both literally and figuratively) and presents a **framework** for a **swarm-based surveillance** and **task-completion system** using **autonomous** robots. The system is designed to monitor large areas and detect obstacles through a coordinated swarm of robots.

Q. What "**framework**" are we using to simulate/run this system?

Ans We are using **ROS** (Robotic Operating System - noetic) on Ubuntu 20, which is the standard for many robotics applications at an industrial and commercial level. We are Also using **Gazebo** in order to simulate a large collection of robots along with the built in **RVIZ** tools.

Q. What are we using to map our surrounding and provide a "**swarm-based surveillance**"?

Ans Every robot maps the surrounding using LiDAR (2D) data, this was done using **slam_gmapping**, but the challenge was to take the 'local' maps created by 'n' robots and merge them into one 'global' map which would be used by all the robots, then the robots would explore their surroundings and create a map. The map merging part was implemented by using the **map_merge** package and then we had to implement this technology with **explore_lite** to create an autonomous map.



(a) Robots exploring map in *'idle'* state. (b) The explored map saved by *map_server* package.

Figure 1: Mapping the environment in real time which are are going to navigate in.

The robots then share this map on topic made by **map_server**. In our solution, if we want then we can also save this map after which we can load later for either further exploration or if we want then we can also preload an explored map.

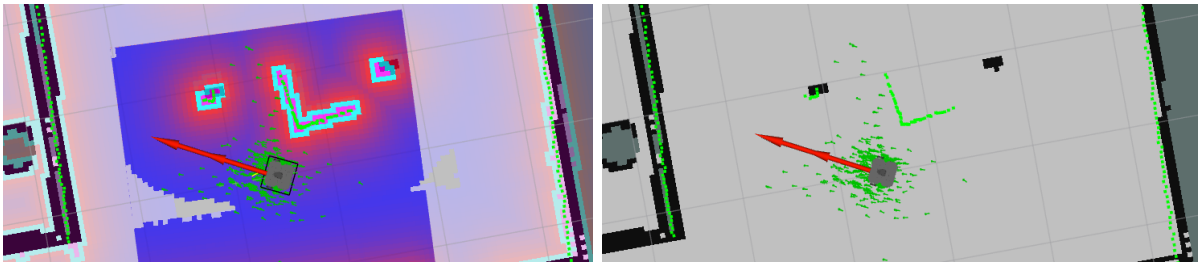
Q. How are you ensuring "**task-completion**" given that there are dynamic obstacles and a large environment?

Ans We understand that the environment given to us is *very* dynamic and also *very* large. So, we ensure that the robots detect and avoid on a "local" level, i.e. every robot not only has a "*global costmap*" which it uses for long distance navigation, but also a rapidly changing "*local costmap*" which a robot uses to avoid nearby obstacles.

Given this, the robot will first try to loacte itself on the map by using **AMCL** (Adaptive Monte-Carlo Algorithm) (Note that we are using this because we don't have GPS) and then by using the **move_base** package will plan "***a route from point A to B by avoiding obstacles and without getting lost on the way***".

Q. What is so "**autonomous**" about this? How are you detecting tasks? What algorithm are you using to scale task distribution to multiple robots?

Ans Our integrated robotics system is designed to execute the task of navigation, obstacle avoidance, task allocation autonomously. Once a task is manually typed or given using a chatbot interface, we



(a) *local* costmap. (b) *global* costmap

Figure 2: Robot detecting dynamic obstacle in local costmap while global costmap remains unchanged.

map the given task to its coordinates using a centralized database which provides us the coordinates, and then we use the **Hungarian algorithm** to distribute tasks between multiple robots. Each robot is equipped with a state-of-the-art object detection system utilizing the **YOLO** model. This model enables real-time object recognition by analyzing images captured by the robot's cameras. Then using our LiDARs we find out the coordinates of that object and store it in our database.

Our Algorithms and Technologies

The technologies we employ are crucial for the functioning of our robotic swarm. Here's a detailed explanation of each:

- **move_base** manages navigation and path planning, considering both local and global cost maps to guide robots safely from point A to point B, avoiding obstacles dynamically. This package provides the interface for configuring the trajectory planner and deals with the intricacies of dynamic path planning based on incoming sensor data and the existing environmental model.

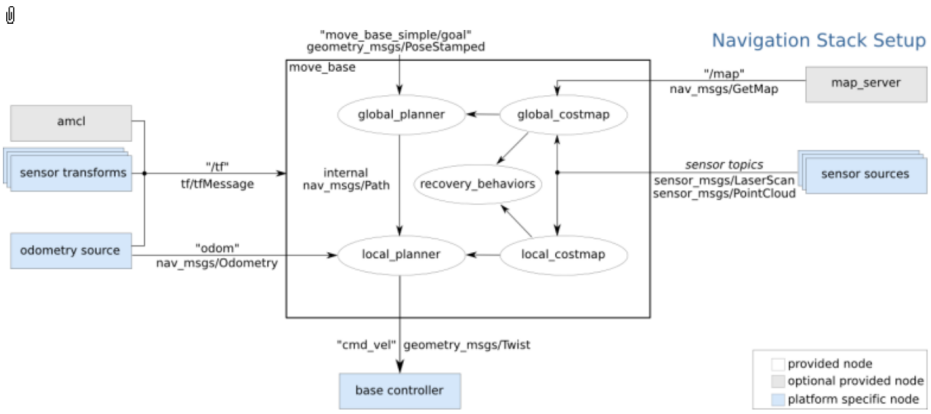


Figure 3: Flow of move_base (*http://wiki.ros.org/move_base*)

- **map_merge** is utilized to integrate individual maps from multiple robots into a cohesive global map, essential for coordinating activities across the swarm. This tool is vital for swarm robotics as it allows for a unified view of the explored area, ensuring that all units operate under a shared spatial understanding, which is critical for efficient task execution and navigation.
- **explore_lite** directs robots to systematically cover all accessible areas, updating the map with newly discovered information and aiding in efficient exploration and surveillance. This package is designed to be lightweight and efficient, enabling robots to autonomously explore and map environments without human intervention.

- **The Hungarian algorithm** $O(t^2n)$ is critical for distributing tasks efficiently among the robots, ensuring each unit's operations are optimally aligned with the overall objectives of the swarm, minimizing redundancy and maximizing coverage. This combinatorial optimization algorithm solves the assignment problem, matching robots to tasks in a way that optimizes the overall efficiency and effectiveness of the swarm's operations.. Here 't' are the tasks and 'n' are the robots.

Worker \ Task	Clean bathroom	Sweep floors	Wash windows
Alice	**$8**	$4	$7
Bob	$5	$2	**$3**
Carol	$9	**$4**	$8

Figure 4: Example of optimization (*https://en.wikipedia.org/wiki/Hungarian_algorithm*)

AI/ML Models used

We are using the **YOLO** (You Only Look Once) model to detect the object present in our environment. It is a pretrained model which comes with weights and a large set of classes.

(a) Waffle Pi robot camera detecting person in gazebo. (b) YOLO falsely identifying "*cartoonish*" objects.

Figure 5: Integrated AI Model to avoid use of ArUco Markers

We first detect object using our camera, after which we find out exacty where it is using *odometry* and *LiDAR data*. Since we know the FOV (Field of View) of camera, we first find out where on the screen a particular object is, after which we find out the depth at a particular angle (nearest angle which has the LiDAR data) which will, given the odometry data of the robot, tell us the coordinates of the object which we will update in the database.

By doing this we get all the locations of all the detectable objects on the global map.

Our Package Architecture

Below is the architecture of our package:

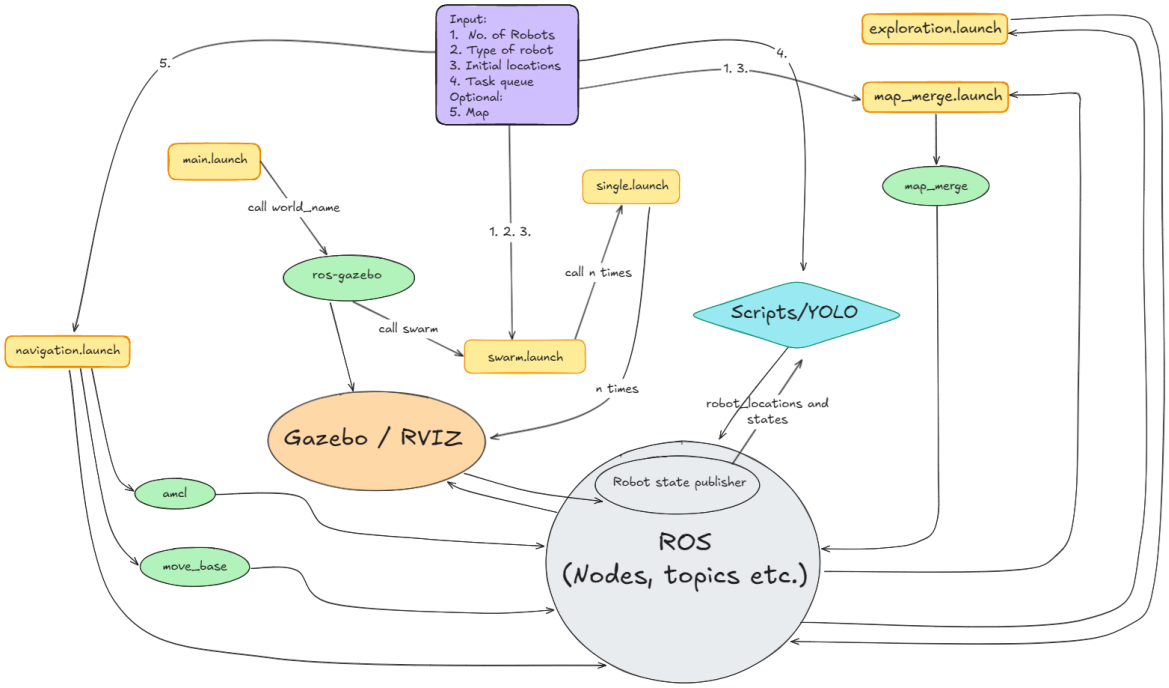


Figure 6: Flow of our swarm_nav_inter_iit

Our implementation has the capacity to deploy any environment given to us and deploy any no of robots in specific locations and code has been made in order to give independent commands to a specific robot. Our implementation architecture allows for deploying very big and dynamic environments and allows for **scaleability** which is one of the main criterion.

Also given any algorithm, this allows for communication between all the robot nodes essentially making the **singular brain** which can run any algorithm.

References:

- http://wiki.ros.org/move_base
- http://wiki.ros.org/amcl
- https://en.wikipedia.org/wiki/Hungarian_algorithm
- https://emanual.robotis.com/docs/en/platform/turtlebot3/nav_simulation/
- https://emanual.robotis.com/docs/en/platform/turtlebot3/slam_simulation/