Reinforcement Learning Techniques for Navigation and Exploration using Drones in Indoor Environments

Ritwik Sharma
2022A3PS0470G
B.E. EEE

Vimarsh Shah
2022B5A71060G
MSc. Phy + B.E. CS

2024

AITG Group at CSIR CEERI, Pilani
A Practice School-I Station
of
BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

Station: Central Electronics Engineering Research Institute (CSIR–CEERI), Pilani
Centre: AITG
Duration: 45 days

Date of Start: 28th May, 2024
Date of Submission: 21st July, 2024
Title of the Project: Reinforcement Learning Techniques for Navigation and Exploration using Drones in Indoor Environments

Name(s) and designation(s) of the expert(s):
Dr. Kaushal Kishore
Senior Scientist at CSIR CEERI, Pilani

Name of the PS Faculty: Prof. Meetha Shenoy

Key Words: Indoor drone navigation, Reinforcement Learning, Exploration techniques, DDPG, ICM, ROS, Gazebo
Project Areas: Reinforcement Learning Techniques for Navigation and Exploration in indoor Environment with drones

Abstract

This report presents a comprehensive study on the application of Reinforcement Learning (RL) techniques for enhancing navigation and exploration capabilities of drones in indoor constrained environments. Autonomous drones are increasingly utilized for various applications, including surveillance, inspection, and mapping, where their ability to navigate and explore complex indoor spaces is crucial. The focus of this report is on leveraging state-of-the-art RL methods to improve drone performance in these challenging settings.

The study begins with an overview of preliminary work related to drone navigation tasks, including algorithmic flows and hardware implementation specifics. A literature review of agile high-speed drone control using imitation learning is presented, highlighting comparisons of various algorithms. The report then introduces 3D LiDAR camera sensor fusion techniques, crucial for accurate environment perception.

Further sections explore indoor drone navigation challenges, propose solutions including a multilayered DDPG architecture and hybrid rewards structure, and discuss simulation setups and results. The investigation into different RL techniques for navigation and exploration is detailed, covering simulation environments, sensor setups, reward strategies, and tested algorithms. The impact of intrinsic rewards on performance is also analyzed, with a focus on the Intrinsic Curiosity Module (ICM) and its integration with Proximal Policy Optimization (PPO).

Through this detailed examination, the report aims to contribute to the development of more effective and adaptable autonomous drone systems, capable of efficiently navigating and exploring constrained indoor environments.

Contents

Chapter 1

Introduction

The field of autonomous robotics has made remarkable strides in recent years, with drones emerging as a key technology due to their versatility and applicability across various domains. Among the many advancements, Reinforcement Learning (RL) has shown particular promise in enhancing the capabilities of drones, especially in the realms of navigation and exploration. RL is a type of machine learning where an agent learns to make decisions by interacting with its environment and receiving feedback in the form of rewards or penalties. This approach allows drones to develop adaptive and intelligent behaviours by optimizing their decision-making policies based on real-time interactions with their surroundings. Unlike traditional methods that rely on pre-defined heuristics, RL provides a robust framework for handling dynamic and uncertain environments.

Indoor constrained environments present unique challenges for drone navigation and exploration, characterized by limited space, complex layouts, and various obstacles. These constraints necessitate sophisticated algorithms capable of precise control and real-time adaptation. Drones operating in such settings must navigate around obstacles, avoid collisions, and effectively map out their surroundings despite the restricted space and potential interference from indoor conditions. By applying state-of-the-art RL techniques, this report aims to address these challenges and enhance the performance of drones in indoor environments. Through advanced RL methods, we seek to develop more capable autonomous systems that can efficiently navigate and explore constrained spaces, paving the way for new applications and improvements in indoor robotic systems.

Chapter 2

Preliminary Work - Drone Navigation Task for Leachate Site Detection and Inspection

Leachate site detection in dumps and landfills is critically important due to the environmental and public health risks posed by uncontrolled leachate. Leachate, a liquid that forms when water percolates through waste materials, can contain hazardous substances like heavy metals, organic pollutants, and pathogens. If not properly managed, leachate can contaminate soil, groundwater, and surface water, posing significant threats to ecosystems and human health. Effective detection and management of leachate sites are essential to prevent these pollutants from spreading and to ensure compliance with environmental regulations.

Drones offer a highly effective solution for leachate site detection. Equipped with advanced sensors, including thermal cameras, multispectral imaging, and LiDAR, drones can quickly and accurately identify areas of concern. Thermal cameras can detect temperature variations indicating leachate outbreaks, while multispectral imaging can reveal changes in vegetation health that might signal contamination. LiDAR can map the topography and identify potential flow paths for leachate. The use of drones for this purpose is not only more efficient but also safer than traditional methods. Drones can cover large areas quickly and access difficult or hazardous locations without putting human surveyors at risk. Additionally, the data collected by drones can be processed and analyzed rapidly, allowing for timely interventions to mitigate environmental damage. Overall, drones enhance the precision, safety, and speed of leachate site detection, contributing significantly to effective landfill management and environmental protection.

2.1 Task Description

The task assigned to us comprised of the following steps:

1. Fly the drone to a GPS waypoint
2. Switch to offboard mode
3. Start the camera and initiate search for leachate site
4. If a site is detected, draw a bounding box and find its centroid

5. Make the drone hover at a certain predefined altitude

6. Maneuver the drone to center itself over this centroid

Figure 2.1: Leachate Site

The task was to be executed in simulation, since the conditions outside were not favorable for flying. Also, simulations provide a good way of testing out an algorithm before deploying it in the real world. The platform used for simulation was ROS 1 (Noetic) and Gazebo simulator. The PX4 package was used to simulate the drone in Gazebo, while the popular mavros protocol was used for communication.

In the end goal, the detection of the leachate site will be done through a trained ML model. Development of such a model was not of significant importance to us right now. Instead, that part of the code was replaced with detection of an arbitrary color, in our case RED for the simulation and an ML model was used during hardware testing.

2.2 Algorithmic Flow

1. The code detects if the GPS waypoint is reached and switches to offboard mode.

2. Detection of the red colored object starts using feed from an RGB camera. The RGB stream is converted into HSV (Hue, Saturation, Value) and set the threshold to segment the red color.

3. Once a red object is detected, a bounding box is drawn around it and the centroid of this bounding box is calculated and stored.

4. Control inputs for roll, pitch, and yaw are calculated using a simple proportional controller which was tuned by intuition.

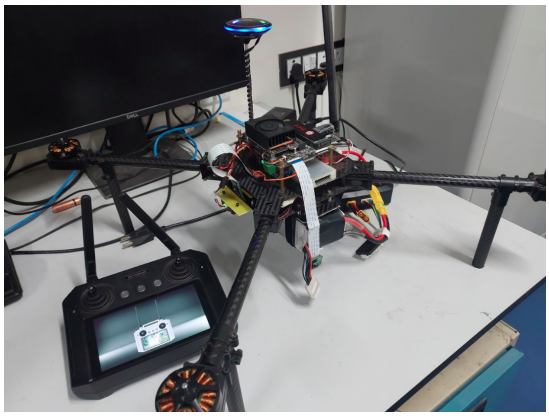
5. Commands are sent to the drone in terms of velocity setpoints to hover at the setpoint height and center itself over the red object, so the centroid is right at the center of the camera frame.

2.3 Hardware Implementation

Our next goal was to do Hardware implementation for the same. For the hardware implementation, we utilized the YOLOv8m-seg model, which contains 27.3 million parameters, to conduct end-to-end testing on the onboard computer (Nvidia Jetson Orin nano). The primary objective was to detect a person from aerial imagery and maintain the drone at a safe altitude (of 5 meters) above the detected individual. This process necessitated the fine-tuning of the proportional-integral (PI) controller to achieve enhanced stability and precise control, ensuring the drone could reliably centre itself above the target completely autonomously.

The sequence commenced with mission setup, progressing through navigation to specified waypoints, and with the final goal being precise alignment over the target. This was accomplished by converting the YOLOv8m-seg model to TensorRT to enable faster inference. Furthermore, we employed the gstreamer pipeline for accelerated GPU-based image capture, thereby streamlining the image processing workflow. Integration with the flight controller was achieved via mavros.

Extensive testing of the entire sequence revealed initial challenges, which were systematically addressed. The iterative process of fine-tuning and validation resulted in a robust system that performed flawlessly. This comprehensive approach ensured that the drone could autonomously navigate to waypoints and accurately position itself above the target, demonstrating the effectiveness and reliability of the implemented solution.



a. Drone equipped with CSI camera, 1D Time of Flight sensor

b. Drone equipped with gimble stabilized camera, 2D LiDAR

Figure 2.2: Both Drones use a CUAV-5 flight controller along with a Nvidia Jetson Orin Nano on board computer

Chapter 3

Literature Review of Agile High-Speed Drone Control Using Imitation Learning

A study was conducted on 'learning high speed in the wild' for aerial vehicles. This research paper aims to address the limitations of current autonomous quadrotor navigation systems, particularly in high-speed and complex environments. While quadrotors are known for their agility and ability to traverse intricate terrains, their autonomous operation has been hindered by the traditional method of breaking down navigation into sequential subtasks: sensing, mapping, and planning. This method, although effective at low speeds, results in increased processing latency and error accumulation, making it less suitable for high-speed navigation in cluttered environments. [1]

The paper proposes a novel end-to-end approach that enables quadrotors to autonomously navigate through complex natural and human-made environments at high speeds using purely onboard sensing and computation. The core principle of this approach is the direct mapping of noisy sensory observations to collision-free trajectories in a receding-horizon manner. This method significantly reduces processing latency and enhances robustness to noisy and incomplete perception. A convolutional network, trained exclusively in simulation via privileged learning—imitating an expert with access to privileged information—performs the sensorimotor mapping. By incorporating realistic sensor noise into the simulation, the approach achieves zero-shot transfer to real-world environments that were not encountered during training, such as dense forests, snow-covered terrain, derailed trains, and collapsed buildings.

This research demonstrates that end-to-end policies trained in simulation can outperform traditional obstacle avoidance methods in high-speed autonomous flight through challenging environments.

3.1 Comparison of Algorithms

The paper performed a systematic comparison of its proposed novel and other popular algorithms like blind trajectory following, reactive methods, and global planning-based methods. This is summarized in the table below:

The proposed novel architecture performed better than its competitors when provided ground truth sensor data, especially at high speeds. What however surprised me the most

Method	Components	μ [ms]	σ [ms]	Perc. [%]	Total Proc. Latency [ms]
FastPlanner [18]	Pre-processing	14.6	2.3	22.3	65.2
	Mapping	49.2	8.7	75.5	
	Planning	1.4	1.6	2.2	
Reactive [32]	Pre-processing	13.8	1.3	72.3	19.1
	Planning	5.3	0.9	27.7	
Ours	Pre-processing	0.1	0.04	3.9	10.3 (2.6*)
	NN inference	10.1 (2.4*)	1.5 (0.8*)	93.0	
	Projection	0.08	0.01	3.1	
Ours (Onboard)	Pre-processing	0.2	0.1	0.4	41.6
	NN inference	38.9	4.5	93.6	
	Projection	2.5	1.5	6.0	

Figure 3.1: Comparison [1]

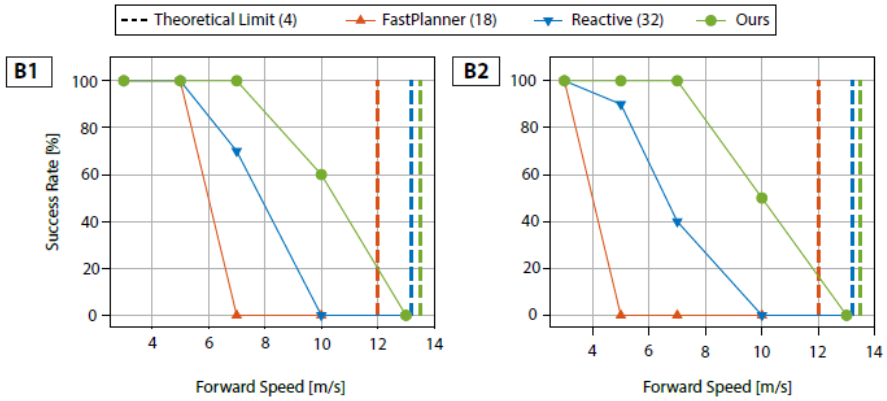


Figure 3.2: B1 - Ground truth, B2 - Noisy data [1]

was how much better it did than its counterparts when provided with noisy sensor input data.

To conclude, the review of this paper proved to be extremely intriguing and fruitful. It provided us with a fascinating research topic and can serve as a gateway into several new and interesting problems. An example could be 'high speed flying in constrained environments with moving obstacles'. We hope to work further on this if possible in our tenure here.

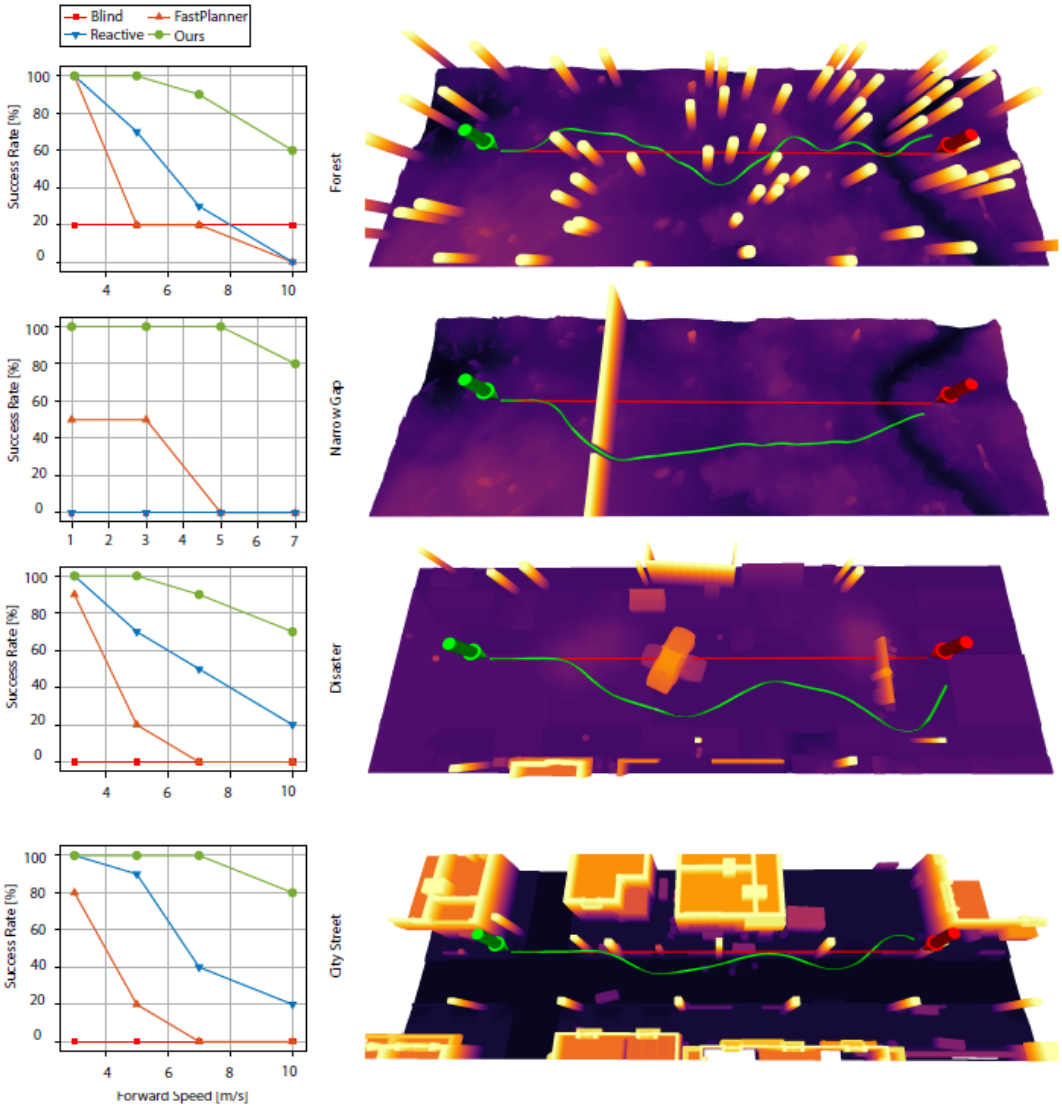


Figure 3.3: Simulations in a variety of environments [1]

Chapter 4

Introduction to 3D LiDAR Camera Sensor Fusion Techniques

Literature review on R3Live [2]

4.1 Abstract

A specific LiDAR-Inertial-Visual sensor fusion framework was studied which can provide tightly-coupled state estimation and mapping in real-world environments. This research paper introduces R3LIVE, a novel LiDAR-Inertial-Visual sensor fusion framework designed to enhance robust and accurate state estimation by integrating measurements from LiDAR, inertial, and visual sensors. R3LIVE consists of two subsystems: LiDAR-Inertial Odometry (LIO) and Visual-Inertial Odometry (VIO). The LIO subsystem, known as FAST-LIO, utilizes LiDAR and inertial sensor data to construct the geometric structure of global maps, determining the position of 3D points. The VIO subsystem processes visual-inertial sensor data to render the map's texture, or the color of 3D points, by minimizing frame-to-map photometric error. Built upon the previous work R2LIVE, R3LIVE features a carefully designed and implemented architecture. Experimental results demonstrate that R3LIVE outperforms existing systems in robustness and accuracy of state estimation.

4.2 Introduction

R3LIVE is a versatile, well-engineered system applicable to a range of uses, including serving as a SLAM system for real-time robotic applications and reconstructing dense, precise, RGB-colored 3D maps for surveying and mapping. To enhance extensibility, the authors developed offline utilities for mesh reconstruction and texturing, bridging the gap between R3LIVE and various 3D applications like simulators and video games. To contribute to the community, R3LIVE has been open-sourced on GitHub, providing access to all codes, software utilities, and mechanical designs.

4.3 Methodology

We plan to put together a similar setup on the same drone which we used for the leachate detection project. The setup will consist of a 3D LiDAR (Realsense L515, Velodyne Puck,

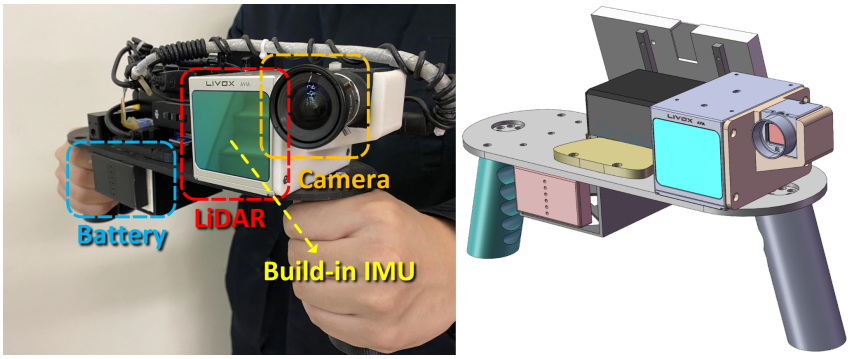


Figure 4.1: R3LIVE handheld setup

Livox, etc.) and an RGB camera. The next step would be to calibrate the two sensors by first storing point cloud data from the LiDAR and an image from the RGB camera and then running the calibration tool provided by the researchers.

After that, we should be able to effectively and accurately map real-world environments with high precision in real-time. This will give the drone a sense of depth perception which can be very useful to implement complex navigation algorithms like the one discussed in the previous section.

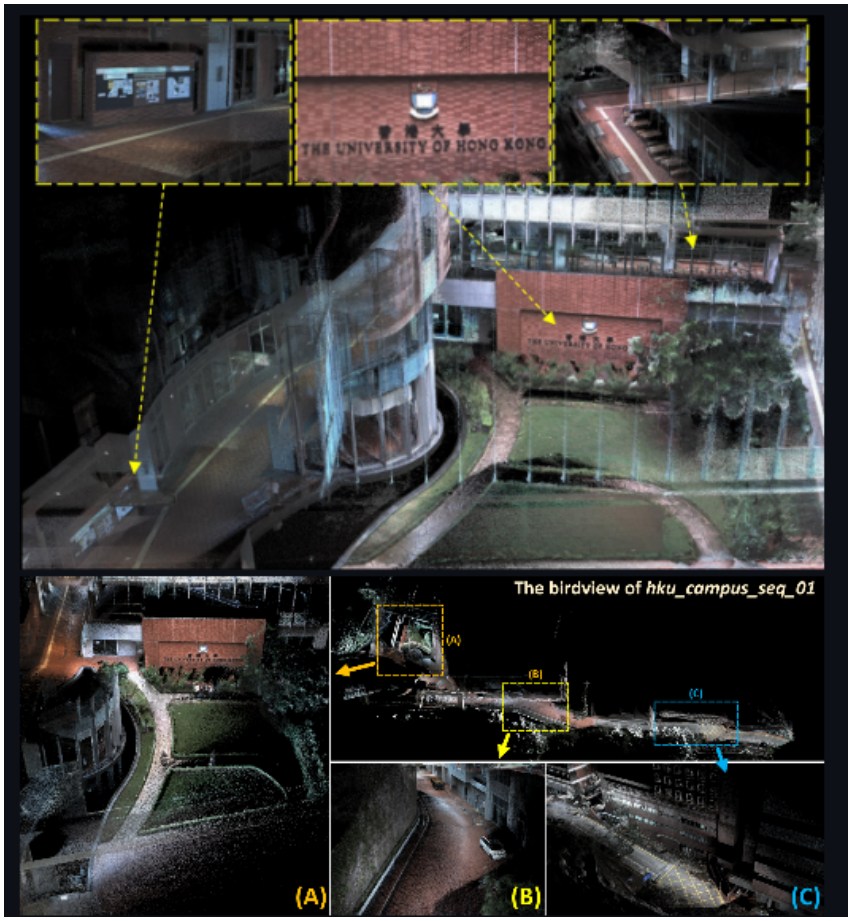


Figure 4.2: Mapping the campus of HKU in real time with R3LIVE

Chapter 5

Indoor Drone Navigation

Implementation: **Exploring Unknown Indoors with Drone using Multilayered DDPG Architecture**

Our original work was focussed on using a novel Multilayered DDPG architecture for exploration of unknown indoor environments. The work which we continued introduces a novel algorithm based on Deep Deterministic Policy Gradient (DDPG) for autonomous indoor drone navigation in confined and unmapped environments. The proposed approach employs a multi-layered DDPG architecture, enabling the drone to navigate, adapt, and respond to its surroundings effectively. The reward structure addresses three key aspects of drone navigation control: 1) obstacle avoidance using artificial repulsive potential hills, 2) centre alignment via artificial attraction valleys, and 3) variable pitch control for accelerated exploration. This multi-control strategy grants the drone a higher degree of autonomy for effective exploration of unmapped environments. Our goal was to test this algorithm in a simulation environment to validate its effectiveness.

5.1 Introduction

With the rapid developments in drones in the last few years, a variety of use cases have been developed. These range from delivery of important supplies to surveillance as well as monitoring and automated reporting for any potential hazards in the desired area. Drones have become the staple for inspections in challenging terrains where it is almost impossible for humans to traverse through. As the usage of drones in various industries keeps on increasing, especially now in dense areas with humans - it has become more important to ensure safe and efficient navigation in these spaces.

Along with all the outdoor use cases of the drones for monitoring and mapping of terrains, recently a lot of people have started using it in indoor spaces for urban delivery, regular monitoring in indoor factory floors and such. This has also increased with the cost of good quality sensors (LiDAR and stereo cameras) going down in recent years.

Path planning algorithms like A* and Artificial Potential Fields (APFs) are very efficient for navigation in known areas. The challenge arises in navigating and exploring unknown regions. In rescue mission scenarios in tight, unexplored territories, these algorithms, along with frontier-based exploration techniques, often fail due to the complexity and absence of a predetermined map. This additional complexity has led to significant recent work on various deep learning techniques.

Deep Deterministic Policy Gradient (DDPG) is a reinforcement learning algorithm that combines the benefits of Q-learning and policy gradient methods. It is particularly suited for continuous action spaces, making it ideal for drone navigation where precise control is required. DDPG uses a deterministic policy, which allows for efficient exploration and exploitation in complex environments. By leveraging a multi-layered DDPG architecture with a unique reward structure, drones can effectively navigate, adapt, and respond to their surroundings. This makes DDPG a robust solution for the challenges associated with autonomous indoor drone navigation in unmapped and confined environments.

The architecture controls the following three degrees of freedom: roll, pitch and yaw. This enables seamless manoeuvres in confined indoor environments. The work proposes special reward functions, memory buffer and a novel multilayered architecture.

5.2 Proposed Solution

5.2.1 Multilayered DDPG Architecture

About Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is an algorithm that merges the principles of Q-learning with policy gradient methods, specifically designed for environments with continuous action spaces. It leverages an actor-critic architecture where the actor determines the action to take given the current state, and the critic evaluates the action by computing a value function. [3]

Actor and Critic Networks

In DDPG, the actor network $\mu(s|\theta^\mu)$ parameterized by θ^μ outputs a deterministic action for a given state s. The critic network $Q(s, a|\theta^Q)$ parameterized by θ^Q evaluates this action by estimating the expected return.

Temporal Difference Learning

The Q-value is updated using the temporal difference (TD) error. The TD target y is computed as follows:

$$y = r + \gamma Q'(s', \mu'(s'|\theta^{\mu'})|\theta^{Q'}) \tag{5.1}$$

where r is the reward, γ is the discount factor, s' is the next state, Q' and μ' are the target critic and actor networks respectively.

The critic network is updated by minimizing the loss function:

$$\mathcal{L}(\theta^Q) = E\left[\left(y - Q(s, a|\theta^Q)\right)^2\right] \tag{5.2}$$

Replay Buffer

To stabilize training, a replay buffer $\mathcal{D}$ is used to store transitions (s, a, r, s'). Mini-batches of experiences are sampled from the replay buffer to break the correlation between consecutive transitions, leading to more stable and efficient learning.

Policy Update

The actor network is updated by using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx E_{s\sim\mathcal{D}} \left[\nabla_a Q(s, a|\theta^Q) \Big|_{a=\mu(s|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu) \right] \tag{5.3}$$

This updates the actor network to maximize the expected return by following the policy gradient.

Multi-Actor System

In a multi-actor system, multiple actors are responsible for generating potential actions based on the current state of the environment. Each actor learns a policy that aims to maximize the expected reward. However, in this work, a single actor system is employed, where one actor is used to generate the actions for the next state.

Multi-Critic System

A multi-critic system comprises several critic networks that estimate the quality of actions proposed by the actor. Each critic provides a different perspective on the action's effectiveness, allowing for a more robust evaluation. The loss function for the critic network is defined as follows:

$$\mathcal{L}(J_t) = \sum_{i=1}^{N} \sum_{j=1}^{M} -\frac{Q_{i,j}}{3k} \tag{5.4}$$

where $Q_{i,j}$ represents the Q-value estimated by the i-th critic for the j-th action, N is the number of critics, and M is the number of actions.

Semi-Gradient Descent Approach

The loss function utilizes a semi-gradient descent approach to update the weights of the multi-critic system. This approach involves updating the critic weights using three critic targets. The actor target generates the action for the next state, which is then evaluated by the critic target networks to determine the Q-values for those actions. This iterative process ensures that the actor learns to generate actions that maximize the expected reward.

Soft Update Approach

The weights of the critic networks, denoted as ϕ, are updated using a soft update approach with a degree of smoothness τ. This is formulated as:

$$\phi' \leftarrow \tau\phi + (1 - \tau)\phi' \tag{5.5}$$

where ϕ' represents the updated critic weights and τ is a small positive constant that determines the smoothness of the update. This approach ensures that the critic networks gradually adapt to new information, leading to more stable learning.

Summary

The proposed approach employs a multi-layered architecture, featuring a multi-critic single actor system. This system is designed to enhance the drone's navigation capabilities by leveraging multiple critics to evaluate the actions proposed by a single actor. This actor outputs three actions squashed between [-1,+1] which are then scaled proportionally and fed into the drone's controller for Pitch, Roll and Yaw.

Each of the three critics uses a different reward function to evaluate each action independently. The critic networks are trained by minimizing the *MSE loss* between the current Q-values and the TD targets. The actor network is trained to maximize the expected Q-value as evaluated by the critic networks. While the weights of the target networks are updated periodically using the soft update approach.

5.2.2 Hybrid Rewards Structure

The hybrid reward structure in the proposed DDPG-based algorithm consists of three distinct rewards, each addressing a specific aspect of drone navigation control. These rewards are designed to enhance the drone's agility, obstacle avoidance, and trajectory alignment. We slightly modified the rewards from the original literature to improve training times and model performance by giving it more direct of a reward.

Reward 1: Pitch Evaluation

Reward 1 is used to evaluate the pitch value generated by the actor, promoting agility and speed. A threshold of 1.5 is set as the upper limit for the pitch value. The reward function is given by:

$$R_1(F) = \begin{cases} \beta(F - F_t)^2 & \text{if } F_t < F \\ -\eta \exp\left(\frac{1}{F - F_t} + \epsilon\right) & \text{if } F_t \geq F \end{cases} \tag{5.6}$$

where F is the pitch value, F_t is the threshold (in our case - 1.5), $\beta(= 0.2)$ is a positive constant, $\eta(= 30)$ is a scaling factor, and ϵ is a small positive constant to avoid division by zero.

Reward 2: Obstacle Avoidance

Reward 2 is based on artificial potential hills, which transform sensed obstacles into virtual repulsive potential fields. The reward is inversely proportional to the distance of the drone from walls or obstacles. The repulsive potential field U_{rep} is mathematically described as:

$$U_{rep}(d) = \begin{cases} \lambda \left(\frac{1}{d} - \frac{1}{d_0}\right)^2 & \text{if } d \leq d_0 \\ 1 & \text{if } d > d_0 \end{cases} \tag{5.7}$$

where d is the distance to the obstacle, $d_0(= 1.0)$ is the influence distance, and λ $(= 0.5)$ is a scaling factor.

Reward 3: Trajectory Alignment

Reward 3 aligns the drone's trajectory with the central axis of the corridor by using an artificial attraction valley. This reward is based on the difference between the distances to the right (R) and left (L) walls. The attractive potential field U_{attr} is given by:

$$U_{attr}(R, L) = \alpha(R - L)^2 \tag{5.8}$$

where $\alpha(= 0.1)$ is a positive constant.

5.2.3 Precision Playback

Precision Playback (PP) is a specialized memory buffer that accumulates experiences when the loss is lower than the preceding state. The essence of PP is rooted in the principle of training the actor with high-quality examples, expediting the learning process. By storing and replaying experiences that result in lower loss, the actor network can learn more efficiently from prime quality examples, leading to faster convergence and improved performance. The PP buffer ensures that the actor is exposed to the most beneficial experiences, optimizing the training process.

The hybrid reward structure and precision playback are critical components of the proposed DDPG-based algorithm for autonomous indoor drone navigation. These elements ensure that the drone can navigate safely and efficiently, avoiding obstacles, maintaining alignment, and adapting to its environment with agility.

5.3 Simulation Setup

Our biggest challenge was setting up a simulation setup for training our model. We decided on using PX4 SITL (Software in the loop) [4] simulation using the Gazebo [5] Physics simulator and Robot Operating System (ROS Noetic) [6]. We developed gym bindings for the environment using Gazebo engine, as there did not exist any standardized bindings for the same. We used the iris drone with an attached LiDAR sensor.

However, the biggest hurdle was resetting of the environment after each episode. In summary, after the end of any episode or the drone crashing into a wall in the simulation, we needed to reset the world to its original state. This caused several issues with the PX4 simulation, primarily due to its EKF2 engine. We could fix this by applying a very specific patch in the 2019 version of PX4-Autopilot.

We then used a simple maze environment for the agent to learn in. We also employed the use of other settings like setting the clock of the physics engine multiple times the real time, reducing iterations of physics solver and more. Unfortunately, we could not run it in parallel due to configuration issues and compute limitations.

5.4 Results & Conclusion

Our experimentation with the proposed multilayered DDPG architecture for indoor drone navigation did not yield successful outcomes. The drone consistently exhibited a behaviour where it would move in circles rather than navigating the environment effectively.

This result was attributed to the independent nature of the reward updates for each critic, which hindered the learning process.

Results

Despite the theoretically sound design of employing multiple critics and a hybrid reward structure, practical implementation revealed significant flaws:

1. Agent not able to learn: The drone consistently failed to navigate the maze environment and would move in circles. This repetitive behaviour indicated that the learned policy was not effective in generating appropriate navigation actions.

2. Independent Critic Updates: The major flaw lay in the independent update mechanism of the critics. Each critic was designed to evaluate a specific aspect of the drone's navigation (pitch, obstacle avoidance, trajectory alignment). However, these aspects are *not* interdependent in practice. For instance, to navigate a turn, the drone's pitch and yaw need to be coordinated. The independent update mechanism failed to capture these interdependencies, leading to suboptimal learning.

3. Inefficiency in Learning: The independent critic updates made the learning process highly inefficient. Each critic evaluated actions based on its own reward, without considering the influence of other actions. This led to a scenario where the drone could not learn a cohesive strategy for navigation, resulting in prolonged training times without significant improvements.

Conclusion

The proposed multi-critic single actor architecture, although innovative in its approach to leveraging multiple perspectives for action evaluation, did not succeed due to the inherent interdependencies in the drone's control actions. Each critic evaluated actions independently, which did not align with the practical requirements of coordinated control inputs for effective navigation.

1. Interdependent Critic Updates: The failure highlighted the importance of considering the interdependencies between different control actions. Future architectures could focus on integrating these dependencies within the critic evaluations to enable more coherent policy learning.

2. Potential for Improvement: Given enough time and possibly additional training data, there is a potential that the network could learn the appropriate value functions. However, this would require significantly more computational resources and time, making the current approach inefficient for practical applications.

3. Reward Structure: While the hybrid reward structure aimed to address different aspects of navigation, it inadvertently contributed to the inefficiency by promoting independent updates. A more integrated reward mechanism that considers the overall navigation objective might lead to better results.

In conclusion, while the multilayered DDPG architecture with a hybrid reward structure presented an interesting approach, it ultimately proved to be inefficient and ineffective for indoor drone navigation due to the independent nature of the critic updates. Future research could explore more integrated approaches that consider the interdependencies of control actions to enhance the learning efficiency and effectiveness of autonomous drone navigation systems.

5.5 Further Work

The independent multi-critic approach, as tested in our setup, did not yield the expected results primarily due to the lack of integration among the different critics. Each critic evaluated actions based on distinct reward functions, which led to inefficiencies in learning and ineffective navigation. To address these issues, several improvements and alternative strategies can be considered:

1. Combining Value Functions

One promising approach is to combine the value functions generated by the different critics into a single, coherent value function. This can be achieved by computing a weighted mean of the value functions from the multiple critics. The weighted mean would be calculated as:

$$Q_{\text{combined}}(s, a) = \sum_{i=1}^{N} w_i Q_i(s, a)$$

where $Q_i(s, a)$ represents the value function from the i-th critic, and w_i denotes the weight assigned to the i-th critic. The weights w_i could be fixed or trainable, depending on whether we want to maintain some level of independence among critics.

To ensure stability, normalization techniques can be applied to the combined value function. This approach would address the issue of independent critics by providing a unified evaluation of actions, thereby facilitating better learning.

Training the Weights: If the weights w_i are trainable, this would involve adjusting them during training to optimize the overall performance. While this introduces some dependence among critics, it allows for a more adaptive approach where the importance of each critic's value function can be learned based on the performance.

2. Task Specific Goals

Multi-critic networks can be highly effective when the critics are designed to evaluate different, yet complementary aspects of a task. For instance, as seen in Siddharth M et al.'s work on Multi-Critic Actor Learning [7], different critics can be used to evaluate various play styles or goals in game environments. In the context of drone navigation, we could have a Navigation Critic which focuses on the drone's ability to navigate the environment and avoid obstacles and an Exploration Critic whose goal is to emphasize exploration and learning of the environment's layout.

Each critic would use different environmental features and reward functions tailored to their specific goals. This approach leverages the strengths of multi-critic systems when tasks are independent but complementary.

3. Using different sensors

Another alternative involves using different sensory inputs and environmental information for the critics. For example:

- Actor: Receives a primary RGB camera feed, and the drone is supposed to navigate and explore an unknown region using this limited sensory input.

- Critic 1: Uses depth segmentation maps derived from the RGB camera feed to evaluate obstacle avoidance and depth perception.
- Critic 2: Incorporates 2D LiDAR data to assess distance measurements and environmental features.
- Critic 3: Utilizes global map features or high-level environment context for navigation and spatial awareness.

By feeding distinct sensory inputs into each critic, the system can learn to utilize diverse types of information, potentially leading to a more robust and effective policy. Each critic's value function would then reflect different aspects of the environment, allowing for a more comprehensive evaluation of actions.

While implementing these techniques, we need to ensure that each critic's input is appropriately preprocessed and normalized. Possibly some use of techniques such as feature fusion or attention mechanisms to integrate the different sensory inputs effectively. It would also be important to balance the contributions of each critic to avoid over-reliance on specific sensory modalities.

Conclusion

While the independent multi-critic approach did not succeed in our current setup, exploring methods such as combining value functions, leveraging task-specific critics, and integrating diverse sensory inputs offers promising directions for improving multi-critic systems. Future research could investigate these approaches to address the limitations observed and enhance the effectiveness of multi-critic architectures in complex environments like drone navigation.

We experiment with some of the aforementioned approaches in the subsequent sections of this report.

Chapter 6

Work on different RL Techniques for Navigation and Exploration

6.1 Simulation Environment Setup and Deployment

6.1.1 Simulation Environment Challenges

The simulation for the drone environment proved to be tedious to set up and difficult to train reliably. The primary challenges arose from the complexity of accurately modelling the drone's dynamics and the cluttered environments in which it would operate. Variabilities in sensor data, environmental interactions, and the high computational cost of simulating aerodynamic effects added layers of difficulty. Consequently, after numerous attempts to stabilize the simulation parameters and achieve consistent training results, we decided to pivot our approach. Deploying the algorithms on a grounded robot, TurtleBot3, emerged as a viable alternative. This transition was motivated by the relative simplicity of ground-based navigation and the well-documented capabilities of TurtleBot3 in ROS environments.

6.1.2 Advantages of Using TurtleBot3

Despite the inherent differences between aerial and ground-based robots, the TurtleBot3 provided a pragmatic solution for initial algorithm deployment and testing. The similarities in terms of size, sensor data, and environmental interactions between the TurtleBot3 and a typical drone enabled a seamless transition. The robot's platform, equipped with LiDAR and IMU sensors, allowed for realistic simulations of navigation tasks that a drone would perform. By successfully implementing and validating our algorithms on TurtleBot3, we established a robust foundation. This validation indicated that porting the algorithms to a drone platform would require minimal adjustments, primarily related to the transition from ground to aerial dynamics, rather than a complete redesign.

6.1.3 Custom Gazebo Environment

In our quest for suitable simulation tools, we explored various open-source Gazebo gym bindings. However, we found that many of these existing solutions were overcomplicated with features that did not align with our specific needs. These extraneous functionalities often led to unnecessary computational overheads and hindered the customization re-

quired for our research focus. Therefore, we opted to create our own custom environment in Gazebo, integrated with ROS. This bespoke setup allowed us to tailor the simulation precisely to our requirements, ensuring efficient and effective training processes. Our custom environment streamlined the simulation, focusing solely on the essential features needed for reliable navigation and control algorithm testing.

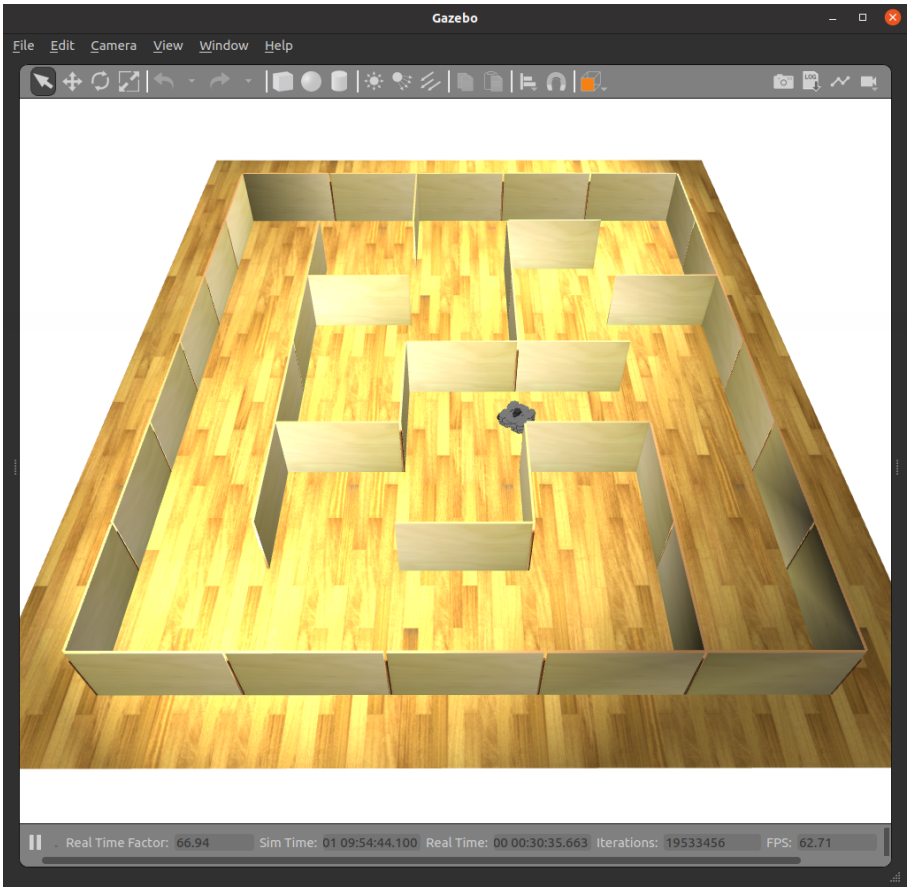


Figure 6.1: Turtlebot3 Gazebo simulation

6.1.4 Research on Reward Strategies and RL Algorithms

Our research involved an extensive review of various reward strategies and reinforcement learning (RL) algorithms. We systematically evaluated different reward paradigms to determine which would most effectively drive the desired navigation behaviours in cluttered environments. This comparative analysis extended to a range of RL algorithms, including both model-free and model-based approaches. By rigorously testing these algorithms under controlled conditions in our custom simulation environment, we gained valuable insights into their relative performance and suitability for high-speed, agile drone navigation. This comprehensive evaluation enabled us to identify the most promising strategies for implementation, providing a strong basis for subsequent real-world testing and deployment.

6.2 Sensor Setup

6.2.1 Importance of Sensors for Navigation and Exploration

Sensors play a crucial role in the navigation and exploration of autonomous systems. They provide essential data about the environment, enabling the robot to perceive obstacles, identify paths, and make informed decisions. Commonly used sensors include depth cameras, stereo cameras, LiDAR, and monocular cameras. Depth and stereo cameras, while offering rich and detailed data, are often expensive and computationally demanding. LiDAR, on the other hand, strikes a balance between cost and data richness, making it a popular choice for many applications. Monocular cameras, despite being cost-effective and low on computational requirements, present challenges in feature extraction and learning for navigation and exploration tasks.

6.2.2 Sensors and Data Collection Methods Used

Odometry

For odometry, we utilized the default TurtleBot3 odometry topic provided in ROS, which is based on wheel odometry and Inertial Measurement Unit (IMU) data. This setup ensures reliable tracking of the robot's position and movement over time, providing foundational data for navigation and path planning.

LiDAR

Given its cost-effectiveness and the richness of the data it provides, we chose LiDAR as the primary sensor for our project. LiDAR is particularly advantageous for its ability to generate detailed 3D maps of the surroundings. In our setup, we divided the LiDAR's 360-degree field of view into nine sectors and used the minimum distance in each sector as observations for the reinforcement learning (RL) agent. This method allowed for efficient and informative environmental sensing, crucial for obstacle avoidance and navigation tasks. Additionally, combining LiDAR data with camera inputs is a potential future enhancement, promising even more robust perception capabilities.

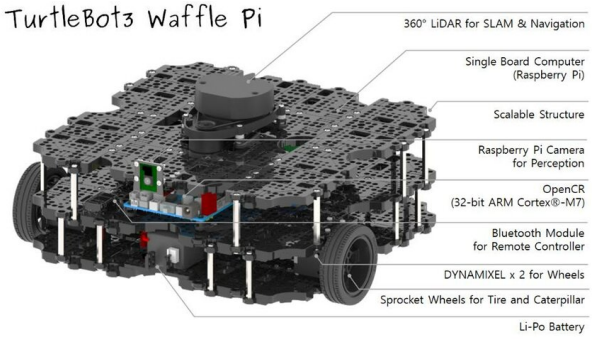


Figure 6.2: Turtlebot3 Waffle-Pi

6.2.3 Custom Occupancy Grid Mapping Solution

Occupancy grids are invaluable as they provide a direct representation of the area explored by the agent, which is essential for devising rewards and accurate value estimation during RL training. Initially, we used the standard `gmapping` ROS library for 2D mapping and occupancy grid creation. However, we observed that `gmapping` introduced overheads such as loop closure, which slowed down the process over time. To address this, we developed a custom, barebones 2D mapping solution from scratch using Python, NumPy, and OpenCV. By employing Numba and the threading library, we optimized the code, achieving speeds nearly 4-5 times faster than the original implementation. We also separated this functionality into a distinct node, allowing it to operate in parallel with the RL code, thereby enhancing overall system efficiency.

Figure 6.3: Example Occupancy Grid

Algorithm 1 Custom 2D Mapping Solution Pseudocode

Initialize occupancy grid with zeros
Input LiDAR data, robot position, grid resolution, yaw angle
for each LiDAR angle and distance **do**
    Calculate global angle using LiDAR angle and yaw
    Convert polar coordinates to grid coordinates (`end_x`, `end_y`)
    Initialize start coordinates (`x0`, `y0`) with robot position
    Calculate differences Δx, Δy and steps `sx`, `sy`
    Initialize error term `err`
    while (`x0`, `y0`) not equal to (`end_x`, `end_y`) and within grid bounds **do**
        Set occupancy grid cell at (`x0`, `y0`) as occupied
        Calculate double error term $e2$
        if $e2 > -\Delta y$ **then**
            Update error and `x0`
        end if
        if $e2 < \Delta x$ **then**
            Update error and `y0`
        end if
    end while
    if distance within threshold **then**
        Mark end grid cell as free space
    end if
end for
Output updated occupancy grid

This custom mapping solution provided a streamlined and efficient method for generating occupancy grids, significantly improving the speed and accuracy of the data used for training the RL agent. The integration of these various sensors and data collection methods ensured a robust and reliable navigation system capable of effective exploration in cluttered environments.

6.3 Reward Strategies

6.3.1 Literature Review and Reward Testing

To develop effective reward strategies for drone navigation and exploration, we performed a comprehensive literature review and tested several reward functions. This review encompassed various approaches used in reinforcement learning (RL) for autonomous navigation, including potential field methods, information-theoretic rewards, and frontier-based exploration techniques. Each reward function was implemented and rigorously tested within our custom simulation environment to assess its effectiveness in promoting desired behaviors such as obstacle avoidance, exploration, and efficient path planning.

6.3.2 Artificial Potential Field for Obstacle Avoidance

One of the primary rewards we implemented was based on the Artificial Potential Field (APF) method [8], designed for obstacle avoidance. The APF approach treats obstacles as repulsive forces and free space as attractive, guiding the drone away from obstacles and towards unexplored areas. The repulsion function $R(d)$ is defined as:

$$R(d) = \begin{cases} 0.5\left(\left(\frac{1}{d} - \frac{1}{d_0}\right)^2\right) + 1 & \text{if } d \leq d_0 \\ 1 & \text{otherwise} \end{cases} \tag{6.1}$$

where d is the distance to the obstacle and d_0 is a threshold distance. This reward effectively penalizes proximity to obstacles, ensuring safer navigation paths.

6.3.3 Occupancy Grid Entropy-Based Reward

To encourage exploration, we chose a reward based on the entropy of the occupancy grid [9]. This reward aims to maximize the information gain by exploring less certain areas. The entropy $E(p)$ of a cell with occupancy probability p is given by:

$$E(p) = -p\log(p+\epsilon) - (1-p)\log(1-p+\epsilon) \tag{6.2}$$

where ϵ is a small constant to avoid logarithm of zero. The total entropy-based reward R_e is computed as:

$$R_e = \frac{\max(E)}{1+\sum E} \tag{6.3}$$

This reward structure promotes exploration of areas with high uncertainty, driving the drone to gather more environmental data.

6.3.4 Frontier Exploration-Based Reward

For promoting global exploration, we implemented a frontier exploration-based reward. This method identifies frontiers, which are boundaries between explored and unexplored regions. The reward R_f for reaching a frontier viewpoint is inversely proportional to the distance d between the robot's current position and the frontier:

$$R_f = \frac{1}{d+1} \tag{6.4}$$

This reward was designed to push the drone towards unexplored frontiers, thereby expanding the known environment.

6.3.5 Other Helper Rewards

In addition to the primary rewards, several helper rewards were implemented to refine the drone's behaviour:

- **Distance Travelled Reward:** Encourages movement by rewarding the distance travelled between two positions (x_1, y_1) and (x_2, y_2):

$$R_d = -\frac{0.1}{0.1 + 10\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \tag{6.5}$$

- **Motion Penalty:** Penalizes abrupt changes in velocity to ensure smooth motion:

$$P_m = |v_{current} - v_{previous}| \tag{6.6}$$

- **Spinnage Penalty:** Penalizes excessive angular velocity to prevent unnecessary spinning:

$$P_s = \omega^2 \tag{6.7}$$

6.3.6 Conclusion and Findings

After extensive testing, we found that the APF and entropy-based rewards were exceptionally effective in achieving safe navigation and thorough exploration, respectively. The frontier-based reward, while conceptually promising, proved to be computationally expensive and less effective in practice, leading to its eventual exclusion. Helper rewards, such as distance travelled, motion penalty, and spinnage penalty, played crucial roles in fine-tuning the drone's behaviour. These rewards were scaled based on intuition and a trial-and-error approach, with final scaling values chosen to balance the different aspects of navigation:

$$\text{Total Reward} = -0.025 \cdot \text{APF} + 5 \cdot \text{Entropy} - 0.1 \cdot \text{Spinnage} + 0.5 \cdot \text{Distance} \tag{6.8}$$

Positive rewards were converted to penalties that decayed over time to encourage rapid exploration. The utility of these rewards was tested using the state-of-the-art Proximal Policy Optimization (PPO) algorithm with the Stable Baselines 3 library, demonstrating significant improvements in navigation performance.

6.4 Exploration and Navigation Algorithms

6.4.1 Utility of Reinforcement Learning (RL) Techniques

Reinforcement Learning (RL) techniques offer substantial advantages for exploration and navigation tasks, particularly in dynamic and cluttered environments. Unlike traditional methods that often rely on pre-defined heuristics or control laws, RL techniques can adaptively learn optimal policies through interaction with the environment. This adaptability is crucial for handling the uncertainties and complexities inherent in such tasks. RL methods excel in optimizing performance by continuously improving their strategies based on feedback from the environment, which is particularly beneficial for high-speed and agile drone control.

6.4.2 Algorithms Tested

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) [10] is a state-of-the-art reinforcement learning algorithm known for its balance between ease of implementation and performance. It is designed to improve the stability and reliability of policy updates by introducing a clipped surrogate objective function. PPO is commonly used for benchmarking due to its effectiveness in various reinforcement learning tasks and its ability to handle continuous action spaces with relatively lower computational costs compared to other methods.

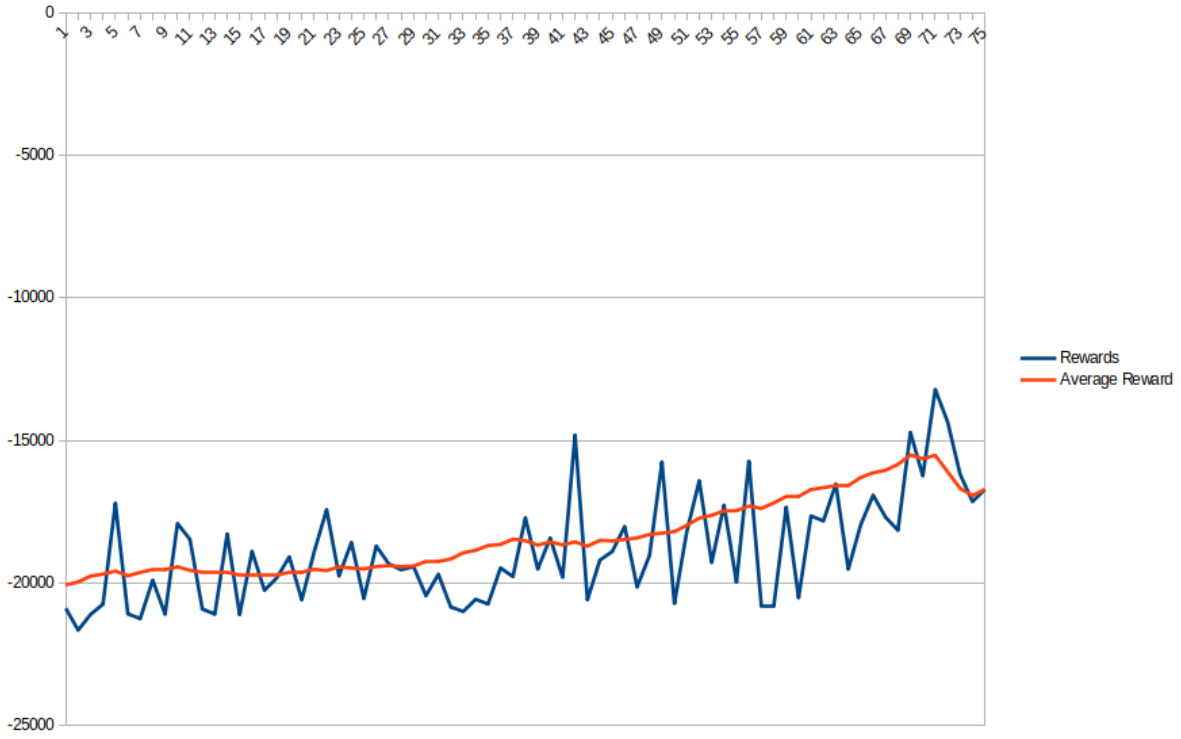


Figure 6.4: Rewards over 150 episodes

Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) [11] is another leading algorithm in the realm of reinforcement learning. It integrates the principles of maximum entropy to encourage exploration and improve robustness. SAC optimizes a stochastic policy and value functions using off-policy learning, which enables it to achieve high performance in continuous action spaces. Its capacity for efficient exploration and stable learning makes it a prominent choice for complex navigation and control tasks.

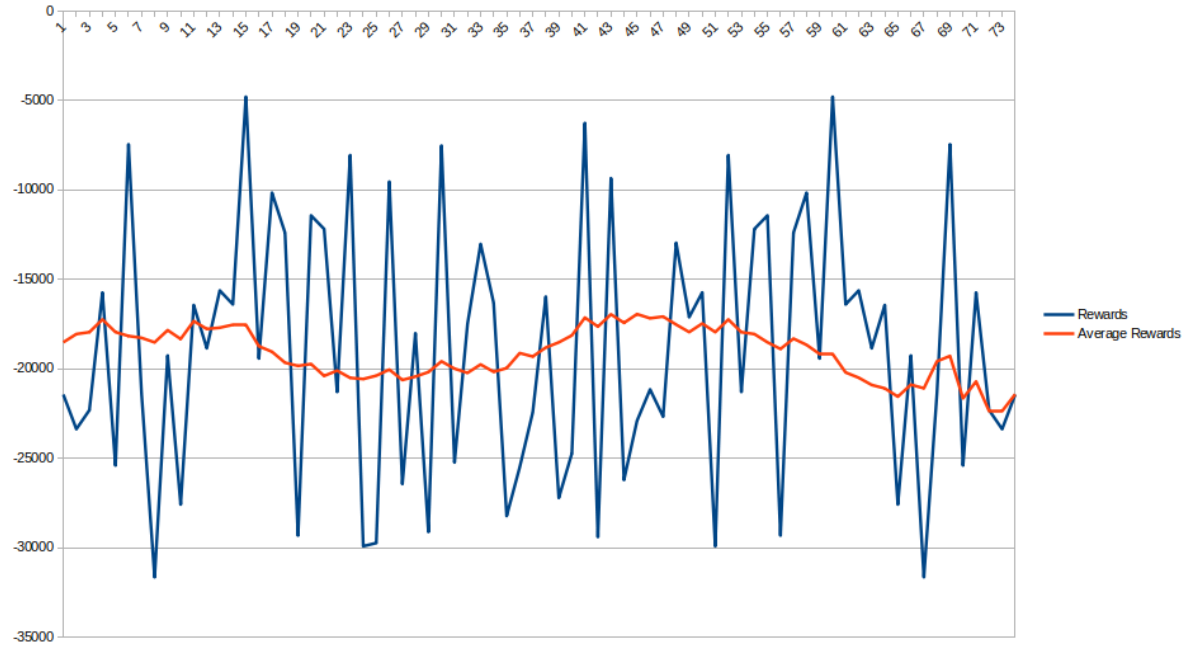


Figure 6.5: Rewards over 150 episodes

Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) [3] is a well-established reinforcement learning algorithm designed specifically for continuous action spaces. It combines the benefits of deep learning with policy gradient methods, using an actor-critic architecture to handle high-dimensional state and action spaces. While DDPG is effective for tasks requiring precise control over continuous actions, it has some limitations, such as sensitivity to hyperparameters and less robustness in exploration compared to more recent algorithms like SAC.

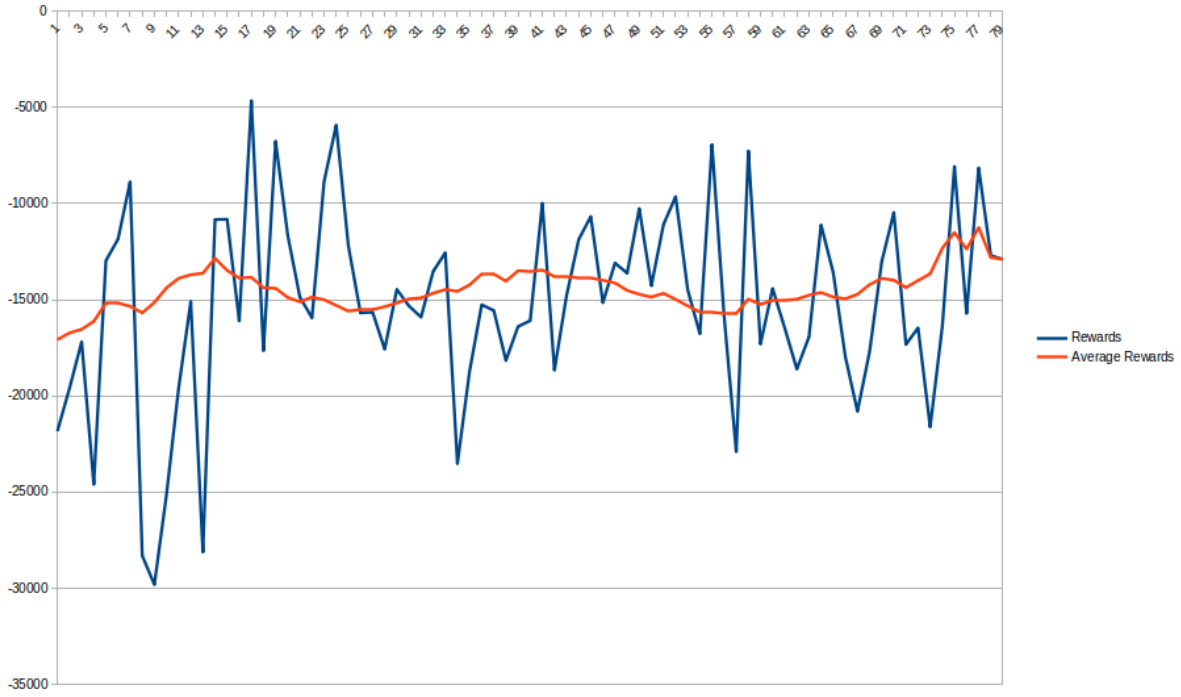


Figure 6.6: Rewards over 150 episodes

6.4.3 Results

The tests were conducted in the Gazebo environment depicted before for 150 episodes, using the same rewards for consistency across all algorithms.

- PPO performed the best among the tested algorithms. It demonstrated very little oscillation in the reward values and showed a steady rise, indicating stable and efficient learning.
- SAC performed the worst. It exhibited a significant amount of oscillations in the rewards and did not show a clear increase, suggesting instability and inefficiency in learning.
- DDPG just outperformed SAC. Although it showed more oscillations than PPO, there was a general upward trend in the rewards, indicating some level of learning progress.

Results for longer training sessions could not be compiled due to limitations on time and resources, restricting a more comprehensive evaluation.

These results strengthen our decision to build our novel architecture through modifications in PPO and DDPG. Potential enhancements include the incorporation of the Intrinsic Curiosity Module (ICM) or a multi-critic approach to improve performance and stability.

Chapter 7

Curiosity Driven Exploration?

Exploration in reinforcement learning is a critical aspect of training agents to navigate and understand environments, particularly when the complete map of the environment is not known a priori. Traditional exploration strategies often rely on extrinsic rewards—rewards provided by the environment based on specific actions or states. However, when faced with environments that are vast, complex, or partially observable, relying solely on extrinsic rewards can be insufficient. This limitation is particularly pronounced in scenarios where the environment contains hidden information or when the agent must discover new strategies to succeed.

An alternative approach to conventional exploration strategies is Curiosity Driven Exploration, which leverages intrinsic rewards to guide the agent's exploration. Intrinsic rewards are generated internally by the agent based on its own representation and understanding of the environment, rather than being directly provided by external sources. This method enables the agent to develop an internal reward function that reflects its curiosity and desire to explore new or novel states.

The core idea behind Curiosity Driven Exploration is to incentivize the agent to seek out and engage with aspects of the environment that it finds unfamiliar or challenging. This intrinsic motivation is based on the agent's internal representation of the environment, which evolves as it learns more about its surroundings. By incorporating intrinsic rewards, the agent is encouraged to explore regions of the environment that might not be immediately rewarding based on extrinsic criteria but are deemed valuable for learning and discovery.

Our goal is to evaluate whether Curiosity Driven Exploration yields superior results compared to traditional extrinsic reward-based methods, particularly in environments with limited extrinsic feedback. We do so by limiting the extrinsic rewards provided and add the *intrinsic rewards* while training the agent with both models simultaneously.

7.1 Intrinsic Curiosity Module (ICM)

The Intrinsic Curiosity Module (ICM) is a reinforcement learning mechanism designed to enhance the exploration capabilities of an agent by providing intrinsic rewards. These rewards are based on the agent's ability to predict and learn from its interactions with the environment, beyond the extrinsic rewards provided by the task. [12]

7.1.1 Theoretical Background

Intrinsic rewards in reinforcement learning can drive an agent to explore more effectively by rewarding it for novel experiences or for learning. The ICM consists of two primary components: an inverse model and a forward model.

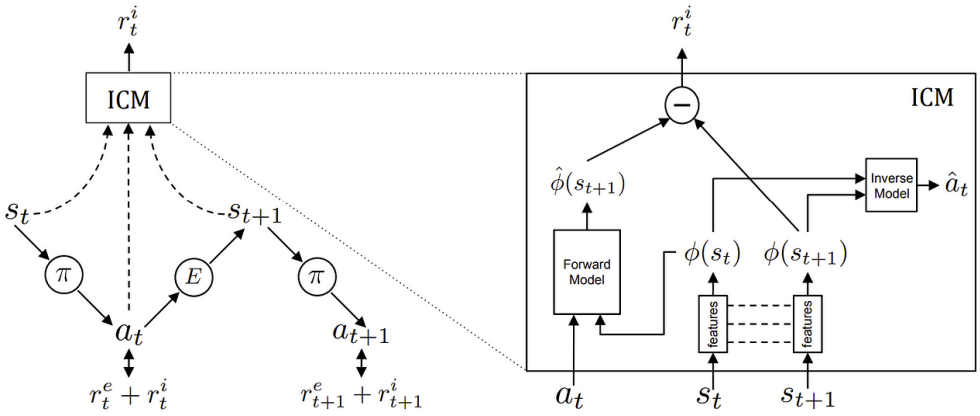


Figure 7.1: Diagram of the Intrinsic Curiosity Module [12]

7.1.2 Components of ICM

1. Feature Encoder: This component maps the raw state observations into a feature space where the predictions are made. We implemented the feature encoder as follows:

$$\text{Feature Encoder: } f_{\text{feat}}(s) = \text{ReLU}(W_1 s + b_1) \rightarrow \text{ReLU}(W_2 h + b_2)$$

2. Inverse Model: This model predicts the action taken by the agent given the current and next state features. The inverse model is described by:

$$\text{Inverse Model: } p(a|s, s') = \text{softmax}(W_a[f_{\text{feat}}(s) \| f_{\text{feat}}(s')] + b_a)$$

3. Forward Model: This model predicts the next state feature given the current state feature and the action taken. The forward model can be expressed as:

$$\text{Forward Model: } \hat{f_{\text{feat}}}(s', a) = \text{ReLU}(W_f[f_{\text{feat}}(s) \| a] + b_f)$$

where s is the current state, s' is the next state, and a is the action.

7.1.3 Intrinsic Reward Calculation

The intrinsic reward r_{intr} is computed based on the prediction error of the forward model:

$$r_{\text{intr}} = \frac{1}{2} \left\| f_{\text{feat}}(s') - \hat{f_{\text{feat}}}(s, a) \right\|^2$$

where $\hat{f_{\text{feat}}}(s, a)$ is the predicted feature and $f_{\text{feat}}(s')$ is the actual feature of the next state.

7.2 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is an on-policy reinforcement learning algorithm designed to improve the stability and reliability of policy gradient methods. PPO achieves this by employing a clipping method that restricts the extent of policy updates, which mitigates the issue of large policy changes.

7.2.1 Algorithm Overview

PPO operates by alternating between collecting data through interaction with the environment and updating the policy using the collected data. The policy is updated by maximizing a clipped surrogate objective function, which balances between improving the policy and maintaining stability.

7.2.2 PPO Objective Function

The PPO objective function is defined as:

$$L^{\text{PPO}}(\theta) = \hat{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right], \tag{7.1}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ represents the probability ratio, $\hat{A}_t$ is the estimated advantage function at time step t, and ϵ is the clipping parameter. The clipping function ensures that the new policy does not deviate too far from the old policy, thereby maintaining stable and reliable updates. The advantage function $\hat{A}_t$ can be computed using Generalized Advantage Estimation (GAE), which reduces variance while maintaining a low bias:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \tag{7.2}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the temporal difference error, γ is the discount factor, and λ is a parameter that controls the trade-off between bias and variance in the advantage estimates.

To optimize the objective function, PPO uses stochastic gradient ascent. The policy and value functions are typically parameterized by neural networks, and their parameters are updated using the Adam optimizer. The value function is trained to minimize the mean squared error (MSE) between the predicted value and the empirical return:

$$L^{\text{value}}(\theta_v) = \frac{1}{T}\sum_{t=1}^{T}(V_{\theta_v}(s_t) - R_t)^2, \tag{7.3}$$

where $V_{\theta_v}(s_t)$ is the value function with parameters θ_v, and R_t is the cumulative return from time step t.

7.3 Algorithm Details

7.3.1 PPO with ICM

Incorporating ICM into PPO involves adding an intrinsic reward component to the reward signal and optimizing both the policy and value functions to account for this intrinsic reward. The intrinsic reward encourages the agent to explore novel states, enhancing

overall learning performance. We calculate ICM loss as a summation of both forward and inverse loss. This is then added to the actor, critic loss after multiplying by a scaling factor. Gradient Descent is performed on this total loss calculated.

7.3.2 Implementation Details

The implementation of the Proximal Policy Optimization (PPO) algorithm augmented with the Intrinsic Curiosity Module (ICM) involves several key components and design choices. Here, we outline the specific details of our implementation, covering the environment setup, network architectures, hyperparameters, training process, intrinsic reward calculation, and evaluation metrics. Our environment setup was the exact same as in the last section 6.1. Using turtlebot3 in a gazebo simulation environment.

7.3.3 Network Architecture

The policy and value networks, collectively known as the actor-critic network, share a similar architecture designed to balance computational efficiency and learning capacity. Both networks are composed of fully connected layers with ReLU activations. The specific architecture details are as follows:

1. Actor Network: Consists of one hidden layers with 64 neurons and a ReLU activation, followed by a TanH final activation for output.
2. Critic Network: Mirrors the actor network with one hidden layers of 64 neurons, outputting a single value prediction.

Additionally, the ICM is structured to process states and actions, comprising a feature encoder, an inverse model, and a forward model:

1. Feature Encoder: Two fully connected layers with 256 neurons each.
2. Inverse Model: Takes concatenated state and next-state features, passing through two fully connected layers with 256 neurons each, to predict the action with Softmax activation.
3. Forward Model: Takes the concatenated state feature and action, passing through two fully connected layers with 256 neurons each to predict the next state feature.

Hyperparameters

Following hyperparameters were chosen based on testing and some tuning in our environment:

1. Learning rate: 3×10^{-4}
2. Discount factor (γ): 0.99
3. Clipping parameter (ϵ): 0.2
4. Number of epochs per update: 10
5. ICM scaling factor: 0.1
6. ICM inverse and forward model loss coefficient: 0.5

Algorithm 2 PPO with Intrinsic Curiosity Module (ICM)

Initialize:
Policy network π_θ and value network V_ϕ with parameters θ and ϕ respectively
Intrinsic Curiosity Module (ICM) network
Replay buffer for storing transitions
Hyperparameters: learning rate α, discount factor γ, clipping parameter ϵ, number of epochs E, ICM scaling factor scale
for each episode **do**
Reset environment and obtain initial state s_0
Initialize episode reward R_{total} and intrinsic reward R_{intr}
Initialize lists: S, A, R, S', log_probs, V, dones, intrinsic_rewards
while not done **do**
Obtain action a_t from policy network π_θ given state s_t
Sample action a_t from the policy distribution
Execute action a_t in the environment
Observe next state s_{t+1} and reward r_t
Compute intrinsic reward r_{intr} using the ICM network:
Compute state and next state features: $f_{\text{feat}}(s_t)$ and $f_{\text{feat}}(s_{t+1})$
Predict next state feature using forward model: $\hat{f}_{\text{feat}}(s_t, a_t)$
Compute intrinsic reward:

$$r_{\text{intr}} = \text{scale} \times \left\| f_{\text{feat}}(s_{t+1}) - \hat{f}_{\text{feat}}(s_t, a_t) \right\|^2$$

Append $s_t, a_t, r_t, s_{t+1}, \text{log\_prob}(a_t), V(s_t), \text{done}, r_{\text{intr}}$ to their respective lists
Update state $s_t \leftarrow s_{t+1}$
end while
Compute returns and advantages:
Calculate R_t and $\hat{A}_t$ using Generalized Advantage Estimation (GAE):

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

$$R_t = \hat{A}_t + V(s_t)$$

Normalize advantages:

$$\hat{A}_t = \frac{\hat{A}_t - \text{mean}(\hat{A}_t)}{\text{std}(\hat{A}_t) + 1e-8}$$

Update policy and value networks:
for epoch = 1 to E **do**
Calculate new log probabilities new_log_probs using the policy network
Compute the ratio:

$$\text{ratio} = \exp(\text{new\_log\_probs} - \text{old\_log\_probs})$$

Compute surrogate losses:

$$\text{surr1} = \text{ratio} \times \hat{A}_t$$

$$\text{surr2} = \text{clip}(\text{ratio}, 1-\epsilon, 1+\epsilon) \times \hat{A}_t$$

Compute policy loss:

$$\text{actor\_loss} = -\min(\text{surr1}, \text{surr2}).\text{mean}()$$

Compute value loss:

$$\text{critic\_loss} = \text{MSELoss}(V(s_t), R_t)$$

Update ICM network:
Compute inverse and forward model losses:

$$\text{inverse\_loss} = \text{MSELoss}(\text{pred\_actions}, a_t)$$

$$\text{forward\_loss} = \text{MSELoss}(\text{pred\_next\_state\_feats}, f_{\text{feat}}(s_{t+1}))$$

Compute total ICM loss:

$$\text{icm\_loss} = \text{inverse\_loss} + \text{forward\_loss}$$

Compute total loss:

$$\text{total\_loss} = \text{actor\_loss} + \text{critic\_loss} \times \text{coeff\_critic} - \text{entropy} \times \text{coeff\_entropy} + \text{icm\_loss} \times \text{beta}$$

Perform gradient descent step:
Zero gradients
Backpropagate loss
Update policy and value networks
end for
Log episode results including losses, rewards, and intrinsic rewards
Save model if episode number is a multiple of save interval
end for

7.3.4 Rewards

Out of all the rewards mentioned in previous section 6.3 only the following were used to validate the effectiveness of ICM. The total rewards are given by

$$\text{Total Reward} = -0.075 \cdot \text{APF} + \text{Entropy} + 1/(1 + \text{Spinnage}) \tag{7.4}$$

7.4 Results

The model was trained in the gazebo environment as per the described algorithm.

On the first long run of 1000 episodes (Figure 7.2), we can observe the following: The intrinsic reward slowly increases to try and match the extrinsic reward. Here we are not using the intrinsic reward in decision-making (it is not combined with extrinsic rewards while testing), just updating its weights in its training process. Over time it should understand the world and better reflect the outcome which we are able to see after a lot of episodes. The intrinsic reward tracks the other total reward.

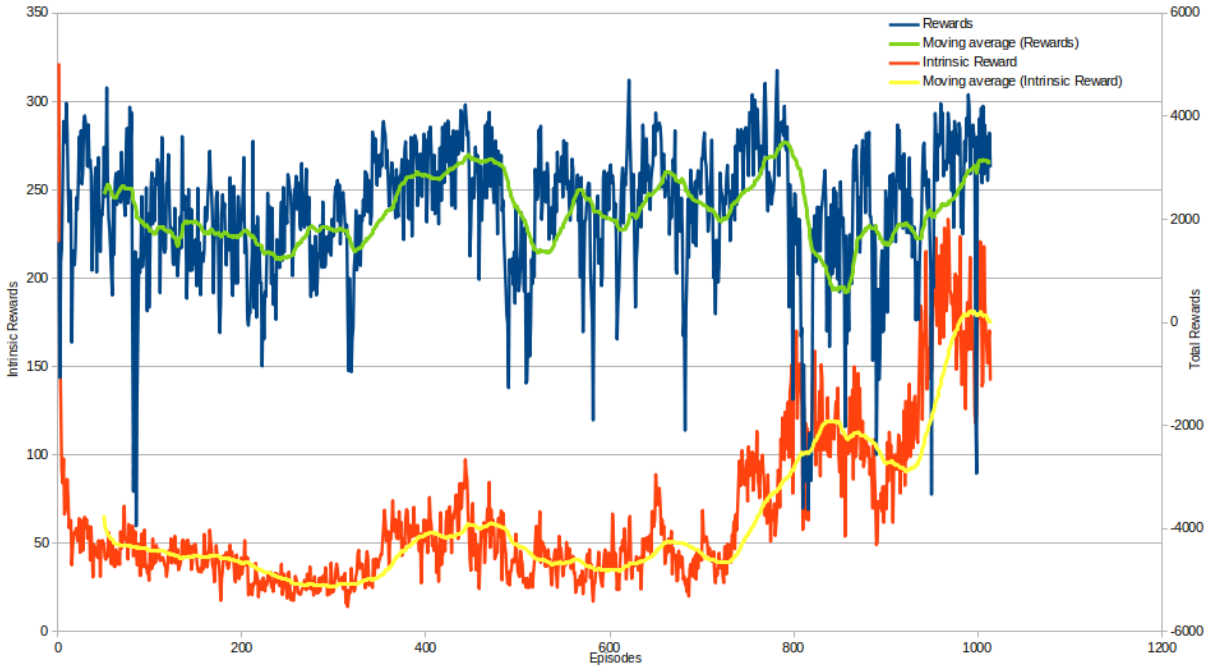


Figure 7.2: Rewards over 1000 episodes

After finetuning some hyperparameters and changing the architecture slightly and retraining for 150 episodes, we can observe (Figure 7.3) that intrinsic reward has actually started to learn the environment much faster. And when compared to other approaches in 6.4.2 it does perform much better.

We cannot coerce this to just be due to ICM and updating the loss together, the model architecture is also slightly different. Our pytorch implementation could be slightly different compared to stable baselines, which is why isolating it just to ICM is slightly difficult.

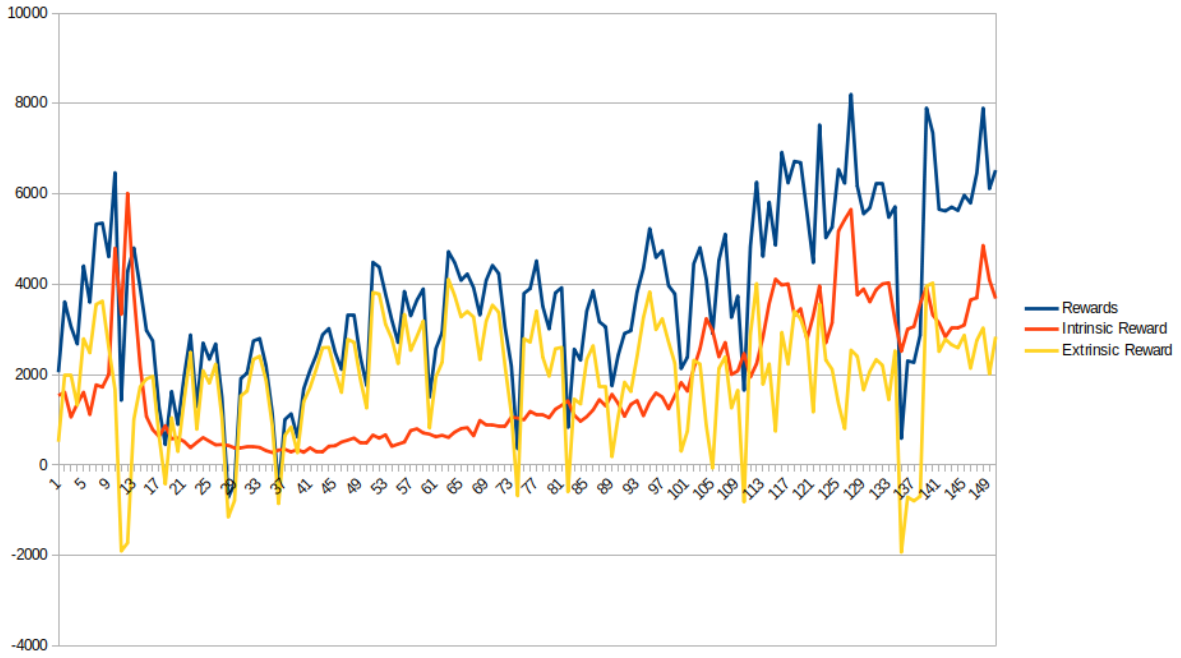


Figure 7.3: Intrinsic and Extrinsic Rewards over 150 episodes

7.5 Conclusion

In Conclusion we see that ICM performs better, although with different reward structure and less number of episodes even qualitative assessment is very difficult. We can see that addition of the ICM does work and the module does learn the features of the world without explicit reward functions being given.

In summary, our study demonstrates that the Intrinsic Curiosity Module (ICM) significantly enhances performance, even when traditional reward structures are sparse and the number of episodes is limited. The integration of ICM allows the agent to effectively learn the features of the environment without relying solely on explicitly defined reward functions. This intrinsic motivation fosters a deeper exploration and understanding of the environment, leading to improved learning outcomes.

Despite these promising results, further testing is required to fully assess the performance of ICM, particularly when intrinsic rewards are incorporated into the actor's decision-making process. Preliminary observations suggest rapid convergence with ICM, raising the possibility that the agent could potentially operate without any extrinsic rewards. However, due to time constraints, we could not explore this hypothesis in detail.

Future work could focus on a comprehensive evaluation of ICM's impact on training efficiency and overall performance. Investigating the feasibility of entirely eliminating extrinsic rewards could reveal a substantial reduction in training time, as indicated by our initial findings.

Chapter 8

Conclusion

In this report, we have thoroughly investigated the application of reinforcement learning (RL) techniques for the navigation and exploration of drones in indoor constrained environments. The study began with a comprehensive review of existing literature and preliminary work on drone navigation tasks, specifically focusing on leachate site detection and inspection. This foundational work provided critical insights into the algorithmic flow and hardware implementation required for effective drone navigation. We explored the integration of 3D LiDAR sensor fusion techniques which could significantly enhanced the drones' environmental perception, leading to more precise and reliable navigation.

The exploration of various reward strategies, such as artificial potential fields, occupancy grid entropy, and frontier exploration-based rewards, underscored the importance of well-designed reward mechanisms in RL-based navigation. We benchmarked state-of-the-art RL algorithms, including Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and Deep Deterministic Policy Gradient (DDPG), to evaluate their performance in complex indoor environments. Our findings demonstrated the superior adaptability of these algorithms in handling static obstacles and constrained indoor spaces. Additionally, the incorporation of intrinsic curiosity modules (ICM) revealed promising results in boosting exploration efficiency and overall performance.

Future work will involve further refinement of these algorithms and the exploration of hybrid models that combine the strengths of multiple RL techniques. Additionally, real-world testing in more diverse and challenging environments will be essential to validate and extend the capabilities of autonomous drones. The results of this study highlight the immense potential of RL in advancing the field of autonomous aerial robotics, paving the way for more intelligent and autonomous navigation systems.

Bibliography

[1] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," 2019.

[2] J. Lin and F. Zhang, "R3live: A robust, real-time, rgb-colored, lidar-inertial-visual tightly-coupled state estimation and mapping package," 2021.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2016.

[4] PX4 Development Team, "Px4 autopilot," 2024.

[5] Open Source Robotics Foundation, "Gazebo simulator," 2024.

[6] Open Source Robotics Foundation, "Robot operating system (ros) wiki," 2024.

[7] S. Mysore, G. Cheng, Y. Zhao, K. Saenko, and M. Wu, "Multi-critic actor learning: Teaching RL policies to act with style," in *International Conference on Learning Representations*, 2022.

[8] S. M. Hosseini Rostami, A. Kumar, J. Wang, and X. Liu, "Obstacle avoidance of mobile robots using modified artificial potential field algorithm," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, 03 2019.

[9] A. Devo, J. Mao, G. Costante, and G. Loianno, "Autonomous single-image drone exploration with deep reinforcement learning and mixed reality," *IEEE Robotics and Automation Letters*, vol. 7, pp. 1–1, 04 2022.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[12] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *ICML*, 2017.