European University Cyprus
School of Sciences | Department of Computer Science and Engineering

Autonomous UAV Navigation in Unknown Environments: Integrating SLAM and AI for Enhanced Robotic Exploration

Master of Science
in Artificial Intelligence

Dimitrios Passos

Acknowledgements

I would like to express my deepest appreciation to my professor Dr. Christos Malliarakis for his insights, feedback, and support that have significantly contributed to the development of this thesis, and I am truly grateful for the opportunity to work under his mentorship.

I would also like to express my heartfelt gratitude to my family for their unwavering support and encouragement throughout this journey. Their love, patience, and understanding have been invaluable, and without them, this achievement would not have been possible.

Abstract

This thesis explores autonomous navigation of Unmanned Aerial Vehicles (UAVs) and a land rover in GPS-denied environments through the integration of SLAM and reinforcement learning techniques. RTAB-Map SLAM is used to create real-time 3D maps, explore lite for autonomous exploration and navigation algorithms for safely navigating the UAV to the target location. The UAV also employs Deep Q-Network (DQN) for discrete action navigation. Additionally, a Twin Delayed Deep Deterministic Policy Gradient (TD3) network is implemented for the land rover, allowing continuous action control. Both systems utilize RGB-D cameras and LiDAR sensors for environmental perception, obstacle avoidance and localization. The performance of these systems is evaluated in simulation.

Table of Contents

1 Introduction

1.1 Overview

Unmanned Aerial Vehicles (UAVs) are increasingly becoming vital tools for a variety of applications, from environmental monitoring to disaster response. However, the challenge of navigating unknown and complex environments autonomously remains a critical barrier to achieving full UAV autonomy. Most UAV systems are heavily dependent on GPS for localization and navigation, which makes them unsuitable for environments where GPS signals are weak or unavailable, such as indoors, underground, or in urban canyons. The project aims to address this limitation by developing an advanced UAV system that combines SLAM, AI, and robust path planning techniques to allow for autonomous navigation in such challenging environments.

The system will not only focus on the UAV's ability to navigate without GPS but will also incorporate object detection to enable it to identify and interact with its surroundings. This combination of technologies makes the system applicable to search and rescue operations, where time is critical, and environments are often unpredictable. By allowing the UAV to map its surroundings, detect obstacles, and navigate efficiently, the project paves the way for smarter and more reliable autonomous systems.

1.2 Aims and objectives

The primary objective of the thesis is to develop a robust navigation system, with the ability to explore and create a map of the unknown GPS-denied environment. It also leverages Deep Q-Network (DQN) for autonomous movement towards predefined targets in 3D environments. Key objectives include:

- Implementing SLAM for real-time mapping and obstacle avoidance.
- Utilizing reinforcement learning algorithms, specifically DQN, to optimize the UAV’s path planning and decision-making processes.
- Validating the system in simulated environments to assess its efficiency, accuracy, and adaptability to dynamic changes.
- Utilizing object recognition and classification
- Implementing autonomous exploration
- Utilizing a TD3 network for a land rover

1.3 Structure of thesis

This thesis is organized as follows:

- **Chapter 2** provides a literature review, discussing existing UAV SLAM, navigation and reinforcement learning techniques by exploring these systems. Mapping strategies in GPS-denied environments are discussed along with ways to explore an unknown environment. Traditional approaches such as frontier-based exploration are discussed. Finally, reinforcement learning methods are mentioned along with their capabilities and limitations.
- **Chapter 3** outlines the tools used in this thesis. It provides an overview of the tools and technologies utilized in developing the UAV navigation system. Key tools include the Hector Quadrotor and Gazebo for simulation, ROS for middleware communication, and RTAB-Map integrated with OctoMap for SLAM and 3D mapping. Reinforcement learning algorithms, such as DQN and TD3, are employed for navigation and control, while OMPL handles motion planning. Additionally, YOLO enables real-time object detection for dynamic obstacle avoidance.
- **Chapter 4** presents the implementation process, including software tools, hardware configurations, and testing scenarios. In this chapter details regarding the practical steps of the thesis will be presented. It outlines how the proposed methodology was translated into a functional system, emphasizing the integration of various components to achieve autonomous UAV operation
- **Chapter 5** presents a detailed analysis of the UAV's performance across a range of test environments. The results of the implemented system are evaluated against key performance metrics, such as navigation accuracy, mapping efficiency, exploration coverage, and obstacle avoidance reliability. This chapter aims to assess how well the system achieved the objectives outlined in the methodology.
- **Chapter 6** summarizes the key findings of the thesis, highlighting the contributions made to autonomous UAV navigation and exploration. The limitations encountered, such as computational constraints and challenges in real-world deployment, are critically discussed. Finally, recommendations for future work are proposed, including improving scalability, enhancing sensor integration, and exploring multi-agent systems for cooperative exploration.

1.4 Summary

The motivation for this project is driven by the increasing demand for autonomous UAV systems capable of navigating and exploring complex and unknown environments without human intervention. Most UAV systems rely on GPS for navigation, which limits their capabilities in GPS-denied areas like indoor spaces, underground, or urban environments where GPS signals can be obstructed. In such environments, traditional navigation methods become unreliable or completely unusable, creating a need for alternative approaches that can ensure continuous, reliable operation.

This project seeks to enhance UAV autonomy through the integration of SLAM, AI, and advanced path planning algorithms, which will allow the UAV to navigate autonomously in these GPS-denied areas. Additionally, the UAV will be equipped with object detection capabilities to support human rescue missions, reducing the need for human intervention in dangerous or inaccessible areas. By improving the UAV's ability to explore and operate in complex environments, this project aims to increase the efficiency and effectiveness of UAV systems in critical, real-world applications such as search and rescue, disaster response, and environmental monitoring.

Moreover, this project aims contribute to advancements in robotic exploration, including frontier exploration, path planning, 3D mapping, and image recognition for both aerial and land-based robots. This broader impact will help drive forward the development of autonomous systems capable of operating in diverse and challenging environments, ultimately making these technologies more accessible and practical for various real-world applications

2 Literature Review

2.1 Introduction

The challenges of localization, autonomous navigation, and full environment coverage for UAVs have gained significant attention over the last few years. Numerous publications propose various methods to address different issues, differing mainly in the types of sensors and the information they use. UAV limitations, such as payload and power consumption—which are important to keep low to ensure higher flight times—and the complexity of movement in the 3D plane compared to ground robots, add additional layers of complexity.

Firstly, the localization techniques for UAVs are discussed, focusing on methods that leverage different sensors and strategies to address the challenges posed by UAV limitations, such as payload and power consumption. This is followed by an exploration of mapping and autonomous navigation techniques, which highlight existing research gaps and areas for future development in achieving comprehensive environment coverage.

2.2 Localization Techniques for UAVs

Localization refers to the process of determining a UAV's position and orientation within its environment. In the context of indoor environments, where GPS signals are unavailable, localization becomes essential for the UAV to understand its location relative to its surroundings. Accurate localization allows the UAV to navigate effectively, avoid obstacles, and maintain a stable flight path. Alternative methods, such as computer vision or lidar, are used to estimate this position, enabling autonomous operation in spaces where GPS cannot be used.

Localization for UAVs in GPS-denied environments can be achieved through several methods, each leveraging different types of sensors and algorithms. Vision-based techniques, such as those using RGB-D cameras, allow UAVs to estimate position by analyzing visual and depth data, often combined with visual odometry to track movement over time. This approach is well-suited for environments with rich visual features but can struggle in low-light conditions or when dealing with featureless surfaces [1].

One common localization method is by using lidar in GPS-denied environments due to its accuracy in providing high-resolution depth information. Localization with lidar relies on point clouds to detect and map the surrounding environment by measuring distances to obstacles. In UAVs, lidar offers significant advantages over vision-based methods, particularly in open environments without many features. [2]. Algorithms such as Hector SLAM, Cartographer, and RTAB-Map use lidar to create detailed 2D or 3D maps, allowing UAVs to localize and navigate effectively. Using only point cloud for localization will need to a significant drift over time. To combat this issue, reference frames are defined along the path of the robot. The reference frames called keyframes, are then used by the Iterative Closest Point (ICP) algorithm to create a pose graph, which on loop closures, provides corrections of the robot' s trajectory [3].

2.3 Graph-based SLAM and RTAB-Map for UAVs

One approach of formulating SLAM is to represent it as a graph, where nodes indicate the robot's poses at different time frames and the edges signify constraints between these poses. These constraints are obtained either by the robot' s movement or from environment observations. Once the graph is built, the map is created by arranging the nodes in a way that fits the measurements along the edges. When a robot encounters a previously visited node (edge) then a loop closure will be detected [4].

Loop Closure refers to the process of recognizing previously visited locations to correct accumulated errors in a UAV's estimated position. As the UAV navigates, small inaccuracies in localization can build up, causing drift over time. When the UAV revisits a known area, loop closure detects similarities between the current environment and the stored map, allowing the system to realign the UAV's position with this reference. This correction helps improve overall map accuracy and reduces localization drift [5].

Three dimensional representations of the environment are called occupancy grids, and each cell or voxel is labelled as either occupied, free, or unknown. In these grids, occupied cells represent areas with obstacles, free cells are navigable spaces, and unknown cells have not yet been mapped. As the UAV explores the area, the occupancy grid is constantly been updated and the system maintains an accurate 3D map that can be used for obstacle avoidance and path

planning. In the case of RTAB-Map, occupancy grids are built using data from sensors like RGB-D cameras or lidar, allowing for precise mapping and navigation in both 2D and 3D environments [6].

This approach is particularly useful in UAV applications that require dynamic exploration, as presented in several studies utilizing RTAB-Map for indoor UAV navigation. On the other hand, in feature-less environments, lidar scans can be used to localize the UAV.

2.4 Mapping Techniques for 3D Environments

Mapping is the process of creating a spatial representation of the environment that a UAV navigates. In autonomous UAV navigation, mapping enables the UAV to understand its surroundings, plan routes, and avoid obstacles. This is particularly important in 3D environments, where the UAV must operate in multiple planes and continuously update its understanding as it moves.

This process is important for autonomous UAV navigation, and several methods use point cloud data to represent 3D environments. For example, the research presented by Beul et al. [7] uses a combination of lidar sensors and stereo cameras to construct detailed 3D maps of warehouses. Using these sensors, the UAV creates a point cloud representation of its environment, which is continuously updated. One of the most common frameworks for creating 3D occupancy grids is the OctoMap framework, widely used in 3D mapping.

On the other hand, the work by Ok, Greene [8] leverages RGB-D cameras to build maps in dynamic environments. They utilize depth information to create 3D maps and apply visual recognition techniques to continuously localize the UAV within the environment. These methods have been proven to work well in complex and feature rich environments such as warehouses, where UAVs need to navigate autonomously and avoid obstacles in real time.

2.5 Exploration Strategies for UAVs

Exploration in robotics refers to the process by which a robot autonomously navigates and investigates an unknown or partially known environment to build a map or collect information. This concept is critical for applications where human control is limited or impossible, such as in hazardous environments, underwater regions, caves, or in search and rescue missions.

The primary goals of exploration are to maximize coverage of the environment, to identify and map unknown areas, and to do so efficiently while minimizing the time and resources expended. For UAVs, this means systematically flying through an area to gather data about its layout, obstacles, and any points of interest.

The need for exploration arises from the challenges associated with unknown environments. Without prior knowledge of the terrain or layout, robots must be capable of making real-time decisions about where to go next to gain the most information. This is essential for applications like mapping, navigation, and environmental monitoring, where detailed understanding and situational awareness are required [9].

The Frontier-based exploration algorithm is a widely adopted technique in which the UAV actively searches for unexplored regions. The algorithm detects frontiers, the boundaries between known and unknown spaces, and directs the UAV to explore those areas [10].
In terms of autonomous navigation, the FUEL [11] paper emphasizes frontier-based exploration for UGVs, detecting frontiers between known and unknown areas to direct exploration in 3D space. Similarly, this frontier-based approach can be applied to UAVs, which dynamically explore environments by identifying transition points to new areas, enhancing coverage efficiency.

2.6 Reinforcement Learning (RL) for Autonomous UAV Exploration

Reinforcement Learning (RL) in UAV exploration allows drones to autonomously navigate by interacting with their environment and learning by rewarding or punishing them. The RL framework, especially 3D exploration algorithms like the one proposed by Bircher et al., focuses on maximizing the UAV's knowledge of unknown environments. UAVs continuously evaluate their surroundings and choose the next best viewpoint, balancing exploration of new areas and exploiting known regions. This technique helps the UAV cover the area efficiently while ensuring maximum information gain in complex environments.

Integrating RL with UAVs has shown significant promise in addressing the challenges of autonomous exploration, allowing UAVs to adapt dynamically to changing environments while learning optimal strategies for navigation [12]. This method enables drones to discover and map environments more intelligently, with minimal human input, through strategies that favor long term benefits over immediate rewards. RL's adaptability makes it particularly effective in environments where predefined rules for navigation are difficult to establish, such as indoor or GPS-denied spaces.

2.7 Challenges in UAV Localization and Navigation

Despite its potential, Reinforcement Learning (RL) in UAV exploration faces several challenges. One issue is the high computational demand, especially in resource-constrained UAVs with limited processing power. Real-time decision making in complex and dynamic environments can be computationally heavy, requiring efficient algorithms to balance exploration and exploitation.

Another challenge is safety and reliability in unpredictable environments, where the UAV must avoid collisions and handle uncertainties like moving obstacles. Training time and sample efficiency are also critical concerns, as RL often requires many interactions with the environment to converge on optimal strategies, which requires significant amount of computational power.

Despite advancements in SLAM and AI for UAVs, several challenges remain. One of the primary concerns is computational efficiency, mainly in resource constrained UAVs. Lightweight drones with limited computing power. struggle to perform complex tasks like real-time 3D SLAM and autonomous decision making. Another challenge is the integration of multiple sensors, such as cameras and lidars, which can increase the UAV's weight and power consumption, reducing flight time and increasing overall cost. Lastly, transferring policies learned in simulation to real-world environments remains a challenge due to the sim-to-real gap.

3 Tools used

In this chapter, tools that used theoretical concepts necessary for understanding this thesis are presented. Specifically, the principles of unmanned aerial vehicles (UAVs) are discussed, as well as the function of a PID controller. Additionally, the tools and libraries used for implementing the localization and full coverage system for a UAV are explained. This includes an analysis of the ROS middleware, which served as the foundation for the entire implementation, along with the Hector Quadrotor ROS packages and libraries such as OctoMap, Particle Filter, and OMPL.

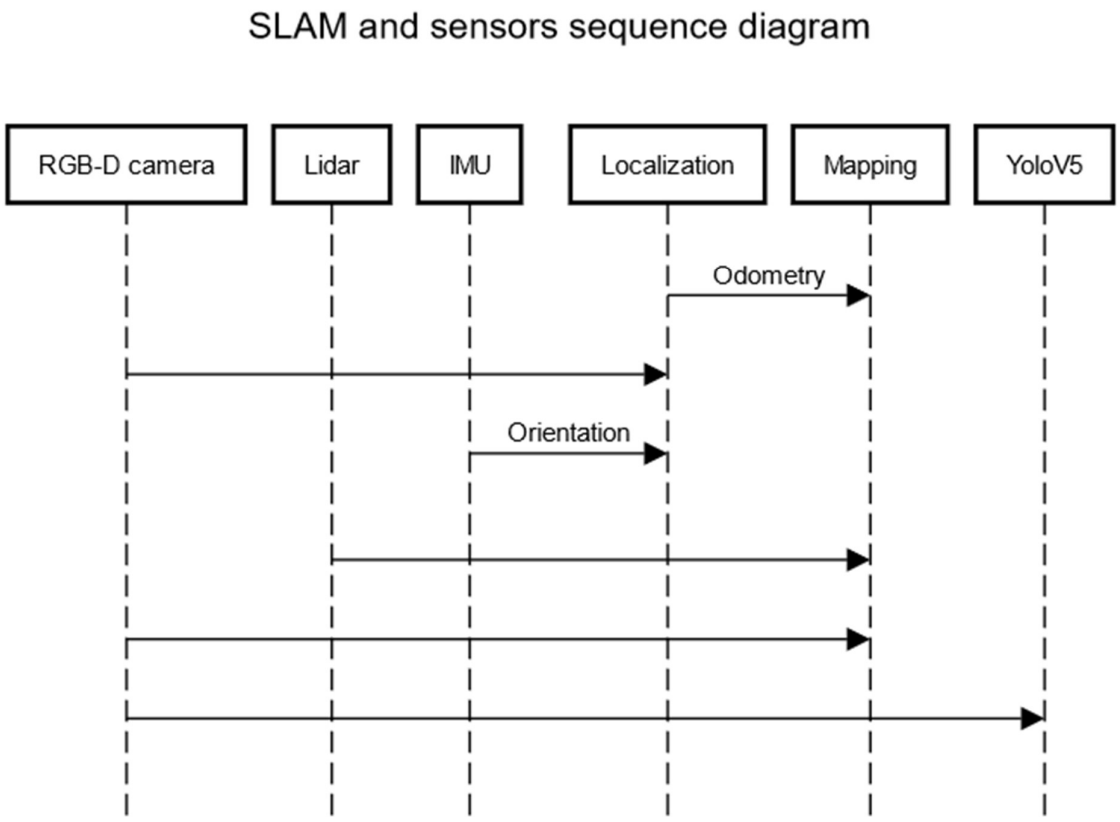


Figure 1. *SLAM and sensors sequence diagram*

In *Figure 1* the sequence diagram to achieve SLAM is presented. As shown in the diagram, we have one node for localization and one for mapping. Localization node takes as input only the RGB-D camera and the IMU sensor, responsible for providing information about orientation, producing the odometry of the UAV in relation to the world (odometry). The mapping node

takes as input the lidar, the RGB-D camera and the previously produced odometry, in order to map the surroundings in relation to the UAV. Finally the camera is also used as input for the yolo framework, providing live image recognition.

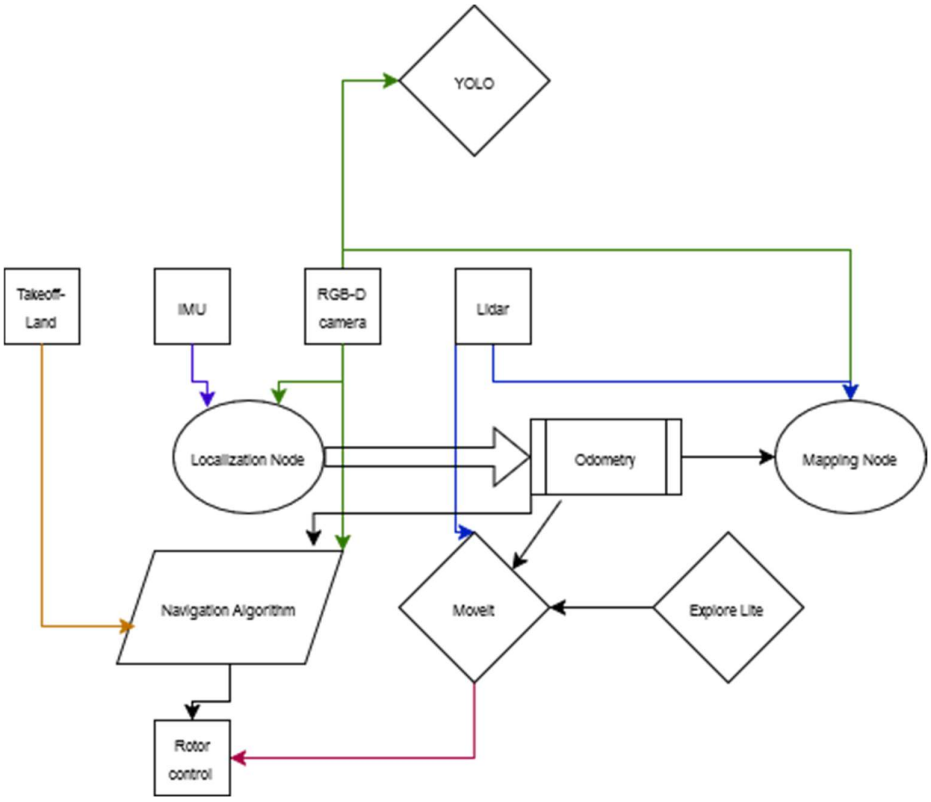


Figure 2. *Thesis diagram*

In *Figure 1* the diagram shows the structure of the thesis. Following the above information, in this diagram we see some additional nodes. Firstly, the navigation algorithm, that is built in this thesis uses as input the odometry for knowing its position and the distance it has to travel, the camera for avoiding collisions, the take-off node for gaining altitude if the UAV is on the ground and has control over the rotors to ensure correct movement. MoveIt node is also responsible for moving in 2D and it is also used for exploration. The node takes as input the odometry produced, the lidar for maintaining a safe distance around obstacles, the exploration node which feeds it with frontiers that needs to be explored and of course it has access to the UAV motor controls.

3.1 UAV Operation

Unmanned Aerial Vehicles (UAVs) are categorized by their motor count and placement, with this thesis focusing on quadcopter, UAVs that rely on four rotors. These rotors are positioned in a symmetrical pattern around the center, enabling it to perform precise movements like take-off, landing, hovering, and rotations. Though quadcopters can theoretically move in six directions, only four of these are controlled directly through adjustments in rotor speed, which allows the quadcopter to stabilize and alter its height, orientation, and position.

Key maneuvers for quadcopters include taking off, hovering, and landing. The rotors spin at equal speeds for take-off, generating an upward lift strong enough to overcome gravity. To land, the rotor speeds are reduced simultaneously, decreasing the lift and allowing for a controlled descent. Hovering, on the other hand, requires the rotors to maintain a precise speed which creates force, equal and opposite in relation to gravity. Rotation around its center, or yaw, is accomplished by changing the speed of opposite pairs of rotors. For example, increasing the speed of rotors 2 and 4 relative to rotors 1 and 3 causes the quadcopter to rotate counterclockwise.

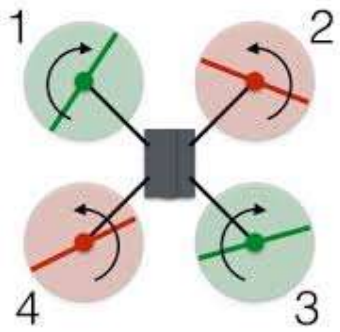


Figure 3. *Rotos on UAV*

For achieving precise control over the quadcopter's movement, PID controllers (Proportional-Integral-Derivative controllers) are commonly used. These controllers adjust the rotor speeds to maintain stability and respond to changes in the quadcopter's position, altitude, and orientation. Each of the three control axes, pitch, roll, and yaw, typically has its own PID controller, which is responsible for fine tuning the rotor speeds to achieve the desired movement.

- Proportional (P) control reacts proportionally to the error, or the difference between the desired and current states. For instance, if the quadcopter tilts away from its intended angle, the proportional control will adjust rotor speeds correcting the tilt.
- Integral (I) control addresses any accumulated error over time. This is essential for correcting long-term deviations, such as persistent drift when hovering.
- Derivative (D) control predicts future errors based on the rate of change, helping to smooth out the response and reduce overshoot.

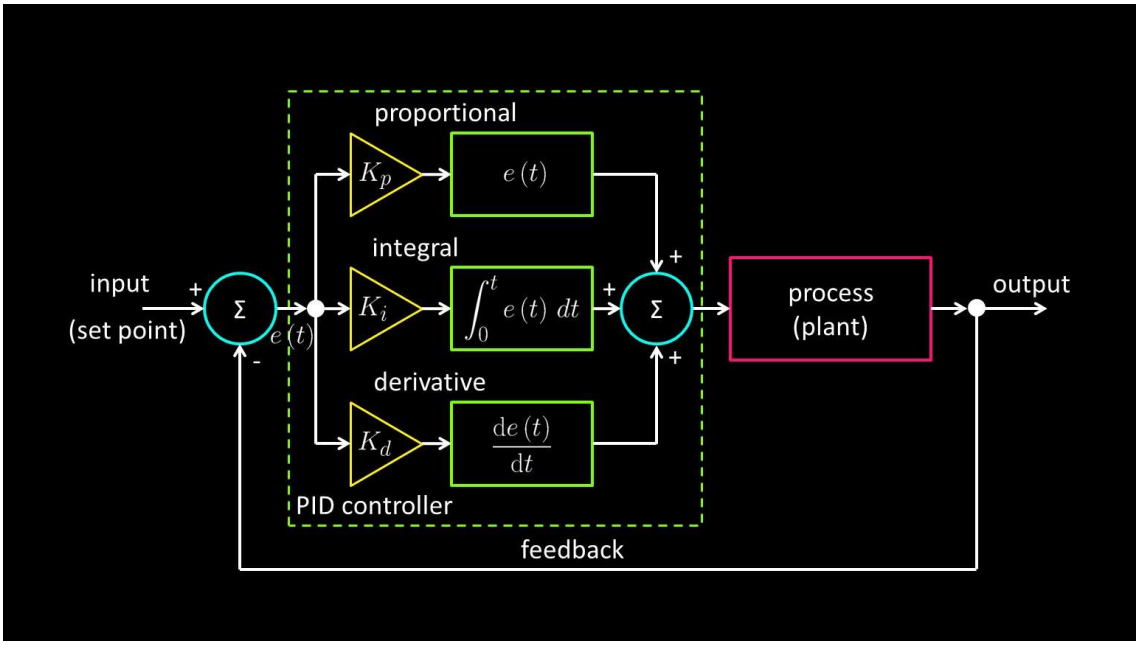


Figure 4. *PID Controller*

Source: https://www.digikey.com/en/maker/projects/introduction-to-pid-controllers/763a6dca352b4f2ba00adde46445ddeb

Figure 4 illustrates the structure of a Proportional-Integral-Derivative (PID) controller, a widely used feedback control mechanism in robotics and automation systems. The PID controller operates by continuously calculating the error e(t), which is the difference between the desired input (setpoint) and the actual system output. This error is then processed through three components:

- Proportional: Corrects the error in proportion to its magnitude, providing immediate corrective action.
- Integral : Accounts for the accumulation of past errors to eliminate steady-state errors.

- Derivative: Predicts future errors based on the rate of change, providing damping to prevent overshoot.

In a quadcopter, PID controllers work together to maintain balance and achieve stable flight by adjusting the rotor speeds according to the combined outputs of the P, I, and D components. This control approach is critical for executing complex maneuvers like hovering, landing, and rotating accurately. For example, when hovering, the PID controller continuously adjusts the rotor speeds to counteract any minor disturbances, ensuring that the quadcopter remains at a fixed altitude and position.

3.2 Robot Operating System (ROS)

The Robot Operating System (ROS) is an open-source, flexible framework, created for building and deploying robotic software. Developed by Willow Garage and now maintained by Open Robotics, ROS has become a standard in robotics research and industry due to its modularity, scalability, and comprehensive toolset that supports complex robotic systems.

At the core of ROS is a distributed, peer-to-peer network of nodes, where each node represents a specific process or function, such as sensor data processing, control, or planning. Nodes communicate through topics using a publish subscribe model, allowing data sharing across processes in real time. Additionally, ROS offers services for synchronous communication and actions for tasks that require feedback, making it suitable for a range of applications.

The ROS Master provides name registration and parameter services, managing communication between nodes. By centralizing the connection setup, the ROS Master enables easy integration and scaling, which is particularly advantageous in multi-robot systems.

ROS functionality is organized into packages, which bundle nodes, configurations, and dependencies together. This package system promotes modularity and reuse, with a large repository of community-contributed packages for sensors, control algorithms, localization, mapping, and other core robotics functions.

It also includes powerful tools like rviz for visualization, rqt for runtime monitoring, and Gazebo for simulation, which allows for testing and validating algorithms in a controlled,

repeatable environment. Libraries such as TF (transform library) enable tracking of coordinate frames, crucial for tasks that require precise spatial transformations.

ROS is widely used in autonomous navigation, manipulation, SLAM (Simultaneous Localization and Mapping), and multi-agent systems, supported by its compatibility with various sensors and actuators. Its open source nature has driven widespread adoption in academia and industry, fostering collaboration and accelerating development. Research applications often leverage ROS for rapid prototyping of complex algorithms, from perception and localization to AI-driven decision-making and robotic planning.

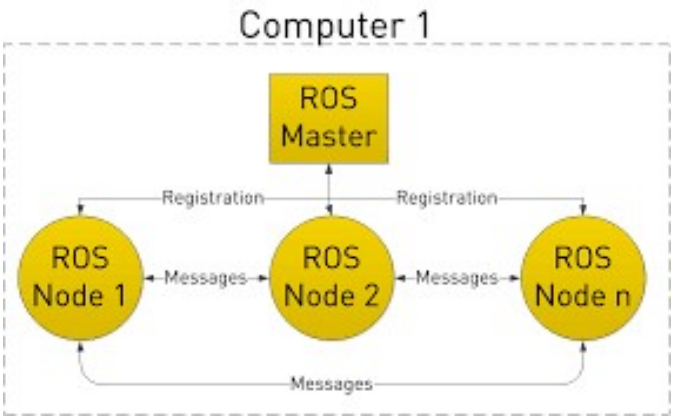


Figure 5. *ROS Architecture*

Source: https://gachiemchiep.github.io/ros/learn-ros-pt1/

Figure 5 shows the architecture of the Robot Operating System (ROS) communication framework. In this setup, the ROS Master acts as the central node responsible for managing and coordinating the registration of all other nodes in the system. Each ROS Node represents a modular process that performs specific tasks, such as sensing, control, or actuation.

The nodes communicate with each other using a publish-subscribe messaging model. Nodes register themselves with the ROS Master, which facilitates message exchange between them by providing the necessary addressing information. This decentralized communication structure allows nodes to exchange data efficiently, enabling seamless integration of components in a robotic system. This architecture supports flexibility and scalability in the design of complex systems.

3.3 SMB Cave Rover

The SMB cave rover is a four-wheel ground robot capable of navigating through rough terrain. In the context of the thesis the rover has been used to train a TD3 network, to navigate through close spaces without colliding with obstacles.
The rover is equipped with a velodyne sensor, offering 360 degrees view of the surroundings, a front based lidar, checking the terrain elevation and four depth cameras on each side.

3.4 Hector Quadrotor

Hector Quadrotor is an open-source simulation model developed for research and development in autonomous UAV (Unmanned Aerial Vehicle) applications. It is part of the Hector Robotics project, created by the Team Hector at the Technische Universität Darmstadt, and is widely used for testing quadrotor control, navigation, and SLAM (Simultaneous Localization and Mapping) algorithms within the Robot Operating System (ROS) framework.

The Hector Quadrotor is primarily designed for use with the Gazebo simulator, allowing researchers to simulate realistic flight dynamics in 3D environments. Gazebo provides a physics engine that models aerodynamics, sensor noise, and environmental factors, making the Hector Quadrotor a valuable tool for safely testing algorithms in virtual environments before deploying them on real hardware.

The model includes a range of simulated sensors typically used in UAVs, such as IMU (Inertial Measurement Unit), laser scanners, GPS, barometers, and cameras. These sensors are crucial for developing perception and localization algorithms, such as SLAM, and allow users to create scenarios with high-fidelity sensor data.

Hector Quadrotor is widely used in SLAM applications, particularly for 3D mapping and exploration in GPS-denied environments. The ROS package includes modules for integrating with Hector SLAM, which relies on laser scan data for mapping and localization, making it suitable for tasks where GPS is unavailable or unreliable, such as indoor navigation or search and rescue.

The package is fully compatible with ROS, providing a set of tools and preconfigured launch files for easier integration. It includes ROS nodes for flight control, sensor data processing, and navigation, enabling users to implement and test advanced flight control and autonomous navigation algorithms.

Due to its open-source nature and comprehensive feature set, the Hector Quadrotor is popular in academic research for applications such as autonomous exploration, obstacle avoidance, and multi-robot collaboration. It supports advanced research on topics like reinforcement learning, SLAM, and trajectory optimization, making it a go-to resource for UAV-related projects within ROS.

3.5 Open Motion Planning Library (OMPL)

The Open Motion Planning Library (OMPL) is an open source library developed for path planning for robotic systems in complex environments. OMPL is highly regarded in robotics research and industry due to its efficiency, flexibility, and broad range of sampling based algorithms, which make it suitable for various applications, from mobile robots and UAVs to manipulators and autonomous vehicles. The library is developed in C++, but it can also be integrated with python, with python bindings.

OMPL specializes in sampling based algorithms, which are particularly effective for three dimensional spaces and complex environments. These algorithms, such as RRT (Rapidly-exploring Random Tree), PRM (Probabilistic Roadmap), and their many variants, allow OMPL to efficiently find feasible paths without needing an explicit model of the free space.

It is designed with modularity in mind, making it easy to integrate with existing robotic systems and frameworks. Users can select from a variety of planners or combine them in a planner pipeline, adapting the library to specific motion planning needs, such as obstacle avoidance, constrained environments, or optimization-based planning.

OMPL is a core component of the MoveIt motion planning framework within ROS. MoveIt leverages OMPL's planners to execute motion planning tasks for robotic arms, mobile bases, and aerial robots, allowing OMPL to benefit from ROS's extensive tools and visualization capabilities. Through MoveIt, users can set goals, define constraints, and visualize motion plans in a simulated or real-world environment.

It can handle high-dimensional configuration spaces, making it suitable for robots with complex kinematics, like manipulators and articulated UAVs. Additionally, OMPL supports planning with constraints (e.g., staying within a specific region or following surface constraints), which is valuable for tasks that require precise motion.

3.6 OctoMap

OctoMap is an open-source 3D mapping framework widely used in robotics for creating volumetric maps of the environment. Developed with mobile robotics and autonomous navigation in mind, OctoMap represents the environment as an octree structure, a type of hierarchical 3D grid that allows for efficient memory usage and fast updates, making it ideal for real time applications.

OctoMap is designed to generate 3D occupancy maps, where each cell in the octree represents a small volume (voxel) and is classified as free, occupied, or unknown based on sensor data. This is useful for applications requiring detailed spatial understanding, like obstacle avoidance, path planning, and exploration. The octree structure enables OctoMap to dynamically adjust the resolution of the map, subdividing space only where necessary. This adaptive resolution makes OctoMap memory efficient, as it allocates more detail only in areas with greater complexity (e.g., near obstacles), while open or empty areas require minimal storage.

It uses a probabilistic framework to determine occupancy, updating the probability of each voxel being occupied based on new sensor measurements. This approach helps OctoMap handle noisy sensor data, improving accuracy and robustness in real-world conditions where noise and measurement errors are common.

It is fully integrated with ROS, allowing it to receive data from a variety of sensors, such as 3D lidar, depth cameras, and RGB-D sensors. Through ROS topics and services, OctoMap can be used seamlessly with other ROS packages, such as navigation and motion planning frameworks, enabling autonomous exploration and mapping.

OctoMap is widely used in mobile robotics, aerial drones, and autonomous vehicles, especially in scenarios requiring 3D environment modeling and real time map updates. Its ability to represent both occupied and free spaces in 3D is essential for applications in environments with

complex structures, such as indoor navigation, autonomous exploration, and multi-robot coordination.

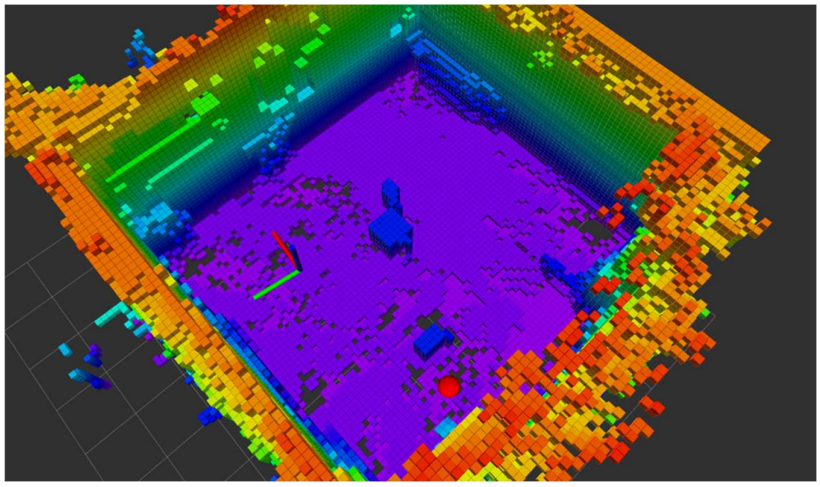


Figure 6. *3D Grid with OctoMap*

Source: https://www.researchgate.net/figure/Octomap-ground-truth-of-the-environment-The-frame-denotes-the-robot-position-and-the-red_fig3_341069094

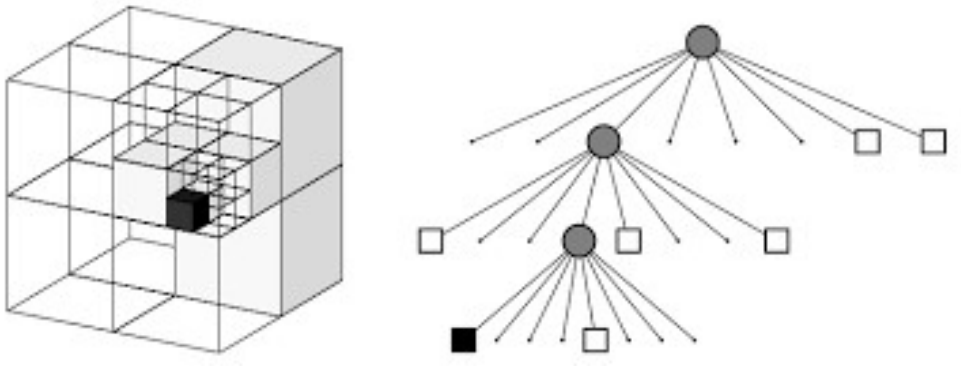


Figure 7. *Octree Representation for Spatial Partitioning and Data Structure Visualization*

Source: http://www.diva-portal.org/smash/get/diva2:1183906/FULLTEXT01.pdf

Figure 6 the 3D representation of the mapped space in Octomap, where each box is considered a voxel in space. In *Figure 7 t*he diagram on the left depicts a voxelized 3D space where each cube (or voxel) represents a discrete unit of space. The smaller cubes within the larger ones represent the hierarchical nature of the octree structure, enabling multi-resolution representation. The diagram on the right shows the corresponding octree representation. In an octree, each node represents a cubic volume of space, and it can have up to eight children, subdividing the parent volume into smaller regions. The black square indicates an occupied voxel, while the white squares represent free or unknown space. Gray circles represent intermediate nodes in the octree that aggregate information about their child nodes.

3.7 RTAB-Map (Real-Time Appearance-Based Mapping)

RTAB-Map is an RGB-D, Stereo and Lidar Graph-Based SLAM approach used for real-time localization and mapping, especially in scenarios where both visual and depth information are critical. It can build a 3D map of the environment using inputs from an RGB-D camera, lidar, or stereo camera. RTAB-Map employs a memory management system that allows for loop closure detection and optimization, which is crucial for creating an accurate and consistent map in large-scale environments. This approach is especially useful for UAVs operating in GPS-denied environments, as it provides robust localization while dynamically exploring unknown areas.

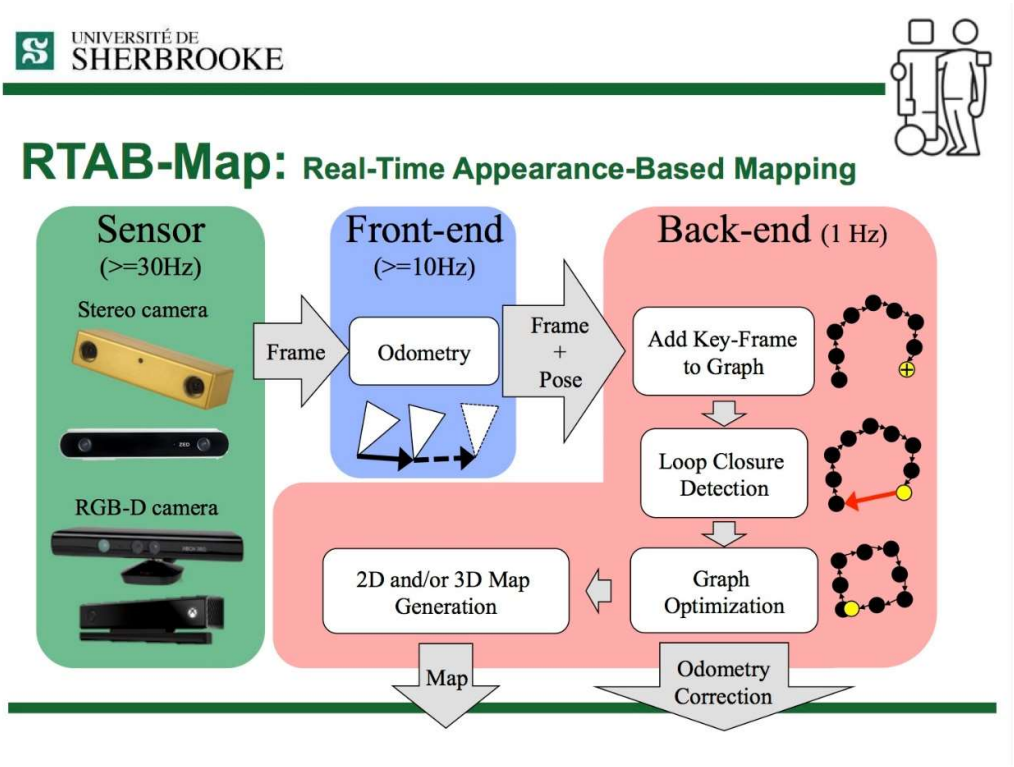


Figure 8. *RTABMap Architecture*

Source: http://introlab.github.io/rtabmap/

Figure 8 shows the architecture of RTAB-Map (Real-Time Appearance-Based Mapping), a SLAM framework for generating 2D and 3D maps in real-time. Sensor data from stereo or RGB-D cameras, operating at high frame rates, provides visual and depth information as input. The front-end processes this data to estimate odometry and generate initial map frames, while the back-end optimizes the pose graph, detects loop closures, and corrects accumulated odometry drift. This combination ensures accurate and consistent map generation, making RTAB-Map suitable for real-time navigation in dynamic environments.

3.8 Deep Q-Network (DQN)

DQN is a popular reinforcement learning algorithm that uses deep learning to solve complex decision-making problems. It combines Q-learning, a classic reinforcement learning algorithm, with deep neural networks to approximate the Q-values of each action in each state. DQN is particularly useful for discrete action spaces, where the agent must learn optimal policies through exploration and exploitation of the environment. In robotic navigation, DQN can be used to help a robot learn effective navigation strategies by receiving rewards based on its actions and improving over time.

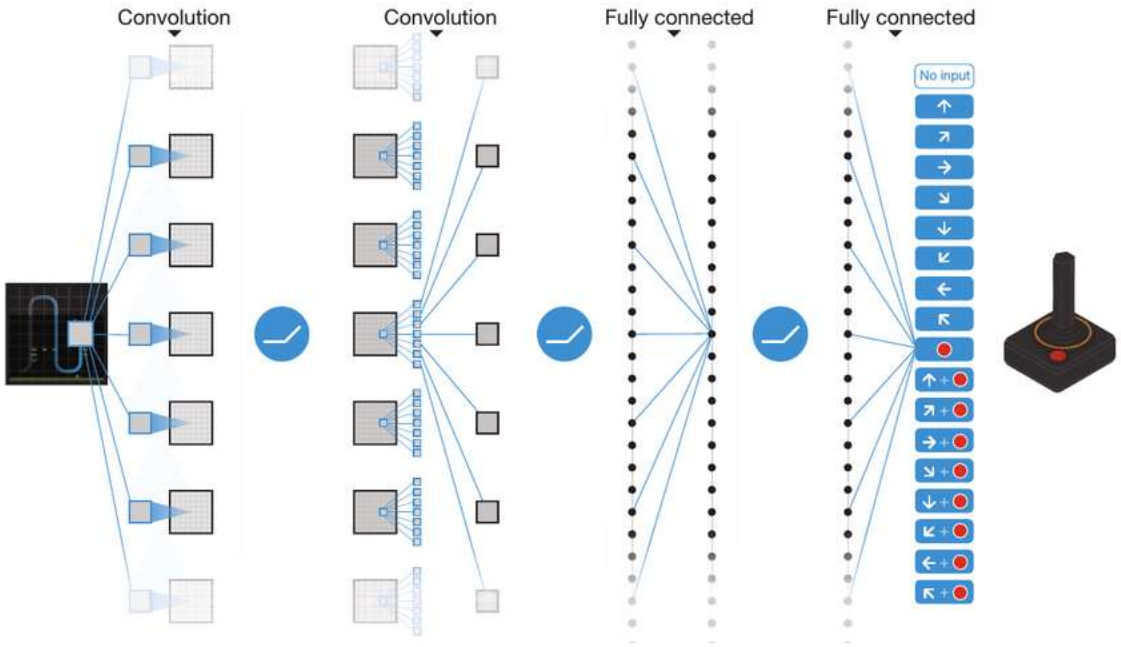


Figure 9. *DQN Network Architecture*

Source: https://www.researchgate.net/figure/Deep-Q-network-architecture_fig2_313156999

The above figure shows the architecture of a Deep Q-Network (DQN), used in reinforcement learning to make decisions. Input data, like images or sensor readings, is processed through convolutional layers to extract important features. These features are passed to fully connected layers, which calculate Q-values for each possible action. The action with the highest Q-value is chosen, helping the agent learn by exploring the environment and receiving rewards. The joystick on the right represents the possible actions the agent can take based on the Q-values.

3.9 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 is an advanced reinforcement learning algorithm designed for continuous action spaces. It is a variant of the Deep Deterministic Policy Gradient (DDPG) algorithm, with improvements to stabilize training and avoid overestimation bias. TD3 uses two Q-networks and delayed updates to improve learning efficiency and robustness. In robotic navigation tasks, TD3 can be used to handle continuous control problems, such as velocity and direction control, by optimizing the robot' s movements to achieve goals such as collision-free exploration and path planning. Key Components of TD3 network are:

- Two Q-Networks: TD3 uses two Q-networks instead of one, which helps to reduce the overestimation bias often seen in DDPG. During training, TD3 computes the target Q-value by taking the minimum Q-value predicted by these two networks. This conservative estimate makes learning more stable by discouraging over-optimistic evaluations of actions.
- Delayed Policy Updates: In TD3, the policy (actor) network is updated less frequently than the Q-networks. This delay ensures that the policy is optimized using more accurate Q-value estimates, which further stabilizes training by reducing the risk of propagating errors through the policy network.
- Target Policy Smoothing: TD3 adds a small amount of random noise to the target policy's actions during training. This prevents the policy from overfitting to narrow peaks in the Q-value estimates, improving robustness by encouraging more exploration of the action space.

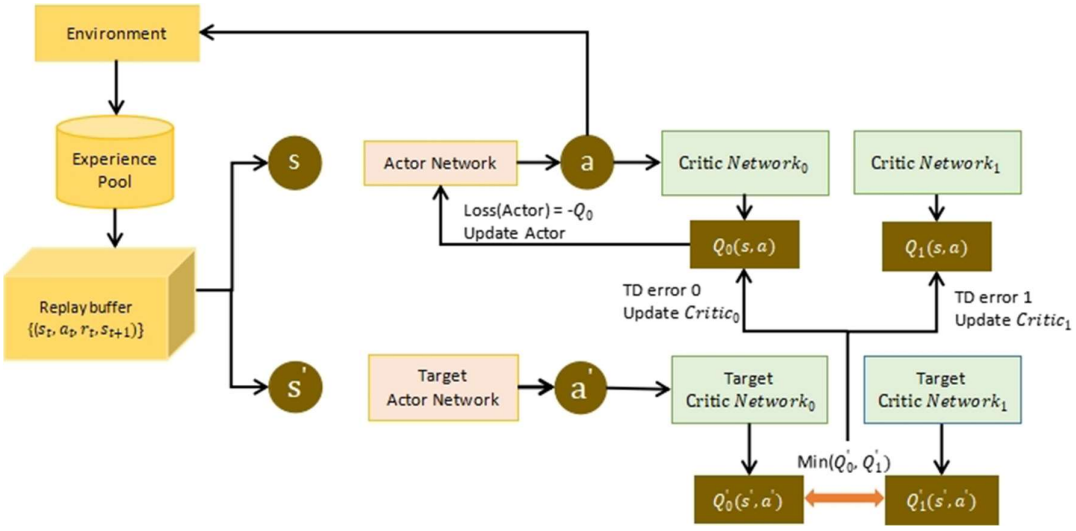


Figure 10. *TD3 Network Architecture*

Source: https://www.researchgate.net/figure/TD3-network-structure_fig3_377551009

Figure 10 demonstrates the architecture of the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, which is an improvement over DDPG for reinforcement learning in continuous action spaces. The process begins with the agent interacting with the environment, storing experiences (state, action, reward, and next state) in a replay buffer.

The Actor Network generates actions based on the current state, while two Critic Networks independently estimate the Q-values to address overestimation bias. Target networks, which are delayed versions of the actor and critic networks, are used to stabilize training by updating more slowly. The Q-value used for training is the minimum of the two critics, reducing overestimation and improving policy learning. This architecture ensures more robust and stable learning in complex.

3.10 Gazebo Simulation Environment

Gazebo is a powerful open-source simulation environment [13] widely used for testing and developing robotic systems. It provides a realistic simulation of both indoor and outdoor environments, allowing robots to interact with simulated sensors and objects under realistic physical constraints. Gazebo supports various robot models, sensors, and environments, and is highly integrated with ROS, making it a go-to platform for developing and testing robotics algorithms before deploying them in real-world applications.

Gazebo's physics engine accurately models real world dynamics such as gravity, inertia, friction, collisions and wind. This makes it particularly useful for testing control algorithms, such as PID controllers, and motion planning algorithms, without the risk of damaging actual hardware. The simulated environment in Gazebo can be configured to include various terrains, obstacles, and sensor data, including lidar, IMU, cameras, and GPS.

In this thesis, Gazebo was utilized to simulate the UAV's environment, enabling the testing of navigation, obstacle avoidance, and mapping algorithms in complex, 3D environments. The Gazebo simulator provides a realistic platform for UAVs to interact with their surroundings, with the added advantage of replicating real-world challenges, such as sensor noise and environmental variability.

3.11 Yolo (You Only Look Once)

YOLO (You Only Look Once) is a deep learning-based object detection framework designed for real-time applications. Developed by Joseph Redmon and colleagues, YOLO has become one of the most popular methods for object detection due to its speed and accuracy. Unlike traditional methods that apply object detection over multiple passes or regions, YOLO processes the entire image in a single pass, making it highly efficient.

It divides the input image into a grid and predicts bounding boxes and class probabilities directly from each grid cell. This one step approach, where detection happens in a single evaluation of the network, allows YOLO to be extremely fast and suitable for real time applications.

Due to its design, YOLO is capable of processing frames at high speeds, which makes it ideal for applications like video analysis, autonomous driving, and real-time surveillance. Its balance of speed and accuracy enables practical deployment on both high-powered GPUs and more limited edge devices.

YOLO uses anchor boxes to predict bounding boxes for objects, with each bounding box containing attributes like the center, width, height, and a confidence score. This structure makes YOLO especially efficient for handling complex scenes with multiple objects.

YOLO is adaptable for training on custom datasets, allowing users to define their own classes for specific applications. Transfer learning is commonly applied in YOLO, making it easier and faster to train a model with limited data, which is valuable for specialized object detection tasks.

YOLO is widely used in areas such as security and surveillance, autonomous vehicles, medical imaging, and retail analytics. Its ability to handle high-speed video feeds and accurately detect multiple objects has made it a go-to solution for applications requiring quick, precise object recognition.
In this thesis the YOLOv5 has been used for the following reasons:

1. Stability and Reliability: YOLOv5 has been extensively tested and widely adopted in the computer vision community, ensuring stability and reliability in a variety of applications. Later versions, while potentially offering improvements, may not yet have the same level of support and thorough validation.

2. Performance and Speed: YOLOv5 is known for its speed and real time detection capabilities, making it ideal for projects requiring fast, efficient processing without significant hardware demands. In cases where hardware or timeline constrained extensive experimentation, YOLOv5 offers a balanced solution.

3. Community and Documentation: Due to its popularity, YOLOv5 benefits from a robust support community and extensive documentation, which can accelerate development and troubleshooting. The ease of access to guides, tutorials, and existing solutions reduces development time compared to newer versions.

4. Compatibility and Resource Optimization: YOLOv5 is compatible with a broad range of hardware setups and does not require the same level of computational power as some newer, more complex models. This can be especially important when working with limited hardware resources or if needing to deploy models to edge devices.

5. Mature Framework with Customization Support: YOLOv5 provides mature support for transfer learning and customization, allowing efficient fine-tuning for specific datasets.

This might make it easier to adapt to unique project needs without the experimental risks associated with newer versions.

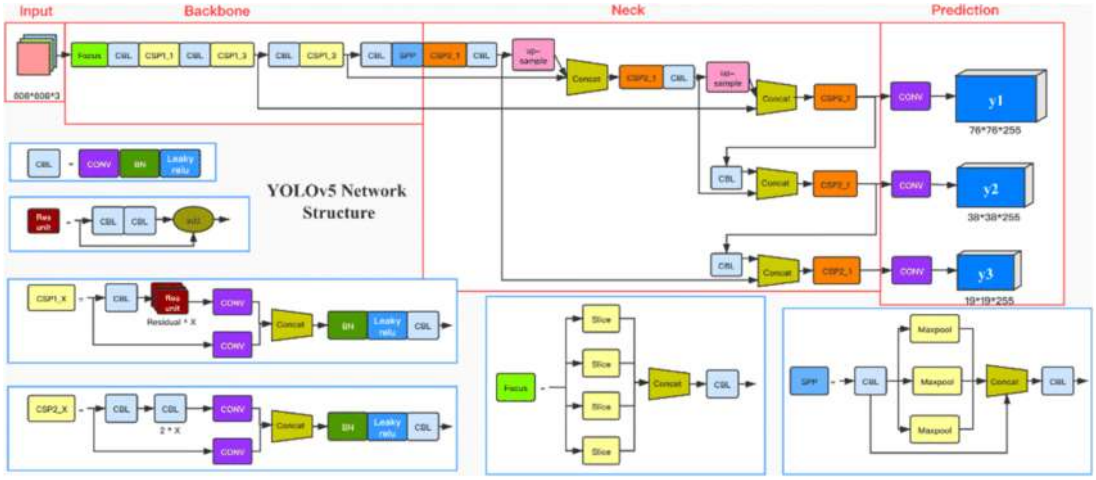


***Figure 11**. YOLOv5 diagram*

Source:

https://www.researchgate.net/publication/359079542/figure/fig1/AS:1152916178120725@1651888045971/The-diagram-of-YOLOv5-structure.png

This figure illustrates the architecture of the YOLOv5 network, which is divided into three main sections: Backbone, Neck, and Prediction. The Backbone extracts features from the input image using convolutional layers and CSP (Cross Stage Partial) blocks for efficient representation. The Neck enhances feature fusion across scales using layers like PANet and FPN, ensuring robust multi-scale detection. The Prediction layer outputs the final bounding boxes and class probabilities for objects at different scales, optimized for real-time object detection.

4 Implementation

This thesis focuses on advancing the capabilities of Unmanned Aerial Vehicles (UAVs) for autonomous navigation and exploration in unknown or GPS-denied environments through the integration of Simultaneous Localization and Mapping (SLAM) and Artificial Intelligence (AI) techniques. The project aims to develop a robust system that enables UAVs to dynamically understand and interact with their surroundings, making real-time decisions for navigation, obstacle avoidance, and mission-specific tasks. By leveraging the latest advancements in robotics, computer vision, and machine learning, this research seeks to significantly enhance the autonomy, efficiency, and safety of UAV operations in complex environments.

The thesis is implemented by two robots. The first robot is the UAV hector quadrotor

The quadrotor is equipped with:

- A 360-degree lidar sensor
- A sonar sensor which is responsible for providing crucial information when the uav is flying in low altitude
- An RGB-D camera with a 120-degree field of view, which is responsible for mapping the 3D environment
- An IMU sensor

Figure 12. *Hector Quadrotor*

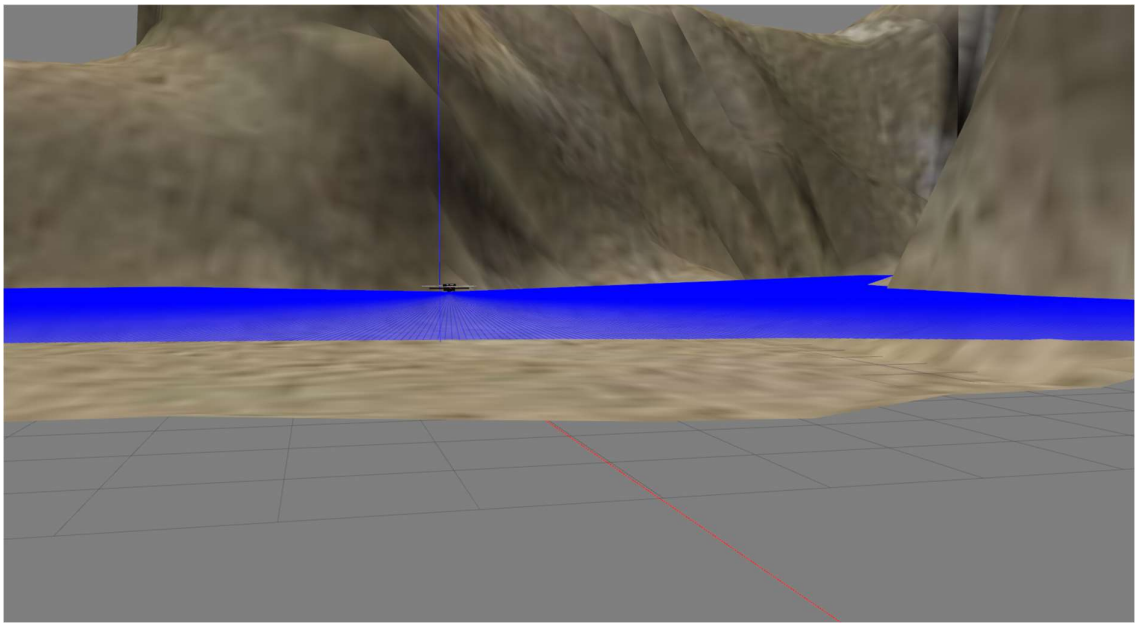


Figure 13. *Lidar beams*

4.1 SLAM

For the first requirement of this thesis, RTAB-Map SLAM has been employed with a focus on RGB-D odometry. The primary inputs for localization are visual and depth data from the UAV's RGB-D camera, which provides direct odometry. Lidar is only used to assist in generating the 2D map and for collision avoidance but does not contribute to the odometry.

The algorithm receives data from the UAV' s RGB-D camera, and the lidar sensor, and creates a 2D map. By integrating Octomap we can also make a 3D map/occupancy grid of the area. The 2D map has no value for the UAV, since it navigates in 3D space, but it can be used with other land robots.

In this implementation, RTAB-Map utilizes the UAV's RGB-D camera and lidar sensor to continually track the UAV's movement through visual odometry. As the UAV explores its surroundings, RTAB-Map creates a pose graph that records the robot's trajectory over time. Each node in this graph represents a pose (position and orientation) of the UAV, along with the sensor data captured at that moment.

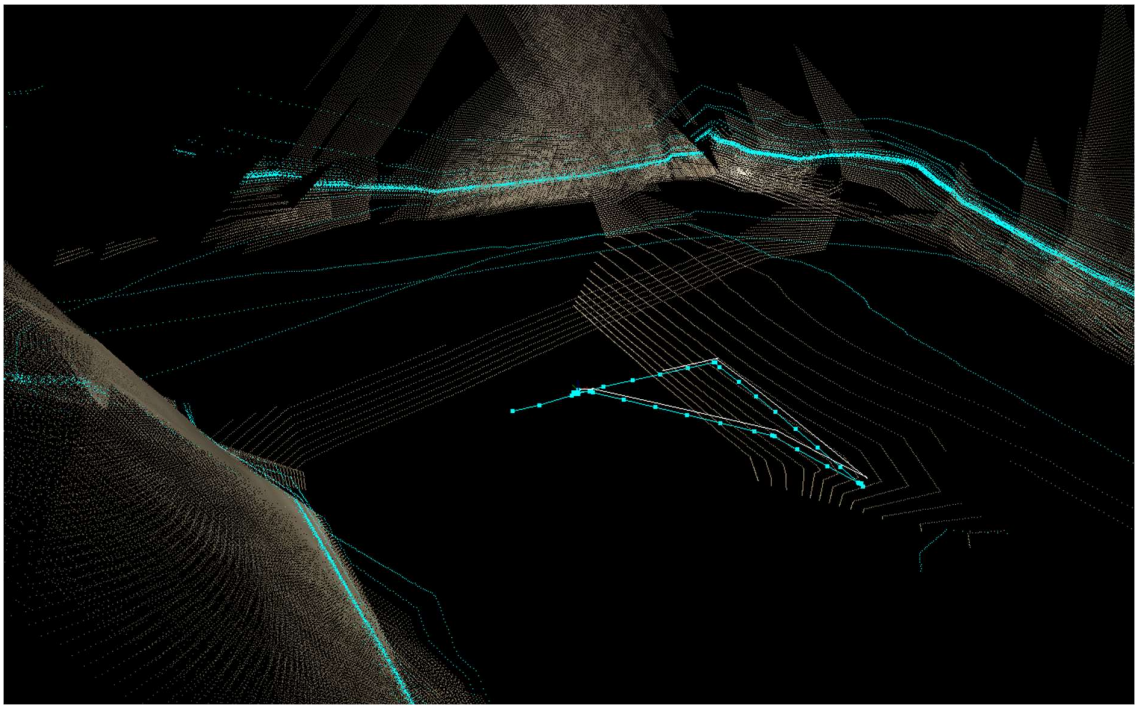


Figure 14. *Loop Closure*

To mitigate drift in odometry, RTAB-Map leverages loop closure detection by identifying previously visited locations. This correction aligns current and past poses, which improves the accuracy and reduces cumulative error over time. Since odometry is based solely on RGB-D, this system is particularly suited to environments where visual features are abundant.

As for the odometry strategy the frame to map method has been selected. In Frame-to-Map (F2M) [14] odometry, each sensor frame (e.g., from a lidar or depth camera) is matched to a previously built local map rather than the last sensor frame. This strategy reduces overall drift, as the system uses the map to align new frames. By relying on this broader map for odometry, F2M improves localization accuracy, especially in feature rich environments. This method ensures the UAV maintains a precise trajectory over longer distances, making it well suited for complex, GPS denied environments.

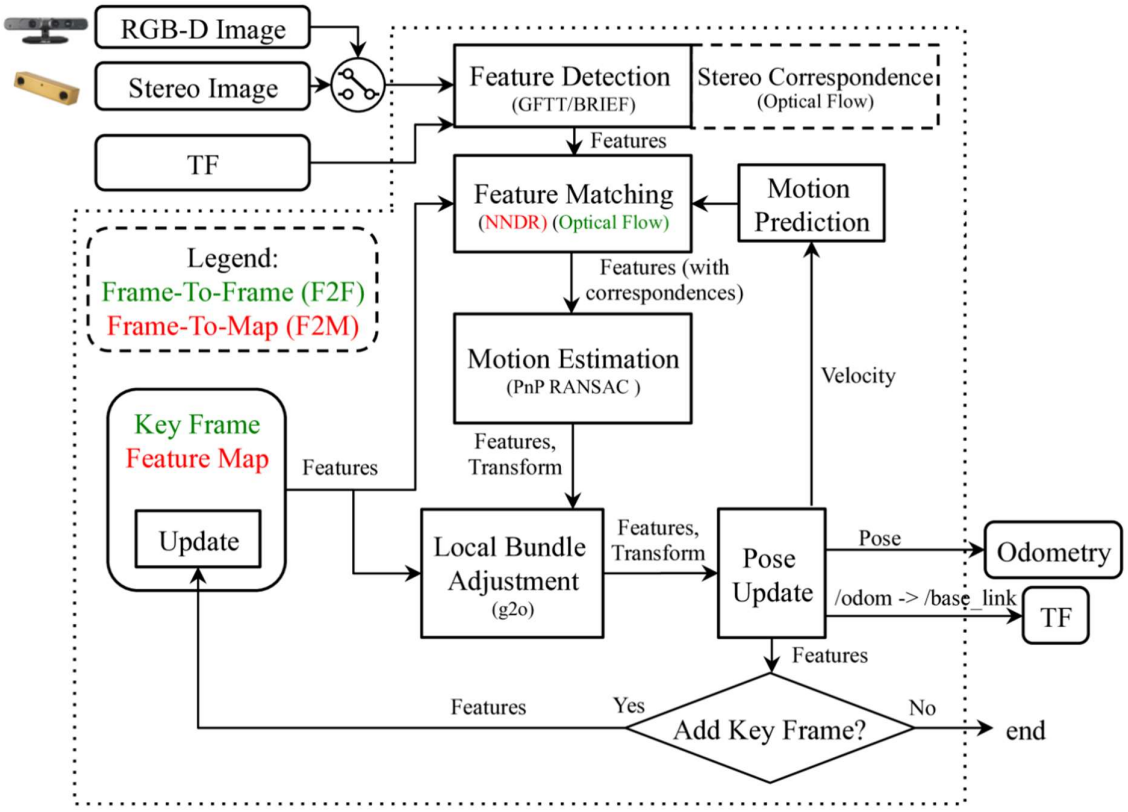


Figure 15. *Frame-to-Map and Frame-to-Frame*

Source: https://www.researchgate.net/figure/RTAB-MAP-visual-odometry-13-diagram_fig2_340813497

In Frame-to-Map odometry for RTAB-Map, visual odometry is primarily achieved through features derived from RGB-D data, with methods like BRIEF descriptors and the Nearest Neighbour Distance Ratio (NNDR). These features allow the system to match the RGB-D camera's visual data against a map, enhancing localization accuracy.

To handle situations where RGB-D odometry is less effective, such as in environments with low texture or poor lighting conditions, we used Iterative Closest Point (ICP) odometry. ICP is a method that aligns 3D point clouds by iteratively minimizing the distance between corresponding points in successive frames, providing robust odometry even in feature-poor areas. By utilizing data from the 2D LiDAR on the /scan topic and the 3D point cloud from the /camera/depth/points topic, ICP can accurately estimate the UAV's pose based on spatial geometry alone, without relying on colour or texture cues. This approach ensures reliable navigation by complementing RGB-D odometry, particularly in scenarios where visual information is insufficient.

Regarding OctoMap, it updates the 3D occupancy grid to reflect changes detected through depth data. This occupancy grid leverages probabilistic updates, using Bayesian principles where the probability of a voxel's occupancy is refined based on sensor data, which enhances the accuracy and robustness of the mapping process.

The occupancy probability in OctoMap is updated as follows:

$$P(m \mid z) = P(z \mid m)P(m)P(z)$$
$$P(m \mid z) = P(z)P(z \mid m)P(m)$$

Where P(m|z)P(m|z) is the posterior probability of a voxel being occupied given the sensor data zz, P(z|m)P(z|m) is the sensor model likelihood, and P(m)P(m) and P(z)P(z) are the prior and evidence probabilities, respectively.

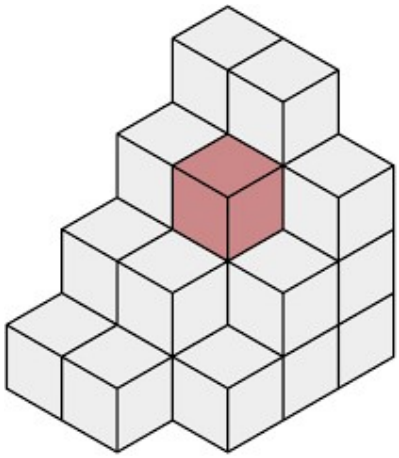


Figure 16. *Voxel Grid Representation with Highlighted Voxel*

Source: https://en.wikipedia.org/wiki/Voxel

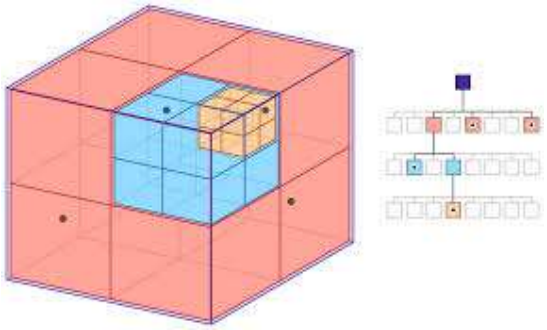


Figure 17. *Octree Structure for 3D Space Partitioning*

Source: https://koukan.github.io/voxomap/doxygen/html/index.html

OctoMap ensures that the UAV maintains a real-time understanding of its position in 3D space, crucial for navigation in GPS denied environments.

4.2 AI-Driven Decision Making

DQN

In this thesis, two AI models were developed, one for a UAV and another for a land vehicle, to enable autonomous navigation to a goal point while avoiding obstacles. Both systems operate without prior knowledge of the map and rely solely on sensor data for navigation.

For the UAV, a Deep Q-Network (DQN) was implemented, leveraging an existing codebase from a publicly available GitHub repository. DQN is well suited for environments with discrete action spaces and requires less training than a continuous action space model, making it an appropriate choice for UAV navigation tasks. For the land vehicle, a TD3 (Twin Delayed Deep Deterministic Policy Gradient) network was used, which excels in continuous action spaces.

The DQN algorithm used in this project was integrated with the UAV's sensor data, allowing real time decision making based on the UAV's perception of its environment. The DQN model is designed to approximate the Q-value function, which estimates the expected future rewards for each action the UAV can take from a given state. Over time, the UAV learns to take actions that maximize cumulative rewards, guiding it to navigate efficiently and safely.
While the base implementation was sourced from GitHub, several key modifications were made to adapt it for this project. These include:

Modifying the reward function to account for the UAV's specific navigation goals.
Fine tuning hyperparameters like the learning rate and exploration decay rate to suit the UAV's environment. Integrating depth camera and LiDAR data, enabling the UAV to perceive and react to its surroundings in real-time.

The existing DQN implementation provided a solid foundation for developing a reliable UAV navigation system. The modifications ensured that the system could meet the specific

requirements of this thesis, while proper citation and credit have been given to the original author of the code. Key Components of the DQN Model are:

- **Replay Buffer:**

  The replay buffer stores the UAV's experiences (state, action, reward, next state), which are sampled randomly during training. This helps break the temporal correlations between consecutive actions, stabilizing the training process.

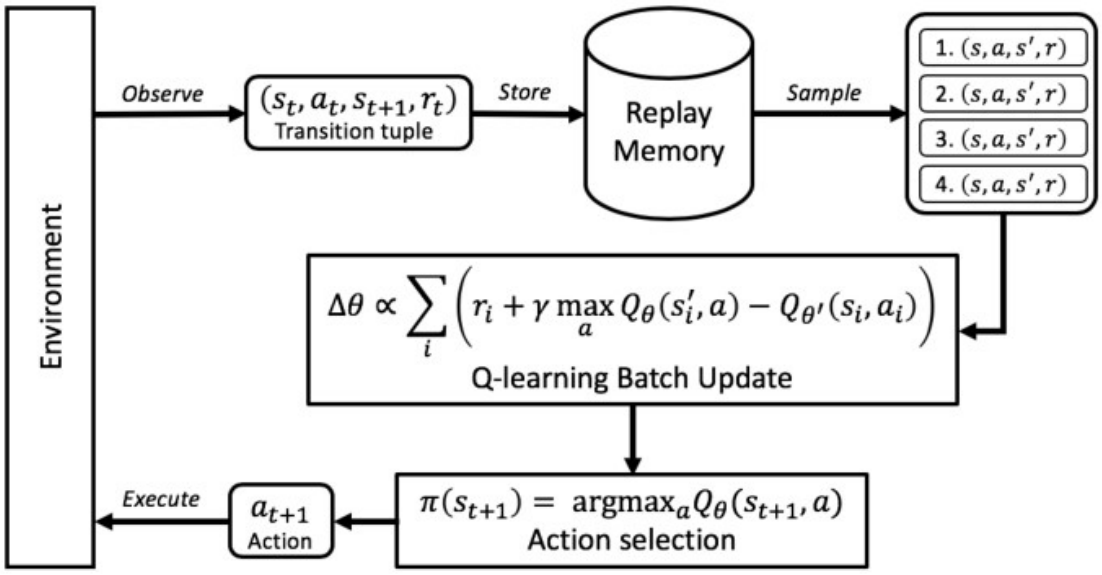


Figure 18. *Deep Q-Network (DQN) Training Process Flowchart*

Source: https://www.researchgate.net/figure/Lifelong-Reinforcement-Learning-using-Replay-memory-to-learn-tasks_fig1_331318744

- **Q-Network:**

  The Q-network approximates the Q-value for each possible action given a state. It is iteratively updated by minimizing the difference between the predicted Q-values and target Q-values, calculated using the Bellman equation.

- **Target Network:**

  A separate target network is used to compute target Q-values, providing greater training stability by updating less frequently than the main Q-network.

  Exploration-Exploitation Trade-off: The DQN algorithm uses an epsilon-greedy policy to balance exploration (choosing random actions) and exploitation (choosing the best-known

action). This ensures that the UAV explores the environment sufficiently while learning an optimal navigation policy.

The UAV relies on sensor data, primarily from a depth camera and LiDAR, to perceive its environment. The DQN model processes this data using Convolutional Neural Networks (CNNs) to extract meaningful features from the depth images, which guide the UAV's decision-making.

The first convolutional layer applies filters to the raw depth images to detect basic features like edges and textures. This is followed by non-linear activation functions like ReLU to enhance feature detection.

Subsequent convolutional layers process these features to identify more complex patterns, such as object shapes and distances.

- **Pooling and Dimensionality Reduction:**

  After each convolutional stage, max pooling is used to reduce the size of the feature maps while preserving essential information. This reduces computational complexity and speeds up the network's processing.

  After the convolutional layers, the output is flattened and passed through fully connected layers that compute the Q-values or action values. These layers combine the extracted features and use them to make navigation decisions, such as whether the UAV should move forward, turn left, or turn right.

  The CNN processes images frame by frame, creating a sequence of depth images representing the UAV's surroundings. These frames are used to:

  1. **Detect obstacles:**

     The CNN identifies obstacles by analyzing depth data and determines the appropriate action to avoid collisions.

  2. **Plan goal navigation:**

     By recognizing free space in the images, the CNN helps the UAV plan a path toward the goal while avoiding obstacles.

  3. **Make real-time decisions:**

     The processed image data, along with other sensor inputs, is fed into fully connected layers to decide the next action based on the UAV's current state.

     The UAV receives positive rewards as it moves closer to the goal.

     Negative rewards are given for collisions with obstacles or going out of bounds. While a large positive reward is awarded when the UAV successfully reaches its

target. This reward structure incentivizes the UAV to avoid obstacles and navigate toward the goal efficiently.

In addition to integrating sensor data and adjusting the reward structure, several other enhancements were made to improve the UAV's performance:

4. **Hyperparameter tuning:**
   Key hyperparameters such as learning rate, exploration decay, and batch size were fine-tuned to optimize the UAV's learning process.
5. **Sensor fusion:**
   The DQN was modified to integrate multiple sensor inputs, including both the depth camera and LiDAR, allowing for more robust decision-making in 3D space.

In this approach, the Twin Delayed DDPG (TD3) algorithm, an extension of the DDPG architecture, is used for training a mobile robot to navigate in complex environments. TD3 employs an actor-critic architecture with two critic networks to mitigate the problem of overestimating Q-values, which is common in deep reinforcement learning algorithms. By selecting the smaller Q-value estimation between the two critic networks, TD3 improves the stability of the learning process.

The actor network is responsible for determining the robot's actions, linear and angular velocities, based on the environmental state. The architecture uses fully connected layers with ReLU activation, and the final output is bounded by a Tanh function, capping the action range to ensure the robot stays within its physical limits.

The critic networks estimate the quality of the actions suggested by the actor network. They take both the environmental state and the actions as inputs and output the Q-values, representing the expected cumulative reward of the state-action pair. By maintaining two critics, TD3 ensures that the learning is more conservative, leading to better overall performance.

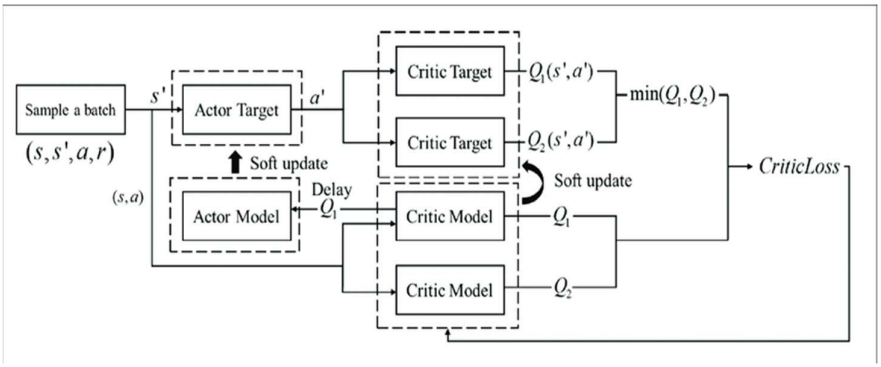


Figure 19. *Twin Delayed Deep Deterministic Policy Gradient (TD3) Architecture Diagram*

Source: https://www.researchgate.net/figure/Network-structure-of-TD3_fig4_366486764

Soft Updates are applied to the actor and critic networks to maintain stability. The idea is to gradually adjust the target networks by incorporating a small portion of the learned network's parameters. This approach prevents overfitting to a particular set of experiences and ensures more generalizable policies.

In training, experiences (states, actions, rewards) are stored in a Replay Buffer, and the network is optimized by sampling batches from this buffer. The actor network is updated less frequently than the critics to ensure more accurate policy learning. This delayed update mechanism reduces instability and ensures that the critic networks provide reliable feedback.

The TD3 training process optimizes both actor and critic networks through gradient descent, using the Bellman equation to compute the target Q-values. Noise is added to the actions during training to encourage exploration of the state space, while noise clipping ensures that this exploration remains controlled.

Through this method, the mobile robot learns to navigate autonomously, choosing actions that maximize long-term rewards while avoiding obstacles. TD3's ability to handle continuous action spaces makes it particularly suitable for tasks like autonomous driving, where precise control of the robot's movement is critical.

This architecture aligns with the goal of creating robust, efficient, and adaptable navigation algorithms for autonomous systems, as demonstrated in the training of your land rover.

The environment is represented numerically through the robot's sensors, primarily using a laser scanner (LiDAR) for distance readings. The robot's state includes laser readings, the distance and angle to a goal, and its prior actions. These inputs are processed by the TD3 network, which outputs the optimal motion commands for the robot.

TD3's delayed updates of the actor network, compared to the critics, ensure more accurate policy learning. The algorithm's use of a replay buffer allows the model to optimize over a history of interactions with the environment. To encourage exploration, noise is added to the robot's actions during training, allowing it to explore various paths.

This method enhances autonomous navigation by helping the robot make decisions that maximize long-term rewards, leading to efficient path planning in complex environments.

4.3 Robust Navigation Algorithms

In this chapter, we will analyse how the UAV navigates and explores the unknown environment while keeping track of its position and creating the map.

During the initialization phase a command will be send to the UAV for take-off, if the UAV is not already airborne. After taking off, the UAV will rotate around itself once, to uncover the unexplored area. The exploration algorithm will take as input the 2D occupancy grid, created by the SLAM algorithm, using its sensors (depth camera, lidar). Frontiers are the edges between the known map and unknown map. Detected frontiers are grouped into clusters based on proximity. This clustering helps reduce the complexity of exploration by treating several closely located frontier points as a single exploration target.

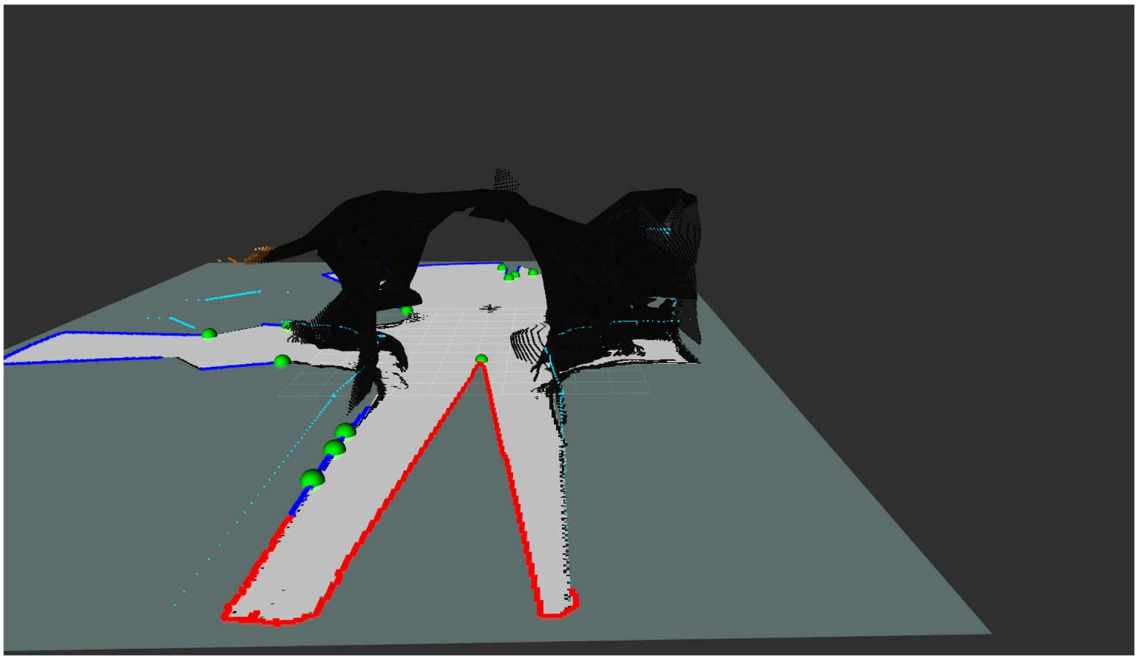


Figure 20. *UAV exploring the map*

Once the frontiers are identified, the exploration algorithm determines the best frontier for the UAV to visit. At this point, the navigation and movement are controlled by the move_base node, which takes the selected frontier as a goal. The global planner (in this case, global_planner/GlobalPlanner) generates a global path to the frontier, considering the overall map. The local planner (in this case, teb_local_planner/TebLocalPlannerROS) generates feasible motion commands for the UAV using the A Star algorithm, avoiding obstacles and ensuring smooth movement towards the goal.

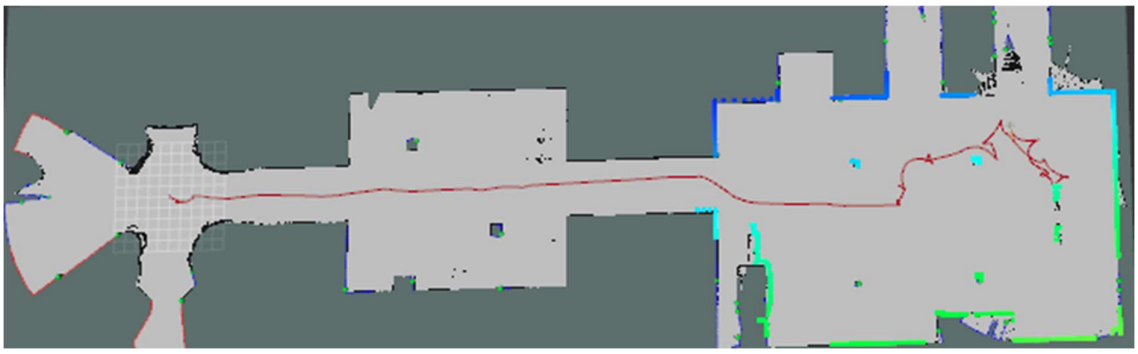


Figure 21. *UAV path followed (a)*

Figure 22. *UAV path followed (b)*

The local planner adjusts the UAV's movements in real-time, factoring in dynamic obstacles, terrain changes, and sensor data. This ensures that the UAV can continuously explore while avoiding collisions. Once a frontier is reached, the algorithm reevaluates the map, searching for new frontiers to explore until the environment is fully mapped.

For 3D navigation a method has been developed, utilizing OMPL and OctoMap and a simple PID controller has been implemented.

The exploration node is responsible for planning the trajectory of the UAV and navigating it to the user selected coordinates. The node receives the occupied cells via the OctoMap and transforms every voxel into a collision object. Because we do not have knowledge of full map and what obstacles might be ahead, the algorithm checks for future collisions after every waypoint it has visited. The node produces around 250-400 waypoint which are then smoothed

to 20 waypoints making the movement of the UAV more natural. After reaching a waypoint, the algorithm will check if at any future point an object is present in the path and will begin calculating a new path. This process will continue until the UAV will reach the point or find the nearest point in comparison to the given one.

The algorithm been used is the RRT Informed star. RRT* (Rapidly-exploring Random Tree Informed) is an advanced version of the RRT* algorithm used for efficient path planning, especially in high-dimensional spaces. The primary goal of the algorithm is to improve the convergence speed toward an optimal path between a start and goal point while maintaining probabilistic completeness.

The algorithm starts by building an initial random tree like RRT*. Once a feasible solution is found, the algorithm switches to elliptical sampling, focusing on regions that could yield a shorter path between the start and goal.

The cost of a path is computed by summing distances between consecutive nodes, aiming to minimize total cost.
Nodes are rewired to reduce path costs. The cost-to-come C(x)C(x) is updated as:

$$C(x) = C\left(x_{parent}\right) + \left|x - x_{parent}\right|$$

As new nodes are added, RRT* Informed rewires the tree to connect new points that provide shorter or more efficient paths, refining the solution over time.

The algorithm has been given a low bias goal with the intention of exploring as much space as possible before reaching the goal. In the RRT Informed* algorithm, goal bias refers to a strategy where the sampling process is biased toward the goal with a certain probability. Instead of randomly sampling the entire space, the algorithm generates samples that are closer to the goal region, increasing the likelihood of finding a path that reaches the goal efficiently. The probability represents how often the algorithm samples directly from the goal region, which helps to speed up convergence by focusing on more promising areas near the goal. A low goal bias favors broad exploration, which can be useful in highly constrained environments but may delay finding the goal in less complex spaces.

In summary, the UAV navigates through an unknown 3D space by continuously updating its map, identifying frontiers, and selecting optimal exploration points based on information gain and navigability. The system avoids revisiting explored areas while dynamically adjusting its trajectory based on the discovered frontiers. The integration of OctoMap and RTAB-Map enables the UAV to create a robust and detailed 3D map of the environment while efficiently covering the unexplored regions.

4.4 Enhanced Perception Systems

To complement the UAV's SLAM-based mapping and localization, the implementation integrates YOLOv5 for real-time object detection. YOLOv5's neural network is lightweight yet powerful, enabling the UAV to detect and classify objects in its environment, such as obstacles, humans, or other points of interest. This capability is particularly important for search-and-rescue missions or dynamic environments where static obstacle avoidance is insufficient.

The object detection pipeline is tightly coupled with the SLAM framework, where detected objects are spatially referenced in the 3D map generated by RTAB-Map. YOLOv5 processes the UAV's camera feed, allowing detected objects to be tagged and incorporated into the occupancy grid. These detected objects not only aid in situational awareness but also influence path-planning decisions, ensuring that the UAV navigates both efficiently and safely.

The YOLOv5 model is trained on datasets relevant to the environment, such as indoor structures or objects found in rescue scenarios. By providing real-time feedback on the location and type of objects, the system enhances the UAV's autonomy and allows for more informed exploration strategies. Additionally, the detection results are continuously updated as the UAV moves, refining its perception of dynamic environments.

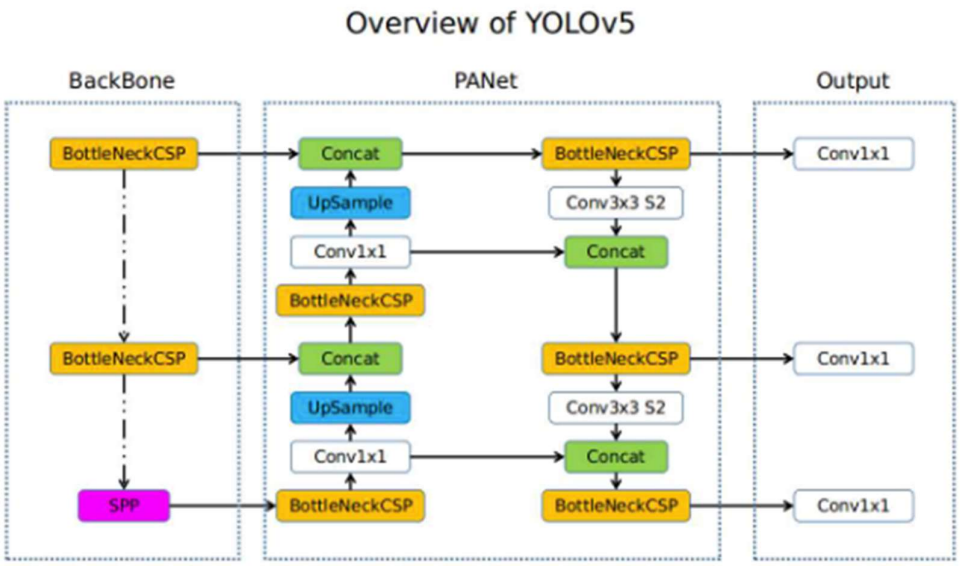


Figure 23. *YoloV5 Architecture*

Source: https://github.com/ultralytics/yolov5/issues/280

Figure 24. *Detection of fire hydrant*

Figure 25. *Detection of fire truck*

5 Results

5.1 SLAM

Localization The results are not accurate in comparison with a real environment. Gazebo introduces perfect shapes, making visual odometry, even with noise, work unrealistically well. The metrics for the height are taken by the camera and imu in relation to the ground. From this data the Absolute Position Error has been calculated and the following metrics reflect the position error in meters and orientationally in rads.

Absolute Position Error = $\sqrt{error_x^2 + error_y^2 + error_z^2}$

Mean = $\frac{1}{n}(\sum_{i=1}^{n} \text{error}_\text{i}) = \frac{error_1 + error_2 + \cdots + error_n}{n}$

Median = $\frac{\left(error_{\left\lfloor \frac{\#n+1}{2} \right\rfloor} + error_{\left\lceil \frac{\#n+1}{2} \right\rceil}\right)}{2}$

Min Error = min(error)

Max Error = max(error)

Root Mean Square Error = $\sqrt{\sum_{i=1}^{N} \widehat{error_i}^2}$

Sum of Squared Error = $\sum_{i=1}^{n} error_i^2$

Standard Deviation = $\sqrt{\frac{1}{N-1} * \sum_{i=1}^{N} (error_i - \widehat{error})^2}$

These metrics are used to evaluate the performance and accuracy of a system, such as a UAV, in terms of its positional estimates compared to ground truth values. Here's each metric represents:

Absolute Position Error: is the Euclidean distance between the estimated position and the ground truth, combining errors in all three dimensions (x, y, z). It provides a single metric for overall accuracy.

Mean: of all positional errors gives an average indication of how far off the system's estimates are over multiple samples. It is useful for understanding general accuracy trends.

Median: gives the middle value of the positional errors, making it less sensitive to extreme outliers compared to the mean. It provides a robust measure of central tendency.

Minimum Error (Min): The smallest observed error indicates the best-case performance of the system. It helps understand the system's potential accuracy in ideal conditions.

Maximum Error (Max): The largest observed error shows the worst-case performance, highlighting the situations where the system performed poorly.

Root Mean Square Error (RMSE): gives a weighted measure of error magnitude by penalizing larger errors more significantly. It is often used as a standard metric for comparing performance across systems.

Sum of Squared Error (SSE): is the total squared error over all observations and is commonly used in optimization to assess how well the model fits the data.

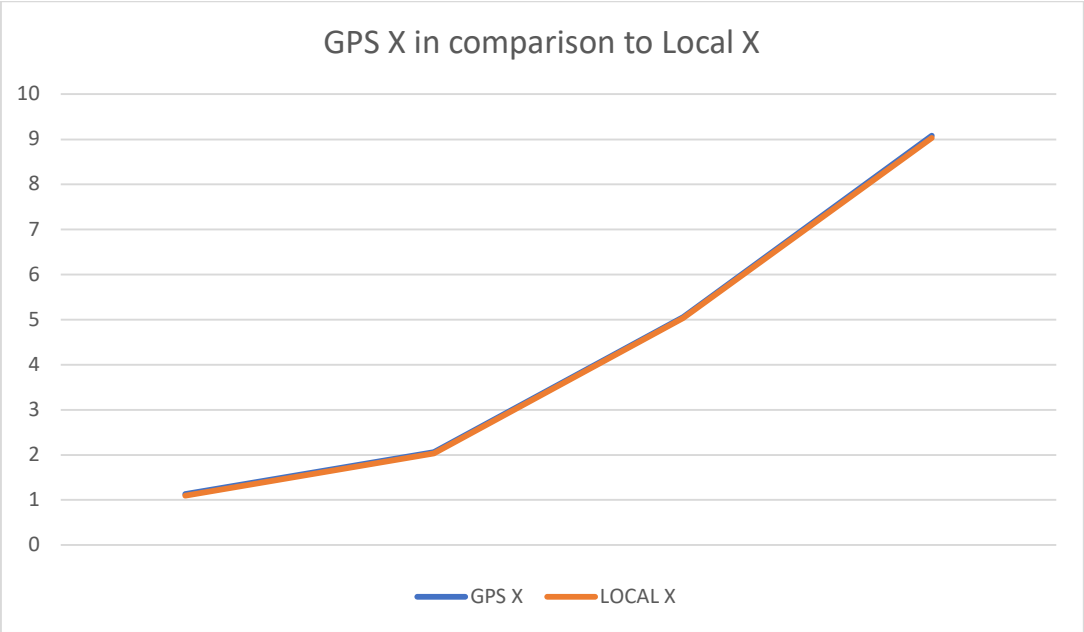
Standard Deviation: measures the spread or variability of positional errors, indicating how consistently the system performs. Lower standard deviation implies more stable performance.

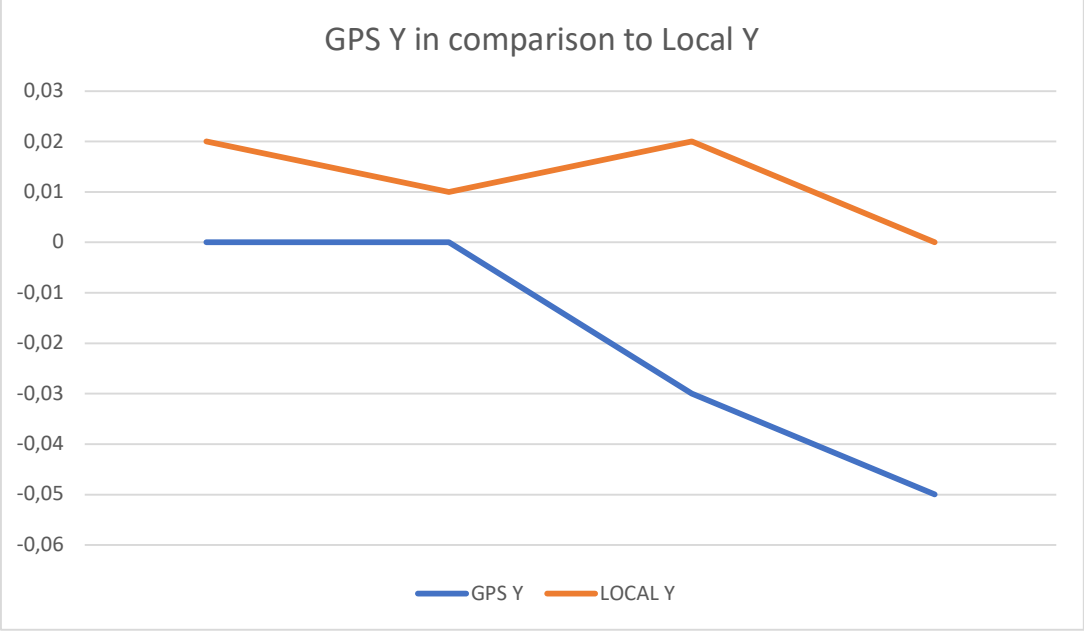
Movement in straight line

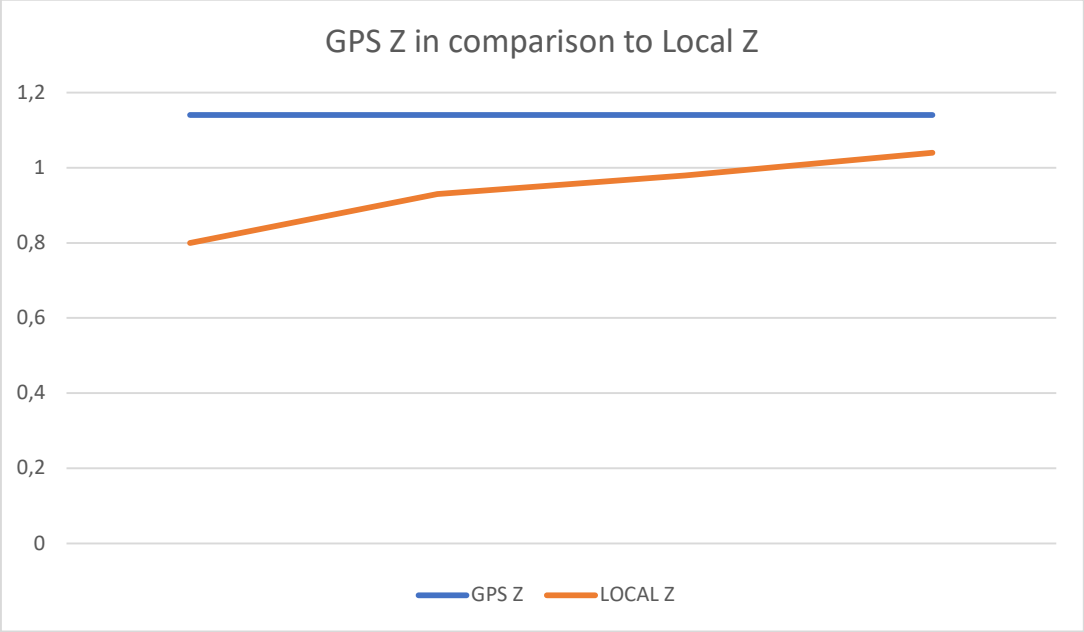
Distance	GPS X	GPS Y	GPS Z	Local X	Local Y	Local Z
1m	1.13	0.00	1.14	1.09	0.02	0.80
2m	2.06	0.00	1.14	2.03	0.01	0.93
5m	5.05	-0.03	1.14	5.03	0.02	0.98
9m	9.08	-0.05	1.14	9.03	0.00	1.04

Distance	Mean	Median	Min	Max	SSE	STD	RMSE
1m	0.3429	0.3429	0.3429	0.3429	0.117	0.00	0.3429
2m	0.2123	0.2123	0.2123	0.2123	0.045	0.00	0.2123

5m	0.1688	0.1688	0.1688	0.1688	0.028	0.00	0.1688
9m	0.1224	0.1224	0.1224	0.1224	0.15	0.00	0.1224
Overall	0.2116	0.1906	0.1224	0.3429	0.205	0.00	0.227







From the above tables the initial error is larger than in the subsequent metrics primarily due to the height difference caused by the drone starting from an elevated position on a small hill. This initial positioning introduces a more pronounced discrepancy in the local height, as the system requires time to accurately align its estimated position with the true terrain profile. At the start, the sensors—such as LiDAR or RGB-D cameras—may have limited data to properly map the environment, resulting in a higher localization error.

As the drone begins to move and explore, its sensors collect more data, allowing the mapping and localization algorithms to refine the environment model. The camera progressively captures new areas and incorporates them into the global map, reducing height-related discrepancies. This iterative mapping process improves the system's understanding of the surroundings, leading to a significant decrease in overall error. Additionally, as loop closures and sensor corrections occur, the alignment of the drone's position with the true environment becomes more precise, further minimizing the errors in subsequent metrics.

At 1 meter, the mean absolute error is the largest at 0.3429, which aligns with your observation regarding the initial height discrepancy. The drone's starting position on a small hill introduces a notable initial misalignment between the GPS and local coordinates, contributing to this higher error. The Standard Deviation (STD) is 0.00, indicating no variability in error across dimensions, which is expected given the short distance traveled. At 9 meters, the mean error is

the smallest at 0.1224, showing a significant improvement as the system stabilizes. This suggests that the combination of continuous sensor data integration and refined localization allows the drone to achieve better alignment with the environment over longer distances.

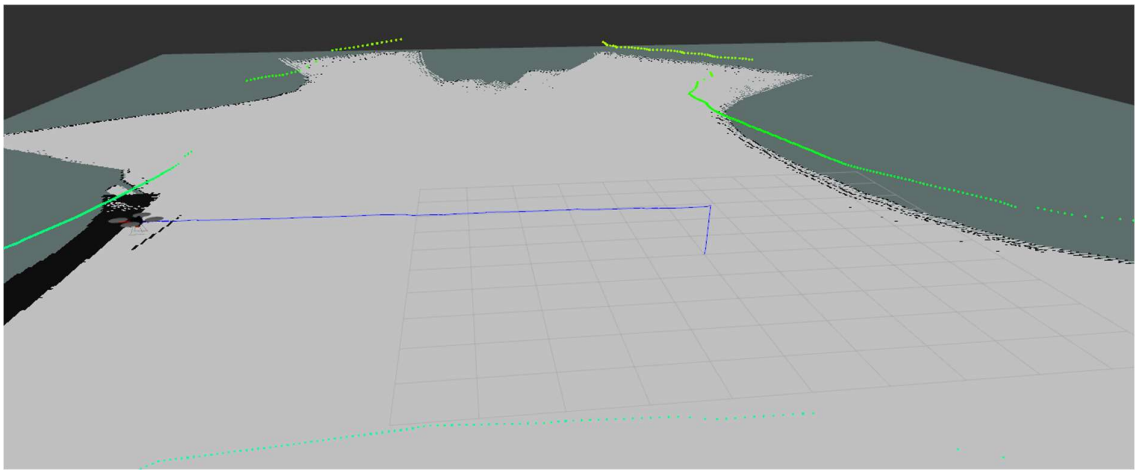


Figure 26. *Map Graph straight movement in Rviz*

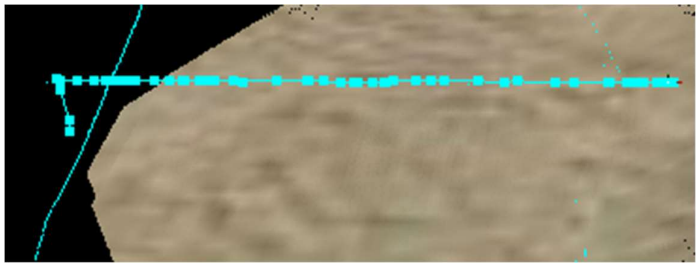


Figure 27. *Map Graph straight movement in RTABMap (a)*

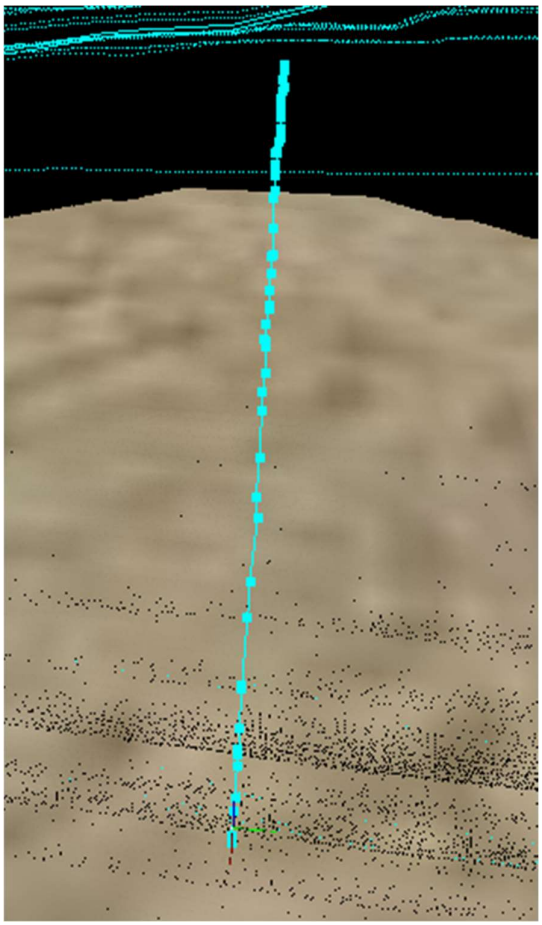


Figure 28. *Map Graph straight movement in RTABMap (a)*

Movement in square after returning to initial position:

Distance	GPS X	GPS Y	GPS Z	Local x	Local y	Local Z
1m	0.04	-0.01	1.28	-0.05	0.00	1.00
2m	0.14	0.01	1.23	0.03	0.04	0.92
5m	0.09	-0.21	1.24	0.09	-0.29	0.87

Distance	Mean	Median	Min	Max	SSE	STD	RMSE
1m	0.2943	0.2943	0.2943	0.2943	0.086	0.00	0.2943
2m	0.3303	0.3303	0.3303	0.3303	0.1091	0.00	0.3303
5m	0.3785	0. 3785	0.3785	0.3785	0.1433	0.00	0.3785
Overall	0.3344	0.3303	0.2943	0.3785	0.3384	0.00	0.3361

In this experiment, the drone follows a square trajectory and returns to its initial position. The data shows the discrepancies between GPS coordinates and local coordinates in terms of x, y, and z at different distances along the square path. This test evaluates the system's ability to handle cumulative errors in localization and mapping when revisiting previously traversed points.

The results highlight that while the system maintains reasonable accuracy, the errors are slightly higher compared to straight-line movement and it is caused by height discrepancy. This is expected because the test is restarted after every iteration, preventing the UAV from accumulating or leveraging prior mapping and localization data for subsequent iterations.

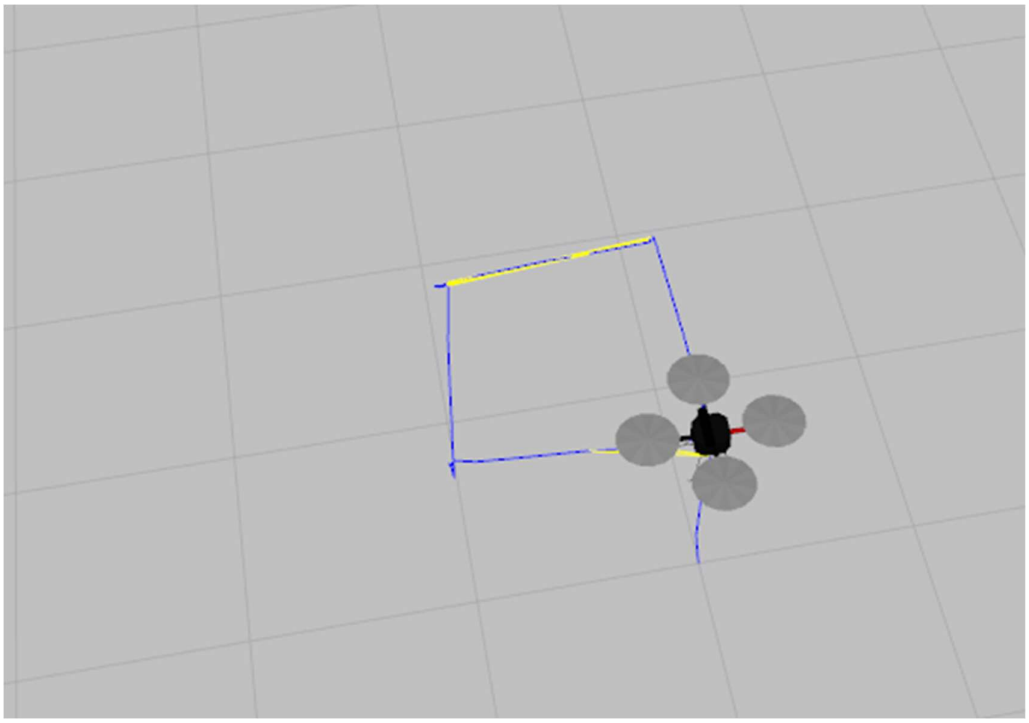


Figure 29. *Map Graph square movement of one-meter Rviz*

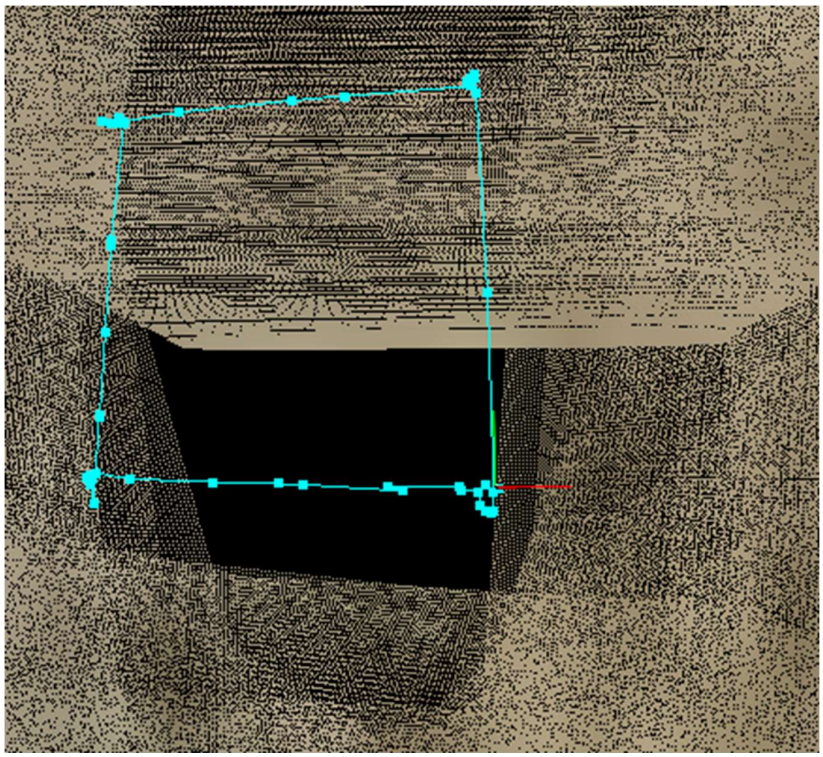


Figure 30. *Map Graph square movement of one-meter RTABMap*

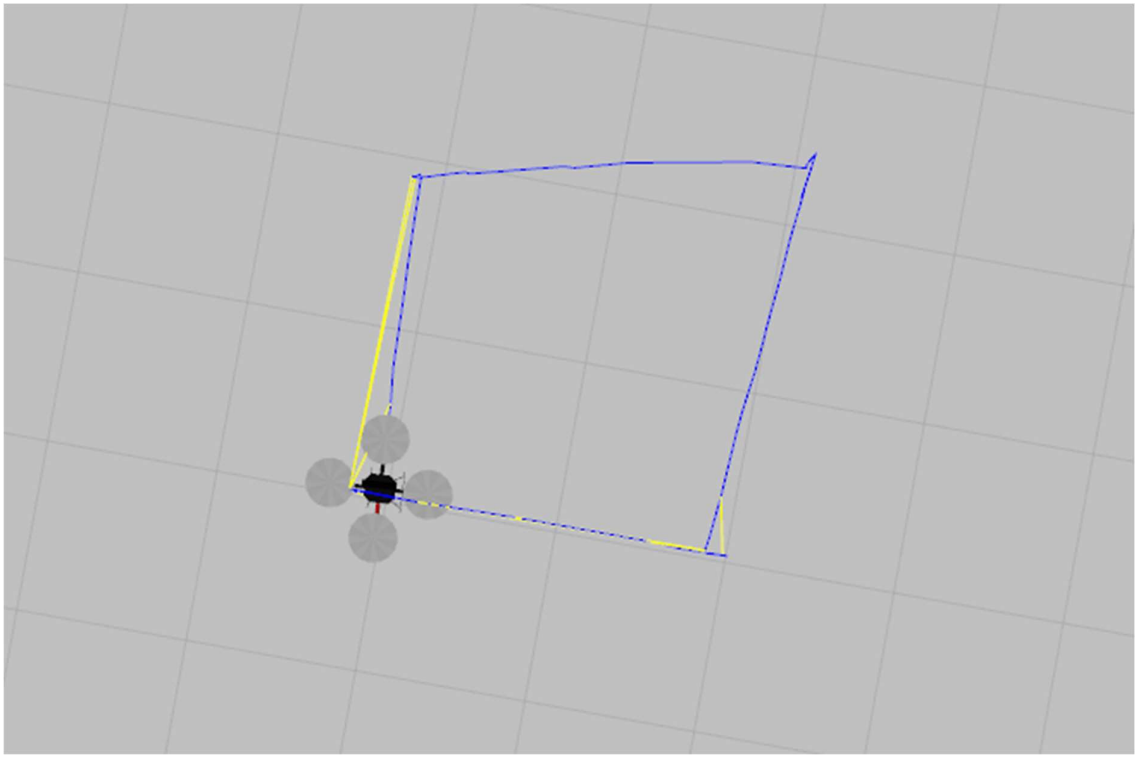


Figure 31. *Map Graph square movement of two meters Rviz*

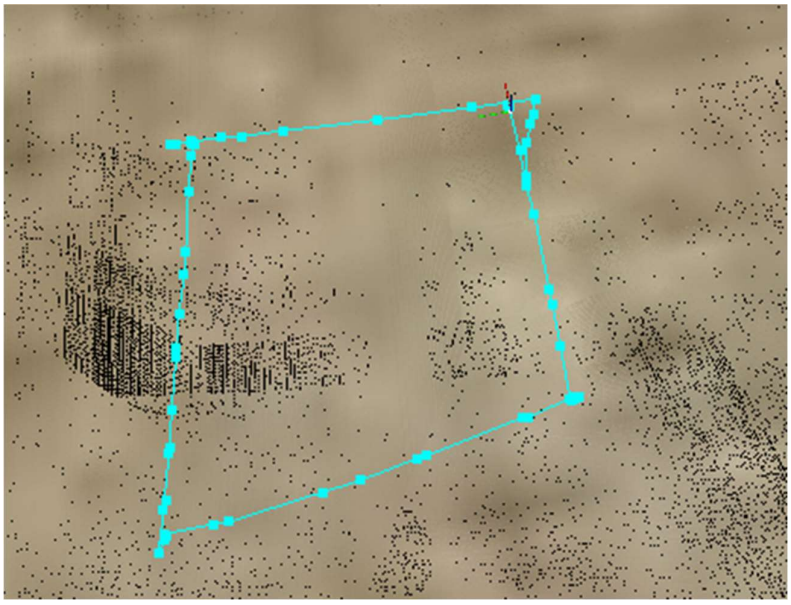


Figure 32. *Map Graph square movement of two meters RTABMap*

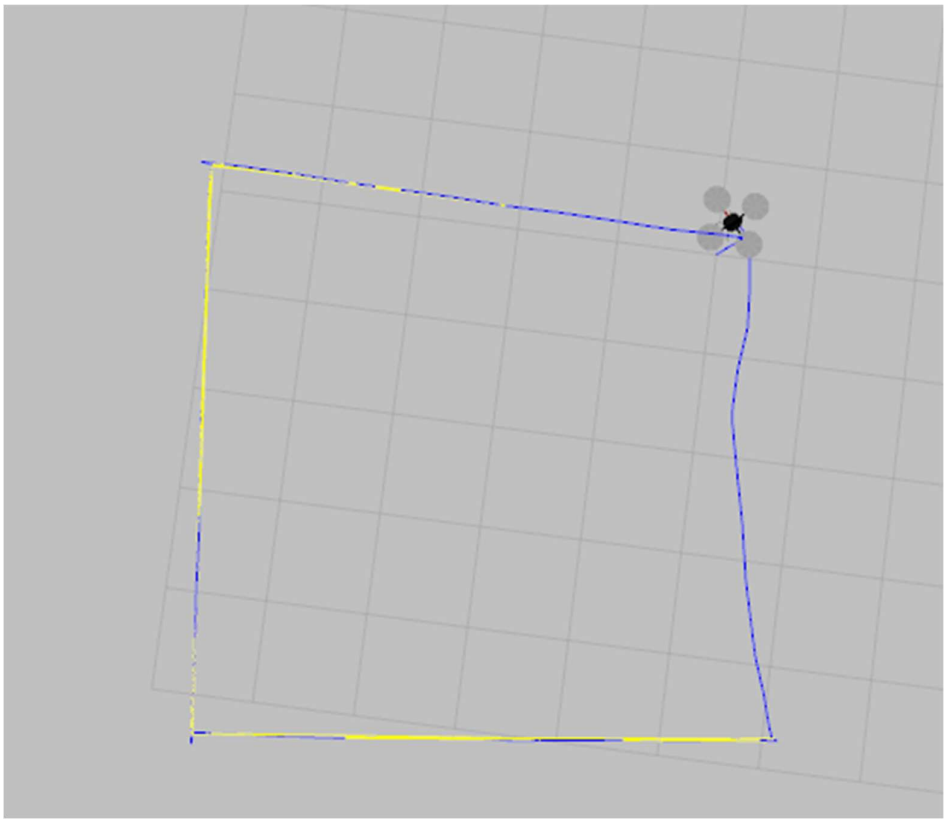


Figure 33. *Map Graph square movement of five meters Rviz*

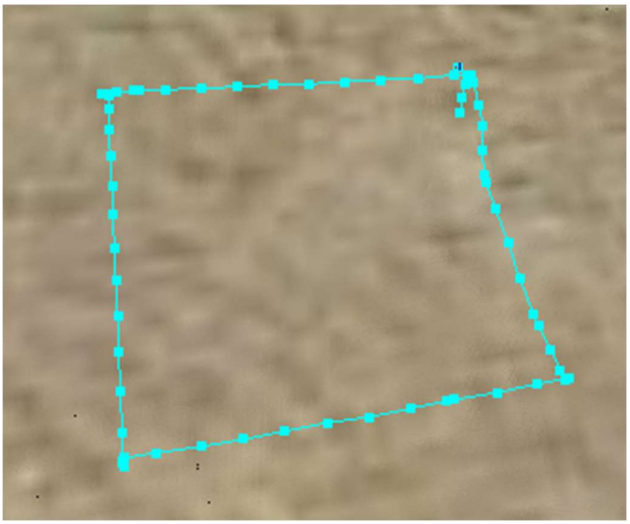


Figure 34. *Map Graph square movement of five meters RTABMap*

Recovery after lost odometry, random movement. Rotation recovery initiated

Distance	GPS X	GPS Y	GPS Z	Local x	Local y	Local Z
55m	0.85	0.74	1.39	1.04	0.6	1.54
75m	15.1	-70.0	1.00	15.4	-70.87	1.20

Distance	Mean	Median	Min	Max	SSE	STD	RMSE
55m	0.2796	0.2796	0.2796	0.2796	0.078	0.00	0.2796
75m	0.9418	0.9418	0.9418	0.9418	0.8896	0.00	0.9418
Overall	0.6107	0.6107	0.2796	0.9418	0.9676	0.00	0.6947

This test evaluates the UAV's ability to recover from lost odometry and navigate during random movement, including rotation recovery. The data reflects the discrepancies between GPS and local coordinates as the UAV regains stability and adjusts its positioning.

At 55 meters, the mean absolute error is 0.2796, with no variability (STD = 0.00), indicating that the system achieves reasonable accuracy after initial recovery. The GPS and local coordinates are closely aligned, showing effective error correction during shorter recovery distances. However, the z-coordinate still shows some discrepancy, highlighting minor challenges in altitude estimation. We should point out that since this is a cave environment, the surface is not flat.

At 75 meters, the mean error increases significantly to 0.9418, indicating the accumulation of larger discrepancies during extended random movements. This suggests that the UAV faces greater difficulty maintaining accurate localization over longer recovery distances. The larger error is primarily due to discrepancies in the y-coordinate, where the local y (-70.87) deviates notably from the GPS y (-70.0). Below are some pictures that were taken during the test.

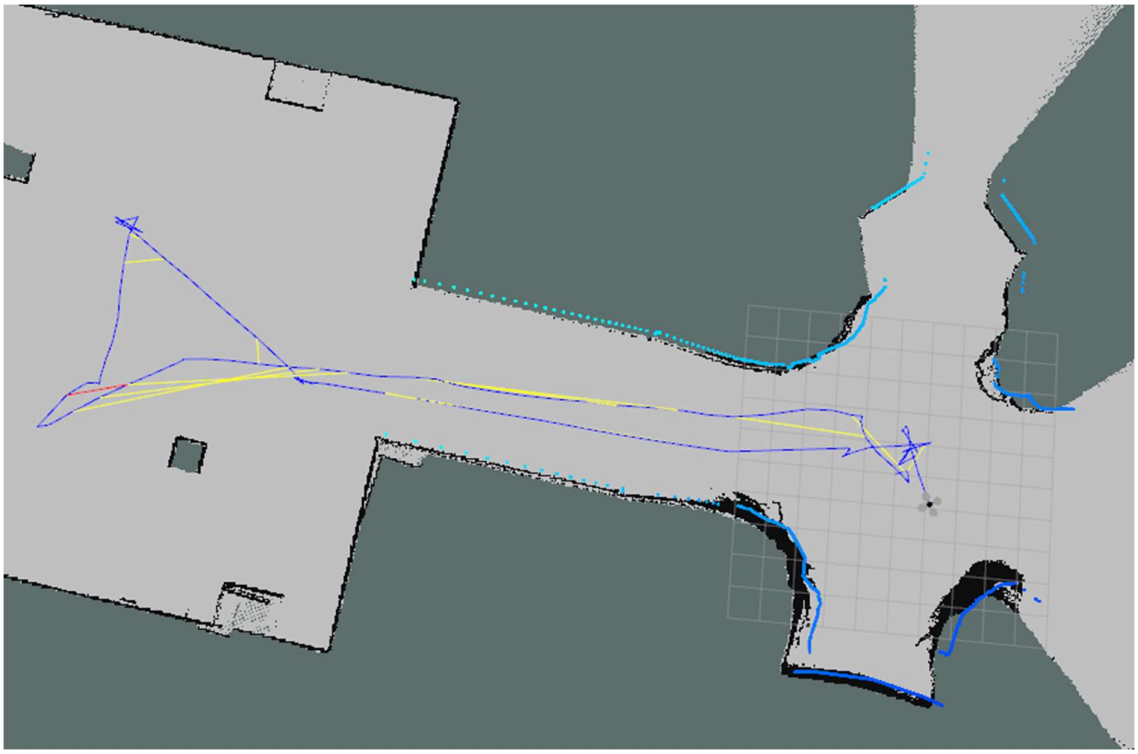


Figure 35. *Exploration fifty-five meters movement Rviz (a)*

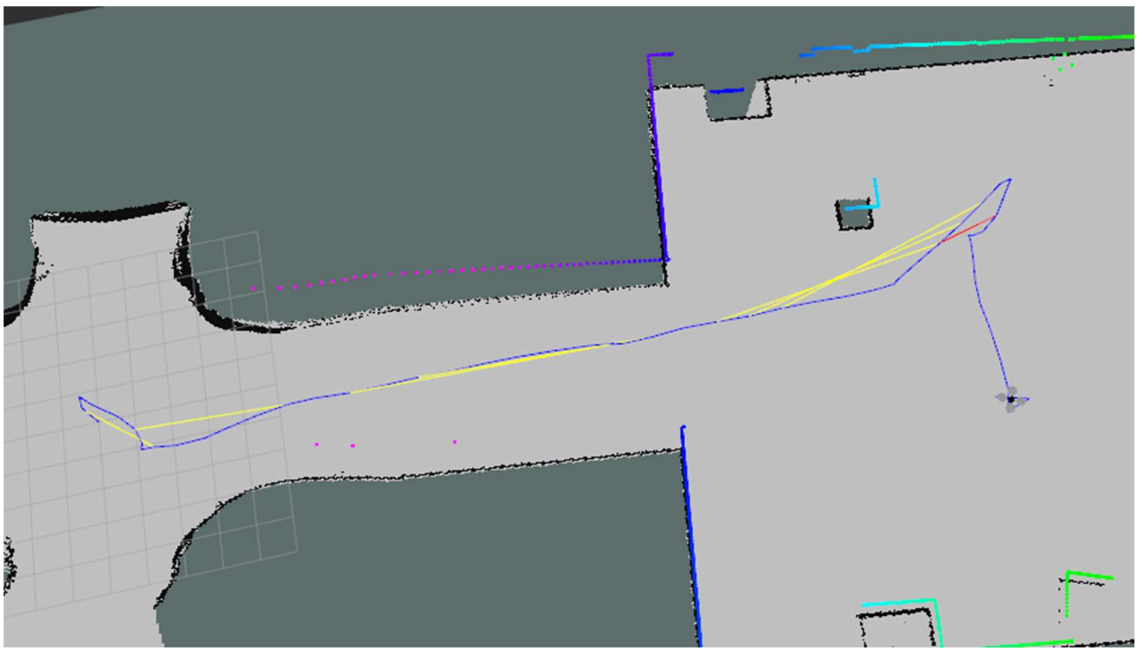


Figure 36. *Exploration fifty-five meters movement Rviz (b)*

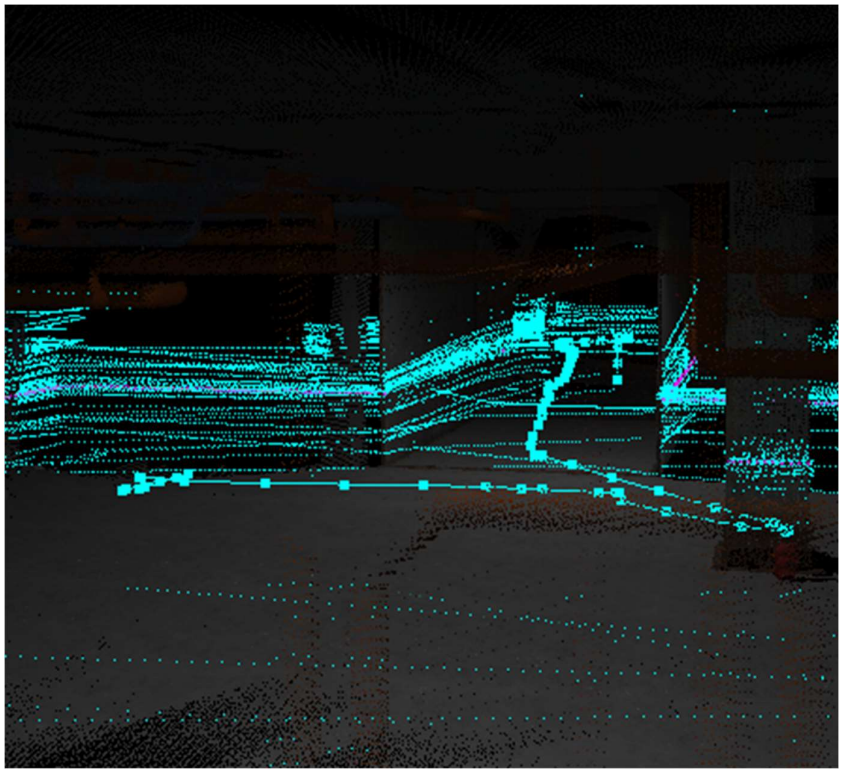


Figure 37. *Exploration fifty-five meters movement RTABMap*

Figure 38. *Exploration seventy-five meters movement Rviz*

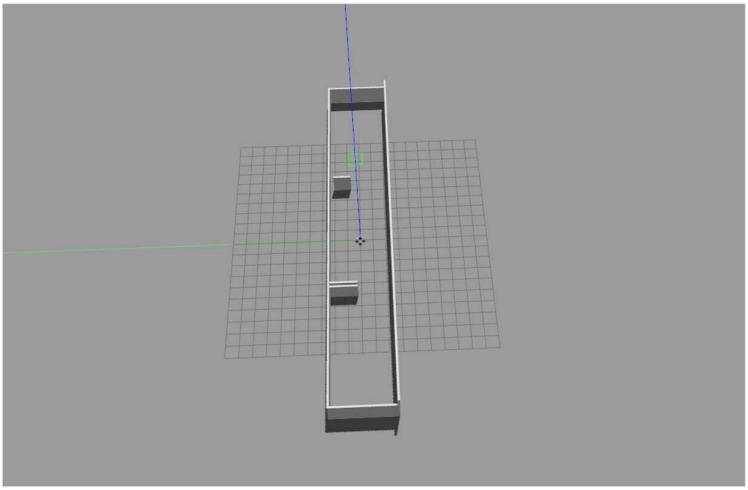


Figure 39. *Small map*

Explored Area 2D	Full Coverage
Explored Area 3D	Full Coverage
Explored Duration	5 minutes
Environment Type	Small map with obstacles
Average Exploration Time	5 minutes
Number of Collisions	0
Attempts Made	20
Slam method	ICP

Explored Area 2D	Odometry failed
Explored Area 3D	Odometry failed
Explored Duration	--
Environment Type	Small map with obstacles
Average Exploration Time	--
Number of Collisions	--
Attempts Made	20
Field of View	120 degrees

Slam method RGB-D	1.0

The UAV's performance in the two exploration trials demonstrates the impact of the chosen SLAM method on localization and mapping. Using ICP (Iterative Closest Point) for SLAM, the UAV achieved full coverage in both 2D and 3D explored areas, completing the task in an average duration of 5 minutes with 0 collisions across 20 attempts. The ICP method excelled due to its reliance on geometric alignment of point clouds, making it highly effective in environments with obstacles that provide distinct depth and structural information.

In contrast, when the RGB-D camera was used with its SLAM method, the UAV encountered significant challenges. The lack of unique features and the featureless nature of some shapes in the environment led to failures in odometry, preventing successful localization and exploration. This outcome is consistent with the limitations of visual-based SLAM systems in environments where sufficient visual landmarks are not present. The 120-degree field of view of the RGB-D camera was insufficient to capture the necessary distinctive features for effective mapping, resulting in complete exploration failure in both 2D and 3D areas.

These observations underscore the importance of selecting a SLAM method suited to the environment. While ICP relies on structural data and point clouds, RGB-D-based SLAM requires environments rich in unique visual features for reliable operation.

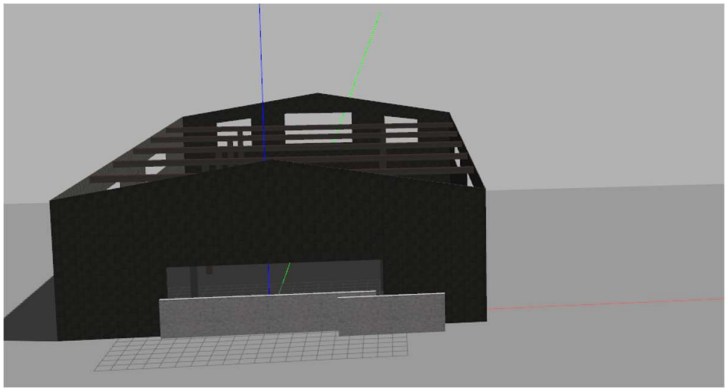


Figure 40. *Warehouse map outside*

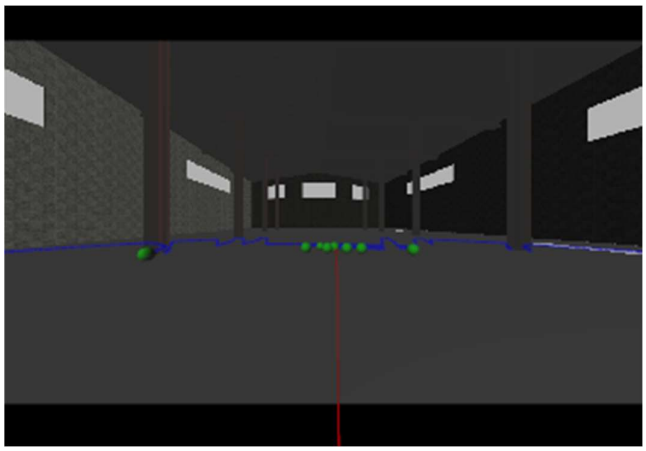


Figure 41. *Warehouse map inside*

Explored Area 2D	Full Coverage
Explored Area 3D	78 %
Explored Duration	6 minutes
Environment Type	Small map with obstacles
Average Exploration Time	5 minutes
Number of Collisions	0
Attempts Made	20
Slam method	ICP

Explored Area 2D	Full Coverage
Explored Area 3D	72 %
Explored Duration	13 minutes
Environment Type	Small map with obstacles
Average Exploration Time	13 minutes
Number of Collisions	1
Attempts Made	20
Slam method	RGB-D

The warehouse exploration was conducted under real-time conditions, with two SLAM methods—ICP and RGB-D—demonstrating contrasting levels of performance. Using ICP, the UAV achieved full 2D coverage and 78% 3D coverage in an average time of 6 minutes, with 0 collisions across 20 attempts. ICP performed efficiently by leveraging structural depth data, allowing the UAV to accurately localize and navigate the small obstacle-filled environment. The slightly incomplete 3D coverage indicates minor challenges in capturing complex vertical features, but overall, the method demonstrated reliability and consistency.

In contrast, the UAV's performance with RGB-D SLAM was less efficient, with full 2D coverage but only 72% 3D coverage. The exploration took 13 minutes on average, with 1 collision recorded across 20 attempts. The system frequently lost odometry due to the warehouse's large empty spaces and lack of distinct visual features, leading to significant positional drift. These odometry failures not only delayed exploration but also impacted the UAV's ability to maintain a stable navigation path, resulting in slower exploration and reduced coverage.

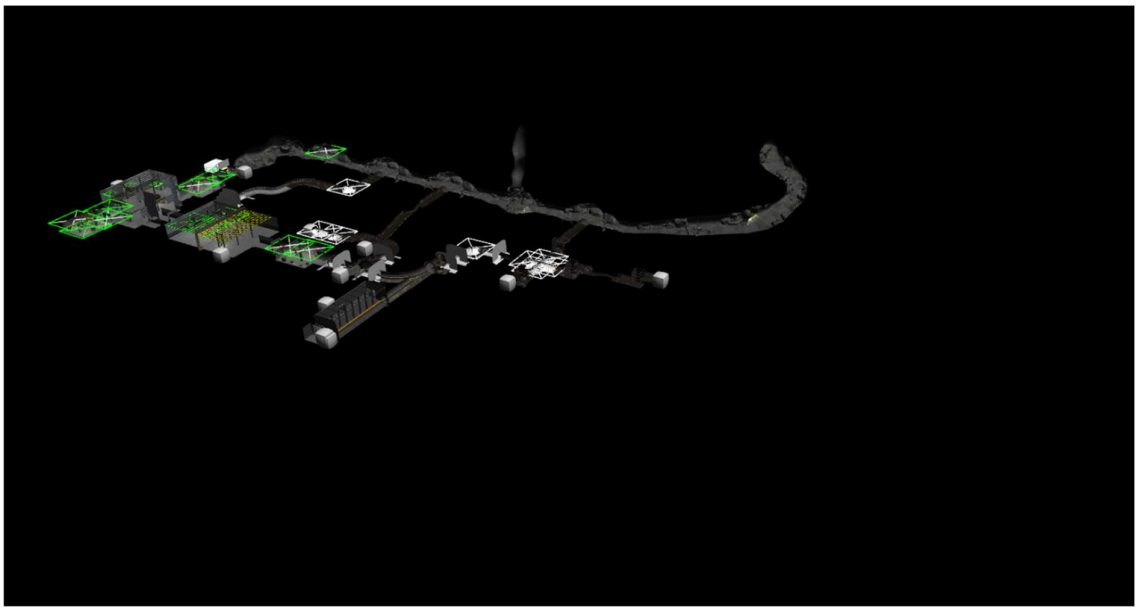


Figure 42. *Cave map*

Explored Area 2D	65 %

Explored Area 3D	50 %
Explored Duration	12 minutes
Environment Type	Huge map with obstacles
Average Exploration Time	12 minutes
Number of Collisions	1
Attempts Made	20
Slam method	ICP

Explored Area 2D	65 %
Explored Area 3D	50 %
Explored Duration	21 minutes
Environment Type	Huge map with obstacles
Average Exploration Time	12 minutes
Number of Collisions	0
Attempts Made	20
Slam method	RGB-D

The exploration in the cave environment demonstrated the UAV's ability to navigate challenging conditions despite limitations in the simulation setup. Using ICP, the UAV achieved 65% 2D coverage and 50% 3D coverage in an average time of 12 minutes, with 1 collision recorded across 20 attempts. While ICP effectively handled obstacle avoidance and frontier selection, its performance was limited by the scale of the map, as the method struggled to maintain consistent localization over larger areas with increased voxel mapping demands.

When using RGB-D SLAM, the UAV also achieved 65% 2D coverage and 50% 3D coverage, though the exploration took 21 minutes on average with 0 collisions recorded. RGB-D SLAM demonstrated better localization accuracy compared to ICP, particularly in the feature-rich sections of the cave. However, the system was similarly affected by the simulation's limitations, as the large number of mapped voxels caused crashes during extended operations. Despite this, the UAV maintained a steady speed, avoided obstacles efficiently, and consistently selected optimal frontiers for exploration.

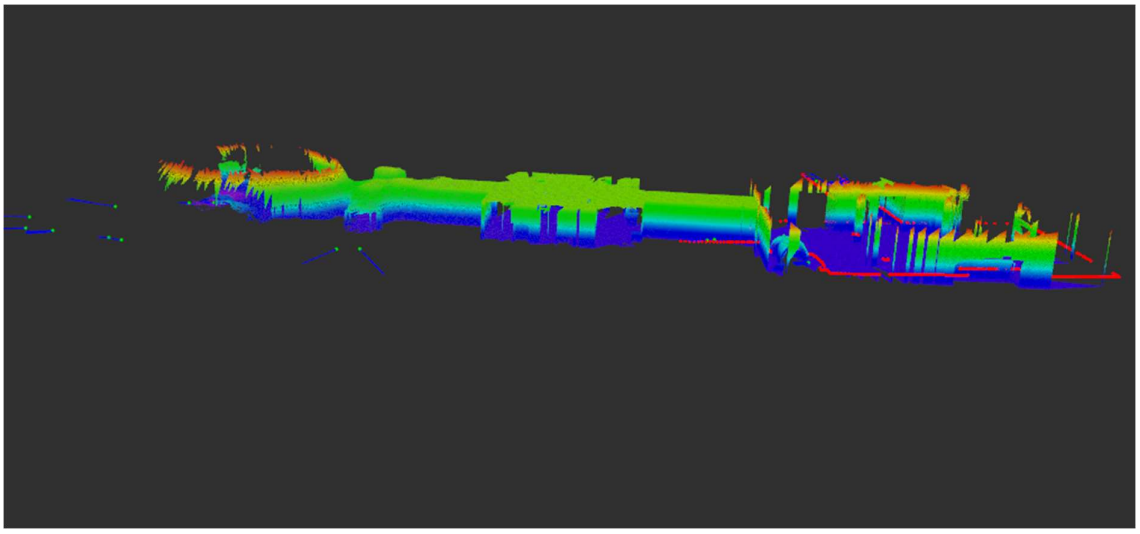


Figure 43. *3D map of huge map (a)*

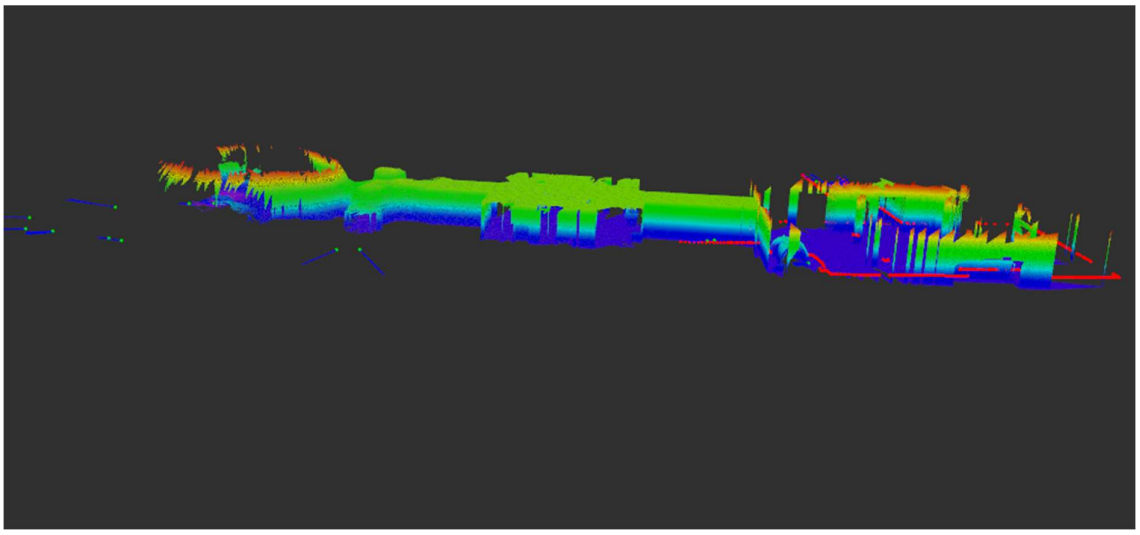


Figure 44. *3D map of huge map (b)*

5.2 TD3 Network

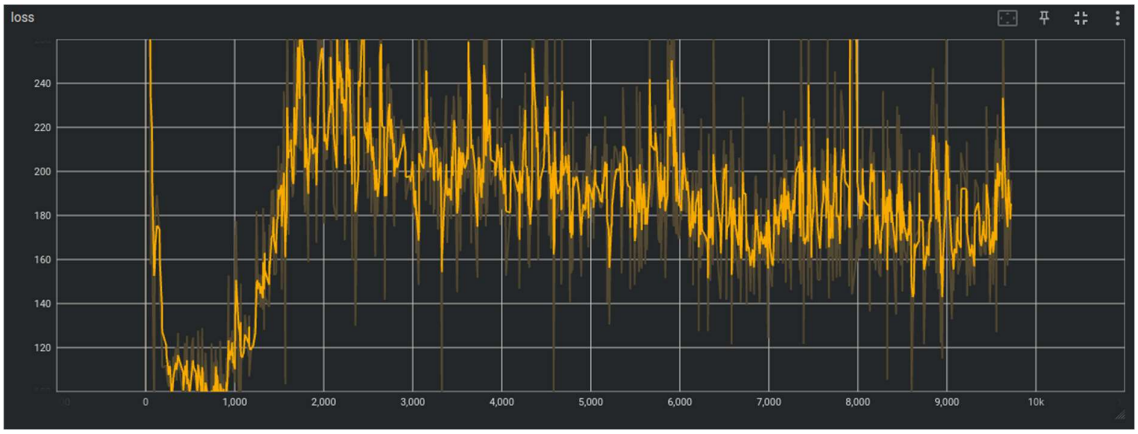


Figure 45. *TD3 loss function graph*

X-axis (Iterations/Steps): This shows the progression of training over time, with the model running for about 10,000 steps.

Y-axis (Loss): This measures how far off the model's predictions are from the true values. Lower loss means better performance.

The graph shows a significant drop in loss in the first 1,000 steps, indicating that the model quickly learned the basic patterns in the data. This initial rapid decline is common in models that effectively grasp early-stage learning and shows that your setup is successfully guiding the model to reduce errors quickly. After the initial drop, the loss stabilizes within a range, fluctuating but staying within a certain band (around 150 to 250). These fluctuations can be expected in more complex models, especially when dealing with real-world data and by introducing noisy patterns or require the model to fine-tune its understanding. The model was trained for 24 hours.

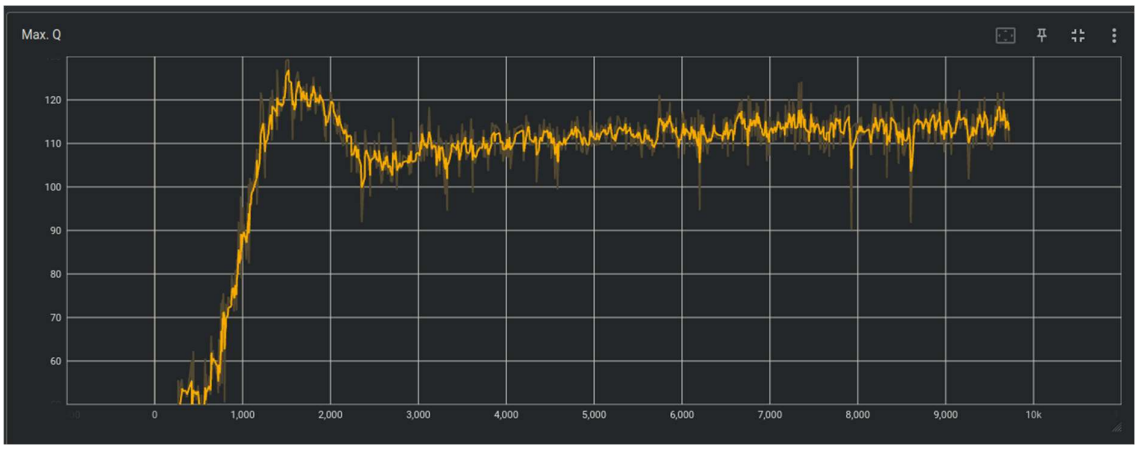


Figure 46. *TD3 Max Q-value graph*

X-axis (Iterations/Steps): This shows the number of training steps, up to around 10,000.
Y-axis (Average Q-value): This reflects the average Q-values that the model is learning. Q-values represent the expected future rewards for a given action in a specific state.

The Q-values decrease rapidly at the start, reaching a low around -40. This initial drop is typical in reinforcement learning, as the model starts with random actions, which often lead to poor rewards initially. After the sharp drop, the Q-values steadily increase, signifying that the model is learning and adjusting its policy to take better actions that lead to higher expected rewards. The curve smooths out after around 7,000 steps, indicating that the model is starting to stabilize and learn a consistent policy with improved average rewards. The Q-values reach a relatively stable range around 10-20. This shows that the model has likely converged to a more optimal policy. Small fluctuations are normal, but the model has generally learned to predict better outcomes from its actions at this stage.

Figure 47. *TD3 average Q-value graph*

X-axis (Iterations/Steps): Represents the number of training steps (up to 10,000).
Y-axis (Max Q-value): Reflects the maximum Q-value observed during the training process at each iteration. Q-values represent the expected future rewards for a given action in a specific state.

The Max Q-value increases sharply from around 60 to 120. This suggests that the model quickly learns better policies, leading to higher expected rewards for certain actions. The steep rise in the early stages reflects that the model is effectively discovering which actions lead to better long-term rewards. After the initial sharp rise, the Max Q-value fluctuates but remains relatively stable between 110 and 120. These small fluctuations are expected as the model continues to learn and fine-tune its policies, exploring different actions and adjusting its Q-values accordingly. This indicates that the model is finding and refining optimal or near-optimal actions but still exploring alternative strategies to maximize rewards. There's no major decline in Max Q-values after stabilization, suggesting that the model is not diverging or experiencing major issues like overfitting. Instead, it maintains high Q-values, indicating consistent performance over time. This graph is a good indicator of a well-performing reinforcement learning model. The steady rise and stabilization of Max Q-values suggest that the model is learning to make decisions that lead to higher long-term rewards, and the process has reached a point of stability after around 1,500 steps.

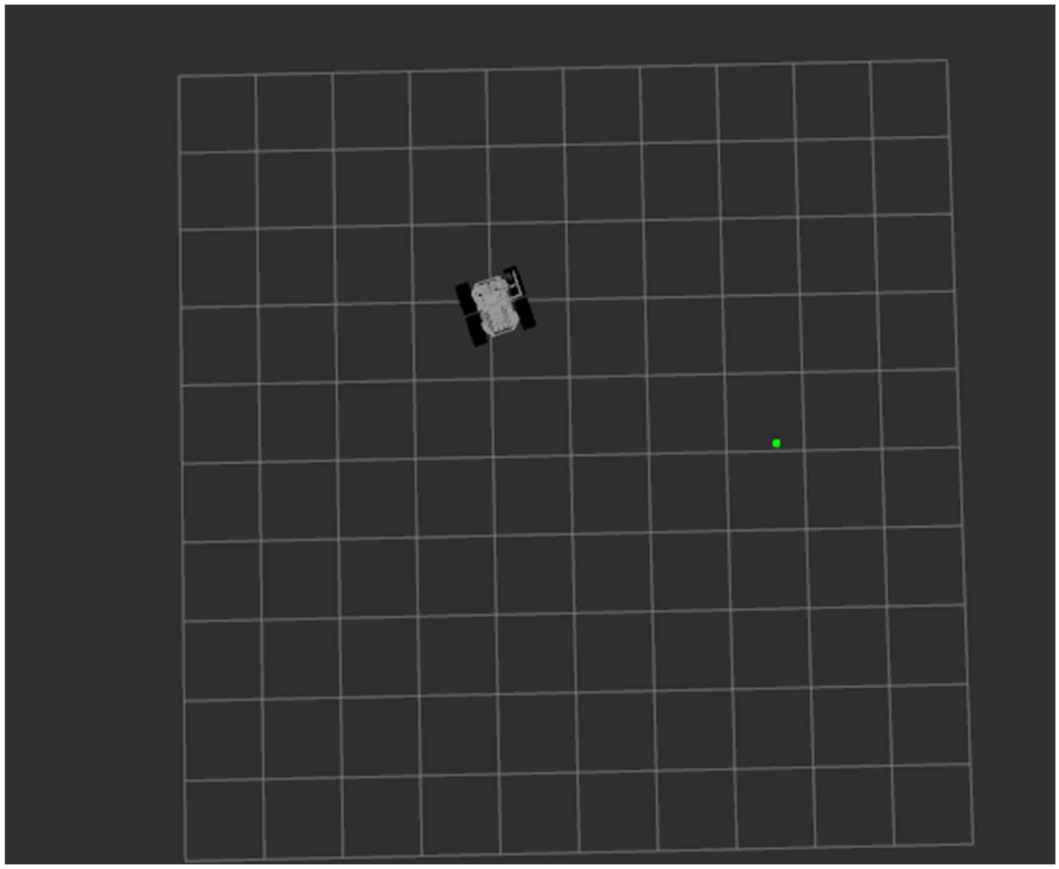


Figure 48. *Validation of TD3 model, robot going to target*

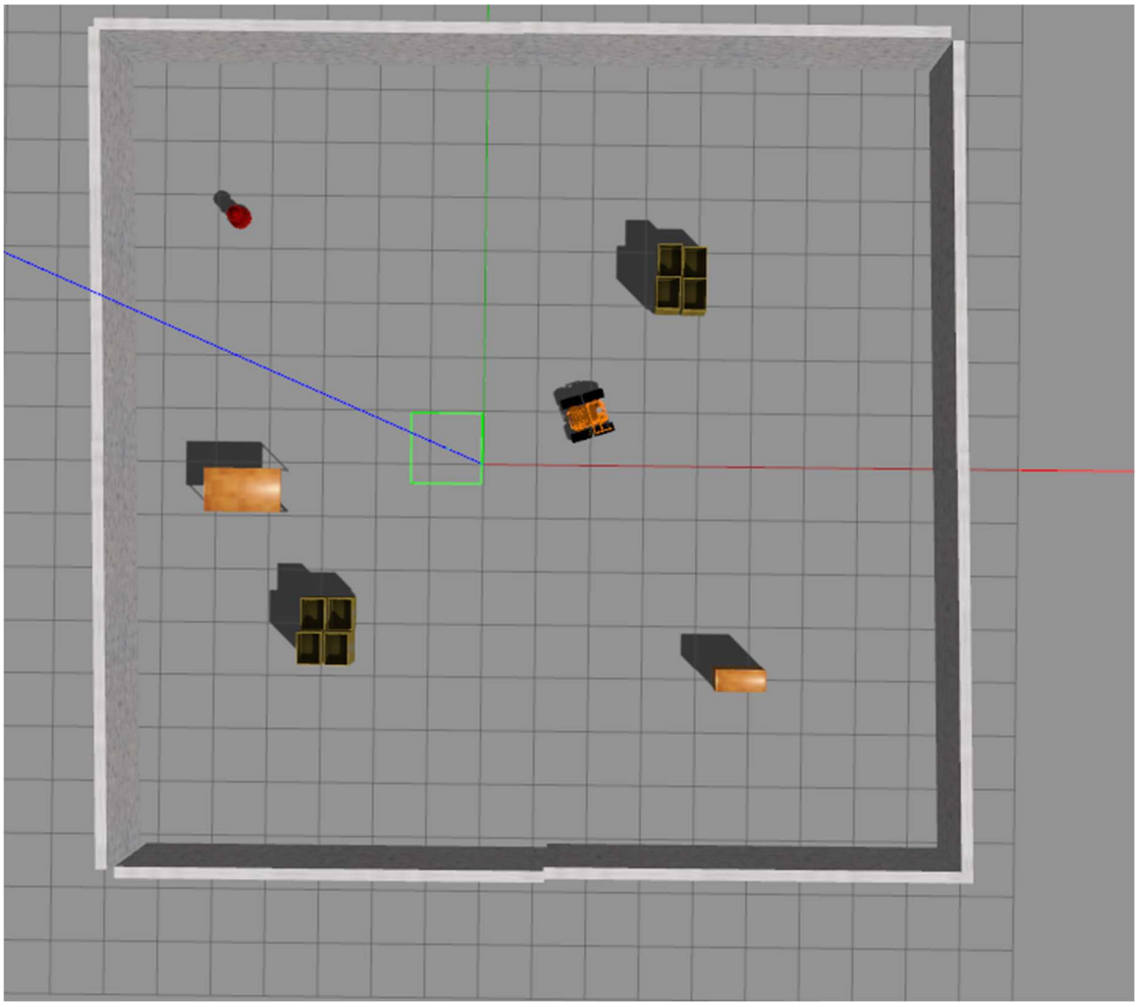


Figure 49. *Map of validation robot movement*

In twenty evaluation cycles the robot has a perfect success rate avoiding obstacles and reaching the target. The model has been trained in a different environment and with a different robot. The model has generalized well, meaning it learned underlying principles of navigation that are not environment-specific or robot-specific. This is a key success factor in machine learning, particularly in reinforcement learning, where transferring knowledge between environments or robots is challenging.

Below are the results from the validation of the DQN model, which was trained using the UAV's onboard camera for object detection. The validation process was designed to test the model's adaptability and generalization by altering the environment. Objects were deliberately moved and placed along the previously generated path the UAV followed to reach its target. This ensured that the UAV was not overfitted to a specific environment or obstacle configuration but could adapt to new layouts.

Collision checks were conducted using LiDAR, which continuously monitored the UAV's surroundings. If the UAV approached an obstacle too closely, it was registered as a collision. The validation setup tested the UAV's ability to navigate dynamically, evaluating its performance in unfamiliar obstacle placements.

During validation, the UAV occasionally collided with obstacles on the first attempt after their positions were changed. This behavior is expected as the model adjusts to new conditions. However, after the initial collision, the UAV consistently recalibrated its strategy and successfully avoided obstacles on subsequent attempts. This indicates that the DQN model effectively learns and adapts during real-time navigation, leveraging its reinforcement learning framework to make better decisions as it gathers more information about the altered environment.

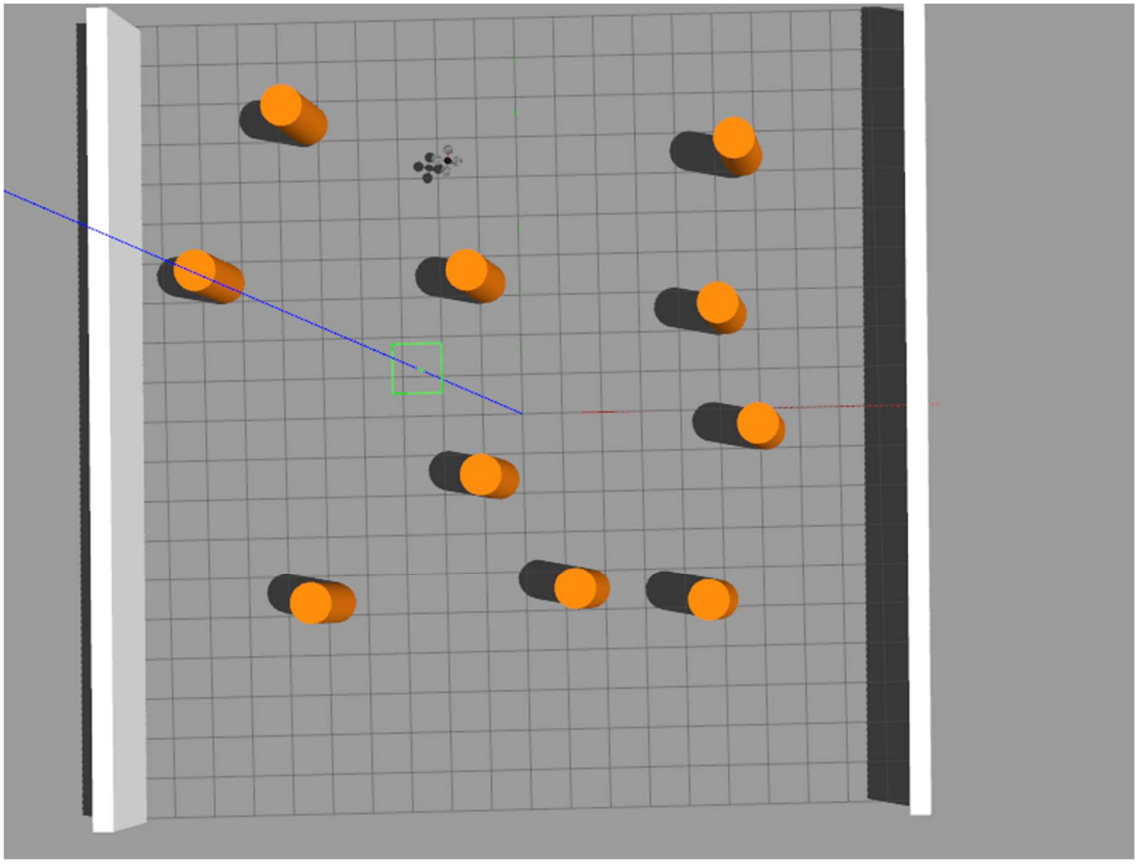


Figure 50. *Validation of UAV Target Navigation Using Deep Q-Network (DQN) obstacle orientation a*

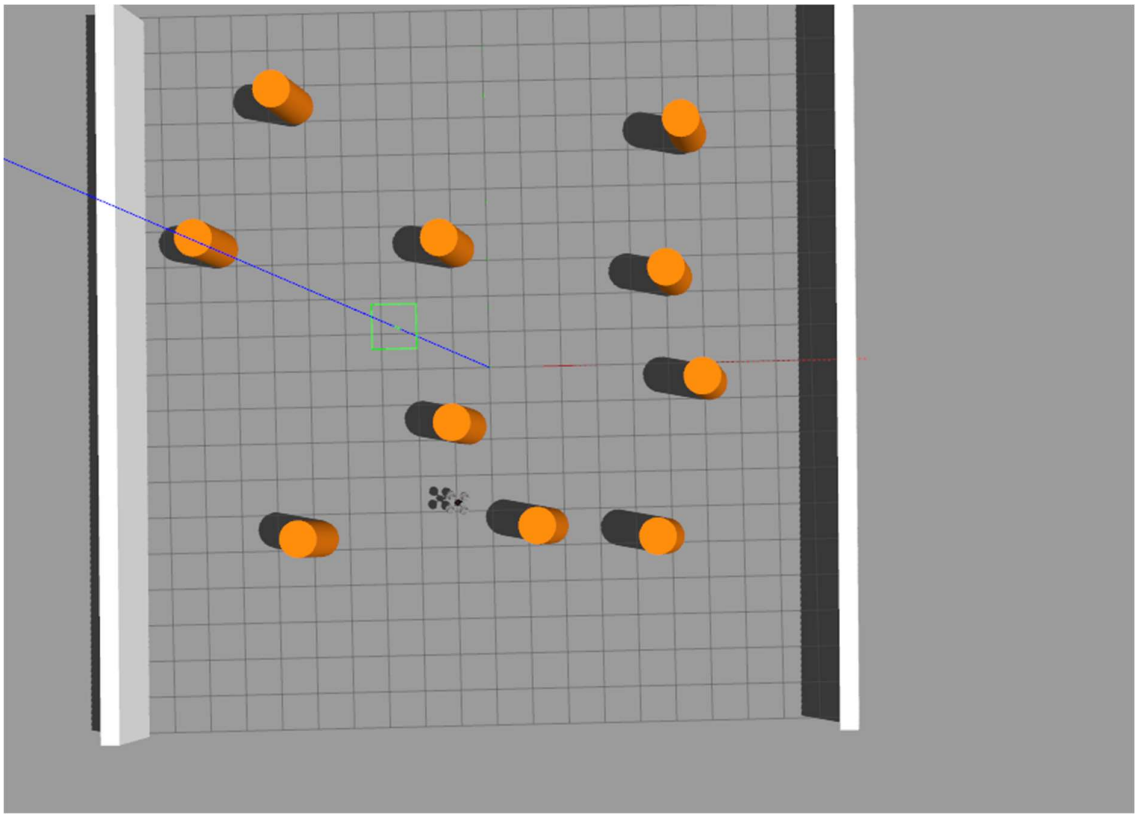


Figure 51. *Validation of UAV Target Navigation Using Deep Q-Network (DQN) obstacle orientation b*

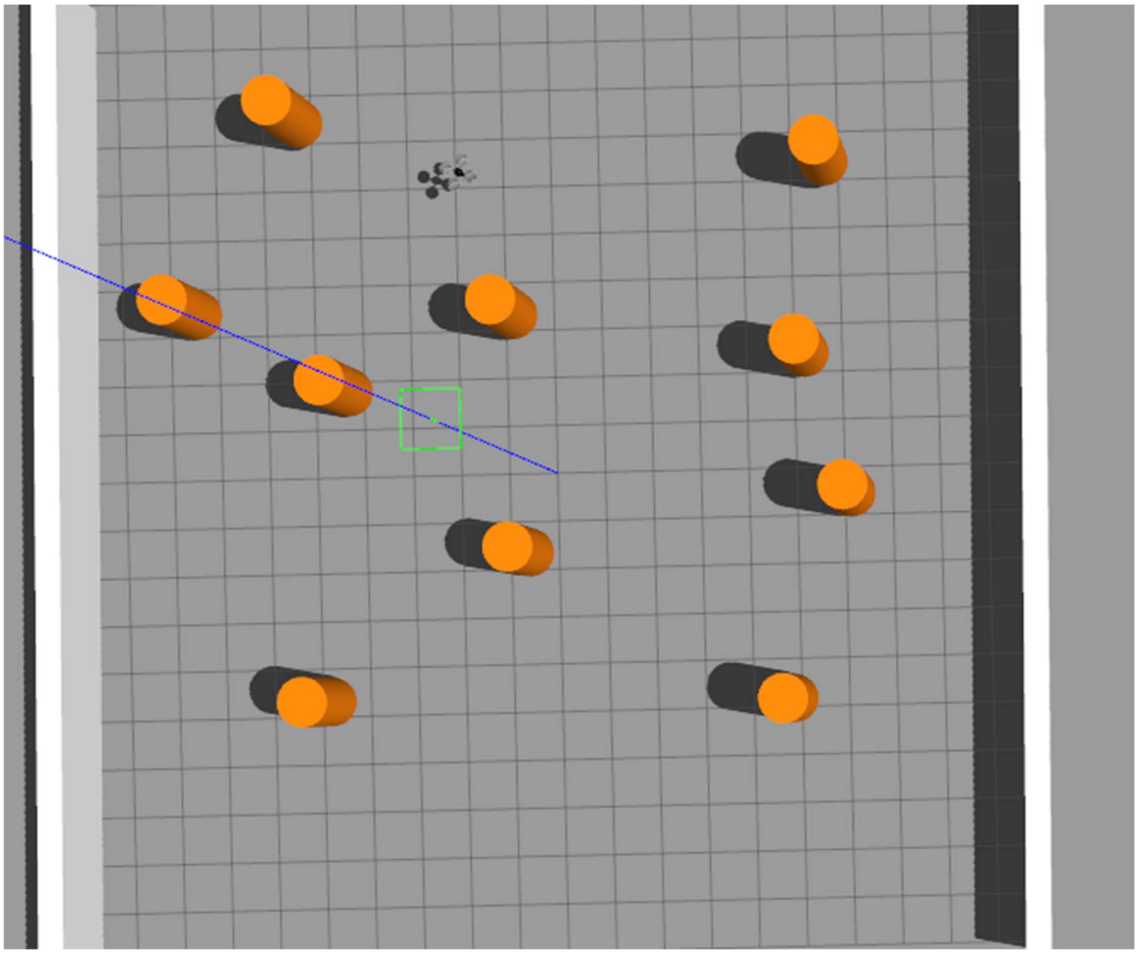


Figure 52. *Validation of UAV Target Navigation Using Deep Q-Network (DQN) obstacle orientation c*

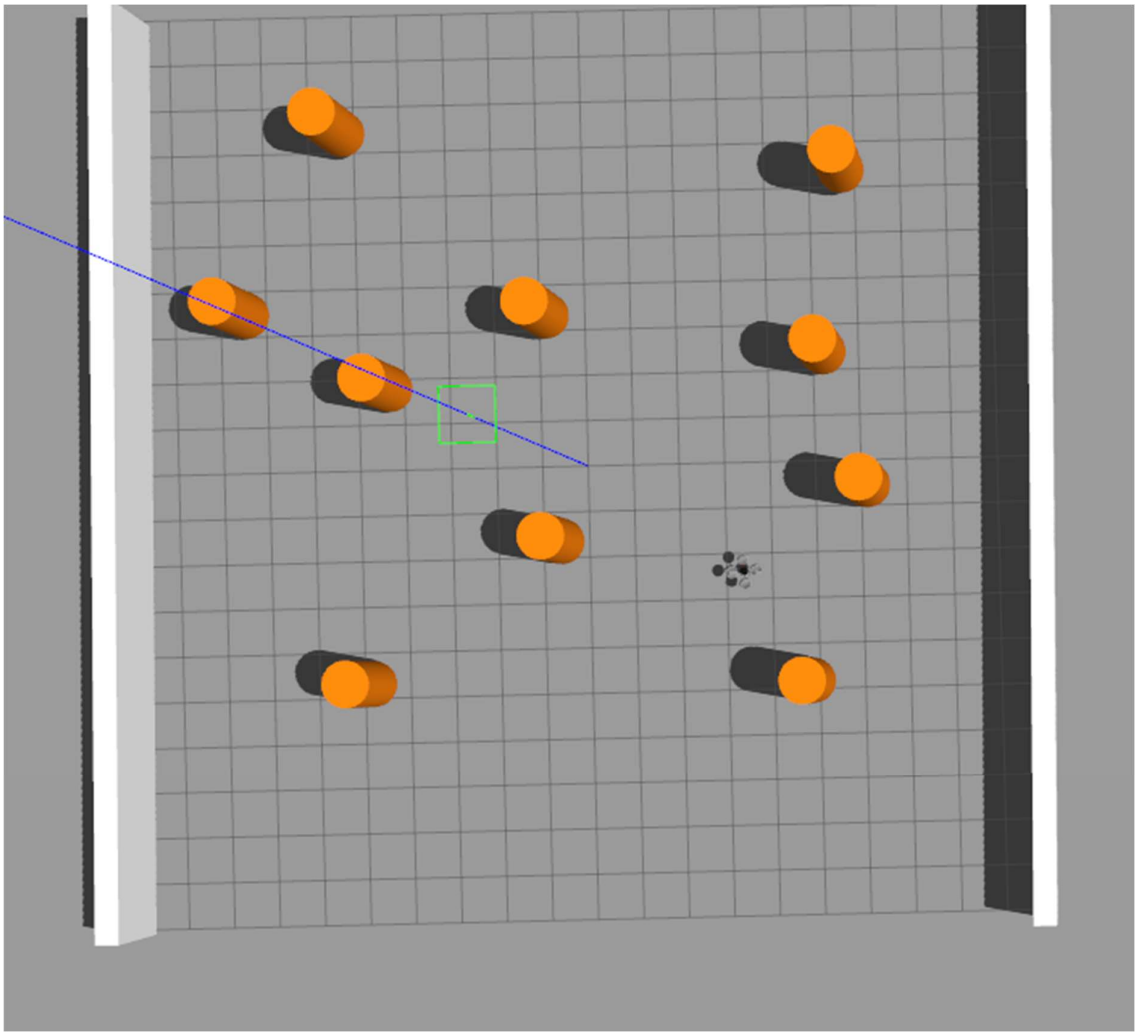


Figure 53. *Validation of UAV Target Navigation Using Deep Q-Network (DQN) obstacle orientation d*

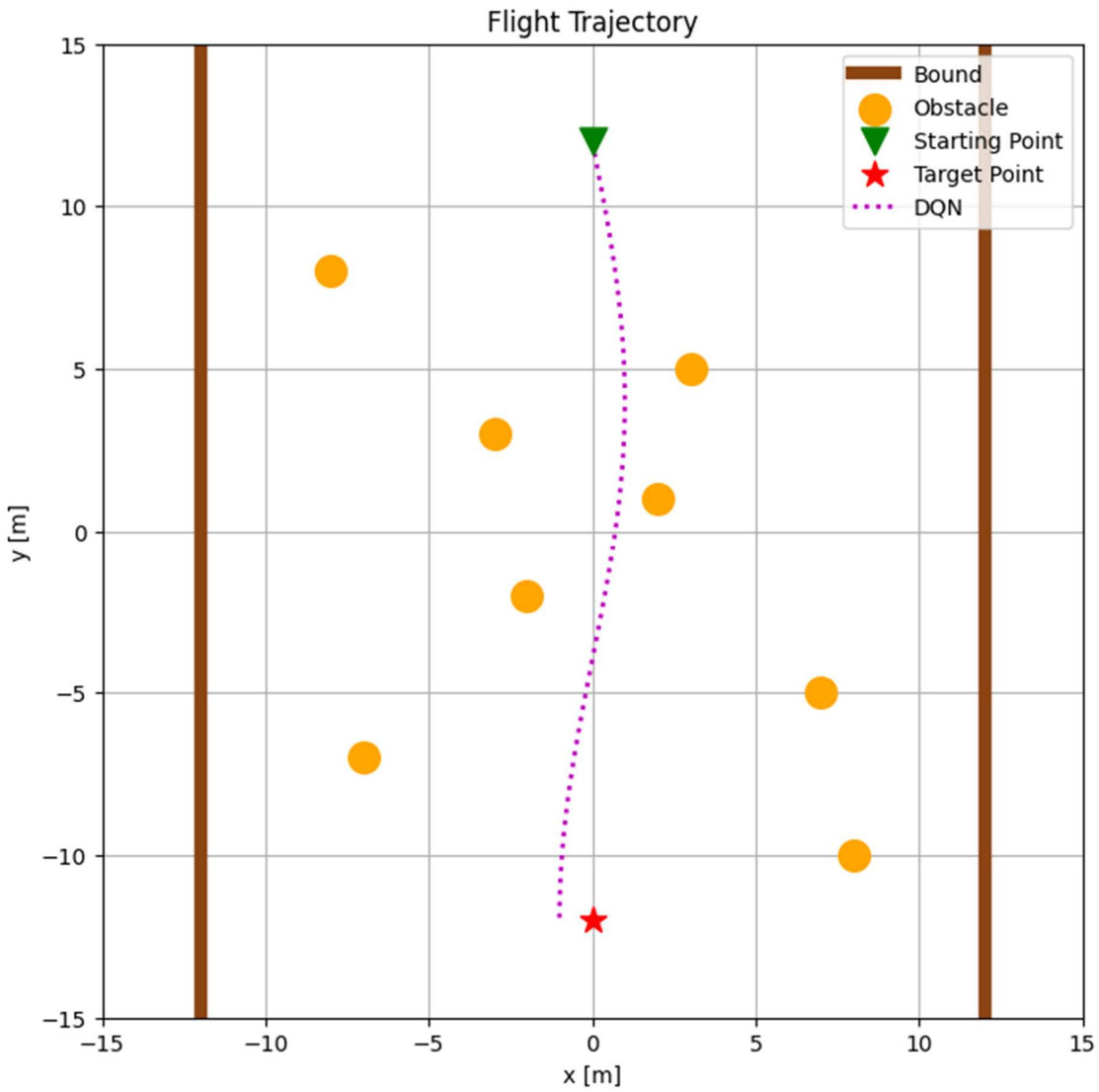


Figure 54. *Movement of UAV*

5.3 Navigation algorithm

In this section the navigation algorithm' s results will be presented. For the navigation algorithm to work correctly, it must always know its current location. Since the distance and the orientation of the target position is calculated based on UAV' s current position. When odometry is lost, the algorithm will wait until the odometry is restored before making any corrections. In this timeframe the UAV may collide with an obstacle because it makes calculations if on the current position or next possible positions an obstacle will be present. In all cases the localization algorithm is RGB-D because we need the height (z) in order to move in a 3D space, since ICP localization does not provide height localization.

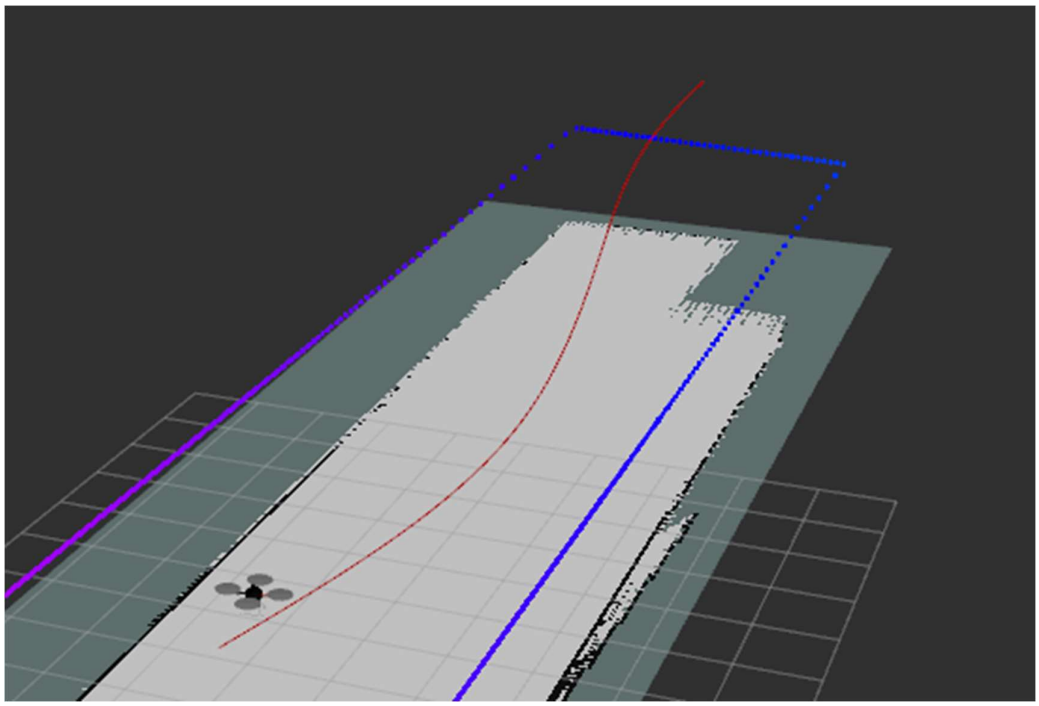


Figure 55. *Informed RRT Star navigation from point A to B*

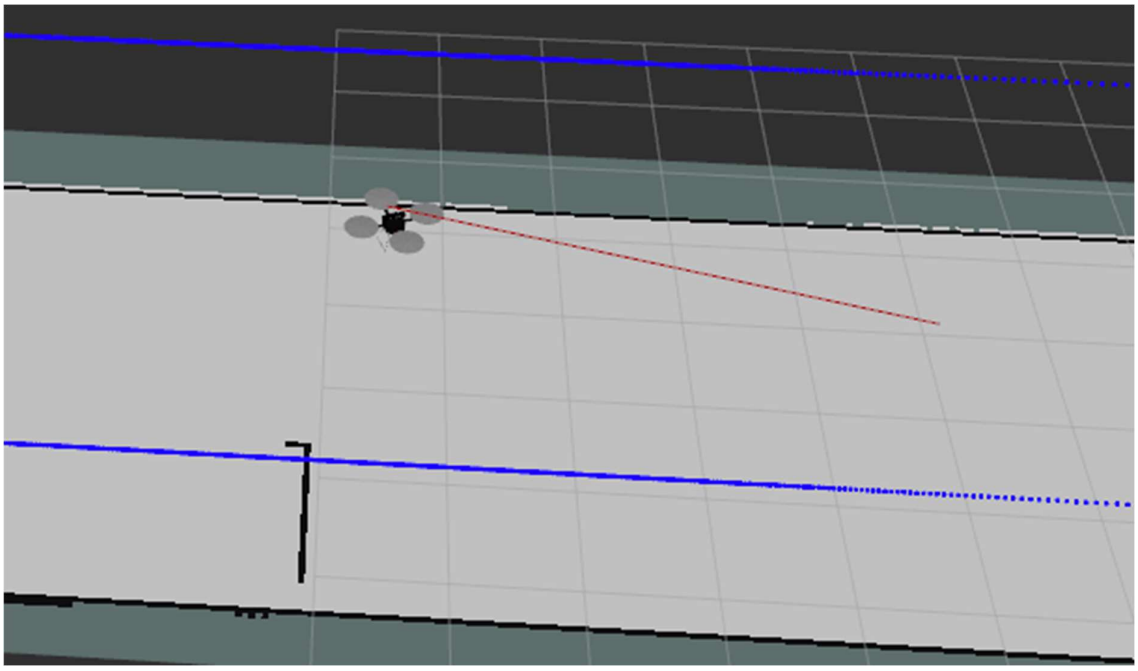


Figure 56. *Informed RRT Star navigation from point C to D*

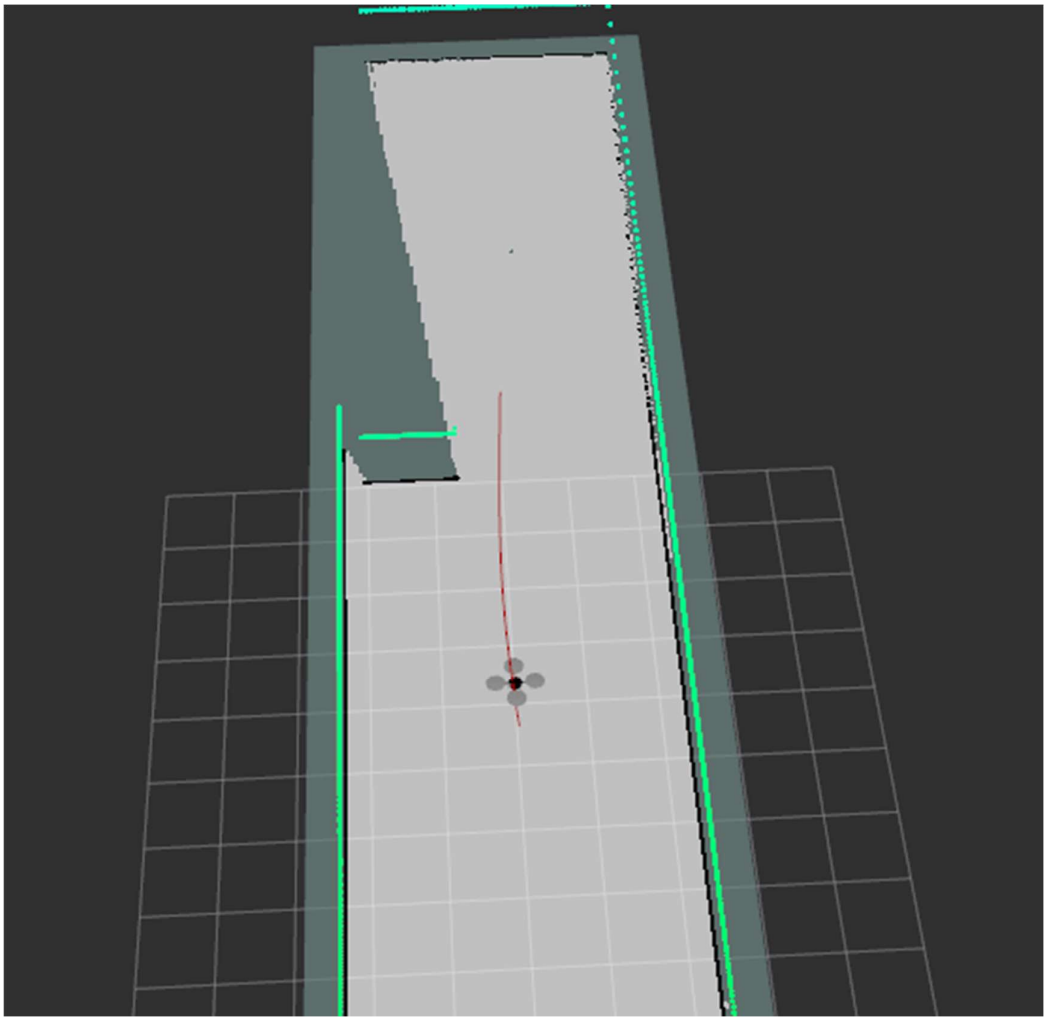


Figure 57. *Informed RRT Star navigation near obstacle*

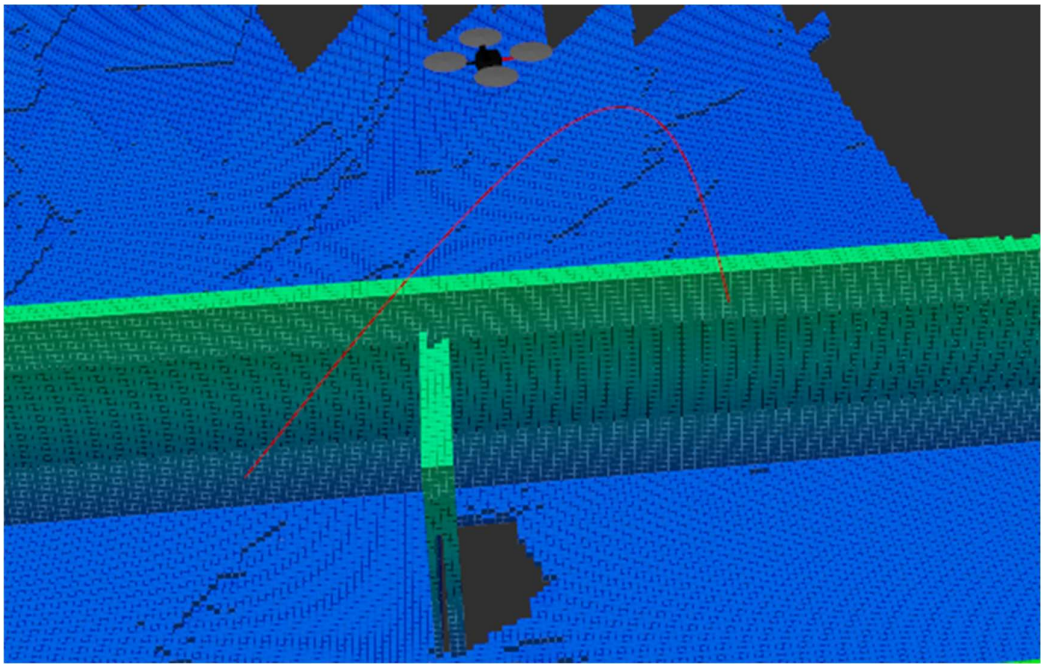


Figure 58. *Informed RRT Star avoiding obstacle*

From the above figure we can see that the navigation algorithm successfully detected the obstacle in its environment and created a trajectory that allowed the UAV to navigate over it. Since the obstacle was already present in the UAV's memory from prior mapping, it efficiently calculated a valid path without requiring additional processing. This highlights the strength of the algorithm in handling pre-mapped obstacles and optimizing the navigation path. However, if the obstacle had appeared during the UAV's movement phase, the system would halt to reassess the environment. The navigation algorithm would then restart, incorporating the newly detected obstacle into its updated map before resuming its trajectory.

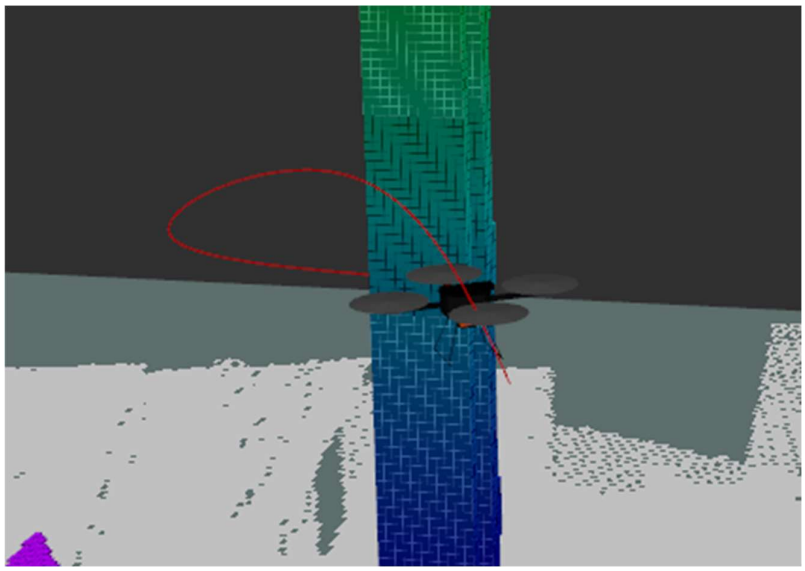


Figure 59. *Trajectory for obstacle avoidance*

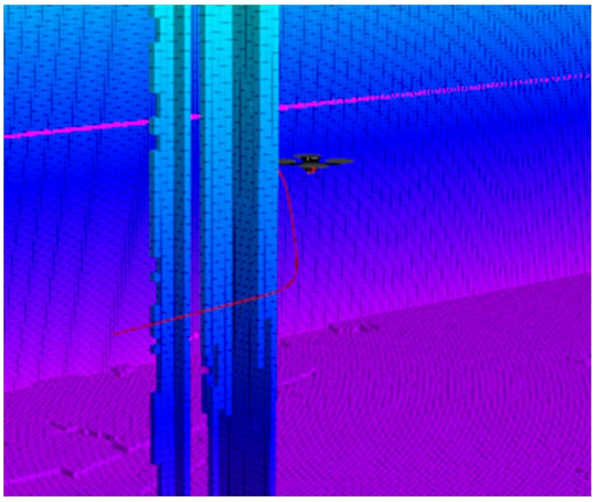


Figure 60. *Collision, odometry lost*

In another scenario, the UAV collided with an obstacle due to a delay in localization odometry. The delayed odometry data caused the UAV to deviate from its intended path, ultimately leading to a collision. This incident underscores the critical importance of real-time odometry updates for maintaining accurate positioning and ensuring collision-free navigation. Delays in processing or sensor data inconsistencies can result in the UAV drifting off-course, especially in dynamic or obstacle-dense environments.

The figures provided visually demonstrate these scenarios, showing how the UAV plans and navigates its path while responding to environmental changes. In one image, the trajectory over

the obstacle showcases successful planning, while another highlights the deviation and collision caused by localization errors. These cases emphasize the need for robust odometry and adaptive navigation to ensure reliable performance in real-world applications.

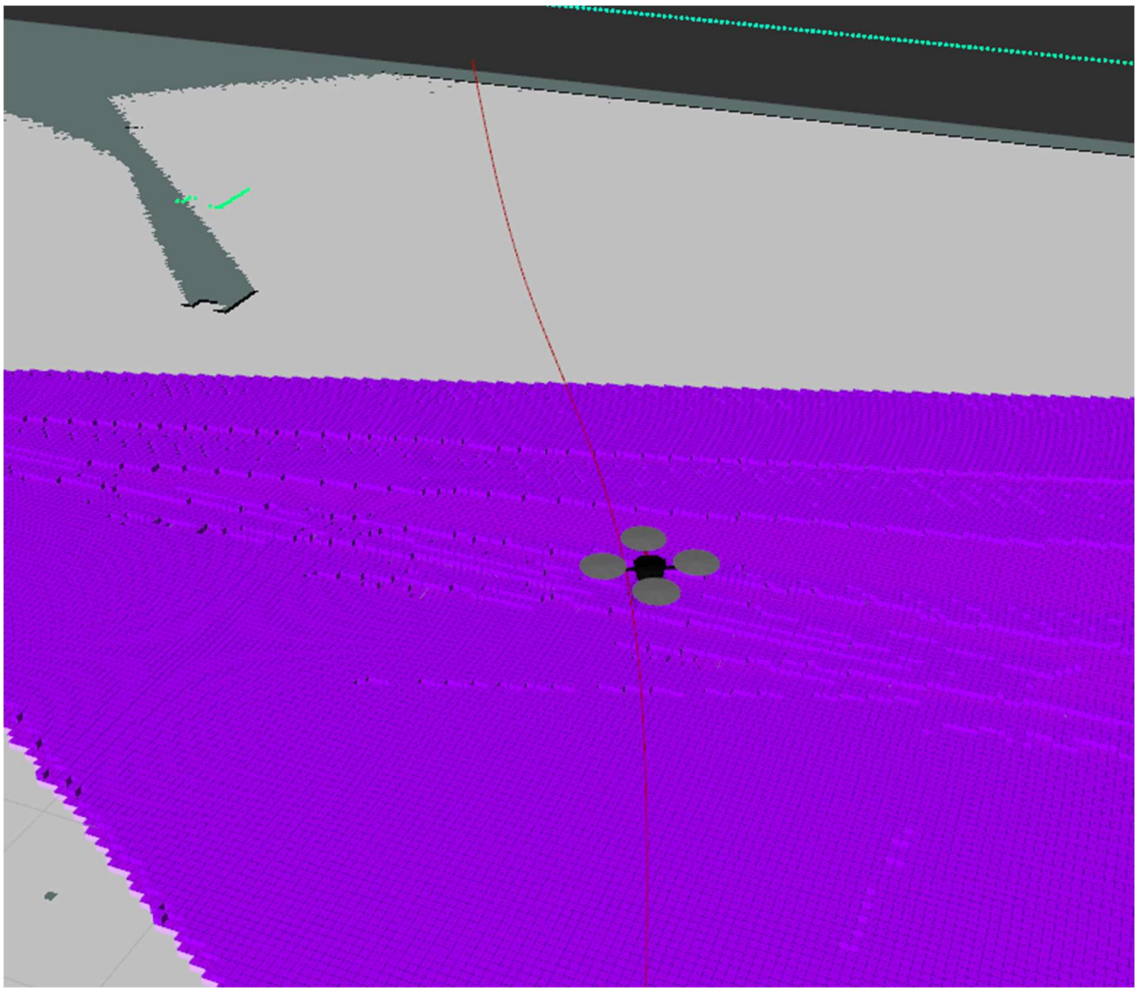


Figure 61. *Trajectory upwards movement*

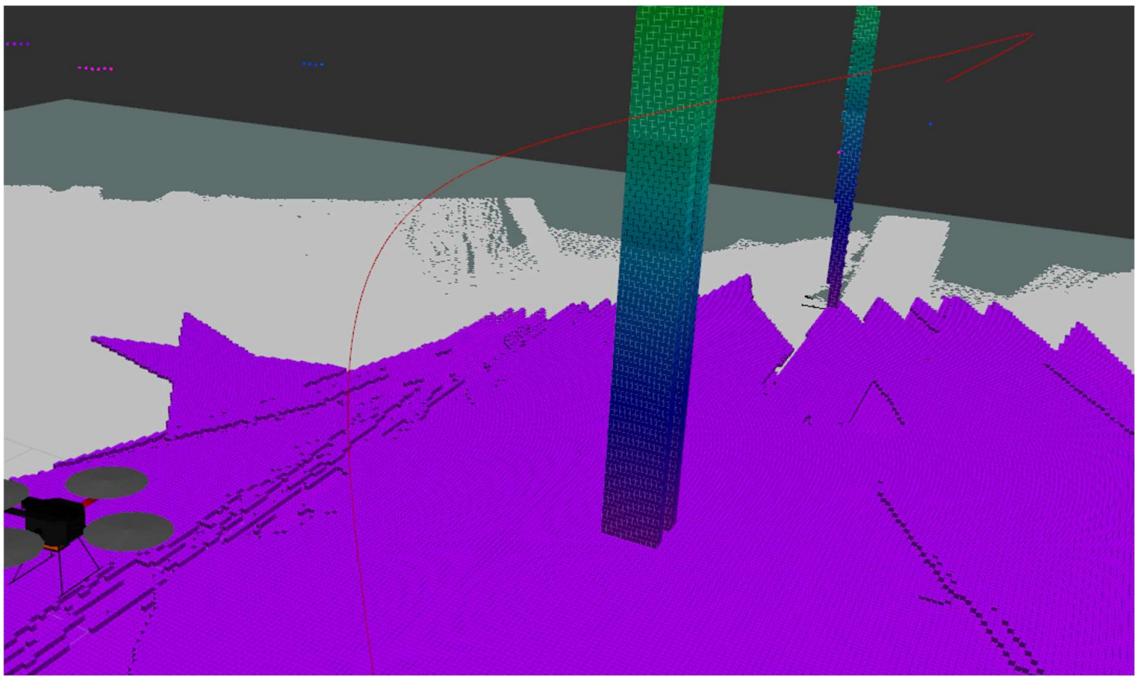


Figure 62. *Trajectory spiral movement*

6 Conclusions and Future Work

6.1 Conclusion

In conclusion, this thesis successfully implemented and validated a comprehensive framework for autonomous navigation and decision-making in UAVs and land rovers. SLAM (Simultaneous Localization and Mapping) was employed to enable real-time mapping and obstacle avoidance, with techniques such as Iterative Closest Point (ICP) and RGB-D SLAM demonstrating their respective strengths in different environmental conditions. The UAV was able to create accurate 2D and 3D maps, navigate efficiently, and avoid collisions, showcasing the robustness of the SLAM integration for real-time operations.

Reinforcement learning, specifically through the implementation of Deep Q-Networks (DQN), optimized the UAV's path planning and decision-making processes. The model demonstrated adaptability to dynamic environments by successfully recalculating navigation paths when obstacles were added or moved. Validation in simulated environments proved critical in assessing the system's performance, revealing strengths in collision avoidance, trajectory optimization, and adaptability to varying obstacle configurations. Furthermore, object recognition and classification enhanced situational awareness, enabling the UAV to identify and respond to objects effectively during navigation.

Autonomous exploration was achieved with systematic and efficient area coverage in both 2D and 3D using SLAM techniques, even in complex environments with featureless regions or dynamic changes. The UAV relied on mapping algorithms, such as ICP and RGB-D SLAM, to construct accurate maps and navigate effectively. The UAV demonstrated improvements in exploration and obstacle avoidance over repeated trials as it incorporated real-time sensor data into its navigation process.

Additionally, the integration of the Twin Delayed Deep Deterministic Policy Gradient (TD3) network was specifically applied to the land rover to optimize its decision-making and path-planning processes, showcasing the versatility of reinforcement learning for ground-based autonomous systems. The UAV's navigation and mapping were entirely based on SLAM and heuristic exploration strategies, ensuring adaptability and efficiency without relying on

reinforcement learning algorithms. This approach highlights the effectiveness of SLAM in achieving real-time navigation and mapping in dynamic, obstacle-dense environments.

6.2 Exploration Improvements

The thesis implements a 3D exploration strategy where the UAV selects the free voxel with the highest gain to maximize the explored space efficiently. The current approach focuses on identifying individual voxels that expand the mapped area, providing a simple but effective mechanism for exploration. However, this method could be further enhanced by selecting multiple voxels to create a more optimized trajectory, smoothing the path, and following it in a continuous manner. Such an approach would minimize abrupt movements, improve efficiency, and enable more seamless exploration. A potential improvement would involve integrating advanced exploration strategies, such as the FUEL (Frontier-Based UAV Exploration in 3D Environments)[11] algorithm, which offers a structured and robust solution for systematic exploration. Incorporating FUEL into this project could significantly improve exploration performance, particularly in complex or large-scale environments.

6.3 Swarm Tactics

In this thesis, the exploration process is demonstrated using a single UAV. However, a more advanced system could employ multiple UAVs operating collaboratively to achieve faster and more efficient area coverage. A central node could be implemented to coordinate the efforts of multiple autonomous UAVs, enabling them to share information about the environment, divide tasks, and reduce exploration time significantly. Such swarm tactics could be extended to include heterogeneous systems, such as a UAV working in tandem with a land rover. For instance, the land rover could handle ground-level mapping while the UAV focuses on aerial exploration, providing complementary perspectives. The integration of a multi-agent system, inspired by projects like GBPlanner, could add significant scalability and versatility to the exploration framework. Project GBplanner [15][16] is an exceptional project, focusing on exploration both for UAV's and land vehicles.

6.4 Obstacle list refinement

For 3D navigation, obstacles are currently managed by inserting them into a Python set, allowing the UAV to check for potential collisions while reaching a waypoint. While this

approach ensures safety, it introduces computational inefficiencies, particularly when dealing with a large number of obstacles or high-resolution voxel grids. The computational complexity scales with O(n×x)O(n×x), where nn is the number of obstacles and xx is the number of waypoints. This could be optimized by implementing a more efficient data structure, such as a spatial hash map or a k-d tree, which would allow for faster querying of obstacles in the UAV's path. Such optimizations would significantly reduce computation time and enable smoother navigation in dense environments.

6.5 Sensor fusion

The current implementation uses visual odometry for UAV localization, which relies on detecting and tracking visual features to estimate movement. While effective in feature-rich environments, visual odometry often fails in areas with few recognizable features, such as plain walls or open spaces. To address these limitations, the project incorporates sensor fusion using extended Kalman filters (EKF) to integrate data from multiple sources. A further enhancement could involve adding an additional odometry source, such as LiDAR-based odometry or inertial navigation systems (INS). By fusing data from visual odometry, LiDAR, and INS, the system could achieve higher accuracy and robustness, particularly in challenging environments. This multi-source approach would ensure reliable localization even in featureless or dynamic conditions, enhancing the overall performance and reliability of the UAV navigation system.

Bibliography

[1] Alam, Mansoor. "Localization in Robotics for Mobile Robots." *Medium*. Accessed [10/11/2024].https://medium.com/@mansooralam129047/localization-in-robotics-for-mobile-robots-ec3ad31f99d4.

[2] Z. Zhang and J. Wang, "Exploring Large Language Models for Code Generation in Robotics and Automation," arXiv, Feb. 2023. [Online]. Available: https://arxiv.org/abs/2302.07433. [Accessed: Nov. 10, 2024].

[3] S. K. L. Lal and M. Li, "Psychophysiological and performance effects of continuous monitoring with Google Glass in a simulated military environment," 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Orlando, FL, USA, 2016, pp. 4411-4414. doi: 10.1109/EMBC.2016.7784298.

[4] K. Qian, R. Mao, L. Tao, and X. Wang, "Enhancing collaborative filtering by user interest expansion via personalized ranking," 2010 IEEE International Conference on Data Mining Workshops, Sydney, NSW, Australia, 2010, pp. 478-484. doi: 10.1109/ICDMW.2010.47.

[5] A. Anaby-Tavor, B. Carmeli, E. Goldbraich, N. Kantor, S. Kour, A. Marmor, S. Mozes, M. Shlomov, R. Tepper, and N. Zwerdling, "Do Not Have Enough Data? Deep Learning to the Rescue!" arXiv preprint arXiv:2105.11344, 2021. [Online]. Available: https://arxiv.org/abs/2105.11344.

[6] T. Kohonen, "The self-organizing map," Proceedings of the IEEE, vol. 78, no. 9, pp. 1464-1480, Sep. 1990. doi: 10.1109/5.58325.

[7] Beul et al. (2018). "A LiDAR-based Approach for Autonomous Navigation in Indoor Environments." IEEE Robotics and Automation Letters.

[8] Ok, Greene, and Roy (2018). "RGB-D SLAM in Dynamic Environments Using Multilevel Semantic Mapping." Journal of Intelligent & Robotic Systems.

[9] Y. Li, H. Zhao, and Y. Shi, "A Survey on Deep Learning-Based Face Recognition," IEEE Access, vol. 9, pp. 99112-99142, 2021. doi: 10.1109/ACCESS.2021.3094823.

[10] S. K. Card, G. G. Robertson, and J. D. Mackinlay, "The information visualizer, an information workspace," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91), New Orleans, LA, USA, 1991, pp. 181-186. doi: https://doi.org/10.1145/280765.280773.

[11] Zhou, B., Zhang, Y., Chen, X., & Shen, S. (2021). FUEL: Fast UAV Exploration Using Incremental Frontier Structure and Hierarchical Planning. IEEE Robotics and Automation Letters, 6(2), 779-786.

[12] Bircher, A., Kamel, M. S., Alexis, K., Burri, M., Oettershagen, P., Omari, S., Mantel, T., & Siegwart, R. (2015). “Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots.” Autonomous Robots, 40. https://doi.org/10.1007/s10514-015-9517-4

[13] M. Quigley, B. Gerkey, et al., "ROS: An Open-Source Robot Operating System," in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan, 2009, pp. 1-6, doi: 10.1109/ROBOT.2009.5152481

[14] Labbé, M., & Michaud, F. (2024). “RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation.” Retrieved from https://arxiv.org/html/2403.06341v1#S2

[15] Dang, T., Tranzatto, M., Khattak, S., Mascarich, F., Alexis, K., & Hutter, M. (2020). Graph-based subterranean exploration path planning using aerial and legged robots. Journal of Field Robotics, 37(8), 1363-1388. Wiley Online Library.

[16] Kulkarni, M., Dharmadhikari, M., Tranzatto, M., Zimmermann, S., Reijgwart, V., De Petris, P., Nguyen, H., Khedekar, N., Papachristos, C., Ott, L., Siegwart, R., Hutter, M., & Alexis, K. (2022). Autonomous Teamed Exploration of Subterranean Environments using Legged and Aerial Robots. 2022 International Conference on Robotics and Automation (ICRA), 3306-3313. https://doi.org/10.1109/ICRA46639.2022.9812401

[17] Perez-Grau et al. (2017). "Multi-modal Mapping and Localization of Unmanned Aerial Robots Based on Ultra-Wideband and RGB-D Sensing." Robotics and Autonomous Systems.

[18] Bircher, A., Kamel, M., Alexis, K., Oleynikova, H., & Siegwart, R. (2020). "Receding Horizon 'Next-Best-View' Planner for 3D Exploration." arXiv preprint, arXiv:2010.11561 [cs.RO].

[19] Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., von Stryk, O. (2011). Hector SLAM for Mapping and Navigation with Autonomously Navigating Mobile Robots. 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems.

[20] R. Cimurs, I. H. Suh, and J. H. Lee, “Goal-Driven Autonomous Exploration Through Deep Reinforcement Learning,” IEEE Robotics and Automation Letters, vol. 7, no. 2, pp. 730-737, 2022, doi: 10.1109/LRA.2021.3133591.

[21] Galceran, E., & Carreras, M. (2013). “A survey on coverage path planning for robotics.” Robotics and Autonomous Systems, 61, 1258–1276.https://doi.org/10.1016/j.robot.2013.09.004

[22] Nam, L. H., Huang, L., Li, X. J., & Xu, J. F. (2016). “An approach for coverage path planning for UAVs.” In 2016 IEEE 14th International Workshop on Advanced Motion Control (AMC) (pp. 411–416). https://doi.org/10.1109/AMC.2016.7496404

[23] Yang, S. X., & Luo, C. (2004). “A neural network approach to complete coverage path planning.” IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 34(1), 718–724. https://doi.org/10.1109/TSMCB.2003.817090

[24] Tsiakas, K., Tsardoulias, E., & Symeonidis, A. L. (2024). “Autonomous Full 3D Coverage Using an Aerial Vehicle, Performing Localization, Path Planning, and Navigation Towards Indoors Inventorying for the Logistics Domain.” Robotics, 13(6), 83.

Installation Manual

Requirements

Ubuntu 20.04.*

Ros noetic

Installation procedure

Install ros noetic using the link http://wiki.ros.org/noetic/Installation/Ubuntu.

Download the thesis into your home folder

Download ompl-1.6.0-cp38-cp38-manylinux_2_28_x86_64.whl from https://github.com/ompl/ompl/releases, Development build -> Assets

Run setup.bash

Source the thesis by executing "source ~/thesis/devel/setup.bash"

Add the following lines in ~/.bashrc ->

1. export LD_LIBRARY_PATH=/home/.local/lib:$LD_LIBRARY_PATH
2. export LD_LIBRARY_PATH=/home/.local/lib/python3.8/site-packages:$LD_LIBRARY_PATH

Change all appearances of "home/dimitris" if present in /home/ubuntu/thesis/src/hector_quadrotor_noetic/hector_gazebo/hector_gazebo_worlds/worlds/TD6

Demos

All demos can be found in thesis/src/demos and can be executed by using the command "roslaunch demos <example.launch> use_icp:=false". If use_icp is set to true, then the SLAM method will be set to icp.

To initiate exploration, execute "roslaunch explore_custom explore.launch".

To execute a 2D path, in rviz, set a 2D goal.

To initiate custom navigation algorithm in 3D space, execute "roslaunch custom_navigation octo_rtt.launch".

To activate yolov5 firstly create the env "conda env create -f environment.yml" then conda activate yolov5, navigate to custom_yolo, catkin_make, source devel/setup.bash and run

“roslaunch my_detector mydetector.launch”. Then the topic can be added in rviz (detections_image_topic).

To run the DQN model, firstly launch the dqn launch in demos and then run “python3 src/gazebo_UAV_RL/validate.py”.

To run the TD3 model just run “python3 src/TD3/test_velodyne.py”. This can only be run if there is CUDA available and installed.