Kalyani Bharat Forge

Final Report: Team_73

"Centralized Intelligence for Dynamic Swarm Navigation"

The topics listed (contents) as well as references in the document are clickable.

Contents

1 Introduction

1.1 Problem Statement

The problem statement focuses on designing a centralized intelligence for dynamic robot swarm navigation , with performing optimized path planning in highly dynamic environments. Autonomous mobile robots are increasingly being deployed in industrial environments for tasks such as maintenance, inspection, and material handling. This problem statement addresses this by developing a robust and intelligent navigation algorithm for AMRs using techniques such as Machine Learning (ML), Deep Learning (DL), Reinforcement Learning (RL) or hybrid approaches.

The goal is to enable AMRs to navigate efficiently in dynamic environments characterized by frequent and unpredictable changes, such as moving obstacles and rearranged layouts. The software should empower AMRs to achieve optimized path planning, efficient task execution, and adaptive learning without manual intervention, ensuring seamless operation in diverse environments. The system should be responsive in real time, environmentally adaptive, and flexible with swarm size. And at last, developing an LLM-based chatbot interface for ease of assignment of navigation tasks and receiving feedback of which robot is performing which task.

1.2 Solution Overview

The proposed solution incorporates a dual-layered architecture: for control of large swarms they found a master-slave type system while for small swarms, they opted for complete autonomy. LiDAR and cameras are incorporated into robots that adaptively create a new map of the surroundings and learn to move towards targets designated by RL strategies. The solution now combines two RL techniques: PPO and TD3, allowing for variable and effective navigation approaches. While navigating, each drone creates a common map database with the obstacle type and location; if they are static or dynamic, object identification is done using a YOLOv8 Nano Object Detection Model. This shared map is stored in a NoSQL MongoDB database suitable for document-based, real-time data generated by robots.

This is because the inclusion of PPO and TD3 as policy optimisation algorithms has its merits afresh. PPO is used since it is reliable and samples efficient especially in a environment with continuously appearing and disappearing obstacles. It navigates the robots to proper pathways while at the same time ensuring they do not crash through LiDAR and camera information. TD3 complements PPO particularly in situations where continuous control and accurate movements are needed, i.e., when it is necessary to move through tight spaces or avoid pushing multiple dynamic obstacles at once. Thus the dual RL approach results in better a system ability to approach a wide range of problem with better performance and stability .

The use of YOLOv8 Nano model complies with computations in object recognition and it runs on an NVIDIA Jetson platform, and provides correct simulation object categorisation. In the current work, the dataset was prepared towards

capturing diverse perspective from a TurtleBot in a simulated environment with the annotations done manually for the training of the model. Feature transformations involved scaling down images, cropping, and image augmentation measures in order to improve generalization.

2 Literature Review

Designed for scalable and efficient multi-robot systems, Swarm-SLAM is a decentralised collaborative SLAM framework. It supports sensors like stereo cameras, RGB-D cameras, and LiDAR, utilising a two-stage loop closure detection process. Global descriptors identify overlaps, prioritised with spectral sparsification, while local matching ensures accuracy [4]. Dynamically elected robots optimise pose graphs using a Graduated Non-Convexity solver, while anchors ensure alignment to a global frame. This framework minimises resource usage and delivers high accuracy in diverse scenarios. The thesis [4] also talks about a decentralised multi-robot SLAM system that uses the Sparse Extended Information Filter (SEIF) to make mapping work well. Each robot independently builds maps, employing sparsification for scalability. Map merging aligns local frames through common landmarks, validated in Gazebo simulations. The system demonstrates robust, scalable performance in dynamic and constrained environments.
[1] Reinforcement learning (RL) enhances Simultaneous Localization and Mapping (SLAM) by optimizing navigation and map construction in unknown environments. Reinforcement Learning (RL) frames Simultaneous Localisation and Mapping (SLAM) as a Markov Decision Process, evaluating three reward functions: sparse, map-completeness, and information-gain. RL agents are better at finding the best paths and finishing maps than traditional frontier-based exploration when rewards are given for map completion or information gain. This is especially true when Rao-Blackwellized particle filters are used for accurate pose estimation and mapping. Deep Recurrent Q-Networks (DRQNs) further improve performance by leveraging memory in partially observable environments. These advancements highlight the potential of RL in Active SLAM, enabling efficient multiagent exploration and real-world applications.
The paper by Gheorge et al. [2] highlights YOLO's efficiency in real-time object detection due to its single-stage architecture, which balances speed and accuracy. YOLOv8 emerges as the most effective, achieving a mean mean average precision (mAP) of 0.99 in various tasks such as traffic monitoring, vehicle detection, and manufacturing defect identification. These tasks involve detection of moving objects and hence suit our requirements.

3 Methodology

3.1 Environment and bot specifications

The process for generating simulation environments was automated. We focused on 2 types of Environments: warehouse and office. For each environment, we created a Python Script that takes the following input from the user: i) Size of World ii) Number of Static Obstacles iii) Number of Dynamic Obstacles. The script generates environments based on these inputs and creates a .world file. Each environment type includes predefined obstacle categories.
Warehouse : Static (Cardboard Box, Bookshelf, Table, Cabinet, Fire Hydrant) dynamic (Human)
Office : static (Conference Table, Chair, Desk, Sofa, Lamp, Dustbin) Dynamic (Human)
The robot is equipped with **LiDAR** and **Velodyne** sensors for mapping and obstacle detection, a **camera** for object recognition, and an accelerometer and magnetometer for motion tracking and orientation. This advanced sensor suite enables efficient navigation and task execution in dynamic, obstacle-rich environments.

3.2 Swarm Architecture

Three distinct designs have been developed for the Swarm Architecture.

3.2.1 Complete Autonomous

In this architecture, each robot in the swarm works individually with complete autonomy in terms of planning and execution of tasks and motion. Every robot decides its environment, selects the tasks to be done, and computes the best route to complete them without being commanded or coordinated from other robots. This decentralised approach makes the swarm more robust and adaptable, as there's no single point of failure. Without hierarchical dependencies, the swarm can sustain high performance under dynamic and unpredictable environments.However, such autonomy involves equipping the robots with advanced onboard algorithms for decision-making, sensor integration, as well as communication. Although the robots can work autonomously, there is a rise in the computational demand and cost generally.

3.2.2 Master-Slave With Liveliness

The original master-slave architecture involves one master robot and several slave robots. The master robot is responsible for task allocation and strategic decision-making, while the slave robots execute commands without independent task or motion autonomy. However, a major drawback in this architecture is that the whole system relies on the master bot, and in case of failure of the master robot, the whole system can collapse. To solve this, a Master-Slave architecture We designed the Master-Slave architecture that incorporates Liveliness. In this architecture, each robot periodically sends liveliness requests to the master to confirm operational status. If a master or a slave node becomes unresponsive, other robots adapt dynamically—for example, if the master node becomes dead, one of the slave nodes can assume the role of the master. This design enhances the reliability of the system, ensuring smooth running even in the event of node failure.

3.3 Object Detection and Labeling

3.3.1 Overview

The object detection part of our solution employs YOLOv8n, a more lightweight, real-time deep learning model appropriate for employing on edge computing devices, such as the TurtleBot, included with a Raspberry Pi 4. The Raspberry

Pi equipped with quad-core ARM Cortex-A72 CPU and VideoCore VI GPU can perform limited computational power hence making YOLOv8n suitable as it consumes lesser computational resources and can provide real-time detection with reasonable accuracy. Proposed models as Faster R-CNN, EfficientDet, and YOLOv11 were not used because of high computational costs that would negatively impact other processes for example reinforcement learning. To achieve the best graph search with YOLOv8n, three training iterations were performed with the step-by-step training dataset refinement, data preprocessing, and training algorithm enhancing to match our application.

3.3.2 Dataset Preparation

To generate a custom dataset for the object detection task, the TurtleBot was maneuvered within a Gazebo simulation environment along predefined coordinates. Upon reaching each target location, the robot performed a 360-degree rotation, capturing images of the surrounding environment from every angle. This motion was controlled using a Python script (move.py), which ensured precise navigation along the designated path. Simultaneously, another script (image_save.py) was employed to manage the timing of the image captures, ensuring uniform intervals between consecutive frames. This process allowed for the systematic collection of a diverse set of images from various perspectives, contributing to the robustness of the dataset used for training the YOLOv8n model.

Dataset Split: Train: Validation: Test = 0.7 : 0.2 : 0.1

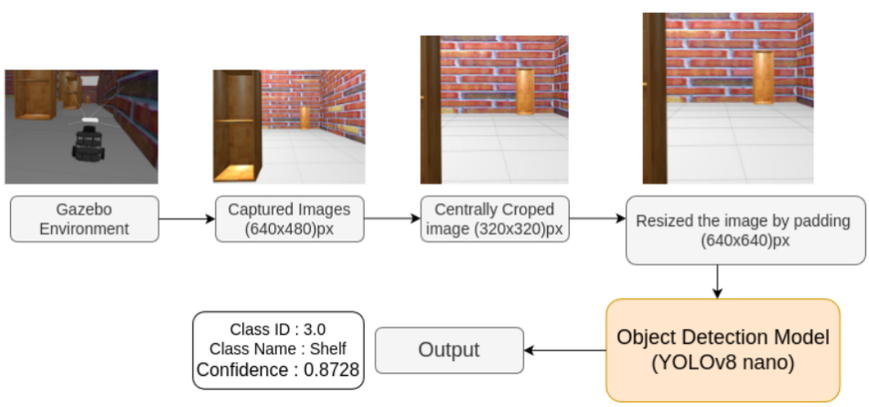


Figure 3.1: Flow chart of the object detection model

3.4 Training approaches

The training is done mainly through 3 approaches:

- **Approach 1:** A dataset of 864 images, each of size 640×480 pixels, is used. Preprocessing was conducted to resize all the images to 640×640 pixels as required by the YOLOv8n model architecture. Images were padded with black edges to maintain aspect ratio consistency during training. Augmentation of horizontal flip was applied to improve generality and size of the dataset. The training was conducted over 25 Epochs with a batch size of 4.
- **Approach 2:** We created another dataset larger in size with 1788 images and the same preprocessing steps as earlier. To improve accuracy, the model from the first attempt was retrained with adjusted hyperparameters, over 50 epochs, and a batch size of 4.
- **Approach 3:** Upon deploying the trained model in the simulated environment, it was observed that performing inference on the entire frame to detect all objects was unnecessary. The primary requirement was to detect objects directly in front of the TurtleBot. To optimize computational efficiency and reduce inference time, the preprocessing pipeline was refined. The original 640×480 pixel images were centrally cropped to focus on a 320×320 pixel region, capturing only the relevant portion of the frame. This cropped region was then resized to 640×640 pixels with padding to maintain compatibility with the YOLOv8 model. The augmentation techniques remained consistent with previous attempts. The model from the second training iteration was subsequently retrained using these updated preprocessing steps, with 50 epochs and an increased batch size of 16.

Attempt	Dataset Size	Epochs	Batch Size	Preprocessing
1	464	15	4	Resized with padding Horizontal flip augmentation applied.
2	1,788	50	4	Resized with padding Horizontal flip augmentation applied.
3	1,788	50	16	Cropped and resized Horizontal flip augmentation applied.

Table 1: Training Attempts

3.5 Shared Database

The swarm robots use LiDAR sensors, cameras, magnetometers, and accelerometers to navigate an unknown environment while performing assigned tasks. Initially guided by a reinforcement learning (RL) model, the robots collaboratively build a shared map of the environment in real-time. This map, stored in a NoSQL database (MongoDB), documents each coordinate with information about obstacles, including their presence (obstacle is present or not), type (static or dynamic), and identity if static (identified through a specific object recognition model). It is continuously updated, which assures that all robots in the swarm can navigate efficiently and perform tasks. Because MongoDB is a type of NoSQL database that manages unstructured and semistructured data efficiently, which are necessarily documents rather than relational by their nature, it is appropriate for our system.

Collection	**Field**	**Description**
Task Collection	Task ID	Unique identifier for each task.
	Textual Description	Detailed description of the task.
	Goal Coordinate	Target position (x, y).
	Status	Task completion status (Pending/Completed).
	Assigned to	Robot ID or group assigned to execute the task.
	Type	Task type (Goal-based/Routine-based).
	Priority	Priority level of the task (e.g., High, Medium, Low).
	Assignment Time	Records the time of assignment of task
	Completion Time	Records the time of completion of task
Map Collection	Coordinate	Tuple (x, y) representing a location on the map.
	Obstacle	Boolean (True/False) indicating whether the coordinate has an obstacle.
	Label	If an obstacle exists: Static or Dynamic classification.
	Timestamp	For dynamic obstacles, records the time the obstacle was detected.
Robot Collection	Robot ID	Unique identifier for each robot in the swarm.
	Position	Current location of the robot, typically (x, y) coordinates.

Table 2: Shared Database Representation

Table 2 provides a detailed representation of the shared database used in the system, structured into three collections: **Task Collection**, **Map Collection**, and **Robo Collection**.

- **Task Collection:** This serves as a storage repository for all tasks that are currently pending in the system. It keeps track of details such as task priority, task ID, assignment and completion times, robot ID, and status.
- **Map Collection:** The Map Collection holds all relevant information about the robot's working environment, including the presence and type of any objects.
- **Robot Collection:** This collection monitors each robot's location by correlating their coordinates with their respective robot ID..

This shared database structure enables efficient task assignment, obstacle detection, and robot positioning in the system, facilitating effective coordination and navigation.

3.5.1 Task Assignment

The bots will regularly check the shared database for any pending tasks. Based on the available task list, prioritization is performed. If a task remains pending and the corresponding bot is deemed the most suitable for completing it based on proximity to goal, the bot will take on the task. Task prioritization is determined by factors such as the bot's current position relative to the task's goal coordinates and the priority level of the task, as illustrated in the example table. 3

Task	**Priority**
Fire Alert: Go to fire hydrant	1
Patrolling	2
Identify items in shelf and cabinet	3
Inspect the carton	4
Go to conference table	5
Go to person 1	6

Table 3: Task Priorities

3.6 Path planning and Collision Avoidance

Path planning and collision avoidance are achieved mainly through 2 methods:

3.6.1 Traditional

Traditional methods like Dijkstra's algorithms and A* use mathematical principles to determine safe routes. These methods work well in surroundings that are static and structured, but they are ineffective in dynamic situations where the surrounding environments change. Improvised versions of A* like Rapidly-Exploring Random Trees (RRT) add flexibility by exploring multiple path options, but they still fall short in handling real-time changes.([7])

3.6.2 Reinforcement Learning (RL)

Reinforcement learning (RL) provides a dynamic solution whereby robots learn navigation strategies through a process of learning from the environment. This is in contrast to the traditional approaches where all the scenarios are programmed into the robots. The robots search their environment to determine the best course of action which improves as they gain experience. This is the case in dynamic and unstructured environments where obstacles and conditions are changing constantly and therefore, RL is suitable for such conditions. Continues learning algorithms like PPO and TD3 make it possible for the robots to adapt to new environments and tasks thus improving the performance of the swarm navigation systems.

Based on the comparisons between traditional approaches and reinforcement learning (RL)-based methods([3]), RL approaches demonstrate better adaptability, precision, and efficiency, particularly in dynamic and unstructured environments from the table 6 in appendix. Thus, for furthur work, we will make use of RL approches for Path planning and Collision Avoidance.

Proximal Policy Optimisation (PPO) : It is an advanced reinforcement learning algorithm that is well-suited for environments requiring continuous action spaces, such as robotic control in dynamic and complex settings[9][8]. PPO strikes an effective balance between exploration and exploitation by using a clipped objective function to restrict the magnitude of policy updates. This characteristic stabilises learning by preventing drastic policy changes that can lead to instability. PPO is especially valuable in multi-agent systems, such as swarm robotics, where agents must adapt to changing environments, avoid dynamic obstacles, and adjust to real-time task assignments. The algorithm's robustness in handling noisy environments and its ability to improve policy efficiency make it a strong choice for swarm navigation tasks, enabling autonomous robots to perform optimally while ensuring stability during learning 7.3.

Key Variables

- $\pi_\theta(a_t \mid s_t)$: Policy probability of taking action a_t given state s_t, parameterised by θ.
- $\pi_{\theta_{\text{old}}}(a_t \mid s_t)$:Policy probability under the old policy (before the update).
- A^t :Advantage estimate at time.t.
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$: Probability ratio for policy update.
- ϵ: Hyperparameter controlling clipping range (e.g., $\epsilon = 0.2$).
- $v_\theta(s_t)$: Value function estimate for state s_t.
- v^t_{target}: Target value of state s_t (calculated using returns).
- c_1, c_2: Coefficients for value loss and entropy bonus terms.
- $S[\pi_\theta](s_t)$: Entropy of the policy at state s_t to promote exploration.

Objective Functions

- **Clipped Policy Surrogate Objective:**

$$L_{\text{CLIP}}(\theta) = E_t\left[\min\left(r_t(\theta)A^t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A^t\right)\right]$$

  $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ restricts $r_t(\theta)$ to $[1-\epsilon, 1+\epsilon]$, avoiding large updates.

- **Value Function Loss:**

$$L_{\text{VF}}(\theta) = Et\left[\left(v\theta(s_t) - v^t_{\text{target}}\right)^2\right]$$

- **Entropy Bonus:**

$$S[\pi_\theta](s_t) = -\sum_a \pi_\theta(a \mid s_t)\log\pi_\theta(a \mid s_t)$$

- **Combined Objective:**

$$L_{\text{PPO}}(\theta) = Et\left[L\text{CLIP}(\theta) - c_1 L_{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t)\right]$$

  Here, c_1 controls the weight of the value function loss, and c_2 controls the entropy bonus weight.

PPO Algorithm Steps

1. **Initialize Policy Parameters:** Start with policy π_θ and value function $v_\theta(s_t)$.
2. **Collect Trajectories:** Execute π_θ to gather trajectory data (states, actions, rewards, etc.).
3. **Estimate Advantages:** Compute A^t using Generalized Advantage Estimation (GAE) or another method.
4. **Update Policy:** Optimize $L_{\text{PPO}}(\theta)$ using stochastic gradient ascent:

$$\theta \leftarrow \theta + \alpha\nabla_\theta L_{\text{PPO}}(\theta)$$

   where α is the learning rate.
5. **Repeat:** Iterate until convergence or maximum training steps are reached.

Twin Delayed Deep Deterministic Policy Gradient (TD3):

The TD3 algorithm[5] is an off-policy actor-critic reinforcement learning algorithm for continuous action space that refines the previous discrete action space algorithm, DDPG, in terms of overestimation and instability. It incorporates two key features: There are two complex methods in the paper; Twin Q-Networks to minimize the impacts of bias, and Delayed Policy Updates to enhance the learning stability. These improvements result to better and more consistent value estimation and policy improvements and thus, TD3 is ideal for high precision tasks such as robot navigation in complex terrains. In this report, our proposed system implements TD3-based DRL with ROS and Gazebo simulation environments; the TurtleBot3 robots employs Lidar, camera and velodyne sensor to identify obstacles while employing odometry for estimating pose 7.2.

Key Features

- **State Space:** A 24-dimensional vector encompassing sensor data (Lidar readings) and robot-specific states (position, velocity, orientation).
- **Action Space:** Continuous control inputs for linear and angular velocities.
- **Reward Function:** Designed to prioritize collision avoidance, goal-reaching, and smooth navigation.

Velodyne Data LiDAR data is processed into 20 discrete bins representing distances to the nearest obstacles in different angular segments. This provides a spatial observation vector for collision avoidance.

TD3 MODEL ARCHITECTURE

The TD3 model 7.2 :

- **Actor Network:** Predicts the optimal action given the current state.
- **Critic Networks:** Estimate expected rewards for state-action pairs.
- **Replay Buffer:** stores experiences for off-policy learning.

State and Action Space

1. **State Space (S):** The state space combines observations of the environment and the robot's current status, enabling comprehensive decision-making.
   - **Laser scan data (s_L):** 20-dimensional vector representing distances to obstacles.
   - **Robot state (s_R):** A 4-dimensional vector: $[d_{\text{goal}}, \theta_{\text{goal}}, v_{\text{linear}}, v_{\text{angular}}]$, describing the robot's relative position, orientation, and velocities.

$$S = \{s_L, s_R\}, \quad S \in R^{24}.$$

2. **Action Space (A):** Defines continuous control inputs for robot movement, facilitating smooth navigation in dynamic environments.
   - Two continuous actions: a_1 (linear velocity) and a_2 (angular velocity), each in $[-1, 1]$.

Reward Function (R) Incentivizes reaching the goal while penalizing collisions and unsafe navigation behavior.

$$R = \begin{cases} 100 & \text{Goal reached,} \\ -100 & \text{Collision,} \\ \frac{a_1}{2} - \frac{|a_2|}{2} - \frac{1-\min(\text{laser\_scan})}{2} & \text{Otherwise.} \end{cases}$$

Network Architectures

1. **Actor Network:**
   - Learns deterministic policy: $a_t = \pi_\phi(s_t)$, mapping states to optimal actions.
   - Fully connected layers: $24 \to 800 \to 600 \to 2$, with ReLU activation for hidden layers and Tanh for the output.
2. **Critic Networks:**
   - Two Q-networks $Q_{\theta_1}(s, a)$ and $Q_{\theta_2}(s, a)$, estimate Q-values to evaluate action quality.
   - Architecture:
     - State branch: $24 \to 800 \to 600$.
     - Action branch: $2 \to 600$.
     - Combined: $600 \to 1$, with ReLU activation.

Training Details Q-value Update: The critic minimizes overestimation bias by considering the smaller of the two Q-values for target estimation.

$$Q(s_t, a_t) = r_t + \gamma \min(Q_{\theta_1}(s', a'), Q_{\theta_2}(s', a')).$$

Actor Update: Optimizes policy by using gradients of the Q-value function, improving action selection.

$$\nabla_\phi J(\phi) = Es \sim \mathcal{B}\left[\nabla_a Q\theta_1(s, a) \nabla_\phi \pi_\phi(s)\right].$$

Critic Loss: Ensures accurate estimation of Q-values by minimizing the error between predicted and target Q-values.

$$L(\theta) = E\left[\left(Q_\theta(s, a) - y\right)^2\right], \quad y = r + \gamma \min(Q_{\theta_1}, Q_{\theta_2}).$$

Delayed Updates: Actor and target networks are updated less frequently to stabilize training.

- Actor and target networks updated every `policy_freq` steps.
- Target networks use soft updates: $\theta_{\text{target}} \leftarrow \tau\theta + (1 - \tau)\theta_{\text{target}}$, blending old and new parameters to improve stability.

Implementation and Evaluation Replay Buffer: Stores (s, a, r, s', d) tuples for random sampling, decorrelating training data for better learning stability.

Exploration: Improves robustness by adding noise to actions during training and enforcing random actions near obstacles.

- Gaussian noise added to actions during training.
- Random actions enforced near obstacles for robustness.

Metrics:

- **Average Reward:** Measures task performance by averaging cumulative rewards across episodes.
- **Collision Rate:** Tracks safety by measuring the proportion of episodes resulting in collisions.

ROS Integration:

- Multi-threaded ROS nodes handle odometry and Velodyne data updates asynchronously, ensuring real-time state estimation.
- Goals are dynamically adjusted for obstacle avoidance, diversifying training scenarios.

3.7 ChatBot

The system uses the Large Language Model (LLM) from OpenAI by incorporating API to help decipher the queries made by the user with respect to Swarm Robotics . If a user typed a query, the LLM assigns the main object based on the matching to a list of predetermined keys (e.g. fire hydrant, chair). The model then generates two outputs: a list of planning region candidates and their closeness, farness or no relation at all. This output is reflected in the form of JSON into the robot control system as shown below. This task is performed in cooperation with shared database that includes position of the objects and robots to identify which robot should perform specific task based on distance between it and the object. For example, in cases when the given task is "there is fire" the LLM prefers fire extinguishers and sets a proximity constraint on the query. This enables the swarm to assign the nearest available robot to complete the task efficiently, ensuring clear communication and real-time feedback to the user.

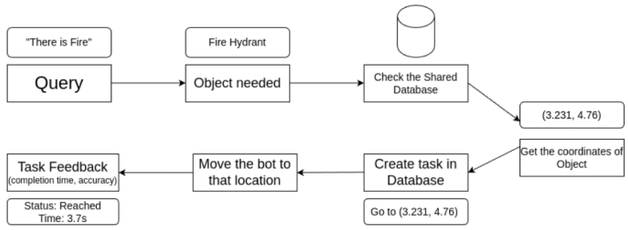


Figure 3.2: Flow diagram of Chatbot

4 Results And Discussions

4.1 Swarm Architecture

In this study, three different swarm architectures were evaluated: **complete autonomous**, **master slave without liveliness**, and **master slave with liveliness.**. The **Complete Autonomous** approach, where each robot operates independently, showed remarkable adaptability and resilience in dynamic environments, as it had no dependency on a central controller. However, it demanded more onboard processing and increased computational costs per robot. The **Master-Slave without Liveliness** configuration, although simpler to implement, proved to be less efficient in dynamic environments due to the lack of feedback from slave robots, which led to limited real-time adaptability. The **Master-Slave with Liveliness**, which included bidirectional communication, significantly improved system efficiency by enabling dynamic task reassignment based on the real-time status of robots. This architecture offered a balanced solution between complexity and adaptability, making it suitable for real-world applications that require fault tolerance and flexibility.

Metric	**Fully Autonomous**	**Master-Slave**
Task Allocation Time	300-500 ms	100-300 ms
Task Redundancy (%)	5-10% (due to independent decision-making)	∼0% (master ensures no duplication)
Real-Time Responsiveness	∼500 ms to adapt to environment changes	∼700 ms (master processes updates for all bots)
Scalability (Bots)	Scales up to 50+ bots with minimal degradation	Scales up to 20-30 bots before master bottleneck
Implementation Complexity	High; complex database synchronization	Moderate; simpler centralized logic
Communication Robustness	High; partial loss still allows operations	Low; master-slave communication loss disrupts system
Fault Tolerance	High; single bot failure does not affect others	Moderate; master failure halts operations
Coordination Efficiency	Requires database consistency	Centralized task assignment ensures coordination

Table 4: Comparison of Fully Autonomous and Master-Slave Systems

4.2 Object Detection Model and labeling

The YOLOv8 Nano model used for object detection and labeling initially struggled with overlapping objects and high object density, resulting in poor performance. We significantly improved performance by changing the image preprocessing strategy, which involved cropping to a central region and applying data augmentation techniques like horizontal flipping. The adjusted dataset improved the model's accuracy and minimised the risk of overfitting, leading to better generalization. The final model was better at detecting and labelling objects, especially smaller and occluded ones, which were previously difficult to identify. This enhanced object detection capability is crucial for effective robot navigation and task execution in complex environments.

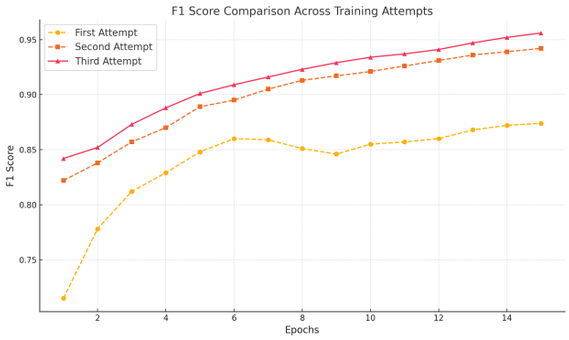
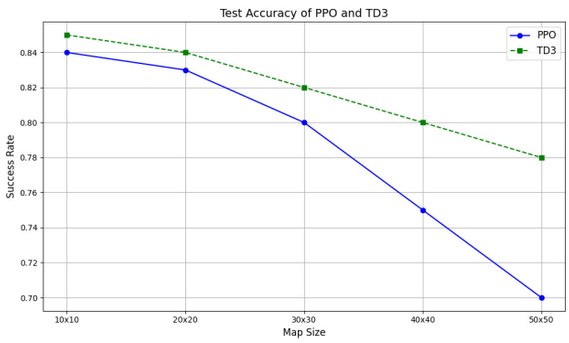


Figure 4.1: Comparison of object detection attempts
For attempt 2 & 3: First 15/50 epochs plotted

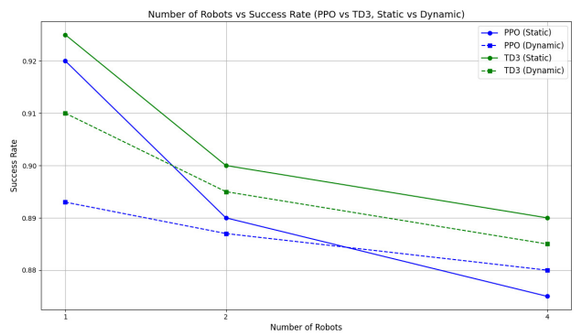
Attempt	mAP50	F1 Score	Key Improvements
1	0.9208	0.9265	Foundational model, lacked generalization.
2	0.9645	0.9571	Better generalization, higher accuracy.
3	0.9788	0.9712	Best balance of accuracy and efficiency.

Table 5: Performance metrics across attempts.

4.3 Path Planning and Collision Avoidance



(a) Comparison between PPO and TD3 model with increasing map size



(b) Comparison between PPO and TD3 model with increasing swarm number

Figure 4.2: Comparison of PPO and TD3 models based on map size and swarm number

Out of two explored RL based models TD3 and PPO the latter was chosen for path planning and collision avoidance. TD3 (Twin Delayed Deep Deterministic Policy Gradient) was more effective than other models in unstructured environments as the model's parameter proved that it is self-learning from experience consistently outdoing other models with time. Those improvements such as twin Q-networks and the delayed policy updates to stability were suitable for real-world decisions in complex situations. However, PPO, by its nature, can work well in certain conditions but does not have the fine-grained solution for fast response to the environment changes. TD3 successfully utilized approximate dynamic programming and learned similar trajectories as the expert, seeing consistent enhancements in navigation while performing way better than a baseline PPO in unknown environments in obstacle avoidance. In his paper Shijie Liu [6] also points out that TD3 is more effective at managing the safety-exploitation trade-off hence making it more suitable for path planning and avoidance during robot navigation.

5 Final Solution and Workflow

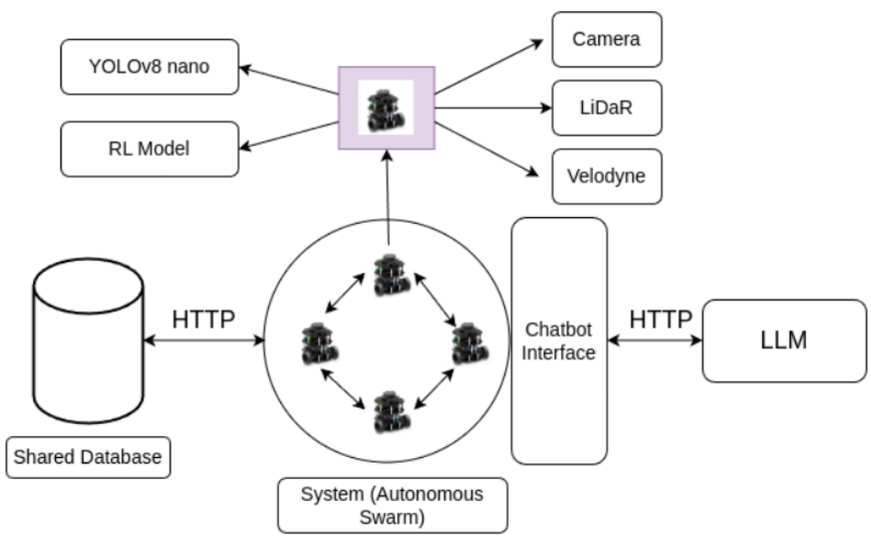


Figure 5.1: System Design

Figure 5.1 illustrates the design of a solution including numerous elements to enable an autonomous swarm of robots. Every module provides a certain feature, therefore allowing the system to be versatile, efficient, and user-friendly. The major components and their uses are the ones listed below:

Sensor Inputs and Data Acquisition

Combining Velodyne sensors, LiDAR, and camera, the system generates real-time environmental data.

- **Camera:** Records images for visual recognition tasks, including item or obstacle detection.
- **LiDAR (Light Detection and Ranging):** Provides perfect depth information and distance measurements, particularly for 3D mapping and navigation.
- **Velodyne:** A high-performance LiDAR sensor enabling full mapping in dynamic surroundings and increasing spatial awareness of the system.

These sensors, which give situational awareness and precise navigation, help the swarm to be guided.

Perception and Decision-Making

- The **YOLOv8 Nano** model integrates camera data for object detection, categorization, and localization. Its lightweight design and real-time operation optimization help small autonomous robots and other resource-constrained scenarios.
- Through **Reinforcement Learning (RL)**, learning optimal rules for task execution enhances the decision-making power of the swarm. This enables interaction with the environment by:
  - Allowing the swarm to dynamically adapt to changes in the environment or employment demand.

These models taken together allow the swarm to see its environment and make sensible autonomous decisions.

Autonomous Swarm System

The basis of the system is the **autonomous Swarm**, which consists of multiple cooperatively functioning robots. Every swarm robot communicates with others to:

- Share sensor data and learned information.
- Plan events to clearly achieve group goals.

The dispersion of the swarm offers robustness since the system can continue operating even if one or more of the robots fail.

Shared Database for Synchronization

The **shared database** stores important information, including:

- Maps produced from sensor data.
- Task assignments and progress reports.
- Policies with regard to performance standards and reinforcement learning.

For real-time data synchronization and accessibility, the swarm and the database interact over HTTP.

Chatbot Interface for User Interaction

Users of a **Chatbot Interface** can interact with the system seamlessly. This interface allows:

- High-level commands, from activity initiation to behavior modification, to be communicated to the swarm.
- Analyzing the state of the system and obtaining explanations for its decisions.

Driving the chatbot is a **Large Language Model (LLM)**, which translates natural language questions into effective inputs for the swarm.

Integration of Components

The system expertly blends perception, decision-making, and user involvement. The information flow can be characterized as follows:

- Sensor data entered into the YOLOv8 Nano and RL model for processing comes from real-time inputs.
- Processed data shows the Autonomous Swarm reporting back to the shared database and completing tasks.
- Users of the Chatbot Interface can send commands or query the system—which the LLM interprets into actions.
- Over time, the system learns and adjusts using user inputs and feedback from the RL model.

Essential Design Elements

- **Adaptability:** Through RL, the system gains compatibility with dynamic environments.
- **Scalability:** The modular construction makes the addition of new sensors or robots straightforward.
- **User-Friendliness:** Driven by the LLM, the chatbot interface makes the system accessible to non-experts.
- **Robustness:** The swarm-based approach ensures reliability even in the event of individual robot failures.

6 Future Scope

Thus, the proposed system has a great number of advantages and a stable basis, though some developments will improve its potential greatly. Key future directions include:

Detection of Small Obstacles

Combining the camera and LiDAR systems may bring the ability to traverse and address some of the failings of the two systems in identifying smaller obstacles that may be missed. By fusing visual and depth data, the system can achieve better resolution and object recognition, enabling:

- Recognition of low-height objects such as parts of debris or curbs.
- Better performance controlling the movements of a vehicle in a crowded or complicated area in which the objects to avoid can be of different dimensions and forms.

Visual Input During RL Training

The use of the feature maps with the RL training of the swarm can improve perception and adaptive criteria. This will enable:

- Recognition of objects that could not be sensed by Velodyne sensors, for instance, transparent or non-standard objects.
- Designing the capability to have multiple approaches to learning so that visual and sensor data greatly increase the accuracy of training data.

Support for Multiple Types of Robots

If the described system is further developed to support the operations of heterogeneous robot swarms, then the areas of its application can become even more diverse, as well as the functionality of the system as a whole. This includes:

- Coordination of aerial, terrestrial, or underwater drones having specific tasks, for example, security, transportation, or exploration.
- A rebuttal of inefficient distribution of work within the swarm, as robots can be specialized to achieve what they were designed for best.

Improved Collaboration with External Systems

The ability to communicate with other systems, for instance, IoT devices or other remote servers or self-automated systems, broadens the usefulness of the system in and of itself. This includes:

- Interaction with other systems with no delay and exchange of data using more sophisticated protocols.
- Integration with other AI models or edge computing hosts and devices for offline analysis and real-time, immediate decision-making.

Real-Time User Feedback and Adaptive Interfaces

Enhancing the Chatbot Interface with real-time feedback capabilities can allow users to:

- Increase the efficiency of monitoring swarm stakeholders and explaining decision-making through the integration of explainability tools.
- Provide dynamic inputs or elements allowing variations, modifications, or feedback in real-time to give users considerable control.

References

[1] N. Botteghi, B. Sirmacek, R. Schulte, M. Poel, and C. Brune. Reinforcement learning helps slam: Learning to build maps. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIII-B4-2020:329–335, 2020.

[2] Carmen Gheorghe, Mihai Duguleana, Razvan Gabriel Boboc, and Cristian Cezar Postelnicu. Analyzing real-time object detection with yolo algorithm in automotive applications: A review. *CMES - Computer Modeling in Engineering and Sciences*, 141(3):1939–1981, 2024.

[3] Ramón Jaramillo-Martínez, Ernesto Chavero-Navarrete, and Teodoro Ibarra-Pérez. Reinforcement-learning-based path planning: A reward function strategy. *Applied Sciences*, 14(17), 2024.

[4] Jaydeep Kshirsagar, Sam Shue, and James M. Conrad. A survey of implementation of multi-robot simultaneous localization and mapping. In *SoutheastCon 2018*, pages 1–7, 2018.

[5] Peng Li, Donghui Chen, Yuchen Wang, Lanyong Zhang, and Shiquan Zhao. Path planning of mobile robot based on improved td3 algorithm in dynamic environment. *Heliyon*, 10(11):e32167, 2024.

[6] Shijie Liu. An evaluation of ddpg, td3, sac, and ppo: Deep reinforcement learning algorithms for controlling continuous system. In *Proceedings of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*, pages 15–24. Atlantis Press, 2024.

[7] Huanwei Wang, Xuyan Qi, Shangjie Lou, Jing Jing, Hongqi He, and Wei Liu. An efficient and robust improved a* algorithm for path planning. *Symmetry*, 13(11), 2021.

[8] Shuhuan Wen, Zeteng Wen, Di Zhang, Hong Zhang, and Tao Wang. A multi-robot path-planning algorithm for autonomous navigation using meta-reinforcement learning based on transfer learning. *Applied Soft Computing*, 110:107605, 2021.

[9] Zhu Xiao, Ning Xie, Guobiao Yang, and Zhenjiang Du. Fast-ppo: Proximal policy optimization with optimal baseline method. In *2020 IEEE International Conference on Progress in Informatics and Computing (PIC)*, pages 22–29, 2020.

7 Appendix

PATH PLANNING AND OBSTACLE AVOIDANCE

Features	**Reinforcement Learning**	**Traditional Approaches(A* or Dijkshtra)**
Training Time	5-100 hours (depending on environment complexity)	0-20 seconds (predefined map)
Computational Cost	0.1 - 0.5 seconds per decision Quality: training)	1-5 seconds per path calculations
Adaptability to dynamic Obstacles	90 - 100% adaptability (real-time learning)	60 - 80% adaptability (path recalculation needed)
Path Planning efficiency	0.1-0.5 sec	1 - 3 seconds per path update (as dynamic obstacle comes in its path)
Scalability	5-10% increase in time/robot	30–50% increase in time/robot
Computational Efficiency in Large Environments	High, with optimized policy	Low, recalculates paths frequently
Path Quality: Optimal or near-optimal over time	Optimal or near-optimal over time	Optimal for static, but suboptimal for dynamic changes
Training Resources (hardware)	High (GPU/Cloud for training)	Low (runs on a standard CPU)

Table 6: Comparison between Traditional Algorithm and RL-based approach

Based on the comparisons between traditional approaches and reinforcement learning (RL)-based methods, RL approaches demonstrate better adaptability, precision, and efficiency, particularly in dynamic and unstructured environments

Reinforcement Learning

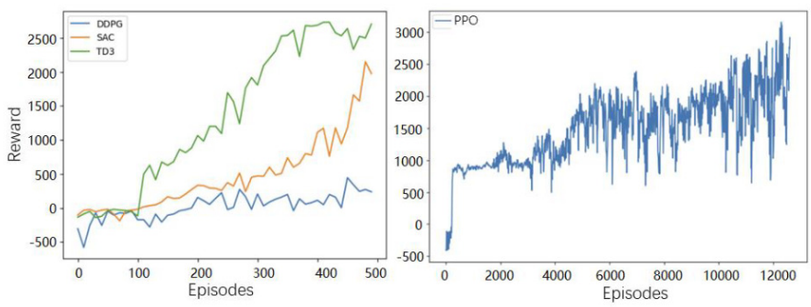


Figure 7.1: Multiple RL approaches in path planning as mentioned in [6]

TD3 architecture

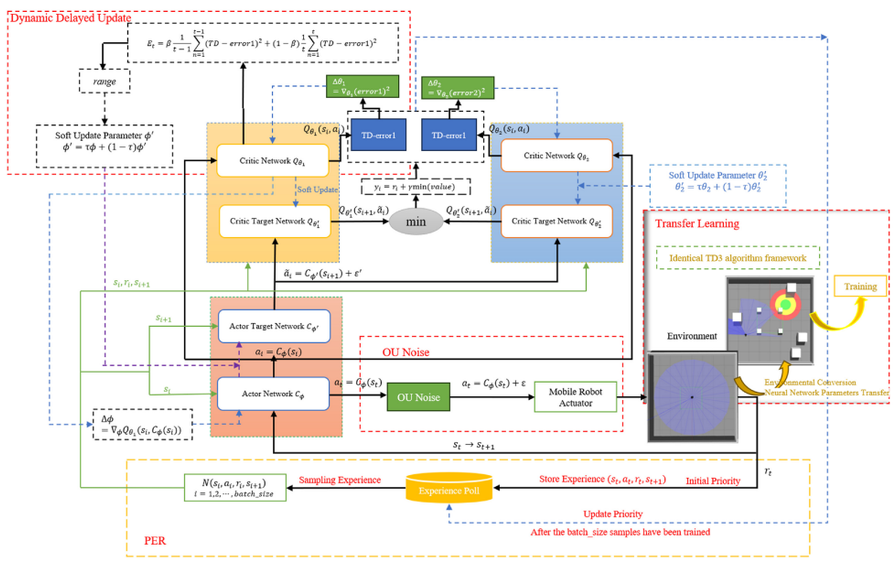


Figure 7.2: TD3 Architecture

PPO architecture

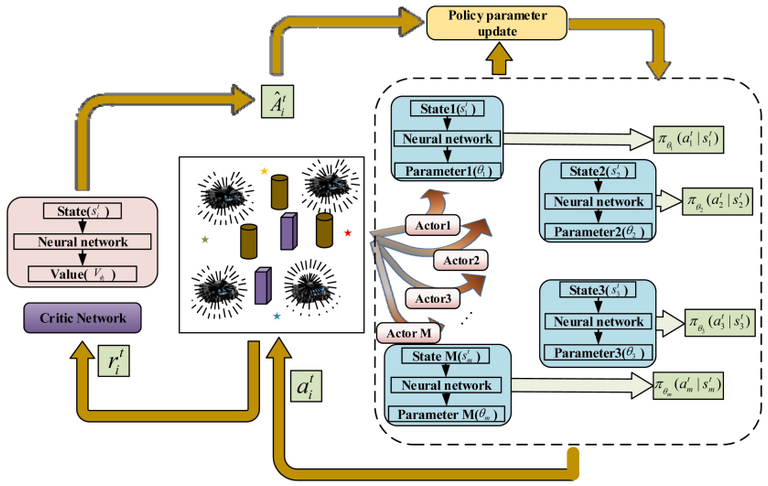


Figure 7.3: PPO Architecture

Code Flow

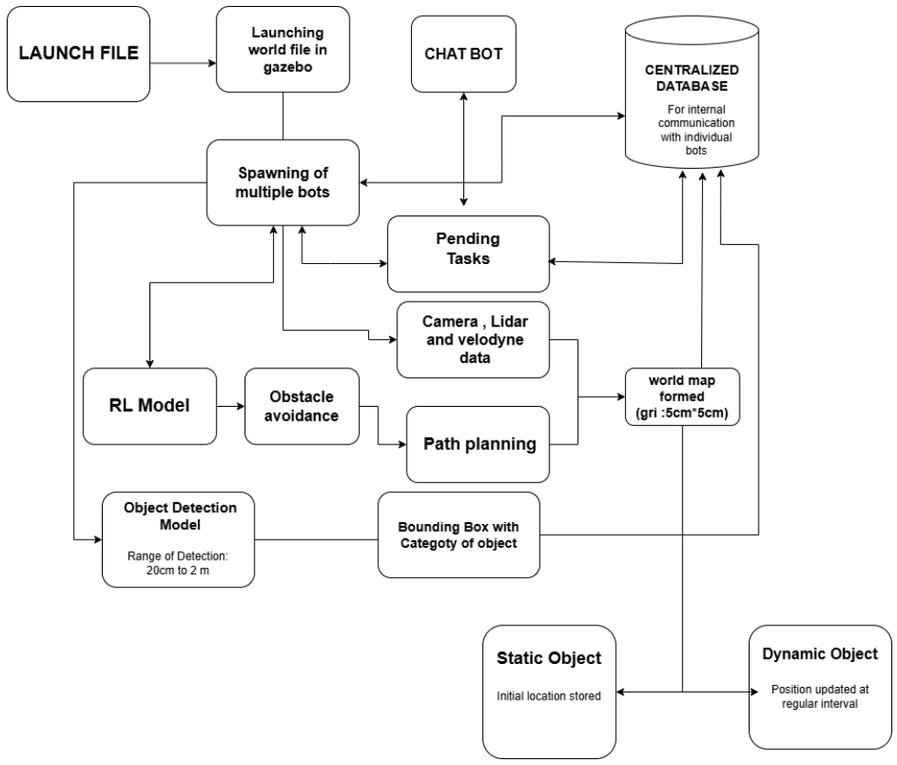


Figure 7.4: Code Flow

Path Planning

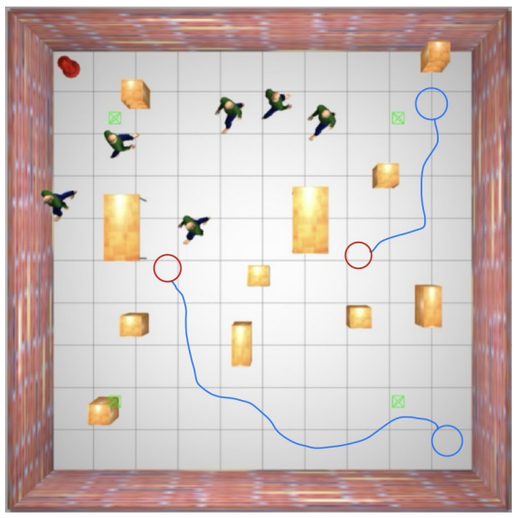


Figure 7.5: Path Planning

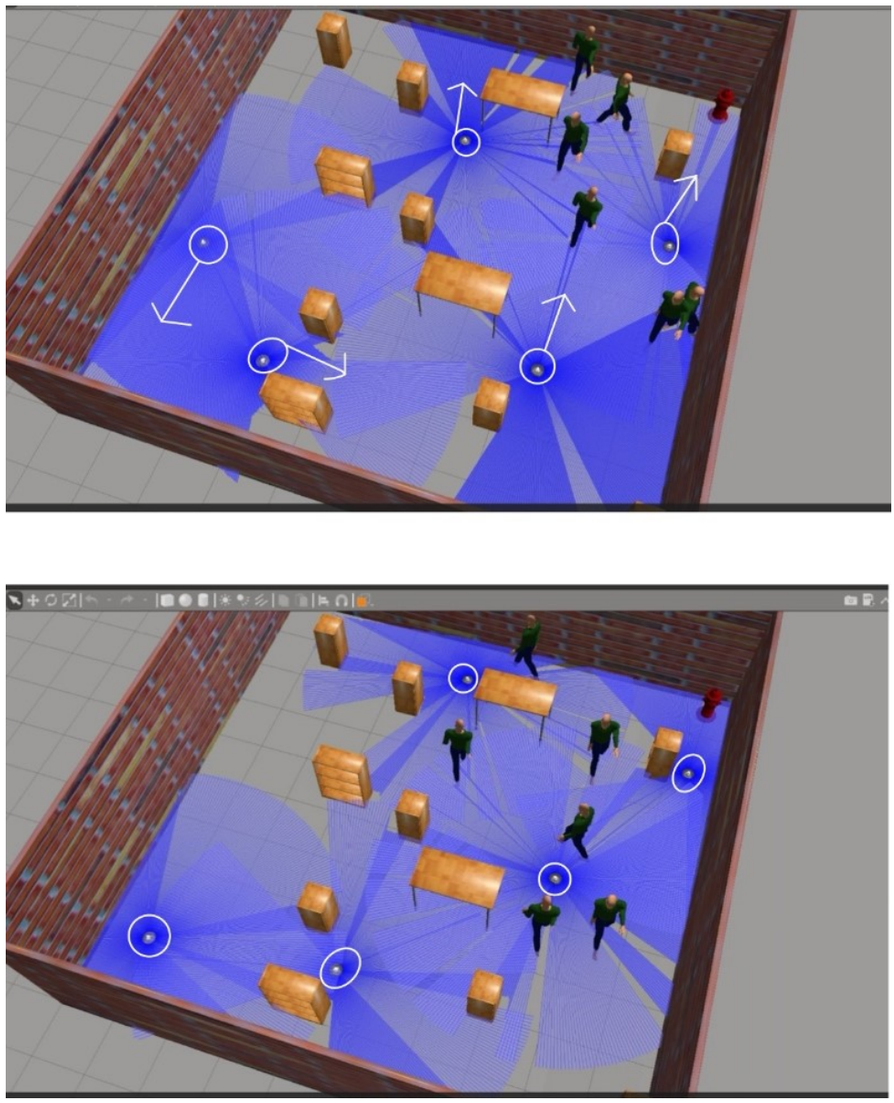


Figure 7.6: Motion of Multiple bots in our actual environment