Multi-Drone Navigation Using Reinforcement Learning

Università di Genova

PADERBORN UNIVERSITY

Ecem Isildar

DIBRIS - Department of Computer Science, Bioengineering, Robotics and System Engineering

University of Genova

Supervisors:
Prof. Erdal Kayacan (UPB)
Assoc. Prof. Stefano Rovetta (UniGe)

In partial fulfillment of the requirements for the degree of

Laurea Magistrale in Robotics Engineering

December 19, 2024

Declaration of Originality

I, Ecem Isildar, hereby declare that this thesis is my own work and all sources of information and ideas have been acknowledged appropriately. This work has not been submitted for any other degree or academic qualification. I understand that any act of plagiarism, reproduction, or use of the whole or any part of this thesis without proper acknowledgment may result in severe academic penalties.

Abstract

This thesis addresses the important problem of locating survivors in collapsed buildings after an earthquake to assist search and rescue efforts. Traditional approaches are mainly based on predefined maps and manual search, making them very time-consuming and dangerous. A new approach where multiple unmanned aerial vehicles can detect a person independently is used in this study. The drones are equipped with cameras and flown in a simulated environment constructed with Gazebo. The autonomous navigation and decision-making of the drones are powered by reinforcement learning, particularly the usage of the Deep Deterministic Policy Gradient algorithm from Stable Baselines. With this, the continuous velocity actions regarding real-time camera observations of the drone could efficiently explore the environment and detect the people. A custom training environment was developed using Robot Operating System nodes in Python. The nodes communicate, guaranteeing a multi-drone system will always cover the whole disaster site, which minimizes search time. This is very critical in optimizing search strategy to avoid overlaps and ensure that the entire area is well covered in its search. In the current research, the performance of the proposed system is tested through simulations. Results from these simulations manifest the potential of reinforcement learning methods to enhance the efficacy of search and rescue missions with a robust framework that would further develop and can be tested in disaster scenarios. This thesis contributes to the area of autonomous systems and disaster response by providing insight into how, in practice, reinforcement learning can be used for search and rescue. The outcome contributes to this area of research by pointing out the various strengths and weaknesses of the approach.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Finding lost people after a natural disaster as quickly as possible is a critical challenge due to the urgent need for first aid and an inability to ask for help since these individuals are traumatized. Actually, in most cases, these people are not only physically incapacitated but are also mentally disoriented, and hence their chances of signalling their locations to the rescuers further diminish. The situation is compounded by the fact that traditional means of reaching people, such as highways, often become impassable or severely damaged in the occurrence of a big disaster. Rescue teams then have a hard time trying to navigate blocked roads and hazardous conditions, a factor that delays vital interventions even further. Besides, communication channels get saturated or disrupted with the quantum of need, making proper coordination for rescue very difficult. Whole regions may get completely cut off from means of communication, leaving victims in isolation without even a glimmer of hope to call for help. Immediate assistance is very vital, especially within the first 72 hours following the disaster, since this is considered the so-called "golden hours" for saving lives. Survival possibilities beyond this window become very low as the non-treatment of trauma and exposure to harsh conditions begin to take hold. Concerning this, one of the interesting responses to these situations is the use of unmanned aerial vehicles to spot injured and trapped people. UAVs can efficiently scan wide areas, and they can access those areas that are not accessible due to debris or collapse of infrastructures. Flying over these hazardous zones, UAVs provide a new type of viewpoint that, from the ground, could barely be accessible to the ground teams and, therefore, allow one to identify survivors much sooner. Using several UAVs shortens the search process substantially, increasing the likelihood of finding survivors within the critical time window. Parallel research with different sectors, as enabled by multi-drones, increases the efficiency of the whole operation once

more. Traditional search methods tend to depend on previously made maps of the environment, which can be highly inaccurate due to the disaster. The very dependence upon maps imposes a serious limitation since a natural disaster can change the looks of the landscape beyond recognition. Buildings that served as orientation points may be fully destroyed while new obstacles can take the form of a building collapse or fire. By contrast, reinforcement learning provides the most flexible approach. Trained reinforcement learning algorithms can work without a detailed map and generalize their search patterns to similar types of buildings and environments quite effectively in settings with dynamic and unpredictable conditions. All this flexibility gives way to UAVs, carrying such an algorithm, to make decisions on the fly in adjusting their paths following prevailing conditions they may come across. By leveraging the capabilities of UAVs combined with reinforcement learning, rescue teams can be more effective and efficient in finding and helping survivors. As technology develops, these autonomous systems will most likely assume responsibilities involving delivering emergency supplies to trapped persons in inaccessible areas. This technological synergy meets not only the immediate need for a rapid response but also accommodates the shifting conditions common in disaster areas. Most importantly, the near-constant stream of data from UAVs can be used to update rescue strategies in real-time, thus enabling teams to tailor their efforts to the evolving situation. Therefore, integrating UAVs with advanced search algorithms is an essential part of disaster responses that may save countless lives in these natural catastrophes. Besides offering a ray of hope for quicker rescues, this innovative strategy opens the way to further developments in automatic disaster relief technologies.

1.2 Introduction

Locating missing individuals after an earthquake as quickly as possible is a critical and challenging task due to the immediate need for medical intervention and the complexity of the post-earthquake environment. The urgency of this challenge is raised by the fact that traditional means of reaching people, such as highways, can be damaged after major earthquakes. Additionally, communication channels may be disrupted by the high volume of need, making it difficult to coordinate rescue efforts effectively Reuters (2023). In such circumstances, the deployment of UAVs to locate wounded and trapped individuals presents a promising solution. Drones are capable of quickly covering large areas and accessing locations that are otherwise unreachable due to collapse. Figure 1.1 summarizes the proposed DRL structure for search and rescue tasks.

It has two parts the upper one shows the training phase and the bottom one shows the test phase. In the training phase, the image data is taken from the

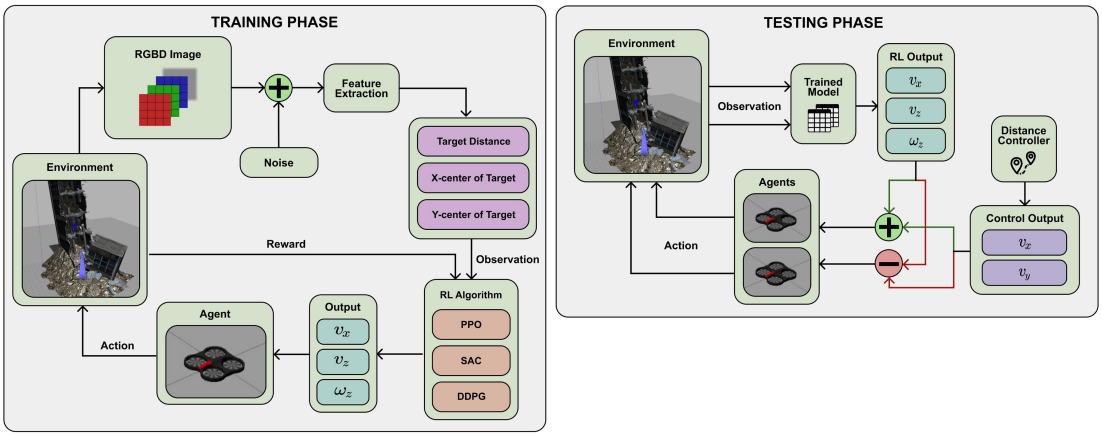


Figure 1.1: Graphical representation of training and the testing phases.

environment with the help of the drone camera and extracts the features of the image frame into three components such as target distance, x-center of the target and, y-center of the target. Then, these observation feeds the three different reinforcement learning (RL) algorithms. These are Proximal Policy Optimization (PPO), Soft Actor Critic (SAC), and DDPG for this case. Then RL algorithms give three velocity outputs to control the drone agent, which are linear x, linear z, and angular z velocities.

To test the success rate of the training, the drone agent is loaded with the RL models and finds the target on the different floors of the building. The key innovation lies in the multi-drone setup tested in this phase, where two UAVs, equipped with the same trained model, searched different floors simultaneously. This parallel search capability drastically improves the speed and efficiency of the operation. If the distance between these agents is dangerous the controller intervenes in the actions by disturbing both agents in slightly reverse directions to prevent a potential crash. A continuous action space is utilized in the RL model to achieve precise navigation and effective obstacle avoidance. This approach enables the UAVs to perform fine-tuned movements, enhancing their ability to maneuver through the unpredictable terrain of collapsed structures. The training of the RL agent takes place within the Gazebo simulation environment, utilizing the ROS2 framework to process camera inputs and issue velocity commands at each operation step. The results from this study indicate that the trained RL agent successfully located the target individuals, even when they were situated on different floors of the collapsed building which is placed in the Gazebo simulation environment. The success rate of these search operations will be discussed in further detail in the results section.

1.3 Thesis Structure

The structure of this thesis is outlined as follows:

1. **Literature Review**: This section gives an overview of the work that has so far been done on this topic and locates it within the broader scope of the field of drone technology and reinforcement learning. It gives a comprehensive account of the different technologies put into the drones used for this simulation, detailing their functions and why they are applied. Further, the chapter details reinforcement learning with some fundamental theories laying the foundation for it. It enumerates various algorithms, explaining their applications for various scenarios and linking them with the aims of this research.

2. **Problem Statement and Objectives**: The focus of this thesis is on elaborating a core problem being addressed in detail. Specific objectives of the research are put forth to act as a guide to subsequent sections. This chapter states the hypothesis on trial, examining the logical basis and implications for the design of the study.

3. **Methodology**: This chapter presents the detailed methodology followed at every step of the research. It provides information about the environmental setup required for the multi-drone system to work, which comprises hardware specifications and software configurations. In this chapter, the algorithms adopted in the framework will be discussed by providing the criterion and working process that form the basis for their selection. The communication protocols followed by the drones to make them work effectively with each other are also explained, and the exact architecture of the system is made amply clear.

4. **Experiments and Results**: This section deals with the experiments conducted to test the hypotheses and objectives of the study. It details how the training process was designed, including the experiment design and data collection methods, the performance metrics used, and the outcomes of the experiments conducted showing the efficacy of the multi-drone system and pointing out all its features.

5. **Discussion**: In this chapter, the results will be discussed in the light of the research questions put forward at the outset. These findings are critically analyzed with a view to the literature and theories set out above. Secondly, there is a discussion of limitations related to this study; this too will provide valuable insights into the problems encountered and the consequences this may have for the reliability and validity of these results.

6. **Conclusion**: This last chapter summarizes what this research added both to drone technology and to reinforcement learning. The significance of the findings and how they could influence further research and practical application have been discussed. Furthermore, suggestions for future research by pointing out some avenues for further exploration and refinement of methodologies and technologies discussed are provided.

7. **References**: This is the last part of the thesis, which involves a comprehensive reference list summarizing those works referenced in this research effort. The section will not only be indicative of foundational works that have, in one way or the other, influenced this study but also serve as a useful companion for other researchers who might be interested in looking into the aspects of drone technology and reinforcement learning.

Chapter 2

Literature Review

There is a growing area demonstrating that DRL can effectively address the challenge of drone navigation, especially in complex and unstructured environments where traditional methods may struggle. In such environments, employment of multiple UAVs, the search process can be significantly accelerated, increasing the chances of finding survivors within the critical time as mentioned in Robotics (2023). Traditional methods as explained in Souissi *et al.* (2013) A* and Dijkstra's algorithm struggle in dynamic and unstructured environments. However, these methods rely on pre-existing maps and structured environments, making them less suitable for post-earthquake scenarios where the environment is dynamic and unpredictable. Simultaneous localization and mapping (SLAM) is another method for the search operations which requires a map of the area as shown in Wang *et al.* (2020). Although slam maps unknown environments while simultaneously navigating through them, it depends on highly accurate sensor data and requires building and updating maps in real-time which can be problematic in a collapsed building environment. Machine learning algorithms used in Sambolek & Ivasic-Kos (2021) can also be applied to this problem but need a huge dataset to achieve high precision. It is also very hard to gather such large datasets in this environment and collecting it takes too much time. In contrast to these approaches, RL presents a novel and advantageous alternative that does not require a pre-existing map of the environment. This flexibility is particularly beneficial in GPS-denied environments, where DRL can effectively guide UAVs without relying on external positioning systems as mentioned in Bodi *et al.* (2023). RL does not require any map of the environment and collects data during training. Although lidars are commonly used in RL such as Ramezani & Atashgah (2024), RGB-depth (RGBD) cameras are preferred due to their easy calibration, affordability, and lightweight design used in Kang *et al.* (2019). One approach, q-learning has been utilized for path planning for UAVs where the drone takes discrete actions to navigate through its environment as mentioned in

Yan & Xiang (2018). Another study uses the deep q-network (DQN) to control drone altitude by selecting discrete actions, showing how DRL can be applied to control specific aspects of drone flight which is shown in Zhou *et al.* (2020). Additionally, DQN has been successfully used in scenarios where images are employed as the observation space, highlighting its versatility in processing visual inputs for navigation purposes as Zuluaga *et al.* (2018) suggested. The potential of DRL extends further into 3D navigation, where images and drone positions are incorporated into the observation space. This approach allows drones to maneuver within complex 3D environments, guided by visual and positional data used in Kim *et al.* (2017). Some studies have explored the use of DDPG for path planning, particularly in 2-dimensional spaces. In these cases, DDPG enables the UAV to take continuous actions based on its position, which is provided as a state at each step of the learning process mentioned in Lillicrap *et al.* (2015). For instance, an improved DDPG algorithm has been proposed for UAVs navigating in large-scale, complex terrains, highlighting its potential in such challenging scenarios shown in Peng *et al.* (2023). In environments where multiple UAVs are used, the search process can be significantly accelerated, increasing the chances of finding survivors within the critical time frame, this approach is used in Pham *et al.* (2018). Moreover, DDPG is effective in teaching drone agents to navigate in 3D environments by taking continuous actions while continuously observing their positions, Bouhamed *et al.* (2020) used continuous action space in the 3D environment. Notably, an improved version of DDPG has been developed to enhance UAV navigation accuracy by incorporating corrective feedback mechanisms, which have proven effective in complex environments as mentioned in Wu *et al.* (2022). Furthermore, the application of DDPG has been expanded to scenarios such as rescue missions, where UAVs need to operate autonomously and adaptively in post-earthquake settings as shown in Ma & Chen (2022). Overall, traditional methods and machine learning methods contribute under specific conditions. However, the versatility of DRL, especially continuous action spaces, makes it the most suitable among all presented approaches for searching and rescuing operations in a building that collapsed. The possibility to navigate without prior maps, the use of efficient sensors like RGBD cameras, and multi-UAV coordination make it a robust framework to deal with the difficulties of this application.

Chapter 3

Problem Statement and Objectives

3.1 Problem Statement

Building collapses resulting from earthquakes or other disasters leave people buried under debris, always a big challenge during search and rescue operations. Conventional methods of locating survivors employ manual searches and dogs that are specially trained, both of which are time-consuming and hazardous for rescue workers. These traditional methods generally require prior knowledge of the structure of the building, which may be missing or unreliable after the disaster.

The crucial challenge this thesis solved was to quickly and efficiently detect the survivors in the collapsed building using an autonomous multi-drone system. So, for such traditional labour-intensive search methods with a variety of limitations, the proposed solution uses an advanced sensor and reinforcement learning-equipped unmanned aerial vehicle. These drones will then autonomously navigate the disaster site, communicate with each other to ensure complete coverage and identify the presence of survivors, independent of any prior maps of the building. The challenge is to devise a system operating in an unpredictable and cluttered environment, effective coordination among multiple drones, and real-time sensory data for making accurate decisions. This problem enables the thesis to work its way into speeding and safeguarding the searching and rescue process for higher salvation chances in the instance of a disaster. In general, the aim of the thesis is the development of an autonomous multi-drone system for detecting survivors in buildings that collapse using reinforcement learning. Specific objectives are defined ın the following section to achieve this goal.

3.2 Objectives

1. **Configure and Calibrate Drones**: Equip the drones with advanced RGBD cameras. Develop and implement publishers and subscribers within the ROS to enable seamless communication and data exchange. This involves ensuring that each drone's camera system is precisely calibrated for accurate data collection and transmission.

2. **Implementing a Controller**: Design and apply a classical Proportional Integral Derivative (PID) controller to the drone system. The objective is to validate the drone's ability to reach predetermined goal positions accurately using consistent velocity commands. This process involves fine-tuning the PID parameters to achieve optimal control performance in various operating conditions, ensuring reliable and stable drone navigation.

3. **Creating Training Environment to Test RL Algorithms**: Adapt the DDPG algorithm from the Stable Baselines library for a multi-drone system. This entails defining the action space and observation space for each drone, as well as designing a comprehensive reward structure that encourages efficient and effective task completion. The training environment must be robust and capable of simulating a variety of realistic scenarios that the drones might encounter in actual deployments.

4. **Train the Drone in the Collapsed Building**: Conduct extensive training sessions within a simulated collapsed building environment to optimize the drones' performance. The goal is to enhance the drones' ability to explore disaster sites and accurately detect survivors. This phase involves iterative training and testing, refining algorithms and strategies to improve search efficiency, accuracy, and overall mission success rates.

5. **Adding Multiple Drones to the Environment**: Integrate additional drones into the operational scene by duplicating all necessary publishers, subscribers, and callback functions. This ensures that the system can manage and coordinate multiple drones simultaneously, facilitating collaborative search and rescue efforts. Each drone must be capable of independent operation while maintaining seamless communication and coordination with other drones.

6. **Analyze and Interpret Results**: Perform a detailed analysis and interpretation of the system's performance data. Compare the results with different DRL algorithms to identify the strengths and weaknesses of the proposed system. This analysis should highlight the system's advantages in terms of efficiency, accuracy, and reliability, as well as potential areas for

further enhancement. The insights gained will inform future development and optimization efforts to improve the overall effectiveness of drone-based search and rescue operations.

3.3 Hypotheses

The following hypotheses are formulated to guide this research:

- **Hypothesis 1:** Utilizing multiple drones equipped with RGBD cameras and reinforcement learning algorithms will significantly reduce the time required to locate survivors in a simulated collapsed building compared to traditional search and rescue methods.
- **Hypothesis 2:** The proposed multi-drone system will achieve a faster solution in detecting survivors compared to single-drone systems, due to improved coverage and inter-drone communication.
- **Hypothesis 3:** The reinforcement learning algorithm, specifically the DDPG algorithm, will enable drones to effectively navigate and search the simulated environment without prior knowledge of the building layout.

Chapter 4

Methodology

4.1 Drone Model

The drone model taken from NovoG93 (2022) is often utilized in research and development for various applications, including aerial robotics, path planning, and control strategies. The drone has a quadrotor configuration, which consists of four rotors positioned at the corners of a rectangular or square frame. This configuration allows for good stability and maneuverability. It can carry various payloads, such as cameras, sensors, or additional equipment for specific tasks.

4.1.1 Mathematical Dynamics

The dynamics of a quadrotor drone can be described using a combination of Newton-Euler equations and kinematic equations. The motion can be analyzed in terms of translational and rotational dynamics.

4.1.1.1 Translational Dynamics

The translational dynamics of the drone can be represented by the following equations:

$$\dot{x} = u_x \tag{4.1}$$
$$\dot{y} = u_y \tag{4.2}$$
$$\dot{z} = u_z \tag{4.3}$$

Where:

- x, y, z are the positions in the world frame.

- u_x, u_y, u_z are the velocity components in the respective directions.

The forces acting on the drone are typically the thrust generated by the rotors and the gravitational force:

$$F = m \cdot g - T \tag{4.4}$$

Where:

- F is the net force,
- m is the mass of the drone,
- g is the acceleration due to gravity,
- T is the total thrust produced by the rotors.

4.1.1.2 Rotational Dynamics

The rotational dynamics can be described using Euler angles (ϕ, θ, ψ) for roll, pitch, and yaw. The rotational motion can be represented as:

$$\dot{\phi} = p + \sin(\phi)\tan(\theta)q + \cos(\phi)\tan(\theta)r \tag{4.5}$$
$$\dot{\theta} = \cos(\phi)q - \sin(\phi)r \tag{4.6}$$
$$\dot{\psi} = \frac{\sin(\phi)}{\cos(\theta)}q + \frac{\cos(\phi)}{\cos(\theta)}r \tag{4.7}$$

Where:

- p, q, r are the angular velocities around the body-fixed axes.

The rotational dynamics can be expressed with the moment of inertia and the torques generated by the rotors:

$$\tau_x = L \cdot (T_2 - T_4) \tag{4.8}$$
$$\tau_y = L \cdot (T_1 - T_3) \tag{4.9}$$
$$\tau_z = K \cdot (T_1 + T_2 + T_3 + T_4) \tag{4.10}$$

Where:

- L is the distance from the center of the drone to the rotors,
- K is a constant related to the rotor characteristics,
- T_1, T_2, T_3, T_4 are the thrust forces produced by each rotor.

4.1.2 Control Inputs

The drone's control inputs typically consist of:

- **Throttle**: Adjusts the total thrust produced by all rotors.
- **Roll, Pitch, Yaw Commands**: Adjusts the distribution of thrust among the rotors to control the orientation and direction of flight.

4.1.3 Environment Interaction

When analyzing the dynamics, the interaction with the environment, such as wind or obstacles, is crucial. Environmental factors can be modeled as additional forces or disturbances affecting translational and rotational motion.

4.1.4 Simulation

This drone model is often simulated in the Gazebo environment, where various factors, including dynamics, control strategies, and environmental interactions, can be tested.

4.2 Classical Control

The first step before starting to implement an advanced method to control the drone is applying a PID controller. This step is necessary to ensure that the drone model in the simulation works properly with the selected velocities. First, a target is created in the Gazebo simulation environment and a PID controller is implemented. It gets the distance between the target and the drone and publishes the correct velocity messages to reach the target.

The control loop feedback mechanism is another critical component of a PID controller in readjusting the movement of the drone proportionally to the error it detects against the desired position and the actual position of a target. Three salient components comprise the PID controller, which is stated as follows:

- Proportional (Kp): This is a term in proportion to the current error. The larger the error, the larger the control output will be to allow immediate correction.
- Integral (Ki): This is accumulated from all the errors that have been realized in the past. The longer this drone has stayed consistently off target, the stronger this term becomes in output to help clear out steady-state errors and ensure the drone reaches a desired position.

- Derivative (Kd): This estimates future errors based on the rate of change of the current error. This dampens the system response by preventing overshooting of the target.

4.2.1 Frame Processing

The overall control logic is coordinated by the function process_frames. It computes the reference and actual positions and then estimates the distance and orientation errors. The PID controllers use these to derive the required velocities. Computed velocities are published as motor commands in order to control movements.

4.2.2 Action Item

The ‘take_action‘ method publishes the computed velocity commands in order to make the drone adjust based on outputs from PID controllers.

This means that the code enables the drone to make its way independently in the environment and, with the help of vision, find the blue objects. For flying accurately, PID controls the drone movement with great precision while the DDPG optimizes it using reinforcement learning. Due to this combination, the interaction capability of the drone with the environment improves and is thus suitable for tasks such as search and rescue or inspection. The algorithm below shows how it works:

Algorithm 1 PID Control

Input: Proportional gain K_p, Derivative gain K_d, Integral gain K_i
Output: Control signal u
Initialize $prev\_error \leftarrow 0$
Initialize $integral \leftarrow 0$
function COMPUTE($error$, $delta\_time$)
    $integral \leftarrow integral + error \times delta\_time$
    if $integral < integral\_min$ **then**
        $integral \leftarrow integral\_min$
    else if $integral > integral\_max$ **then**
        $integral \leftarrow integral\_max$
    end if
    $derivative \leftarrow (error - prev\_error)/delta\_time$
    $u \leftarrow K_p \times error + K_d \times derivative + K_i \times integral$
    $prev\_error \leftarrow error$
    if $u < output\_min$ **then**
        $u \leftarrow output\_min$
        $integral \leftarrow integral - anti\_windup\_gain \times (u - output\_min) \times delta\_time$
    else if $u > output\_max$ **then**
        $u \leftarrow output\_max$
        $integral \leftarrow integral - anti\_windup\_gain \times (u - output\_max) \times delta\_time$
    end if
     return u
end function

4.3 Introduction to Markov Decision Process

Before starting to explain what is reinforcement learning it is important to understand the Markov Decision Process which is the basis of reinforcement learning. It is a mathematical framework used to model decision-making in environments where outcomes are partly random and partly under the control of a decision-maker. An MDP is defined by the following components:

- S: A set of states
- A: A set of actions
- $P(s'|s, a)$: Transition probability function, representing the probability of moving from state s to state s' after taking action a

- $R(s, a)$: Reward function, representing the immediate reward received after taking action a in state s
- $\gamma \in [0, 1]$: Discount factor, which represents the importance of future rewards.

Transition Probability Function

$$P(s'|s, a) = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a) \tag{4.11}$$

This represents the probability of transitioning to state s' given that the agent was in state s and took action a.

Reward Function

$$R(s, a) = \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a] \tag{4.12}$$

This is the expected reward received after taking action a in state s.

Policy

A policy π is a strategy for choosing actions. A deterministic policy π maps states to actions, denoted as $\pi(s) = a$, while a stochastic policy defines a probability distribution over actions:

$$\pi(a \mid s) = \Pr(a_t = a \mid s_t = s) \tag{4.13}$$

4.3.1 Value Functions

The value function gives the expected cumulative reward starting from a state, under a given policy π. There are two main value functions:

- **State-value function** $V^\pi(s)$: The expected return when starting in state s and following policy π thereafter:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s \right] \tag{4.14}$$

- **Action-value function** $Q^\pi(s, a)$: The expected return when starting in state s, taking action a, and following policy π thereafter:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, a_0 = a \right] \tag{4.15}$$

Example: Imagine you are in a video game where you are deciding whether to move left or right in a maze:
State-value function $V(s)$: If you're in a certain location (state s) in the maze, $V(s)$ tells you the expected total score you will get if you follow a particular strategy (policy π) from that location. It averages over all possible actions according to the strategy.

Action-value function $Q(s, a)$: It will tell the expected total score if you are in state s, take action a (e.g., move left or move right), and then follow the strategy π afterwards. It gives more specific information about what might happen after a particular action.

4.3.2 Bellman Equations

The Bellman equations express the value functions recursively to simplify the state-action value calculation. Instead of calculating the expected return for each state or each state-action pair, we can use the Bellman equation according to HuggingFace (2021a).

- **Bellman equation for state-value function**:

$$V^{\pi}(s) = \sum_{a} \pi(a \mid s) \sum_{s'} P(s' \mid s, a) \left[R(s, a) + \gamma V^{\pi}(s')\right] \tag{4.16}$$

- **Bellman equation for action-value function**:

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) \sum_{a'} \pi(a' \mid s') Q^{\pi}(s', a') \tag{4.17}$$

4.3.3 Monte Carlo vs Temporal Difference Learning

An RL agent learns by interacting with the environment by updating its value function or policy according to the received reward. Monte Carlo and Temporal Difference (TD) Learning are two different ways to train the function. While Monte Carlo uses an entire episode of experience before learning, temporal difference only uses a step to learn mentioned in HuggingFace (2021c).

- Monte Carlo: Waits until the end of the episode then it calculates G_t and updates the target $V(S_t)$. That means it needs to wait until one episode ends before updating the value function. Then starts the new episode with this new knowledge.

$$V(S_t^*) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right] \tag{4.18}$$

where:

  - $V(S_t^*)$ is the new value of state t
  - $V(S_t)$ is the estimation of the value of state t
  - α is the learning rate
  - G_t is the return at step t

- Temporal Difference: Waits for only one iteration S_{t+1} to TD target and update $V(S_t)$ using R_{t+1} and $\gamma V(S_{t+1})$.

$$V(S_t^*) \leftarrow V(S_t) + \alpha \left[R_{t+1} - \gamma V(S_t + 1) - V(S_t)\right] \tag{4.19}$$

  where:

  - $V(S_t^*)$ is the new value of state t
  - $V(S_t)$ is the estimation of the value of state t
  - α is the learning rate
  - R_{t+1} is the reward
  - $\gamma V(S_t + 1)$ is discounted value of next state

Optimal Policy

The goal of an MDP is to find an optimal policy π^* that maximizes the expected return. The optimal state-value and action-value functions are denoted as $V^*(s)$ and $Q^*(s, a)$, and they satisfy the **Bellman optimality equations**:

- **Optimal state-value function**:

$$V^*(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a) + \gamma V^*(s')\right] \tag{4.20}$$

- **Optimal action-value function**:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) \max_{a'} Q^*(s', a') \tag{4.21}$$

The optimal policy π^* is the policy that selects actions which maximize $Q^*(s, a)$:

$$\pi^*(s) = \arg\max_a Q^*(s, a) \tag{4.22}$$

4.4 Introduction to Reinforcement Learning

4.4.1 Reinforcement Learning

Reinforcement learning (RL) is an interdisciplinary area of machine learning and optimal control concerned with how an intelligent agent takes action in a dynamic environment to maximize the cumulative reward according to Wikipedia (2024). The RL consists of two main parts which are the **agent** and the **environment**. The agent is the robot which learns to do some task by taking some actions in the environment in which the robot is located. After every action, it observes new states and also gets a **reward** according to how good or bad is the action it selected. The goal is to maximize the cumulative reward which is called **return**. Fig. 4.1 shows the agent and environment interactions in RL.

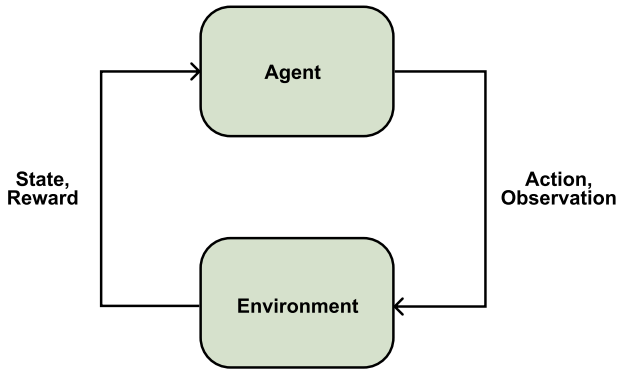


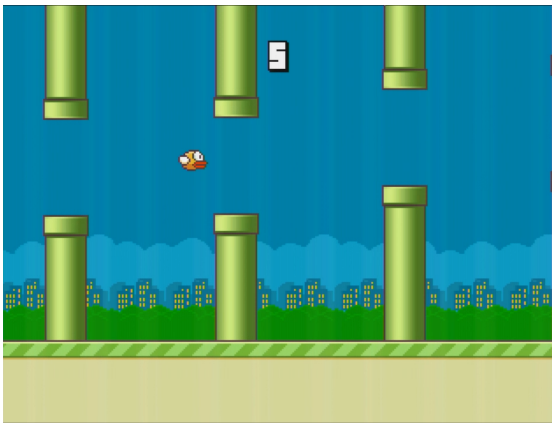
Figure 4.1: RL loop

In RL, the agent does not know the transition probabilities $P(s'|s,a)$ or the reward function $R(s,a)$. Instead, the agent interacts with the environment, learns through trial and error, and improves its policy over time. MDPs provide the theoretical foundation for RL because they describe how the environment responds to the agent's actions. The agent's task in RL is to approximate the optimal policy for this MDP.

- **States and Observations**: The state s describes the environment fully without extracting any information. A robot state can be represented as positions and velocities. While observation o is a partial description of a state, which may omit information. For example, it can be an RGB matrix of the image given in Openai (2018b). The agent can observe the states fully or not depending on the environment. For instance, it fully observes the grid world or frozen lake environment but can not observe fully the

Mario or the flappy bird environments. Fig. 4.2a shows a grid world which is frequently used in RL. Fig. 4.2b shows a non-observable environment.

(a) Example grid world frozen lake



(b) Example environment which is not fully observable

Figure 4.2: (a) shows frozen lake environment (b) shows flappy bird environment

- **Action Spaces**: The valid action that an agent can take depends on the environment. For instance, games such as Atari or Go only accept **discrete action spaces** while controlling a robot in a physical world needs **continuous action spaces**.
- **Discount Factor**: The cumulative reward at each step is written as:

$$R(\tau) = r_{t+1} + r_{t+2} + r_{t+3} + ... = \sum_{k=0}^{\infty} r_{t+k+1} \tag{4.23}$$

In this equation, τ represents the trajectory which means the sequence of states and actions. In the Fig. 4.3 the mouse has to eat the maximum amount of cheese without being eaten by the cat. As we can see in the diagram, it's more probable to eat the cheese near us than the cheese close to the cat (the closer we are to the cat, the more dangerous it is). Consequently, the reward near the cat, even if it is bigger (more cheese), will be more discounted since we're not sure we'll be able to eat it. So the expected cumulative reward should be decreased in each step and it is written as:

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{4.24}$$

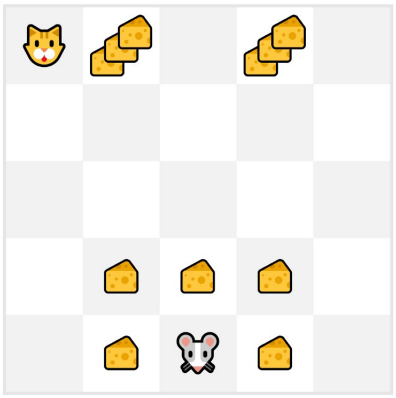


Figure 4.3: Example cheese eating game

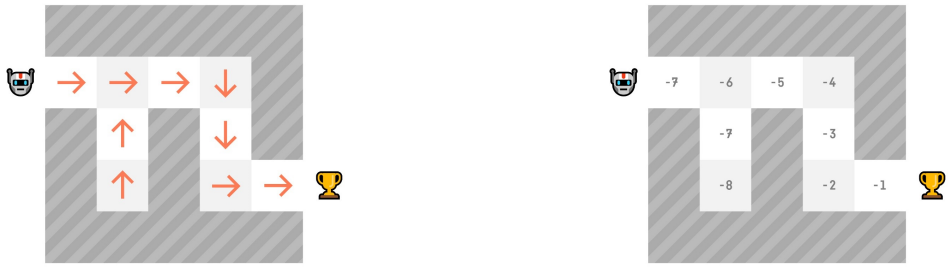
- **Task types**:
  - Episodic: The tasks with a starting and ending point such as Super Mario.
  - Continuing: The tasks that last continuously such as stock trading.
- **Exploration and Exploitation**: Exploration means that the agent takes random actions to find more information about the environment and exploitation is using the known information to maximize the reward.

There are two main approaches to solving RL problems:

- **Policy-Based Methods:** It can be deterministic or stochastic. The agent directly learns a policy function. Fig. 4.4a directly shows which action the agent should take in each step. The deterministic policy always gives the same action $a = \pi(s)$, while a stochastic policy defines a probability distribution over actions $\pi(a \mid s) = P[A \mid s]$.
- **Value-Based Methods:** The agent learns a value function which shows the expected value of being in a state. Fig. 4.4b directly shows the values for each step so the agent can select the highest values in each step. So, this function shows the value-base policy

$$v^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+1} \mid S_t = s\right] \tag{4.25}$$

- **On-Policy**: In these algorithms, the policy being learned and the policy used to generate data (interact with the environment) are the same.



(a) Policy-based approach (b) Value-based approach

Figure 4.4: (a) shows policy-based approach and (b) shows value-based approach

- **Off-Policy**: In these algorithms, the policy being learned is different from the policy used to generate data.

4.4.2 Reinforcement Learning Algorithms

- Model-Based RL: In this model, the agent can simulate the environment, predict future states, and plan optimal actions by looking ahead.
- Model-Free RL: In this model, the agent learns to optimize its behavior by directly interacting with the environment, without constructing or using a model of the environment's dynamics. Instead, the agent learns a value function (like Q-values) or a policy that maps states to actions based on experience. Deep Q-Learning (DQN), PPO, DDPG, and SAC can be counted as model-free algorithms.
  - **Q-Learning**: It is an off-policy value-based method to train its action-value function. Fig. 4.5 shows **the Quality** of the action at a specific state. The Q-table consists of state-action pair values. Figure taken

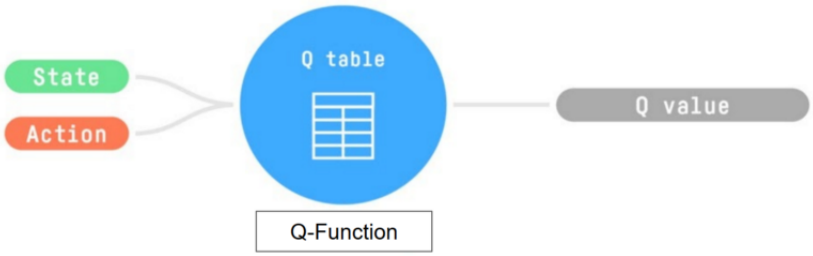


Figure 4.5: Q-learning algorithm.

from HuggingFace (2021d). At the beginning all values are zero, then

it is filled with the Q values during the training. After the training, we obtain the optimal Q-table. So, we have an optimal policy because we know the best action to take in each state. The optimal policy is calculated using the formula below:

$$\pi^*(s) = \text{argmax}_a Q^*(s, a) \tag{4.26}$$

Algorithm 2 Q-Learning

Require: policy π, positive integer *num_episodes*
Ensure: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
1: Initialize Q arbitrarily (e.g., $Q(s, a) = 0$ for all $s \in S$ and $a \in A(s)$, and $Q(terminal\_state) = 0$)
2: **for** $i = 1$ to *num_episodes*
3: $\epsilon \leftarrow \epsilon_i$
4: Observe S_0
5: $t \leftarrow 0$
6: **repeat**
7: Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
8: Take action A_t and observe R_{t+1}, S_{t+1}
9: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$
10: $t \leftarrow t + 1$
11: **until** S_t is terminal
12: **end**
13: **return** Q

- **DQN**: The DQN algorithm is one such agent that combines Q-learning with deep learning for solving reinforcement learning tasks with large or high-dimensional state spaces, which could be represented by images or video frames. In this instance, the agent learns to approximate the Q-function using a neural network. The algorithm makes use of experience replay, wherein the experiences of the past are stored in a replay buffer and sampled randomly to take consecutive samples and stabilize training. DQN also uses a target network for further stability of training by keeping a separate network for computing the target Q-values. It is updated not as often as the main network. The agent follows the policy dictated by the derived Q-values, which help it pick actions to maximize rewards, thus enabling it to learn effectively in complicated environments.
- **PPO**: It is a policy-gradient method, which means it optimizes the policy directly from states to actions. It tries to maximize a surrogate

objective function and the key point is to avoid large updates according to Openai (2017). It uses a ratio that indicates the difference between the current and old policy and clips this ratio to the $[1-\epsilon, 1+\epsilon]$ range given in HuggingFace (2021b). So, the agent can learn stably. The objective function is shown below:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{4.27}$$

where

* Θ is the policy parameter
* $\mathbb{E}_t$ denotes the empirical expectation over timesteps
* r_t is the ratio of the probability under the new and old policies, respectively
* $\hat{A}_t$ is the estimated advantage at time t
* ϵ is a hyperparameter, usually 0.1 or 0.2.

PPO trains a stochastic policy in an on-policy way, which means that it explores by sampling actions regarding the latest version of the policy. Over the training, the policy typically becomes less random, because the update rule encourages it to exploit rewards that it has already found.

– **DDPG**: It is an actor-critic model-free algorithm which learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function and the Q-function to learn the policy. It combines both policy and value-based methods and is suitable for the continuous action space. This approach combines Deterministic Policy Gradient (DPG) and DQN as written in Openai (2018a).

  * Deterministic Policy: Unlike stochastic policies used in PPO, DDPG uses a deterministic policy, so, the action is always the same value instead of being a distribution.
  * Actor-Critic Networks: The actor is a policy network that determines which action to take in the current state. It outputs a deterministic action, so, the action is always the same. The critic is a value network that evaluates the action taken by the actor by estimating the Q-value, which is the expected return (future rewards) from taking a specific action in a given state.
  * Off-policy Learning: It is an off-policy algorithm. So, it learns from past experiences.
  * Experience Replay: It uses experience replay this buffer stores the list of $(state, action, reward, next_state)$ and instead of learning

only from recent experience, we learn from sampling all of our experience accumulated so far. It consists of actor, critic, target actor and target critic networks mentioned in Keras (2018).

* Target Networks: It uses target networks to stabilize the learning for both actor and critic networks. The target networks help reduce the risk of the Q-values diverging during training. The target network update formulas are given below:

$$\theta^{Q}_{\text{target}} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q}_{\text{target}} \tag{4.28}$$

$$\theta^{\pi}_{\text{target}} \leftarrow \tau\theta^{\pi} + (1-\tau)\theta^{\pi}_{\text{target}} \tag{4.29}$$

where:

- τ is a small positive constant (typically equal to 0.005)
- Θ^{Q} and Θ^{π} are the parameters of the main critic and actor networks, respectively.
- Θ^{Q}_{target} and Θ^{π}_{target} are the parameters of the target critic and actor networks, respectively.

– **SAC**: It is an algorithm that optimizes a stochastic policy like PPO but in an off-policy way. So it is an algorithm in between PPO and DDPG. The main feature of this algorithm is **entropy regularization**. It is trained to maximize the trade-off between expected return and entropy, which is the measure of randomness in the policy. Increasing entropy results in more exploration given in Openai (2018c). The entropy regularization formula is given below:

$$\mathcal{H}(\pi) = \mathbb{E}_{s_t}\left[\alpha \log \pi(a_t|s_t;\theta^{\pi})\right] \tag{4.30}$$

where:

* $\mathcal{H}(\pi)$ is the entropy of the policy.
* α is the temperature parameter that controls the trade-off between reward and entropy. A higher α increases the emphasis on entropy, leading to more exploration.
* $\pi(a_t|s_t;\theta^{\pi})$ is the policy's probability distribution over actions given state s_t parameterized by Θ^{π}.

Table 4.1 compares the PPO, DDPG and SAC algorithms which are used to train the drone agent in this project.

Feature	PPO	DDPG	SAC
Type	On-policy	Off-policy, deterministic	Off-policy, stochastic
Action Space	Continuous or Discrete	Continuous	Continuous
Exploration Strategy	No explicit exploration	Noise addition	Entropy maximization
Stability	Highly stable	Can be unstable	Highly stable
Sample Efficiency	Moderate to High	High	High

Table 4.1: Comparison of PPO, DDPG, and SAC algorithms

4.4.3 Deep Reinforcement Learning Framework for Navigation and Rescue Operations

DRL enables autonomous systems like drones to navigate complex environments, make real-time decisions, and adapt to challenges during rescue missions. This can significantly increase the efficiency and effectiveness of search-and-rescue operations, especially in inaccessible areas, by reducing human risk and improving the chances of locating survivors quickly. In order to solve this problem a drone agent equipped with an RGBD camera is trained to find the target position in a collapsed building which is placed in the Gazebo environment using three different RL algorithms which are PPO, SAC and DDPG. Fig. 4.6 shows the general RL structure with these algorithms. This agent has an RGBD camera and this camera images are extracted to three elements to reduce the input observation space. Then these three components are sent to the RL algorithms to calculate the velocity actions by considering the rewards collected in each step.

4.4.4 Observation and Action Spaces

For this project RGBD image information is used for the observation space and the continuous velocity actions are used for the action space. This means that the drone agent needs to learn from the RGBD information and directly take velocity actions. The reason of choosing PPO, SAC and DDPG is that all of these algorithms allow to take continuous actions. Because the drone agent in the environment should take continous actions instead of only discrete actions in order to ensure that it can easily reach the target by doing continuous manuevers.

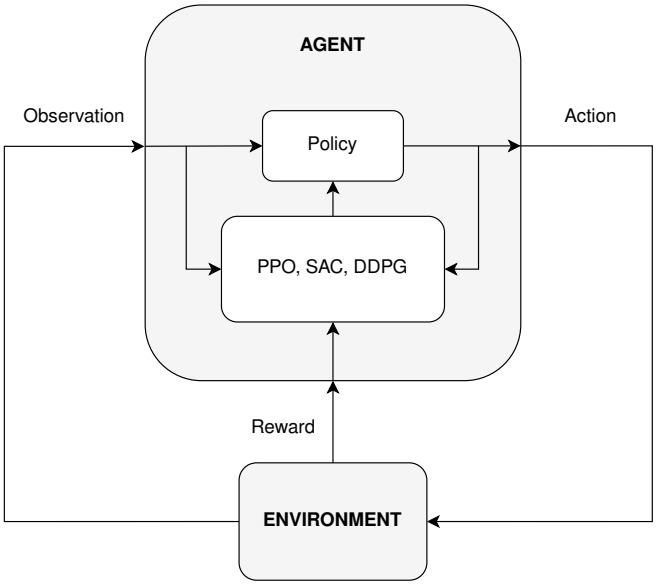


Figure 4.6: Agent-Environment interaction loop with different algorithms

4.4.5 Reward Function Design

4.4.6 Termination Conditions

In this custom environment which is created for the drone agent, there are 3 termination conditions:

- **Termination 1:** The first termination condition occurs if the agent crashes to something in the environment.
- **Termination 2:** The second termination condition occurs if the agent reaches the target position.
- **Termination 3:** The third one is actually a truncation instead of a termination, because it truncates the episode if the agent can not find the target within 60 seconds. This kind of terminal condition is called truncation because the agent is not able to learn how much time it takes and why it is terminated suddenly. For example, the drone agent is flying around and collecting some rewards then it can stay in a position and get so much reward but it is actually not the solution we would like it to be found by the agent. The goal is getting closer to the target position as soon as possible.

4.4.7 Custom Environment for Reinforcement Learning

A custom environment is created for training the agent using a ROS node. The Algorithm 3 describes this environment. The ROS 2 node is initialized, which serves as the communication hub for subscribing to sensor data and publishing commands to control the drone. The environment subscribes to various ROS 2 topics to receive real-time data from the drone. These subscriptions include RGB and depth images from the drone's cameras and odometry data to track the drone's position. Additionally, collision states are monitored to detect any impacts or crashes. The take action function publishes velocity commands according to the RL algorithm in each step when called by the step function. The step function returns observation, reward, termination and truncation. Episodes reset when the drone has collided with or collected the target, in these cases termination flag is true or reset if the agent cannot find the target in 60 seconds, in this case truncation flag is true. The reset function lands the drone to the initial position, resets the reward and the observation space.

4.4.8 Communication Interfaces

The ROS 2 node is initialized, which serves as the communication hub for subscribing to sensor data and publishing commands to control the drone. This setup allows the environment to interface seamlessly with the drone's simulated sensors and actuators. The environment subscribes to various ROS 2 topics to receive real-time data from the drone. These subscriptions include RGB and depth images from the drone's cameras and odometry data to track the drone's position. Additionally, collision states are monitored to detect any impacts or crashes. To control the drone, the environment publishes commands to ROS 2 topics. These commands manage the drone's movement by sending velocity instructions and handling flight operations such as taking off and landing. This bidirectional communication ensures that the environment can observe the drone's state and commands to direct its behavior.

4.4.9 Multi-Drone Communication

The diagram in Fig. 4.7 illustrates the system architecture designed to control and manage multiple drones using an RL model.

This architecture is only applied during the test phase, which means both drone agents have loaded the same trained model and they only tested together to decrease the total searching time. The system consists of two main components: the custom environment and the distance controller. Each drone operates within the environment, which provides position information necessary for navi-

Algorithm 3 Drone Environment

Initialize drone controller clients, publishers, subscribers
Initialize observation space and action space
Initialize agent position, rewards, observation

Function `collision_callback`:
    Check for collisions and update contact status
Function `rgb_callback`:
    Process RGB image data and detect the target if it is in the field of view
Function `depth_callback`:
    Process depth image data and update distance to the target

Function `take_action(v_yaw, v_x, v_z)`:
    Publish velocity commands to the drone
Function `takeOff()`:
    Command the drone to take off
Function `land()`:
    Command the drone to land

Function `step(action)`:
    Execute action and update the drone's position
    Calculate distance to the goal and check for success or collision
    Compute reward based on goal distance and penalties
    return (observation, reward, terminated, truncated)

Function `reset()`:
    Reset simulation, call land function and resets drone state
    Take off the drone and set the goal position
    return initial observation

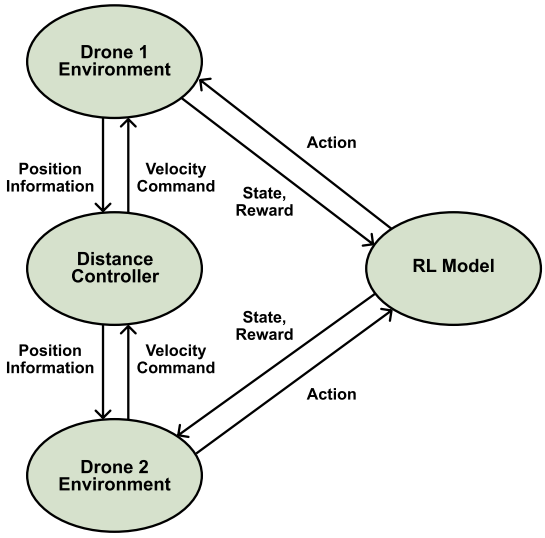


Figure 4.7: System architecture in ROS

gation. This position data is sent to the distance controller, and if the distance between the drones is too large, the distance controller sends speed messages to both drones in the same direction to bring them closer together. In the opposite case, it sends speed messages in opposite directions to separate the drones until the distance is within the safety limits. Otherwise, drones continue to take the actions, coming from the RL model. Algorithm 4 shows how the distance controller node works to control drone distances. This architecture enables the autonomous and adaptive operation of multiple drones using the same trained model.

4.5 Drone Configuration and Sensors

The drone description package is used for the training. The illustration of the quadcopter in the environment is displayed in Fig. 4.8.

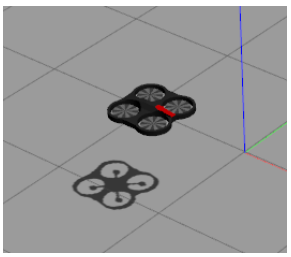


Figure 4.8: The drone model in Gazebo environment.

Algorithm 4 Distance Controller Pseudocode

```
1: Class DistanceController
2: Function DistanceController_init()
3: Initialize node, position subscriptions, velocity publishers
4: End Function
5: Function drone1_position_callback
6: Update drone1_position
7: End Function
8: Function drone2_position_callback
9: Update drone2_position
10: End Function
11: Function check_distances()
12: Compute distance between drones
13: if distance > 10.0 or < 1.0 then
14:     Adjust drone velocities via publishing velocity messages
15: end if
16: Publish new velocity commands
17: End Function
18: Function main()
19: Initialize ROS, create DistanceController node, spin and shutdown node
20: End Function
```

It is suitable for ROS2 and has a Unified Robot Description Format (URDF) file for the drone. This model allows to sending of both linear and angular velocity commands to the drone as well as getting the ground truth position and orientation information of the drone model. This drone model is equipped with an RGBD camera in the front to detect people and their distance from the drone. It has a collision detection topic already which provides feedback when it hits somewhere.

4.5.1 Collaborative Approach for Drones

Search-and-rescue operations in collapsed buildings are enhanced in efficiency and effectiveness through the use of multiple drones. Such teams of drones would be able to survey larger areas faster, providing a more comprehensive search as opposed to single-drone operations. In the experiments, after the training phase of three different algorithms developed for the optimization of drone navigation and target detection, a new drone was added to the setting. This new drone joining the setup was then tasked to find the same target that the other deployed drones were searching for using the same model. The integrated effort solely aimed to test whether using more drones would speed up the target location speed and accuracy, hence reducing the time for its identification and rescue.

4.5.2 Search and Rescue Operation Workflow

When the drones start their flight, they maintain a maximum separation of 10 meters to ensure robust communication between them. If the drones come closer than 1 meter to each other, they are automatically pushed apart to avoid interference. Within the 1 to 10-meter range, the drones operate according to their trained algorithms. They communicate effectively by sending alerts to each other if one of them detects a target, allowing the other drones to cease their search. Custom environments consist of observation, action space, reset logic, reward function and actions for every step taken. The RL model node is to train the agent with different algorithms. The distance controller checks the distance between drones and adjusts their positions in dangerous situations like getting too close or too far from each other.

4.6 Simulation Environment

The custom environment is designed to create a simulation environment for autonomous drone navigation using reinforcement learning techniques. This environment combines OpenAI Gym, ROS 2, and Gazebo to facilitate the training

and evaluation of drone control algorithms. An illustration of the collapsed building environment is shown in Figure 4.9 taken from Viswanathan *et al.* (2023) and the same environment is shown with the drone agent and with the target in Figure 4.10.

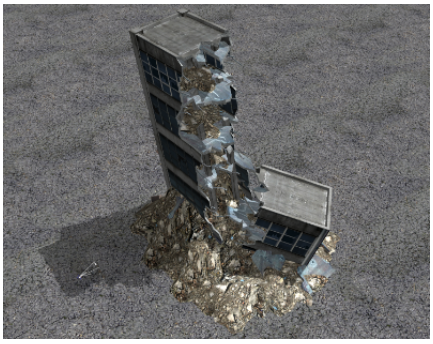


Figure 4.9: Collapsed building environment in Gazebo.

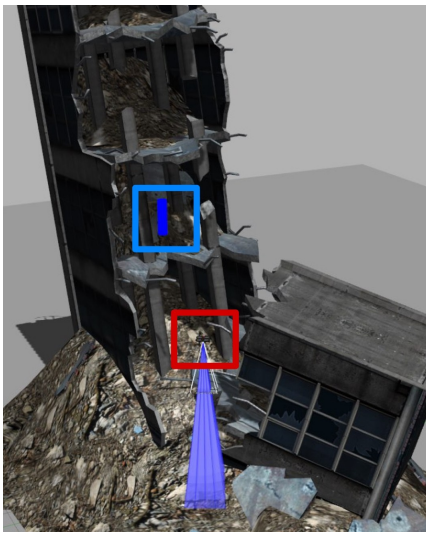


Figure 4.10: Collapsed building environment and the target inside it. The target is shown in blue square and the drone is shown in red square.

4.7 Training Loop

The agent is trained with reinforcement learning using the Stable Baselines3 library, and it is designed to train agents in a custom multi-drone environment. It begins by importing various algorithms from Stable Baselines3, including DDPG,

SAC and PPO. The code creates directories for saving trained models and logging data, using the current timestamp to ensure unique folder names. It then sets up a vectorized environment using the ‘MultiDroneEnv‘ class, configuring action noise to enhance exploration during training. The total training timesteps and save intervals are defined, although the model creation and training process is commented out. In this section, a DDPG model would be instantiated with specific parameters, and a loop would allow the model to learn and save periodically. Finally, the code loads a pre-trained DDPG model from a specified path, resets the environment, and enters an infinite loop to predict and execute actions based on the current observation, effectively running the trained model in the environment. The commented section for resetting the environment upon termination can be activated for proper episode management.

Chapter 5

Experiments and Results

5.1 Reward Function Design

5.1.1 Maze Scenario

Designing the reward function is very essential to obtain a well-trained agent. For that reason, the agent is trained with various reward functions. The main idea was to use negative rewards which are used to be more stable than positive rewards. Then, the actions were sorted from the most useful to the most dangerous for our agent. The first idea was to put only one green ball in a maze in Gazebo and the agent needed to collect these balls one by one. To achieve this the reward function is given in the equation below:

$$R = -1 + 2/(1 + e^{d/n_{gp}+1}) \tag{5.1}$$

where:

- d is the ground truth distance from the drone to the ball
- n_{gp} is the number of green pixels in the field of view of the drone camera

According to this reward function given in the 5.1, the agent got higher rewards when it saw more green pixels and got closer to the green ball. In this figure, while the x-axis shows the distance between the agent and the ball, the y-axis shows the reward.

The agent is trained with this reward function and obtains good training results which can find the green ball even if it is not in the field of view at the beginning. So it was starting to search for it by turning around. However, with this training, the agent was not able to find a green ball which is behind the walls. Also, the training took too long and it was not possible to try many different reward functions just for an easy task.

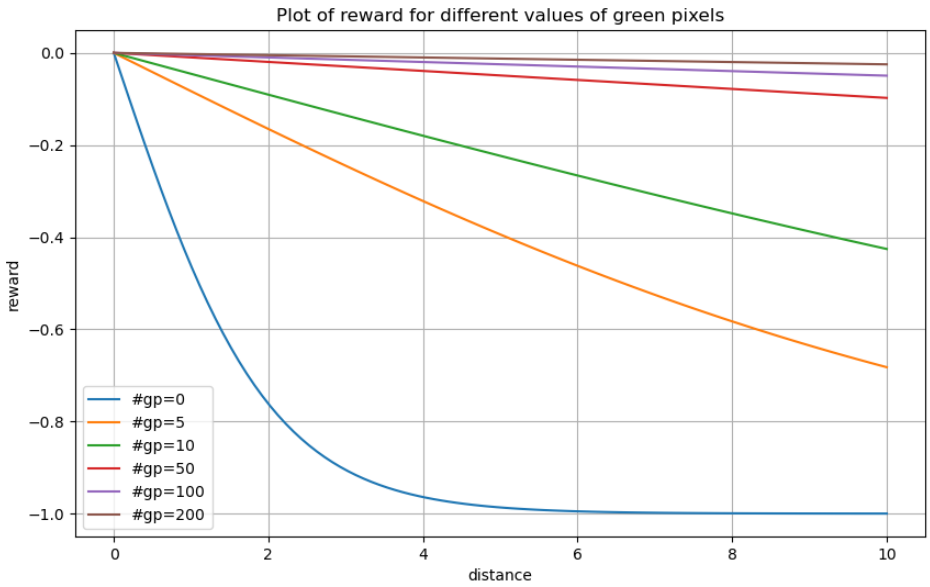


Figure 5.1: Reward curve for different values of the green pixels from 0 to 200

5.1.2 Earthquake Scenario

In this scenario, the observation space was reduced and that gave me the flexibility to change and try more reward functions. The main idea was again to use a negative reward function to ensure the stability of the training. The successful reward equation is given below:

$$R = -1 * dist + pen + coll \tag{5.2}$$

where:

- *dist* is the ground truth distance to the goal position
- *pen* is the penalty if the agent is more than 10 meters far from the origin which means it is also far from the building
- *coll* is the collision penalty when the closest distance is closer than 2 meters to the drone

5.2 Simulation Results

In order to simplify the problem, the agent is trained using only an RGB camera and the scenario was collecting the green ball in the environment. This can be

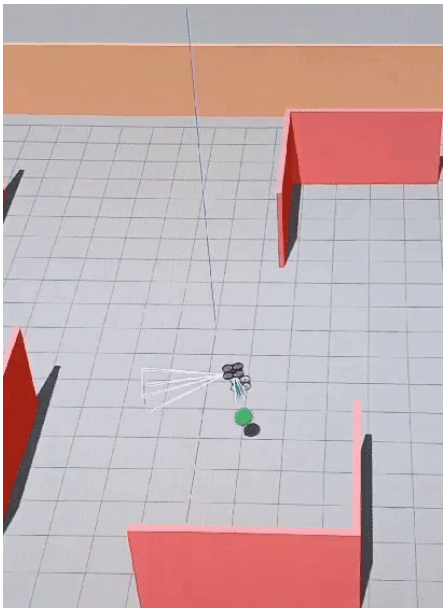


Figure 5.2

seen in the figure 5.2.

This purpose was achieved with the StableBaselines DDPG algorithm with a 10^{-5} learning rate. But it was only able to collect the green ball if it was in the field of view during the training which was not applied for a search and rescue scenario. After that, the idea was extended to train the agent using RGBD camera observation, which means 480*640*4 pixels in each step. While 480*640 is the height and the width of the frame, 4 contains the colour (RGB) and depth values of each pixel. The huge amount of information came into the camera in each step and the agent needed to reason which pixels were the most useful. Most of these pixel values change in each step. So, a compilation of the learning process took too much time even with a powerful graphic card and hard to do parameter tuning as it could converge in any timestep. For that reason, different types of reducing techniques were tried to reduce the learning time.

- **Grayscaling**: The first approach was grayscaling the observation space which reduces the space to 480*640*1. The first intention was that the agent could learn to differentiate shapes either and not need color channels. However, the agent needs to differentiate obstacles from the target according to their colors. Because this approach failed after training with different learning rates.

- **Reducing Resolution**: This approach contains training the agent with different resolutions via changing the camera resolution. However, this ap-

proach also did not work because the agent needed high resolution while moving.

- **Changing Neural Network Sizes**: Also adding extra layers and changing the network dimensions did not make any visible effects during the training phase.

As a result, the pixel values were extracted before putting them into the observation space. This reduced the observation space size drastically from 480*640*4 to 3.

5.2.1 Single Drone

The training of the drone agent is extensive while in a very challenging dynamic environment of a collapsed building, with several RL algorithms. This is a well-simulated case of a post-disaster scene: a non-homogenous structure with each floor presenting a different challenge in layout, obstacles, and navigation difficulties. While other environments may possess uniformly structured floors, the different floors in this building environment were designed to pose an increasing level of difficulty with the intent of making the agent change its behavior each time a new floor configuration is encountered. These included variations in the arrangement of debris, gaps between floors, broken walls, and other obstructions common in real-life collapses. It underscored the strength but, simultaneously, the flexibility of the RL approach regarding so many factors that cannot be forecasted.

In all of the training, the goal position that the drone had to fly toward was fixed and was located on the second floor of the building. With the initial height of the drone close to the second floor, this setup keeps the training consistent and focused on learning optimal paths under challenging circumstances, making sure it is starting in the vicinity of the target with enough complexity to test its navigation capabilities. The more restrictive starting condition provided the RL agent to learn and experience the fine details of vertical and horizontal movements within a confined 3D space. In this paper, three different RL algorithms were used in training the drone: Proximal Policy Optimization, Deep Deterministic Policy Gradient, and Soft Actor-Critic having different strengths concerning stability, exploration, and convergence speed. These algorithms were diverse, ensuring that the methods of reinforcement learning applied to drone-based search and rescue tasks in challenging environments were extensively evaluated.

Following the training phase was a protocol for rigorous testing that sought the performance of the trained agent on various floors of the collapsed building. These tests were designed to stress the learned policy of the drone by randomly varying both the initial starting height and the goal position. For example, although the

goal in the training was always positioned on the second floor, during the test, the positions of the goals were slightly shifted and the initial positions of the drone were randomized across different heights to consider real-world scenarios where exact initial conditions are often unknown or subject to changes. These changes were necessary to check the generalization capability for new situations of the learned policy and to assure that the agent didn't just memorize the training environment but, instead, knew generalizable strategies that could be used to navigate collapsed structures.

Each of these algorithms PPO, DDPG, and SAC is tested on the same set of 10 tests per floor to make sure the comparison among them is comprehensive yet fair. These floors are very different in terms of layout and complexity, and tests on each were conducted from different initial heights to simulate drones starting from multiple levels of elevation. The success criterion of each test was defined as an agent navigating within 3 meters to the target position without collision with obstacles within a time limit of 60 seconds. This metric was chosen based on the assumption that, in real-world search and rescue missions, proximity to the target, like a survivor or point of interest, in a reasonable amount of time with the safety of the drone's operation in fragile environments is very important. The time limit imposed by setting the timer to 60 seconds represented a balance between the need for rapid performance in emergencies and realism regarding the battery life of drones and risks of environmental exposure.

The results for these tests, as highlighted in Table 5.1, were very promising. For the average success rate across all tests conducted on all algorithms combined, it stood at about 84.4%. It means this high success rate that the trained agent smoothly ran in the collapsed building environment, reaching the target position and avoiding collisions in most conditions. If further breakdown is sought for individual algorithm performance, PPO has good stability and reliability, performing extremely well under higher random noise of an environment, while DDPG does the fastest convergence to an optimal path in a less complicated environment. Nevertheless, all three algorithms could demonstrate that they could do the job of coming up with successful agents for this challenging task.

5.2.2 Multi-Drone

The full-scale testing of the DDPG algorithm is conducted on a multi-drone scenario designed to simulate realistic and complex search-and-rescue missions within a collapsed building environment. For this experiment, independently navigating two drones are searching for one target across several floors of the structure. The aim was to see how well the drones could cooperate in efficiently scanning various parts of the building to maximize the chances of finding the target in the minimum amount of time. Each of the drones is initialized at

Table 5.1: Number of Successes in Different Floors for Different Algorithms

Algorithm / **Metric**	**Number of Successes for 10 Trials**		
	PPO	***DDPG***	***SAC***
First Floor	9	10	8
Second Floor	9	9	9
Third Floor	7	8	7

[a]Number of successes in different floors for each algorithm.

different heights so that during the search operations, each of them will focus on different floors. This difference in starting points was fundamental to the division of labour's development, as every drone focused its efforts on one floor until one of them found the target.

Another important feature in this experiment was the role played by the drone controller node in coordinating the movements of both drones during the search process. This controller constantly checked the relative position of each drone and sent corrections in the velocity for each to maintain an optimum distance between them. This was a very important function, ensuring that the drones did not collide with one another or stray too far apart to render the search operation inefficient. The controller hence assured that at all times the two drones flew between one and ten meters apart, a balance optimizing safety and area covered, by sending regular velocity commands. Such dynamic coordination by the controller was among the crucial elements for the overall success of this operation.

The scenario of Figure 5.3 showed the red and blue targets on each floor; red targets were placed on every floor.

During the first test sessions, both robots were programmed to search for only the red targets, since they are relatively easier to detect and approach. Each robot was assigned to a different floor: one to the first and one to the second floor. For example, when the red target was on the second floor, the setup allowed for parallel exploration where the second drone's task would be to find it and the first drone was responsible for scanning the floor below. The drones were, in that sense, independent yet collaborating on a common goal. These experiments were repeated on three different floors of a building 10 times each to check the consistency and performance of the drones under different conditions. The reason for such repetition in tests is essential for verifying whether the DDPG algorithm works reliably in performing complex tasks of searching in multi-drone environments.

Further to the red target search, there was another more challenging experiment concerning the search for blue targets. The blue targets, on the other hand,

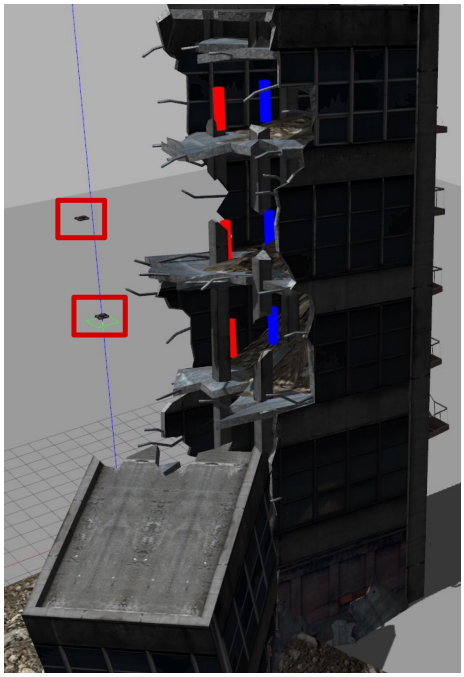


Figure 5.3: Collapsed building environment and different targets inside it. The red squares show the multi-drones in the environment.

were placed such that they should be a little difficult to find within the building. This has been done to test the limits of the drones in terms of their capability for navigation through complex environments and around obstacles. Success rates for locating the blue targets were understandably much lower than those realized during the search for the red target. This nonetheless allowed the collection of important information about the capabilities and limitations of the drones when handling environments with increasing difficulties. Success rates of both target searches are summarized and presented in Figure 5.4, showing the performance of the DDPG algorithm with varying levels of task difficulty.

In summary, this research presented a strong demonstration of the performance that DDPG can ensure in complex, real-world-inspired tasks with the requirement for precise coordination, autonomous decision-making, and efficient exploration patterns. That the drones could autonomously operate while maintaining a cohesive, coordinated search strategy in itself speaks volumes toward DDPG's potential for performance in real-world search-and-rescue missions. Environments like building collapses or cluttered spaces, which are particularly challenging, call for strong, reliable methods to enable multi-agent collaboration, effectively map large and hazardous areas while avoiding as much redundancy as possible, and realize maximum effectiveness.

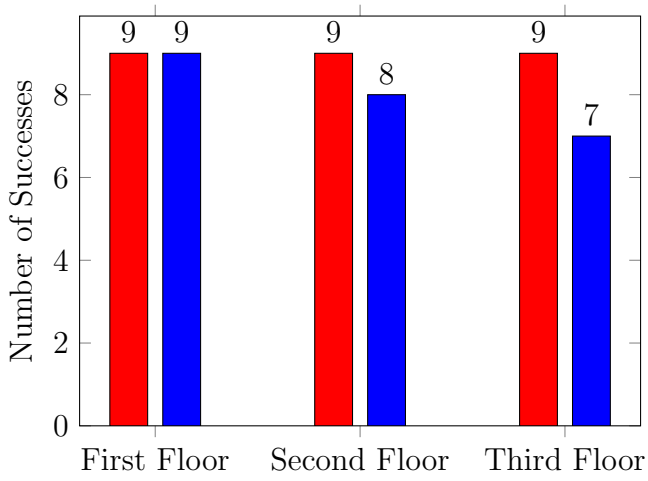


Figure 5.4: Reaching the target successfully on different floors for two different target positions.

The balance that DDPG strikes between exploration and exploitation makes it effective in view of the discussion above. Consider search and rescue, where agents must explore an unknown, probably hazardous environment while remembering to adapt their strategy to leverage learned paths and environmental clues. Mastery by DDPG of these competing goals is critical since it allows drones not only to efficiently travel around complicated spaces but also to find their targets under difficult conditions. This experiment underlines how DDPG allows each drone to perform autonomously for its search area while contributing to one general, coherent mission objective that becomes a critical success factor in environments where the coordination of multiple agents may provide the only difference between success and failure.

Results obtained with this multi-drone setting further support the scalability and suitability of DDPG for collaborative robotic systems. On one hand, DDPG allowed drones to apply fine-grained changes in movement based on continuous control-the result was stable trajectories with smooth navigation between obstacles and adaptation to dynamic scenarios on their own without human intervention. Furthermore, the efficiency of this algorithm in such settings is indicative of its role as a backbone toward further advancements in autonomous search and rescue systems, therefore providing a scalable and adaptive framework that may be easily extended to larger teams of agents.

A framework like this would be really useful in situations when human responders are at a premium or have limited access, while this solution is highly flexible and reliable, as well as able to efficiently cover areas with minimal overlap and waste.

Chapter 6

Discussion

6.1 Interpretation of Results

The agent was tested on its ability to locate targets across multiple floors of a collapsed building, achieving an 84.4% accuracy. Further, a multi-drone setup is evaluated, where two UAVs, equipped with the same trained model, searched different floors simultaneously. This reduced the search time by half. The multi-drone scenario is tested with the DDPG algorithm with an accuracy of 85%, presenting how much this approach can enhance search and rescue missions in difficult, post-earthquake environments.

6.2 Limitations

During the project, there were a lot of limitations such as version competencies, simulation constraints and computational limits which are explained in detail below:

6.2.1 Competence with the versions

Stable baselines3 library is the main part of this project that requires python3 installed in the computer. Since I would like to connect the agent to the Gazebo environment, a drone model which is compatible with ros2 humble is used with Ubuntu 22.04. The drone model is taken by NovoG93 (2022) drone repository. For faster convergence in such a complex problem, the computations need to be done in GPU. So, a computer with a GeForce RTX 4080 graphic card is used. And cuda is installed according to this graphic card.

6.2.2 Simulation Constraints

The gazebo simulation easily crashes when it reaches some boundaries when its publishing to the topics to the drone because it is not designed for reinforcement learning. Also, it highly depends on the wifi connection. So most of the time it does not receive the messages on time so it can cause errors such as multiple resetting of the episodes or not getting the correct drone positions.

6.2.3 Computational Limits

The initial idea was to give the image observation space to the agent. In that case, the agent was receiving 480*640*3 RGB pixel inputs every second which requires a very powerful computer and a very long training time. It took more than 24 hours to learn to find a red ball which is 2 meters away from the drone using RGB observations. And either Gazebo or directly Python were constantly crashing during the training. That is why I could not succeed with the image observation space directly and had to think about how to decrease the size of the observation space. The first attempts were grayscaling the image and lowering the resolution. But even after these reductions, the agent was not able to learn. So, I tried to tune the learning rate, batch size, episode length, and discount factor but could not find a solution. So, I decided to extract only the necessary information for the agent and directly reduce the observation to that size which is 3. The part I exracted was when the drone detects the target it calculates the distance and pixel numbers which show the middle point of the target. If the drone was not seeing the target the observation was given as zeros. This approach helped to reduce the training time to 8 hours for DDPG, and 3 hours for PPO and SAC.

Chapter 7

Conclusion

7.1 Summary of Findings

In this thesis, a UAV simulation framework is designed to enable a UAV agent to locate wounded individuals in a collapsed building. This framework uses three different DRL algorithms within the Gazebo simulator environment. The interaction between the simulator and the agent is facilitated through ROS 2, allowing for the transmission of velocity commands to the UAV and the retrieval of depth information from its camera. Additionally, the framework is tested with two UAV agents working collaboratively to search for target positions across various sections of the building, which enhances the efficiency of the search process. The experiments conducted illustrate that the proposed framework not only functions effectively but also improves the speed and accuracy of the search operations, showcasing its potential for practical applications in emergency response scenarios.

7.2 Contributions

This project has made significant contributions to the field of DRL across various dimensions, addressing complex challenges and exploring innovative methodologies. Below are the key contributions:

- **Observation Space**: One of the prominent novelties introduced by this work is the novel design of the observation space. Traditional approaches to DRL normally exploit raw state information provided directly by the agent's environment. For real-world applications, this raw state information may not be reliable at all times, since the state information might not be available or may not be exact at each instant in time. To this end, this project

removes this limitation because the agent in the project is using exclusively the image data perceived by the RGBD camera mounted on the flying drone. An agent, therefore, learns navigation and decision-making independently of any state information but based purely on the visual streams of information in an operationally more realistic emulation. In this way, the system can be made much more robust and flexible in dynamic environments. This is especially important in realistic applications where sensory input will typically be noisy or incomplete.

- **Utilization of Stable Baselines**: This work is the first project using the Stable Baselines library in a highly complex multi-drone environments domain that has received no works, up until recently, that have published their use of the framework. Stable Baselines provide a more organized and efficient implementation of multiple deep reinforcement learning algorithms, improving training stability and efficiency. Key features include action clipping and advanced replay buffer management methods that significantly speed up the learning and improve the general performance of agents. When implemented, these techniques provide faster convergence and reduce the large variances experienced in the DRL training process. This will set the precedent for further research to better exploit this strong library in other challenging environments.

- **Addressing Search and Rescue Challenges**: This project addresses a very critical and challenging problem, that of search and rescue, in an intrinsically complex and unpredictable domain. Moreover, the unstructured nature of the real environment introduces significant difficulties regarding navigation and target identification. This work has contributed a great deal to these aspects through the proposition of a new approach that marries DRL with the excellent abilities of aerial drones. This would, in the end, assist in coming up with better and more effective approaches toward search and rescue missions. Theoretically rich, this research also addresses the possible practical influence on emergency situations where time and accuracy form the basis of retrieval.

- **Navigating Unstructured Environments**: Most of the existing projects have successfully employed reinforcement learning in structured gridded or two-dimensional environments, but this project takes it a notch higher by addressing challenges in three-dimensional space. An unstructured three-dimensional space has immense complexities in navigation concerning obstacle avoidance, depth perception, and path planning in a conducive way. If such a space allows for successful development and training of a DRL agent, then this research offers an extension of the application domain of

DRL techniques and a stepping stone for future potential studies that would investigate similar challenges in three-dimensional contexts.

- **Advancements in Multi-Robot Systems**: The project also contributes to the development of multi-robot systems because it tests and validates the performance of two drones which may act in a coordinated way for the purpose of the search. That aspect of this research illustrates the possibility of collaborative strategies in robotic systems where multiple agents may work together to accomplish common objectives. The insights obtained with this multi-drone approach go beyond the advances in the state of the art on understanding cooperative behavior in robotic agents and give a basis for further advances in coordinated multi-agent systems. This research opens new perspectives toward efficiency in search operations, proving that collaboration from autonomous drones can significantly cut down times used in searches, raising the overall mission success rate.

7.3 Future Work

The present research work lays a good foundation in the area of DRL for drone navigation and SAR; however, several aspects of the project can be further improved and taken over for better performance, robustness, and applicability. Following are some proposed improvements and expansions:

- **Real-time Testing**: While this project was able to train a model on simulated environments, what matters is real-time tests that would help in the validation of its true performance in a real operation scenario. Real-time testing would mean deploying the trained drone in a controlled real-world environment where one could observe its behavior, navigational capabilities, and decision-making processes under dynamic conditions. Such tests would yield great insight into how the model would generalize to real-world challenges such as illuminations, unexpected obstacles, and the general unpredictability of the environment. Additional refinements of the model could be done with real-time data collection to create a feedback loop to improve learning and adaptiveness. Real-world validation is, therefore, needed, meaning that the gap between simulation and practical application must be bridged, hence assurance can be accorded that the drone will work in real emergency situations.

- **RGBD Observation Space Enhancement**: This might also be significantly enhanced by increasing the observation space to include the complete RGBD pixel data at each timestep for better navigation by the agent. When

the agent is provided with such rich visual information, it could make up its mind more effectively and progress better in complicated environments. However, doing so would bring challenges of their own as it relates to computational resources and processing time. Moreover, the RGBD camera would generate high-dimensional data that require significant computational resources. In the present framework, this may also lead to latency in decision-making. Optimizing the data processing pipeline with techniques such as dimensionality reduction or feature extraction would allow for a better balance between the richness of the observation space and the required computational efficiency for real-time applications. Data granularity and the speed at which it is processed are two ends of a scale that have to be weighed against one another to arrive at the optimal balance.

- **Incorporation of LSTM Layers**: Long short-term memory (LSTM) layers may bring huge benefits to the neural network architecture in terms of agent memory and learning from temporal sequences. The agent uses a replay buffer for training by storing and then subsequently sampling past experiences to break the correlation between consecutive experiences to stabilize training. This mechanism is not utilized in the testing phase. Adding the LSTM layers after the Actor and Critic networks enables the agent to learn from past observations effectively, enhancing its capability for making decisions based on historical context. LSTMs are useful when the current state might not encapsulate valuable information for decision-making, such as following moving targets or navigating areas with obstacles. This would allow the agent to build up an increasingly nuanced understanding of its environment over time and may lead to improved performance on complex tasks.

- **Exploration of Advanced Algorithms**: This also extends into future work on the implementation of more advanced or hybrid reinforcement learning algorithms that extend from the basic ones implemented in this work. Anyone reading this could try using PPO, SAC, or any meta-learning methods that might result in better learning efficiency and robustness. Many of these algorithms incorporate mechanisms for better exploration-exploitation trade-offs, which hopefully results in improved performance within complex and dynamic real-world environments. Further, applying multi-agent reinforcement learning strategies could potentially offer better coordination among multiple drones, hence enhancing collaborative capabilities in search and rescue missions.

- **Integration of Sensor Fusion Techniques**: There is another area of improvement incorporating sensor fusion, which integrates data besides the

RGBD camera. The agent updates the information from other sources, such as IMUs, GPS, or thermal cameras, for the overall understanding it has about its environment. With such a method, one might improve navigation accuracy, especially in conditions that are not very favourable for navigation, such as dark environments or places where obstacles block the view. It could also, by developing an effective sensor fusion framework, enhance the robustness of the system for real-world deployment in emergency scenarios.

- **Longer Training Duration and Fine-Tuning**: This would further improve with increased training time and hyperparameter tuning. The more varied the scenarios the agent has seen during its training, the more generalizable its performance to new, unseen environments. Furthermore, automated hyperparameter optimization may result in even better values of learning rate, batch size, and other critical parameters for the algorithm. The outcome of such systematic fine-tuning would likely be a much more robust and capable agent, able to adapt to many different operational contexts.

- **User Interaction and Feedback Mechanisms**: Finally, embedding mechanisms of users' interactions and feedback may yield a system that is more adaptable and effective. Allowing the operators to give feedback in real-time during actual missions may allow the agent to learn from these interactions, constantly improving its performance and decision-making capabilities. Such mechanisms could further advance human-robot collaboration toward a more synergistic approach in search and rescue operations where human insight would complement the autonomous capability of drones.

In the end, this work has indeed taken great strides in DRL for drone navigation but also has many exploratory paths left toward refinement and extensions for future studies. Addressing these items will continue to enhance the contribution of the research to the state of the art and further advance on the path toward more effective and assured solutions for search and rescue and other challenging operational settings.

Appendix A

Extra

A.0.1 Training Parameters

- PPO: Learning rate is 10^{-4}, the batch size is 128, the replay buffer size is 1000000, the discount factor is 0.99, and tau is 0.005.
- SAC: Learning rate is 10^{-4}, the batch size is 128, the replay buffer size is 1000000, the discount factor is 0.99, and tau is 0.005.
- DDPG: Learning rate is 10^{-4}, the batch size is 128 and normal action noise is used, its sigma parameter is 0.1, the replay buffer size is 1000000, the discount factor is 0.99, tau is 0.005.

A.0.2 Training Curves

The agent is trained with 3 algorithms PPO, SAC and DDPG. Figure A.1, A.2 and A.3 shows the learning curves. The fastest convergence is achieved with SAC and the slowest with DDPG.

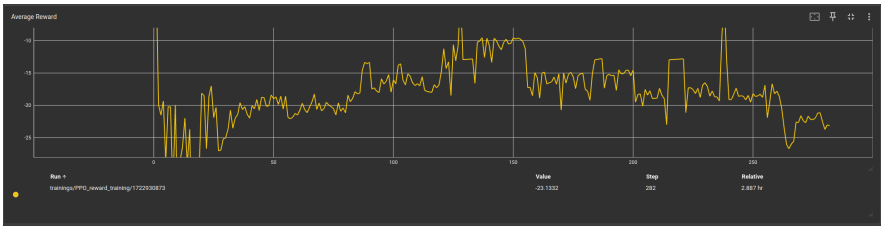


Figure A.1

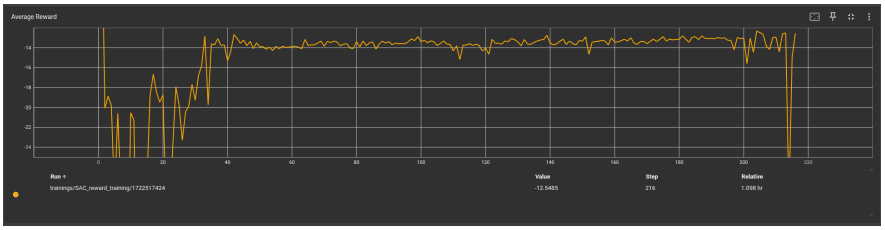


Figure A.2

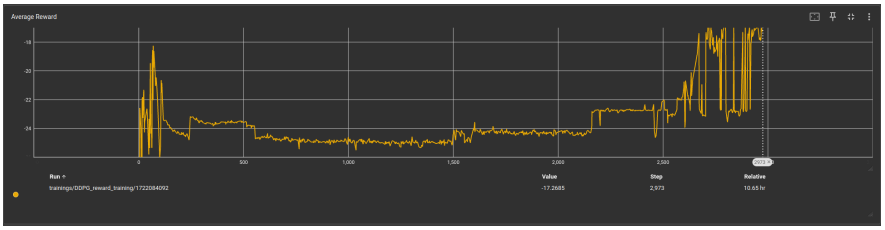


Figure A.3

A.0.3 PPO Algorithm

PPO is an on-policy algorithm that uses a clipped objective to improve stability and performance. It is designed to prevent large policy updates, balancing exploration and exploitation.

Algorithm 5 PPO Algorithm

Initialize policy parameters θ_0, value function parameters ϕ_0
for each iteration **do**
    Collect set of trajectories $\mathcal{D}$ by running policy π_{θ_k} in the environment
    for each trajectory **do**
        Compute advantage estimates $\hat{A}_t$
    end for
    for epoch $1, 2, \ldots, K$ **do**
        Update the policy by maximizing the PPO objective:

$$\mathbb{E}\left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right]$$

        Update the value function using the mean squared error loss:

$$L(\phi) = \frac{1}{T}\sum_t \left(V_\phi(s_t) - \hat{V}_t\right)^2$$

    end for
end for

A.0.4 SAC Algorithm

SAC is an off-policy actor-critic algorithm that aims to maximize both the expected reward and the entropy of the policy, encouraging exploration.

Algorithm 6 SAC Algorithm

Initialize critic networks Q_{ϕ_1}, Q_{ϕ_2} with random parameters ϕ_1, ϕ_2
Initialize actor network π_θ with random parameters θ
Initialize target critic networks $\phi_1^{\text{target}}, \phi_2^{\text{target}}$
Initialize replay buffer $\mathcal{D}$
for each iteration **do**
    for each environment step **do**
        Sample action $a_t \sim \pi_\theta(a_t|s_t)$
        Observe reward r_t and next state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
    end for
    for each gradient step **do**
        Sample a batch of transitions (s_t, a_t, r_t, s_{t+1}) from $\mathcal{D}$
        Compute target value $y = r_t + \gamma \min(Q_{\phi_1^{\text{target}}}(s_{t+1}, a_{t+1}), Q_{\phi_2^{\text{target}}}(s_{t+1}, a_{t+1})) - \alpha \log \pi_\theta(a_{t+1}|s_{t+1})$
        Update critics by minimizing loss:

$$L(\phi) = \frac{1}{|\mathcal{D}|} \sum \left(Q_{\phi_i}(s_t, a_t) - y\right)^2$$

        Update actor by minimizing the loss:

$$L(\theta) = \frac{1}{|\mathcal{D}|} \sum \left(\alpha \log \pi_\theta(a_t|s_t) - Q_{\phi_1}(s_t, a_t)\right)$$

        Adjust temperature α if needed.
    end for
    Update target networks:

$$\phi_i^{\text{target}} \leftarrow \tau\phi_i + (1-\tau)\phi_i^{\text{target}}$$

end for

A.0.5 DDPG Algorithm

DDPG is an off-policy, model-free algorithm that learns deterministic target policies in continuous action spaces.

Algorithm 7 DDPG Algorithm

Initialize critic network Q_ϕ and actor network μ_θ with random parameters
Initialize target networks $Q_{\phi^{\text{target}}}, \mu_{\theta^{\text{target}}}$ with $\phi^{\text{target}} \leftarrow \phi$, $\theta^{\text{target}} \leftarrow \theta$
Initialize replay buffer $\mathcal{D}$
for each episode **do**
    Initialize random process $\mathcal{N}$ for exploration noise
    Receive initial state s_0
    for each step in the episode **do**
        Select action $a_t = \mu_\theta(s_t) + \mathcal{N}_t$
        Execute action a_t and observe reward r_t and next state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
        Sample a random batch of transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{D}$
        Compute target for the critic:

$$y_i = r_i + \gamma Q_{\phi^{\text{target}}}(s_{i+1}, \mu_{\theta^{\text{target}}}(s_{i+1}))$$

        Update critic by minimizing the loss:

$$L(\phi) = \frac{1}{|\mathcal{D}|} \sum (Q_\phi(s_i, a_i) - y_i)^2$$

        Update the actor using the sampled policy gradient:

$$\nabla_\theta J(\theta) = \frac{1}{|\mathcal{D}|} \sum \nabla_a Q_\phi(s, a)|_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s)$$

        Update target networks:

$$\theta^{\text{target}} \leftarrow \tau\theta + (1-\tau)\theta^{\text{target}}, \quad \phi^{\text{target}} \leftarrow \tau\phi + (1-\tau)\phi^{\text{target}}$$

    end for
end for

References

Bodi, M.A., Liu, Z., Jiang, F., Zhao, W., Dang, Q., Wang, X., Zhang, J. & Wang, L. (2023). Reinforcement learning based uav formation control in gps-denied environment. *Chinese Journal of Aeronautics*, **36**, 281–296. 6

Bouhamed, O., Ghazzai, H., Besbes, H. & Massoud, Y. (2020). Autonomous uav navigation: A ddpg-based deep reinforcement learning approach. In *Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5, Seville, Spain. 7

HuggingFace (2021a). Bellman equation. Accessed: 16-Sep-2024. 17

HuggingFace (2021b). Introduction to ppo. Accessed: 16-Sep-2024. 24

HuggingFace (2021c). Monte carlo vs td learning. Accessed: 16-Sep-2024. 17

HuggingFace (2021d). Q-learning. Accessed: 16-Sep-2024. 22

Kang, K., Belkhale, S., Kahn, G., Abbeel, P. & Levine, S. (2019). Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. In *Proceedings of the 2019 IEEE International Conference on Robotics and Automation (ICRA)*, 6008–6014, Montreal, QC, Canada. 6

Keras (2018). Ddpg on pendulum. Accessed: 16-Sep-2024. 25

Kim, I., Shin, S., Wu, J., Kim, S.D. & Kim, C.G. (2017). Obstacle avoidance path planning for uav using reinforcement learning under simulated environment. In *Proceedings of IASER 3rd International Conference on Electronics, Electrical Engineering, and Computer Science*, 34–36, Sapporo, Japan. 7

Lillicrap, T.P. *et al.* (2015). Continuous control with deep reinforcement learning. *arXiv.org*, available: https://arxiv.org/abs/1509.02971. 7

Ma, Z. & Chen, J. (2022). Adaptive path planning method for uavs in complex environments. *International Journal of Applied Earth Observation and Geoinformation*, **115**, 103133. 7

NovoG93 (2022). Sjtu drone. Accessed: 16-Sep-2024. 11, 43

Openai (2017). Openai baselines: Ppo. Accessed: 16-Sep-2024. 24

Openai (2018a). Deep deterministic policy gradient (ddpg) - spinning up. Accessed: 16-Sep-2024. 24

Openai (2018b). Reinforcement learning: An introduction. Accessed: 2024-09-8. 19

Openai (2018c). Soft actor-critic (sac) - spinning up. Accessed: 16-Sep-2024. 25

Peng, J., Lv, B., Zhang, L., Lei, L. & Song, X. (2023). An improved ddpg algorithm for uav navigation in large-scale complex environments. In *Proceedings of the 2023 IEEE Aerospace Conference*, 1–11, Big Sky, MT, USA. 7

Pham, H.X., La, H.M., Feil-Seifer, D. & Nguyen, L.V. (2018). Reinforcement learning for autonomous uav navigation using function approximation. In *Proceedings of the 2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 1–6, Philadelphia, PA, USA. 7

Ramezani, M. & Atashgah, M.A.A. (2024). Energy-aware hierarchical reinforcement learning based on the predictive energy consumption algorithm for search and rescue aerial robots in unknown environments. *Drones*, **8**, 283. 6

Reuters (2023). How search and rescue teams pull survivors from rubble. *Reuters*, available: https://www.reuters.com/graphics/EARTHQUAKE-RESCUE/mopajqojmva. [Accessed: Aug. 1, 2024]. 2

Robotics, I.S. (2023). Intelligent service robotics. *Intelligent Service Robotics*, **16**, 415–430. 6

Sambolek, S. & Ivasic-Kos, M. (2021). Automatic person detection in search and rescue operations using deep cnn detectors. *IEEE Access*, **9**, 37905–37922. 6

Souissi, O., Benatitallah, R., Duvivier, D., Artiba, A., Belanger, N. & Feyzeau, P. (2013). Path planning: A 2013 survey. In *Proceedings of the 2013 International Conference on Industrial Engineering and Systems Management (IESM)*, 1–8, Agdal, Morocco. 6

Viswanathan, V., Satpute, S. & Nikolakopoulos, G. (2023). Flie: First-look enabled inspect-explore autonomy toward visual inspection of unknown distributed and discontinuous structures. *IEEE Access*, **PP**, 1–1. 33

Wang, H., Zhang, C., Song, Y., Pang, B. & Zhang, G. (2020). Three-dimensional reconstruction based on visual slam of mobile robot in search and rescue disaster scenarios. *Robotica*, **38**, 350–373. 6

Wikipedia (2024). Reinforcement learning. Accessed: 2024-09-8. 19

Wu, L., Wang, C., Zhang, P. & Wei, C. (2022). Deep reinforcement learning with corrective feedback for autonomous uav landing on a mobile platform. *Drones*, **6**. 7

Yan, C. & Xiang, X. (2018). A path planning algorithm for uav based on improved q-learning. In *Proceedings of the 2018 International Conference on Robotics Automation and Science (ICRAS)*, 1–5, Wuhan, China. 7

Zhou, S., Li, B., Ding, C., Lu, L. & Ding, C. (2020). An efficient deep reinforcement learning framework for uavs. In *Proceedings of the 2020 21st International Symposium on Quality Electronic Design (ISQED)*, 323–328, Santa Clara, CA, USA. 7

Zuluaga, J.G.C., Leidig, J.P., Trefftz, C. & Wolffe, G. (2018). Deep reinforcement learning for autonomous search and rescue. In *NAECON 2018 - IEEE National Aerospace and Electronics Conference*, 521–524, Dayton, OH, USA. 7