Machine Learning-Based Drone Navigation and Sensor Mapping in Simulated 3D Environments

By Carlan Jackson

A THESIS

Submitted partial fulfillment of the requirements for the degree of Master of Science in the Department of Computer Science in the School of Graduate Studies

Alabama A&M University

Normal, Alabama 35762

June 2025

Submitted by CARLAN JACKSON in partial fulfillment of the requirement for the degree of MASTER OF SCIENCE specializing in COMPUTER SCIENCE.

Accepted on behalf of the Faculty of the School of Graduate Studies by the Thesis Committee:

_____ Dr. Michael Ayokanmbi

_____ Dr. Alak Bandyopadhyay

_____ Dr. Ed Pearson

_____ Dr. Yujian Fu
Major Advisor

_____ Dr. Tau Kadhi
Dean of School of Graduate Studies

_____ Date

MACHINE LEARNING-BASED DRONE NAVIGATION AND SENSOR MAPING IN SIMULATED 3D ENVIRONMENTS

Carlan Jackson, M.S., Alabama A&M University, 2025.

Thesis Advisor: Dr. Yujian Fu

Abstract: This paper presents the developments for an autonomous drone navigation system focused on efficient path planning and collision avoidance in complex environments. Through leveraging a high-fidelity simulation environment developed using Microsoft AirSim and the Unreal Engine platform, as well Reinforcement Learning (RL) for autonomous decision-making, the system was designed to support adaptive and robust drone navigation in complex virtual environments. The methodology involved 3D environment modeling, simulated drone sensor configuration, algorithm design for obstacle avoidance, and iterative optimization of training parameters. Throughout training, drone agents progressively learn to navigate toward target coordinates while minimizing collisions. Qualitative and quantitative evaluations confirmed improvements in learning performance and policy effectiveness. The system consistently demonstrated the ability to reach navigation goals, providing a strong baseline for future research in multi-agent coordination and real-world deployment of autonomous aerial systems.

KEY WORDS: artificial intelligence, machine learning, reinforcement learning, unmanned aerial vehicle,

Table of Contents

CHAPTER 1 - Introduction

The rapid advancement of unmanned aerial vehicles (UAVs) has opened the door to applications across diverse fields such as search and rescue, precision agriculture, infrastructure inspection, and other environmental monitoring. Despite these innovations, UAVs still face significant challenges when operating autonomously in real-world settings. Many environments—such as dense cities, forest areas, or disaster zones—are unstructured, cluttered, or highly dynamic, presenting obstacles that are difficult to anticipate or pre-map. Traditional rule-based systems often fall short under these conditions due to limited adaptability.

To be effectively deployed in such scenarios, UAVs must go beyond basic flight stability and obstacle avoidance. They require robust decision-making systems capable of interpreting sensor input in real time, handling edge cases, and generalizing unfamiliar terrain or mission parameters without direct human oversight. This involves not only the integration of perception, control, and planning modules, but also the use of learning-based approaches that can adapt over time. Moreover, deployment in the physical world introduces additional constraints—such as hardware limitations, battery life, etc.—which are not always addressed in simulation. Bridging this gap requires careful validation, rigorous simulation, and scalable training frameworks capable of producing agents that perform reliably outside of controlled lab environments.

Traditional path planning algorithms such as A* and Dijkstra have been widely used in robotics but often fall short when applied to large-scale or unpredictable environments. These approaches typically rely on explicit environmental models or static maps, making

them ill-suited for tasks that require adaptive, real-time behavior in changing conditions. As smaller UAVs are expected to operate more independently, especially in GPS-denied or visually unfamiliar spaces, new approaches to intelligent navigation have become essential.

One of the key components enabling such autonomy is accurate and robust sensing. In this research, LiDAR (Light Detection and Ranging) serves as the primary perception tool due to its ability to provide precise depth and range information in diverse lighting and environmental conditions. LiDAR-based mapping allows UAVs to perceive obstacles and spatial layouts in real time, forming the foundation of the learning-based navigation pipeline. This sensor choice directly shapes both the observation representation and the reward design throughout this study.

Reinforcement Learning (RL) offers a powerful framework for training agents to make decisions by interacting with their environment and learning from the consequences of their actions. Unlike supervised learning, which depends on labeled datasets, RL enables agents to adapt through trial and error, making it well-suited for the dynamic, uncertain domains of UAV navigation. Deep Reinforcement Learning (Deep RL) builds on this by integrating deep neural networks, allowing agents to generalize from the high-dimensional input data such as images or sensor streams. Algorithms **like Deep Q-Networks (DQN)** and **Proximal Policy Optimization (PPO)** have shown considerable success in complex tasks involving real-time control and planning. When deployed in simulation platforms, these algorithms can leverage continuous, real-time inputs enabling the development of UAV systems that are not only trained in a virtual space but are also

capable of responding to live, unpredictable conditions that mimic real-world environments—highlighting the need for realistic, physics-rich simulation environments where such learning and interaction can be safely and efficiently conducted.

This work expands the investigation to include the **Deep Deterministic Policy Gradient (DDPG)** algorithm, a continuous-control, off-policy actor-critic method well-suited for fine-grained UAV maneuvers. DQN, PPO, and DDPG span both **on-policy** and **off-policy** learning paradigms. On-policy methods such as PPO learn directly from experiences generated by the current policy and are valued for their stability. Off-policy methods like DQN and DDPG, by contrast, learn from replayed experiences and tend to be more sample-efficient, though potentially less stable. This contrast forms a key part of the study—analyzing how algorithmic family, action representation, and policy type influence performance in continuous sensor-rich navigation tasks.

To develop and test our system, we use **Microsoft AirSim**, a high-fidelity simulator for autonomous vehicles built on Unreal Engine. AirSim offers a realistic 3D physics environment that replicates aerodynamics, gravity, and collision mechanics with high precision. It supports a wide range of virtual sensors—including RGB and depth cameras, GPS, lidar, etc.—which can be customized to mirror real-world hardware configurations. This enables the development of perception-driven navigation strategies in complex environments without risking physical hardware. Additionally, AirSim supports real-time API access and is fully compatible with deep learning frameworks like PyTorch, TensorFlow, or Stable Baselines, making it ideal for training and evaluating

reinforcement learning (RL) agents. Our system architecture connects sensor feedback from the simulation to learning agents, enabling end-to-end training for tasks such as **collision avoidance, path planning, and goal-seeking behavior in 3D space.**

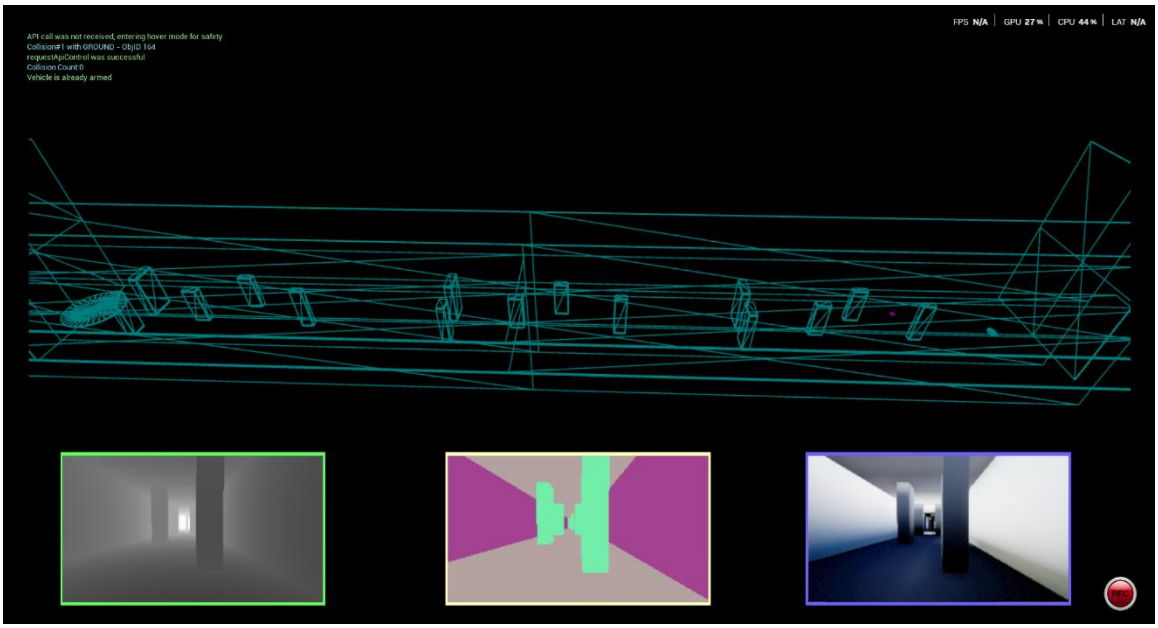

Figure 1 - Wireframe View of 3D Obstacle Course

The training simulations will use standardized LiDAR encodings and performance will be evaluated based on metrics such as **success rate, collision rate, path efficiency**, as well as time-to-goal, and learning stability, providing a comprehensive view of each algorithm's strengths and limitations.

The objective of this research is to create a scalable, learning-driven UAV navigation system capable of safely and autonomously reaching target locations while adapting to stochastic environments. **In this extended work, we implement and compare three algorithms—DQN, PPO, and DDPG—to analyze their behavior across different policy families (on-policy vs. off-policy) and control spaces (discrete vs. continuous).**

Each algorithm will be tested and evaluated under equal configurations and parameter settings to compare performance. This comparison will help determine the most effective approach for future enhancements.

Overall, the key contributions of this work include:

1. Integration of **Deep Q-Networks (DQN)**, **Proximal Policy Optimization (PPO)**, and **Deep Deterministic Policy Gradient (DDPG)** into a unified simulation-based framework for UAV navigation, enabling a systematic comparison of on-policy and off-policy learning strategies.
2. Development of a **LiDAR-centric, modular training pipeline** that connects Microsoft AirSim with deep reinforcement learning models, supporting flexible experimentation with different agent configurations, sensor inputs, and 3D environmental layouts.
3. Implementation of carefully designed **reward structures, discrete and continuous action models**, and tunable configurations that support real-time decision-making and obstacle avoidance in cluttered or unfamiliar environments.
4. A **comprehensive evaluation framework** measuring success rate, efficiency, stability, and generalization to analyze which algorithm learns faster, adapts better, and operates more safely in diverse conditions.

CHAPTER 2 - Review of Literature

The integration of reinforcement learning (RL), deep reinforcement learning (DRL), and simulation tools like AirSim has significantly advanced the field of UAV path planning. DRL, in particular, has enabled agents to handle complex navigation problems by combining traditional reward-driven strategies with the pattern recognition capabilities of deep neural networks. Mnih et al. (2015) introduced the Deep Q-Network (DQN), which demonstrated human-level performance in challenging environments, laying the foundation for many subsequent UAV navigation applications in simulated and real-world contexts.

AirSim has emerged as a popular platform for testing and validating these strategies due to its high-fidelity, customizable physics environment. It allows for the emulation of real-world UAV flight dynamics using configurable sensors such as GPS, RGB cameras, and lidar. Shah et al. (2018) first introduced AirSim as a research tool, and since then, several studies have leveraged it for RL-based UAV navigation. For instance, Tu and Juang (2023) employed Q-learning within AirSim's 3D environment, showing that RL methods could adapt more effectively to dynamic conditions compared to traditional planners.

Conventional path planning methods like Dijkstra's and A* (Dijkstra, 1959; Hart et al., 1968) remain foundational, but they often fall short in large-scale or unpredictable environments. Koenig & Likhachev (2002) and Stentz (1995) attempted to address some of these issues with real-time dynamic variants, while LaValle (2006) and Choset et al. (2005) explored sampling-based methods like RRT. Yet, RL has proven more scalable by

allowing agents to learn policies through interaction, without requiring a complete map of the environment. Zaghbani et al. (2024) compared Q-learning with SARSA for 3D path planning, demonstrating that Q-learning generally outperforms SARSA in both efficiency and obstacle avoidance.

Hybrid approaches have also emerged, combining classical planning with learning-based methods to enhance performance in real-world scenarios. For example, Maw et al. (2021) proposed the iADA*-RL framework, which integrates global path planning using graph search with local DRL-based obstacle avoidance. Similarly, Li et al. (2021) presented a hybrid model that combined A* with Q-learning, significantly improving adaptability and execution speed in complex environments.

More recently, Wang et al. (2024) introduced FRDDM-DQN, an algorithm that merges Faster R-CNN for obstacle detection with a Q-learning variant to enhance real-time decision-making. Their experiments in military-inspired 3D simulations showed improved collision avoidance and navigation accuracy. Sun et al. (2024) tackled trajectory generation using B-splines and gradient-based optimization, showing how advanced mathematical models can produce smooth, safe flight paths in cluttered airspace.

Several studies have also pushed into multi-agent territory. Laudenzi investigated collaborative navigation using multi-agent DRL systems, focusing on applications like search and rescue or industrial automation. These setups demonstrated how multiple UAVs could learn to coordinate while avoiding obstacles and sharing limited airspace. Meanwhile, Xie et al. (2021) proposed a novel DRL method for navigating large-scale

environments using an adaptive sampling strategy, improving both path length and learning efficiency.

Taken together, the literature highlights the rapid evolution from classical planners to deep reinforcement learning as the dominant paradigm for UAV navigation. These works demonstrate that while algorithms like DQN, PPO, and hybrid variants can achieve strong results in simulation, most prior research remains constrained by simplified sensing assumptions and limited comparisons across policy families. In practice, UAV autonomy depends not only on robust planning but also on the reliability of its perception systems and the scalability of its training process.

This study adopts a **LiDAR-centric perspective**, emphasizing range-based sensing as the backbone of UAV decision-making. LiDAR provides geometric precision and resilience in conditions where cameras or GPS may fail, yet its integration into reinforcement learning for UAVs is less explored, especially under noisy or dynamic conditions. By prioritizing LiDAR representations and reward structures, our framework directly addresses this gap.

Equally important is the distinction between **on-policy and off-policy learning paradigms**, which is not as systematically studied in UAV navigation. On-policy methods such as PPO offer stability and robustness but at a higher sample cost, while off-policy methods such as DQN and DDPG promise efficiency but introduce challenges in stability and overestimation. By including **DDPG** alongside DQN and PPO, this work expands the comparative space to cover both discrete and continuous action settings, enabling a fuller picture of what trade-offs exist for UAV control.

Ultimately, this synthesis of literature leads to a central gap that shows there is no unified, sensor-aware evaluation of on- and off-policy algorithms within the same LiDAR-driven framework, under consistent conditions of domain randomization and curriculum training. This study fills this gap by standardizing inputs, environments, and evaluation metrics, allowing us to compare algorithms with equal footing. This analysis not only clarifies which methods learn more efficiently or robustly but also establishes a benchmark for future extensions to multi-agent and real-world deployments.

CHAPTER 3 – Core Concepts

3.1 - LiDAR Sensing for UAV Navigation

Autonomous UAV navigation depends heavily on the quality and reliability of perception systems. Among the many sensors available, **LiDAR (Light Detection and Ranging)** has become a standard in both robotics and aerial systems. LiDAR works by emitting laser pulses and measuring the time-of-flight of reflected signals to calculate precise distances. This data can be assembled into point clouds, occupancy grids, or simplified scan vectors, giving UAVs a detailed understanding of their surrounding environment. Compared to other sensing modalities, LiDAR offers several distinct advantages. Unlike cameras, LiDAR performance is unaffected by changes in illumination, making it reliable in low-light, shadowed, or visually degraded environments. Unlike GPS, it remains functional indoors, in forests, or in urban canyons where satellite signals are obstructed. These strengths make LiDAR particularly effective for **collision avoidance and adaptive path planning**, where UAVs must continuously evaluate both free space and obstacles in real time.

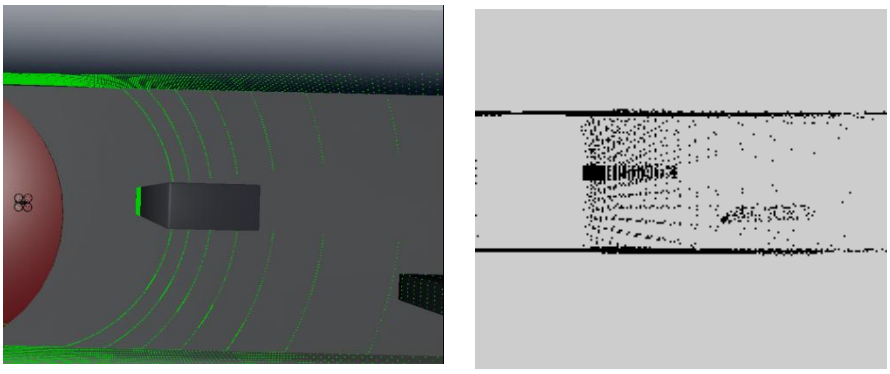

Figure 2 - Top Down Lidar Scan Visualization: Point Cloud Representation

In this study, LiDAR serves as the **primary observation modality**. The data informs not only the UAV's perception of its surroundings but also directly influences the **reward function**. For example, near-collision distances can be penalized while forward progress toward a goal is rewarded, and smoothness of flight can be encouraged through penalties on abrupt directional changes. By centering reinforcement learning around LiDAR data, the training process remains grounded in a sensing approach that is widely adopted in real-world UAVs and can be realistically modeled in simulation. This choice strengthens the relevance of the learned policies for potential transfer beyond simulation.

3.2 - On-Policy vs. Off-Policy Algorithms

Reinforcement learning algorithms can be broadly categorized into **on-policy** and **off-policy** families, each with different strengths and limitations. Understanding these paradigms is critical to evaluating which algorithms are most effective for UAV navigation tasks.

- **On-policy methods**, such as Proximal Policy Optimization (PPO)learn exclusively from data generated by the current version of the policy. This ensures that the training process remains stable and closely aligned with the agent's actual behavior. The primary strength of on-policy methods is their **robustness and stability**, even in highly dynamic environments. However, this comes at the cost of **sample efficiency**, since experiences cannot be reused once the policy is updated.

- **Off-policy methods**, such as Deep Q-Networks (DQN) and Deep Deterministic Policy Gradient (DDPG), learn from replay buffers that store past experiences. This allows them to be **far more sample-efficient**, reusing data across multiple updates. Off-policy algorithms are often faster to learn in terms of interactions with the environment, but they are also more prone to instability, overestimation of value functions, and sensitivity to hyperparameters. The stability of training depends heavily on mechanisms such as target networks, exploration noise, and replay buffer management.

For UAV applications, this distinction has important implications. On-policy methods may provide steady improvements under noise and randomization, while off-policy methods may reach higher performance with fewer environment samples but risk unstable convergence. A direct comparison under identical sensing and environmental conditions is necessary to understand these trade-offs in practice. This thesis investigates three representative algorithms, each illustrating different points along the on-policy/off-policy spectrum:

- **Deep Q-Networks (DQN)**: An off-policy, value-based method operating on a **discrete action space**. It is well-suited for motion primitives such as "move forward," "turn left," or "turn right." DQN demonstrates how discrete decisions can be learned efficiently through replay buffers and target networks.
- **Proximal Policy Optimization (PPO)**: An on-policy, policy-gradient method that can be implemented with either discrete or continuous actions. Its clipped

surrogate objective and entropy regularization provide stability during updates, making it a reliable baseline for UAV navigation under randomization.

- **Deep Deterministic Policy Gradient (DDPG)**: An off-policy, actor–critic algorithm designed for **continuous action spaces**. It allows fine-grained control over UAV velocity and orientation. DDPG offers the potential for smoother trajectories but introduces additional challenges in exploration and critic overestimation.

Together, these three algorithms span both discrete and continuous action settings as well as on- and off-policy families. This enables a **systematic comparison** across stability, sample efficiency, and real-time adaptability—dimensions that are directly relevant to UAV deployment in stochastic environments.

CHAPTER 4 – Algorithm Study

The core of this research lies in applying Deep Reinforcement Learning (DRL) to enable UAVs to autonomously navigate through complex, obstacle-rich environments. This chapter outlines the specific learning algorithms implemented—Deep Q-Networks (DQN), Proximal Policy Optimization (PPO), and Deep Deterministic Policy Gradient (DDPG) and explains how each contributes to the drone's ability to make intelligent, real-time decisions based on continuous feedback from its surroundings. A key component of path efficiency is measured using a trajectory cost function:

$$C = \sum_{k=0}^{n} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

Here, (xi,yi) are the coordinates along the drone's path, and n is the number of waypoints. This function computes the total distance traveled, which is critical for evaluating flight efficiency and energy consumption.

4.1 - Deep Q-Network (DQN)

DQN combines traditional Q-learning with deep neural networks, allowing agents to learn optimal policies in high-dimensional, continuous state spaces where tabular methods become infeasible showing its potential to advance both theoretical understanding and practical applications in autonomous systems (Jang, 2019) In the context of UAV navigation, the objective is to train the drone to traverse from a starting point to a goal while avoiding collisions and minimizing travel distance.

The core learning mechanism of DQN is the Q-value update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Where:

- sts_tst and st+1s_{t+1}st+1 are the current and next states,
- at is the action taken,
- rt+1r_{t+1}rt+1 is the immediate reward,
- α is the learning rate,
- γ is the discount factor.

This update allows the drone to incrementally improve its policy by estimating long-term rewards. Mnih (2013) showed that over time, the DQN agent learns to associate specific actions with higher expected returns, ultimately converging on a policy that favors efficient, collision-free flight paths.

The Bellman equation provides the theoretical foundation for these updates:

$$V(s) = max_a\{\sum_{s'} P(s, a)[R(s, a, s') + \gamma V(s')]\}$$

This reflects the maximum expected return from any state s, accounting for transition probabilities P, immediate rewards R, and the expected value of subsequent states.

In our implementation, the DQN architecture utilizes a convolutional neural network (CNN) to process point cloud based inputs (e.g., lidar scans) and output discrete actions such as turning left, right, or moving forward. Experience replays and target networks are also incorporated to improve learning stability.

4.2 – Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is an on-policy algorithm that improves training stability by restricting the magnitude of policy updates. Unlike DQN, which operates on discrete actions, PPO is capable of handling both discrete and continuous action spaces, making it particularly suitable for dynamic UAV navigation tasks.

The key innovation of PPO lies in its clipped objective function, which prevents excessively large updates that can destabilize learning:

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

Here:

- $r_t(\theta)$is the probability ratio between the new and old policy,
- $\hat{A}_t$is the estimated advantage function,
- $\epsilon$is a small hyperparameter (commonly 0.1–0.2) controlling the clipping range.

This clipped formulation ensures that updates neither overshoot nor under-adjust, leading to stable and consistent performance improvements (Schulman, 2013).

For UAV navigation, PPO excels in environments with complex, continuous state-action relationships, allowing the agent to adapt its trajectory in real-time with smoother flight paths compared to discrete methods.

4.3 – Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is an off-policy algorithm designed specifically for continuous action spaces. Unlike DQN, which selects from discrete choices, DDPG outputs continuous control signals, a helpful capability for UAV systems where fine-grained control of velocity, pitch, or yaw is required.

Lillicrap's (2015) DDPG combines two core ideas:

1. **Actor-Critic Framework**:
    - The **actor network** outputs deterministic actions directly from states.
    - The **critic network** evaluates these actions by estimating Q-values.
2. **Experience Replay with Target Networks**:
    - Transitions are stored in a replay buffer, and minibatches are sampled to stabilize learning.
    - Target networks for both actor and critic are slowly updated to avoid instability.

The policy gradient update is defined as:

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s_t \sim \mathcal{D}}[\nabla_a Q(s, a \mid \theta^Q) \mid_{a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s_t \mid \theta^\mu)]$$

Here:

- $\mu(s_t \mid \theta^\mu)$is the deterministic policy (actor),
- $Q(s, a \mid \theta^Q)$is the critic's evaluation,
- $\mathcal{D}$is the replay buffer.

In UAV navigation, DDPG enables precise control over continuous movements, making it particularly effective in scenarios where smooth maneuvering is required (e.g., tight obstacle avoidance or energy-efficient path adjustments).

CHAPTER 5 - Software Stack & Code Implementation

This project is implemented in Python, chosen for its mature machine learning ecosystem and rapid prototyping workflow. PyTorch provides the core tensor structuring and graphing engine; Stable-Baselines3 (SB3) supplies robust, interchangeable DRL implementations, enabling the same environment, callbacks, and logging to be reused across DQN, PPO, and a DDPG-ready path. NumPy supports numerical preprocessing, and OpenCV is available when visual inputs are required.

As mentioned, the simulator is **Microsoft AirSim** on **Unreal Engine**, accessed through AirSim's Python API for command-and-sensor I/O. Scenario parameters (e.g., environment layout, sensor settings, weather toggles) are maintained in editable **JSON** files so experiments can be reconfigured without touching core code.

5.1 - High Level Architecture

The codebase is structured into **three major modules** to help facilitate a clean and logical flow of implementation. This modular design ensures that the training system remains flexible, organized, and easy to transition when introducing new algorithms or sensors.

The **Environment Interface** serves as the foundation of interaction between the reinforcement learning algorithms and the AirSim simulator. It follows standard RL conventions by implementing two core methods:

- **reset()** – Responsible for spawning the UAV into the AirSim environment, initiating takeoff, and stabilizing the drone at the target altitude. During this

phase, the system applies a short grace period to filter out any sensor noise and to ensure reliable state initialization.

- **step(action)** – Executes a single control command from the learning agent. The method translates this command into AirSim's API calls and returns the new observation, computed reward, and termination status.

The Environment Interface maintains **consistent goal logic** and **reward evaluation** across all algorithms. It also handles the mapping from reinforcement learning ***actions*** to AirSim control commands. In **discrete mode (DQN)**, each action corresponds to a **predefined motion primitive** such as moving forward, ascending, turning left, or descending. In **continuous mode (PPO/DDPG)**, the agent **outputs a vector** that directly represents **yaw rate and linear velocities** $[\boldsymbol{yaw\_rate}, \boldsymbol{v_x}, \boldsymbol{v_y}, \boldsymbol{v_z}]$**,** allowing for smoother and more precise maneuvering. This module also **computes the observation state** by combining **LiDAR scan data** with UAV ego-state information such as **body-frame velocities, altitude error, and the goal vector.** The result is a compact yet **informative state representation** that captures both environmental geometry and task context. The implementation of observation spaces is further is discussed in **Section 5.2**

The **Training Module** coordinates the learning loop of the agent, controlling how episodes are executed, how experience is gathered, and how models are updated and stored. It is designed to operate efficiently while providing checkpoints and evaluation feedback during training.

This module handles:

- **Training loops** – Running episodes, collecting experiences, and updating model parameters.
- **Checkpointing** – Automatically saving model weights and optimizer states at set intervals, ensuring that progress can be resumed or analyzed at any point.
- **Evaluation** – Conducting periodic test episodes using the current model to measure learning progress over time.

The Training Module integrates with **Stable-Baselines3 (SB3)**, a PyTorch-based library that provides well-tested implementations of **DQN, PPO, and DDPG** behind a uniform API. This lets the same pipeline (environment wrappers, callbacks, logging, and checkpointing) run across all three agents without changing core code. SB3 supplies ready-to-use policy classes for discrete and continuous control, action bounding, built-in evaluation routines, TensorBoard logging, and reliable save/load functions (Raffin, 2021). It also supports vectorized environments to speed up data collection when needed. In short, SB3 handles the training loop details so we can focus on observation design, reward shaping, and experiment setup while keeping results reproducible across algorithms.

The **Data Logging Module** records the performance and behavior of the UAV throughout training. Logging plays an essential role in monitoring and diagnosing performance trends and comparing algorithmic efficiency. The system logs data through **TensorBoard Metrics,** these include episode rewards, success rates, collision counts, time-to-goal, and auxiliary signals such as minimum forward clearance and goal distance. These live dashboards make it easy to visualize training stability and detect anomalies.

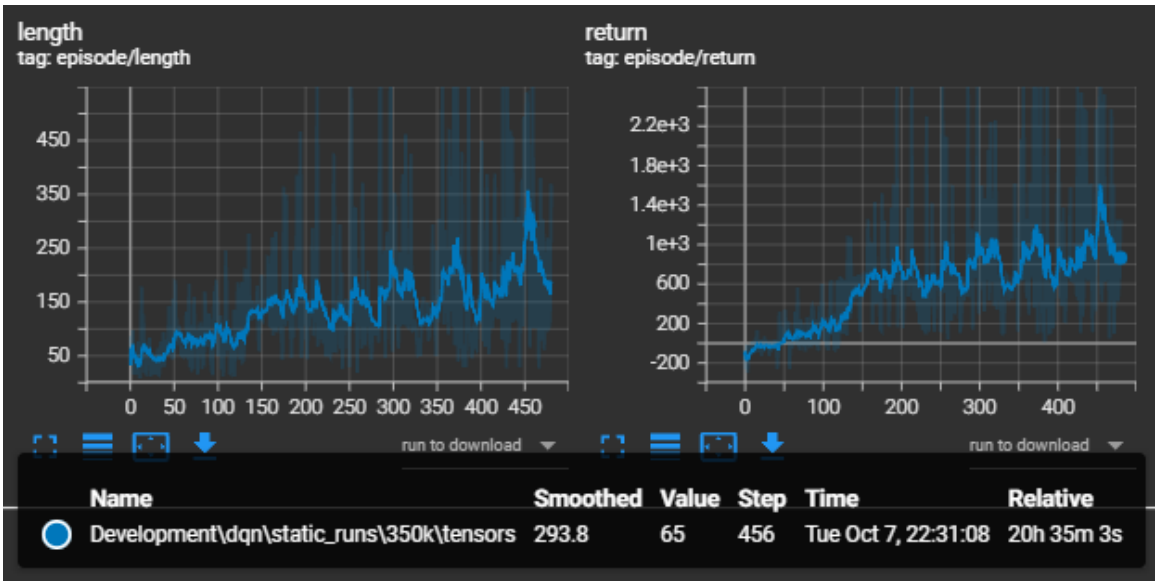

Figure 3 - TensorBoard Logging for 350,000 DQN Timesteps: Episode Length (left) and Reward (Right)

Additionally, the Data Logging Module maintains a structured directory system for all experiment runs and checkpoints. Each training session automatically generates folders for checkpoints, best models, and logs, ensuring that experiments remain traceable and reproducible. Together, these three modules, **Environment Interface**, **Training Module**, and **Data Logging**, form the backbone of the system. Their modular structure allows for clear debugging, easier algorithm swaps, and consistent performance measurement across training sessions.

5.2 – Observation & Action Spaces

To enable the UAV to learn intelligent behavior in simulation, it must first perceive its surroundings and express control through well-defined input and output channels. The Observation Space and Action Space together define this interaction.

5.2.1 - Observation Space

The observation space provides the agent with the sensory and state information it needs to make navigation decisions. In this system, observations are composed of two major components: LiDAR-based environmental perception and ego-state awareness.

1. **LiDAR Perception**

   A simulated 360-degree LiDAR sensor continuously scans the UAV's surroundings. Raw point data are down sampled into a fixed-length vector that represents relative obstacle distance at evenly spaced angular intervals. This transformation serves two goals It provides a compact yet informative representation that captures spatial layout without overwhelming the neural network with raw point-cloud data.

   - It normalizes distance readings into the range $0,1$, allowing the network to generalize sensor values across different scenes and scales.
   - This LiDAR vector effectively tells the agent where free space exists and how close obstacles are in any direction, which is critical for collision-free flight.

2. **Ego-State and Goal Information**

   Alongside LiDAR, the observation includes state features that describe the UAV's own motion and mission objective:

   - Body-frame linear velocities in the x, y, and z directions.
   - Altitude error, computed as the deviation from the desired flight band.
   - Normalized goal distance, indicating progress towards course completion.

Combining these signals allows the agent to interpret not only the layout of its environment but also its own dynamic relationship to the goal coordinate. The resulting observation is a single concatenated vector that encodes geometry, motion, and task context in a compact form suitable for deep-learning models.

5.3.2 - Action Space

The action space defines how the UAV interacts with the environment, whether through discrete motion commands or continuous control signals.

1. **Discrete Control (DQN Mode)**

   In discrete mode, the agent selects from a set of predefined motion primitives. Each action corresponds to a short, bounded movement such as Move forward, Strafe left / right, Ascend / descend and Yaw left / right. This configuration simplifies the control problem and ensures predictable movements during early training. It is especially effective for value-based algorithms like DQN, which benefit from a finite, well-defined action set.

2. **Continuous Control (PPO and DDPG Modes)**

   For policy-gradient algorithms, the system employs a continuous control space represented by a four-dimensional action vector:

$$[\boldsymbol{yaw_{rate}}, \boldsymbol{v_x}, \boldsymbol{v_y}, \boldsymbol{v_z}]$$

Here, v_x, v_y, v_z denote body-frame linear velocities, and *yaw rate* specifies angular rotation around the vertical axis. Each element is constrained within physical limits defined by UAV dynamics to maintain stable flight. Continuous control enables smoother

trajectories and finer adjustment of speed and orientation compared to the discrete approach.

Defining the observation and action spaces in this way strikes a balance between computational efficiency and representational richness. The LiDAR down sampling and state fusion create a compact input vector that still preserves all spatial cues needed for obstacle avoidance. Maintaining identical observation and reward definitions across algorithms ensures that performance differences arise purely from learning behavior rather than from inconsistent perception or control models.

5.3 – Reward Shaping Metrics

The reward structure defines how the UAV interprets progress, safety, and efficiency during training. It acts as the feedback signal that determines whether a flight maneuver contributes positively or negatively toward achieving the mission goal. Each term within the reward function is designed to promote controlled, purposeful, and safe motion while maintaining efficiency in navigation.

5.3.1 - Reward Components

The total reward combines several weighted objectives, each targeting a specific behavioral outcome. Goal Advancement serves as one of the primary reward flags. The UAV receives a positive increment whenever it moves closer to the target coordinate, encouraging forward progression. Obstacle Avoidance of course is applied if objects are detected within a minimum safety margin ahead of the vehicle. This helps prevent reckless approaches toward walls or tight spaces. Reward increments are given for

Centering, or the maintenance of equal distance between lateral obstacles, ensuring that the UAV stays within open corridors rather than hugging one side. Deviations from the desired flight altitude result in small penalties to maintain vertical control, allowing Altitude Stability to be a minor but necessary influence on the decision output. For Motion Smoothness, the system implements a slight penalty when consecutive commands differ drastically, discouraging sharp, erratic changes in motion. Lastly, Terminal Events in the reward calculation result in large positive rewards being granted for successful goal completion, while collisions result in heavy penalties.

This multi-faceted reward structure ensures the agent learns to not only value reaching the goal, but do so with balance, precision, situational awareness, and most importantly optimized efficiency.

5.3.2 - Motion-Influencing Metrics

Beyond the reward function, the learning system's perception of motion depends on how *often* it can observe the world and update its control commands. These temporal parameters form what can be called the "heartbeat" of the UAV's learning loop.

Each decision cycle has two key aspects:

1. **Decision Frequency** – How many times per second the UAV can evaluate sensor data and choose a new action.
2. **Command Duration** – How long each chosen action is applied before the next decision is made.

These temporal characteristics directly shape the UAV's motion and learning behavior.

- **Higher Decision Frequency** leads to more reactive flight control. The UAV can make fine corrections around obstacles but may produce jittery or unstable motion if learning is incomplete.
- **Lower Decision Frequency** smooths out control and stabilizes motion but can cause sluggish responses to sudden changes in the environment.

This decision can be the difference between a drone agent making long uninterrupted strides in open vs making constant calculated decision even when not around any obstacles.

- **Short Command Duration** allows tight maneuvering at the cost of greater computational load.
- **Long Command Duration** produces smoother, more energy-efficient trajectories but limits agility.

Fune tuning this metric results in the drone's ability to determine when constant corrections are necessary around things like curves and corners.

The combination of these parameters defines how granularly the UAV perceives and interacts with its surroundings. They also influence how reward feedback aligns with motion, faster decision loops generate more frequent, smaller rewards, while slower loops yield sparser but larger updates. Different algorithms respond uniquely to these timing characteristics. Value-based methods like DQN generally perform better with slower decision cycles that provide stable, discrete outcomes, whereas policy-gradient algorithms such as PPO and DDPG leverage finer time resolutions to refine continuous actions and improve control fluidity.

Together, reward shaping and motion-influencing metrics determine not only what the UAV should learn, but **how** it learns, the rhythm of its feedback, and the pace at which intelligent flight behavior evolves.

5.4 – AirSim Integration & Training Episode Lifecycle

The AirSim interface is the operational core of this system, managing the interaction between the reinforcement learning agent and the simulated environment. Each training episode represents a self-contained flight mission, beginning at takeoff and concluding upon collision, timeout, or goal completion. Within that episode, the UAV experiences a sequence of decision steps, each one representing a full perception–action–feedback cycle.

5.4.1 - Integration Workflow

The software stack integrates with AirSim through its Python API, which provides real-time access to vehicle state, LiDAR readings, and control commands (Shah et al, 2017). The environment runs as a continuous simulation while the RL agent communicates asynchronously through a loop of reset, action, and observation calls. The cycle proceeds as follows:

1. **Environment Initialization**

   When an episode begins, the environment spawns the UAV at a designated start position and commands a short takeoff maneuver to the target altitude. A brief stabilization period follows, ensuring that early sensor readings are noise-free and that the system has a consistent baseline state before learning begins.

2. **State Observation**

   Once stabilized, the UAV gathers data from onboard sensors, most critically, the LiDAR range scan, body-frame velocity, altitude error, and relative goal vector. These values are fused into a single observation vector that represents the UAV's perception of the surrounding environment and its own state.

3. **Action Selection**

   The reinforcement learning agent processes the observation and outputs an action, either as a discrete command (in DQN) or a continuous control vector (in PPO/DDPG). This represents the UAV's decision at that instant, its next motion intent.

4. **Action Execution**

   The command is sent through the AirSim API, producing a physical motion such as moving forward, rotating, or adjusting altitude. The UAV maintains that command for a fixed control duration before new sensor data are sampled.

5. **Reward Evaluation and Transition**

   At the end of each control duration, the system calculates a reward value based on the UAV's progress, safety, and stability metrics. The environment then transitions from the current state to a new one, forming what is known in reinforcement learning as a *timestep transition*. Each transition records the tuple (state, action, reward, next state), which serves as the foundation for learning.

6. **Termination and Reset**

   An episode continues until a terminal condition occurs (goal reached or collision).

When a terminal event is triggered, the episode concludes, the UAV resets to its starting position, and a new episode begins with fresh conditions.

5.4.2 - Timestep and Episode Definition

Each timestep represents one full cycle of perception, decision, and response, a single unit of interaction between the UAV and its environment. The episode, in contrast, is the complete flight sequence composed of multiple timesteps. The first motion command after takeoff marks the beginning of the episode's first timestep. From that point, each subsequent action taken by the agent corresponds to one transition until termination. These transitions collectively define the UAV's trajectory and serve as the experiential data through which learning occurs.

5.5 – Hardware Architecture

The training and simulation system was executed on a high-performance workstation configured specifically for reinforcement learning workloads. The hardware was selected to ensure stable operation of Microsoft AirSim, Unreal Engine rendering, and concurrent deep learning processes under continuous load.

The configuration includes an **Intel Core i7-12800H processor**, **32 GB of DDR5 RAM**, and an **NVIDIA RTX 3070 Ti GPU**. This combination provides sufficient compute density and memory bandwidth to sustain both real-time simulation and GPU-accelerated training without performance degradation. The RTX 3070 Ti, with its large number of CUDA cores and high VRAM capacity, enables parallelized neural network computation for DQN, PPO, and DDPG agents.

System performance is supported by **NVMe SSD storage** to accelerate dataset reads, model checkpointing, and TensorBoard logging. High throughput storage also minimizes latency during model saving and environment resets, keeping the training loop continuous and efficient.

This configuration was chosen after preliminary experiments on lower-tier machines revealed clear stability and performance limitations. The final setup ensures that:

- GPU acceleration supports frequent gradient updates and experience replay sampling.
- Memory and storage bandwidth handle concurrent LiDAR streaming, environment rendering, and neural network computation.
- CPU multithreading accommodates Unreal Engine physics simulation while maintaining synchronization with the reinforcement learning process.

Collectively, this architecture provides a reliable foundation for large-scale reinforcement learning experimentation, maintaining consistent framerates, minimal episode lag, and reproducible results across algorithms.

CHAPTER 6 – Experiment Methodology: Training & Evaluation

This chapter outlines the experimental procedure used to train, validate, and compare the performance of the three reinforcement learning algorithms. The focus is on how training was conducted across multiple timesteps, how progress was recorded through checkpointing, and how performance was evaluated using consistent success metrics.

6.1 - Training Procedure

Training sessions were conducted using the modular software stack previously described, with AirSim serving as the simulation environment and Stable Baselines3 providing the RL implementations. Each run consists of a sequence of **episodes**, where the UAV begins from a fixed spawn point, completes its flight objective, and resets upon collision, goal achievement, or timeout. The system is configured to save **model checkpoints** at predefined intervals of total timesteps. These checkpoints allow both continuation of training and performance evaluation at intermediate stages. Across all algorithms, training sessions were executed for a range of total timesteps ranging anywhere from *150 000 – 700 000*, adjusted per algorithm based on need for continued training until potential overfitting. Each checkpoint folder contains:

- The serialized model weights and optimizer state.
- Environment statistics required to resume training.
- TensorBoard summaries of cumulative reward, episode length, and success rate.

This structure enables flexible experimentation, model training can be paused, extended, or evaluated independently from any saved state. During training, unified reward and

observation definitions as well as environmental conditions remain consistent across agents. As stated, this ensures that performance differences are attributable to algorithmic behavior rather than environmental variation.

6.2 - Evaluation Methodology

Performance evaluation is conducted periodically by loading saved checkpoints and executing a fixed number of test episodes without further learning.

The evaluation process measures **Episode Reward** by average cumulative return per episode. **Success Rate** is evaluated through a ratio of completed flights to total attempts. **Collision Rate is measured** as a frequency of obstacle contact events. Minimizing this metric shows a model ability to from safe paths. Those paths can be optimized by analyzing **Path Efficiency** as a ratio between optimal and actual trajectory lengths.

These metrics are logged through TensorBoard, providing temporal plots that capture the learning progression and convergence patterns for each algorithm. A standardized testing script replays each trained policy within the same simulation configuration used during training. This allows direct comparison of generalization and stability across algorithms at equal training durations.

6.3 - Timesteps and Convergence Tracking

Each timestep corresponds to one full interaction cycle between the UAV and the environment—observation, decision, action, and reward computation.

An episode consists of a finite sequence of timesteps; when the UAV reaches a terminal condition, the environment resets, and training resumes with a new episode. Model

checkpoints saved every *N* timesteps (e.g., every 10000 or 20000) provide insight into learning progression. By examining these checkpoints, convergence behavior can be inferred: whether learning stabilizes early or continues to improve with additional interaction time.

6.4 – Algorithm Comparison and Control Sensitivity Analysis

To evaluate both learning efficiency and behavioral robustness, multiple training runs were performed for all three algorithms under two control-tuning intentions. The objective was to study how each method responded to variations in control frequency and command duration, the parameters that govern how quickly the UAV reacts and how long each command persists.

6.4.1 - Baseline (Conservative Motion Parameters)

Initial training runs were executed using moderate, conservative control settings. These values correspond to a balanced cadence of decision-making, producing smooth, stable flight trajectories with limited oscillation. Under these conditions, each algorithm was trained for extended durations as needed until reward values rise and level.

The conservative configuration emphasizes flight stability and path efficiency, allowing clear measurement of convergence speed and general control competence. This phase served as a baseline for comparing later results with more aggressive control dynamics.

6.4.2 - Aggressive Control Configuration

Following the baseline evaluation, additional training sessions were conducted using an **aggressive motion profile**. In this mode, the UAV executed faster decision cycles with

shorter command durations, producing quicker, sharper maneuvers. Each algorithm was trained for **150 000 timesteps** under this configuration to test short-term adaptability and resilience.

These aggressive settings increased the agent's reactivity to environmental changes and introduced higher temporal granularity in control decisions. However, they also demanded more precise policy updates, revealing how each algorithm handles faster feedback loops and potentially noisier state transitions.

CHAPTER 7- Results

7.1 - Model Evaluation Protocol

Once a point of training that exhibited consistent success was reached for each model, training was stopped and checkpoints were evaluated over **25 episodes** in the same AirSim scenario. For every episode, the system logged **Reward**, **Duration (secs)**, **and number of steps** using **Reward together with Duration** as the primary proxy for policy effectiveness. Under this reward design, longer, safer, longer flights accumulate more reward, indicating a more capable and robust motion policy. TensorBoard curves highlight per-algorithm **top checkpoints** by mean Reward,

7.2 - Conservative Control Loop

Across conservative control runs, **PPO** is the most efficient, reaching high reward with long, stable durations while needing fewer training steps and showing low variance late in training. **DDPG** is high-capability, but training-hungry, exploratory, continuous actions produce larger early oscillations; with additional exposure it converges and often approaches or matches PPO, giving reasonable expectation that, with enough timesteps and tuning, DDPG can surpass PPO on peak reward. **DQN** improves steadily but is structurally constrained by its discrete action set; its ceiling on reward and episode duration rises more slowly and typically remains below the continuous-control methods. Key takeaways from the evaluation show **PPO** exhibiting the strongest **average** performance across checkpoints and maintaining long flights; its **highest** episode rewards are about **2,500**, paired with durations on the order of a few hundred seconds. **DDPG** records **highest** rewards about **2,750** with very long flights—often several hundred

seconds—though with greater variance across episodes and checkpoints before stabilizing. **DQN** achieves **highest** rewards at about **2,520** with shorter typical durations than PPO and DDPG, consistent with a discrete-action ceiling and more conservative motion. Overall, under the more conservative control loop, **PPO** is the most efficient and stable, **DDPG** reaches comparably high peaks but with higher variance and more training demand, and **DQN** trails in cumulative reward and sustained flight time.

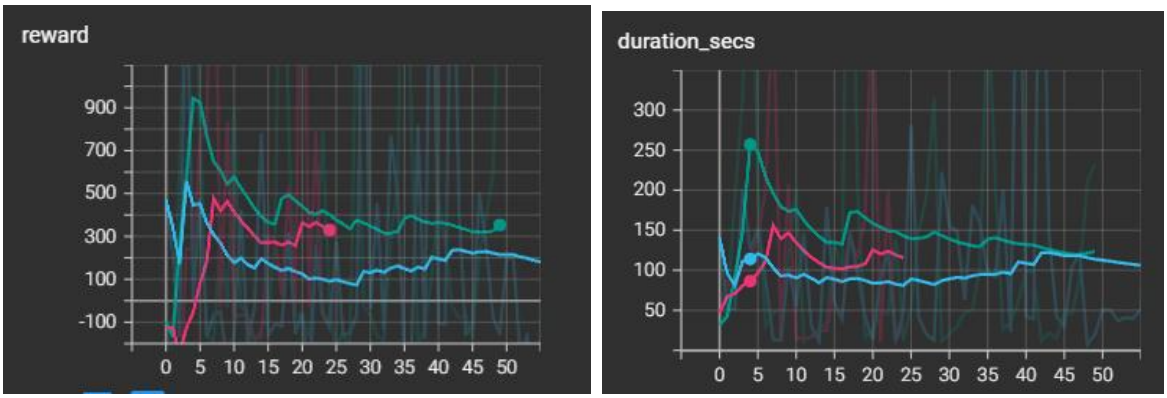

Figure 4 - TensorBoard Logging Showing PPO (Green) with higher reward/duration values than DQN (pink) and DDPG (Blue)

7.3 - Aggressive Control Loop

Under the more aggressive control settings, **DQN** exhibits the strongest peak capability and broad robustness at this horizon, with **highest** episode rewards about **2,200** and long flights on the order of **450–520 seconds**, including several clear successful trajectories; its discrete motion primitives appear to benefit from the faster decision cadence. **DDPG** lands in the middle: it shows **highest** rewards about **600–650** with flights roughly **70–140 seconds**, but with large variance across episodes (ranging from negative returns to strong positives), reflecting its exploratory continuous control needing more exposure to

stabilize under rapid commands. **PPO** underperforms at this training budget in the aggressive setting, with rewards clustered around **−25 to −30** and short flights of roughly **5–10 seconds**, indicating that 150k timesteps is generally insufficient for it to adapt to the faster control loop and shorter action persistence. This extra analysis where each model trained for 150,000 timesteps has **DQN** leading. **DDPG** shows partial adaptation with high variability, and **PPO** lags, consistent with the notion that aggressive timing favors discrete, well-bounded actions early while continuous policies require more interaction to realize their advantages.

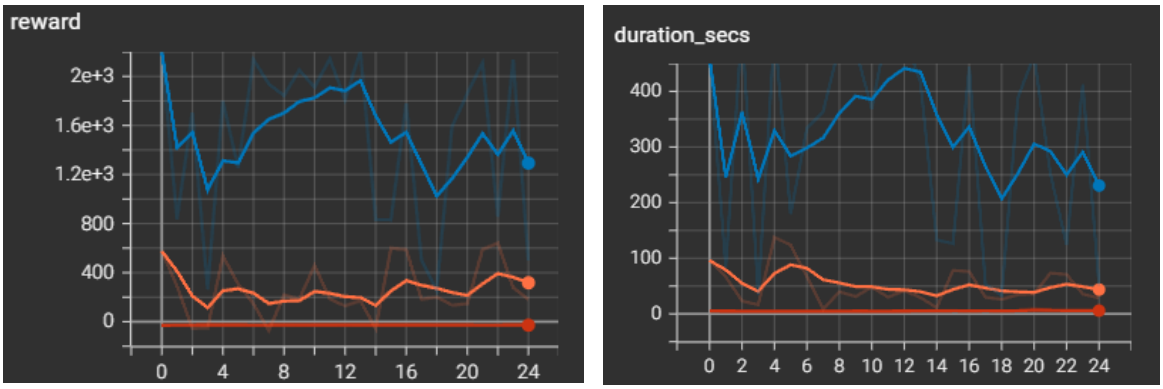

Figure 5 - TensorBoard Logging Showing DQN (Blue) with higher reward/duration values than both

CHAPTER 8 - Discussion & Interpretation

8.1 - Conservative Control Loop Interpretation

Under the conservative cadence (longer action persistence, lower decision frequency), PPO's clipped policy-gradient updates yield stable, incremental policy improvement. The algorithm resists over-correction and quickly finds a reliable sequence of actions, which translates into high average rewards with long, steady episode durations at comparatively lower training budgets. DDPG shows a higher potential ceiling, continuous actions allow fine-grained control and long flights, but it is training hungry. Early phases reflect broader exploration and variance; with sufficient exposure and tuning, performance converges and can reach or exceed PPO's peaks. DQN improves consistently under these metrics but remains action-space limited. Discrete primitives limit motion fluency, so reward and duration rise more slowly and typically remain below the continuous-control methods.

8.2 - Aggressive Control Loop Interpretation

With faster decision cycles and shorter command durations, bounded discrete actions become easier to exploit early. At 150,000 timesteps, DQN benefits from a reduced action search space and clear affordances (e.g., brief heading holds, small yaw corrections), producing high reward peaks and long flights more frequently than continuous methods. DDPG lands in the middle, continuous control plus exploration noise yields mixed early returns, from negatives to strong positives, before settling. Indicating that more interaction is needed to calibrate critic and actor under rapid dynamics.

PPO lags at this short horizon in aggressive timing; on-policy sampling and conservative updates mean it typically requires more data to adapt to the faster loop and shorter action persistence.

8.3 – State Transition Cadence: Conservative vs. Aggressive

Conservative timing smooths dynamics, lengthens the temporal footprint of each action, and makes credit assignment easier. Policies that update cautiously (PPO) compound small gains into long, stable flights. Aggressive timing injects rapid state changes; early learning favors simple, bounded choices (DQN) while continuous controllers (DDPG/PPO) need more samples to stabilize critic/policy estimates.

Discrete primitives are quick to exploit at fast cadence (DQN excels at 150k) but have a lower ceiling in rich maneuvers. Continuous actions (DDPG/PPO) have higher expressiveness and long-term ceiling, but they face struggle under more aggressive conditions due to less decision-making control before their advantage appears, implying that under aggressive conditions, discrete control methods prove more effective.

CHAPTER 9 - Conclusions, Limitations, and Future Work

9.1 – Conclusion

The results demonstrate that **control cadence** metrics are an impactful decision choice that must align with the learning algorithm and the available training budget. When **reliable stability is needed quickly**, a **conservative timing** pairs naturally with **PPO**, which converges to high reward and long, steady flights with fewer timesteps and low late-stage variance. When the goal is **fast early competence** under tight budgets, an **aggressive timing** setup favors **DQN**: bounded discrete actions exploit the rapid decision cadence to achieve strong peaks and frequent long flights at short horizons, after which control can be handed off to a continuous method for further refinement. If the aim is **maximum performance** and there is room to train and tune, **DDPG** (or modern continuous variants) benefits from progressively tightened timing as the critic and policy stabilize, its higher expressiveness supports very long flights and high peak rewards once variance collapses with additional exposure.

Under a **conservative control loop**, PPO is the most efficient achieving high reward with long durations in fewer steps and remains stable late in training. **DDPG** shows a higher ultimate ceiling, producing very long flights and high peaks, but it requires more interaction to settle, reflecting its exploratory continuous control; variance diminishes with sufficient exposure. With an **aggressive loop at 150k**, **DQN** leads on peak capability and robustness because discrete, well-bounded actions align with the fast cadence; **DDPG** is mixed but improving as training proceeds; **PPO** lags at this short horizon. Overall, the **timing of decisions and command duration** materially shape

which algorithm appears “best” for a given budget—**cadence is part of the problem definition**, not an afterthought, and should be chosen strategically to match stability needs, sample budgets, and performance targets.

9.2 – Limitations & Future Works

This study’s findings should be read in the context of several constraints that shape both sample efficiency and apparent ranking among algorithms. All experiments were conducted in simulation on a single high-end laptop GPU, which bounds batch sizes, parallel rollouts, and the breadth of hyperparameter exploration. While AirSim offers high fidelity, sim-to-real gaps remain. Scenario diversity was limited to static or moderately complex layouts; narrow corridors, dense clutter, and moving obstacles can be added to make training conditions more strenuous.

Future work addresses these limits directly. Training can be extended to substantially longer horizons, particularly for PPO and DDPG under aggressive timing, to observe late-stage variance collapse and potential asymptotic gains. Environment complexity will be expanded with tighter corridors, denser clutter, and moving obstacles to surface failure modes absent in simpler maps. Dedicated GPUs/CPUs and parallelized environments will enable larger batches, deeper model capacity, and broader hyperparameter analysis. Together, these steps will test whether the advantages observed under the conservative cadence persist in harder settings, how aggressive timing reshapes learning with longer exposure, and how cadence selection, algorithm choice, and compute scale interact when pushed beyond the present budget.

References

1 Jang, Beakcheol & Kim, Myeonghwi & Harerimana, Gaspard & Kim, Jong. (2019). Q-Learning Algorithms: A Comprehensive Classification and Applications. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2941229.

2 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," *arXiv preprint* arXiv:1509.02971, 2015.

3 A. Maw, M. Tyan, T. A. Nguyen, and J. W. Lee, "iADA*-RL: Anytime graph-based path planning with deep reinforcement learning for an autonomous UAV," *Applied Sciences*, vol. 11, no. 9, p. 3948, 2021.

4 V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

5 Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M., "Playing Atari with Deep Reinforcement Learning," arXiv preprint arXiv:1312.5602, 2013.

6 W. Sun, P. Sun, W. Ding, *et al.*, "Gradient-based autonomous obstacle avoidance trajectory planning for B-spline UAVs," *Scientific Reports*, vol. 14, p. 14458, 2024. doi: 10.1038/s41598-024-65463-w

7 G.-T. Tu and J.-G. Juang, "UAV path planning and obstacle avoidance based on reinforcement learning in 3D environments," *Actuators*, vol. 12, no. 2, p. 57, 2023.

8 A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

9 R. Xie, Z. Meng, L. Wang, H. Li, K. Wang, and Z. Wu, "Unmanned aerial vehicle path planning algorithm based on deep reinforcement learning in large-scale and dynamic environments," *IEEE Access*, vol. 9, pp. 24884–24900, 2021.

10 Zaghbani, R. Jarray, and S. Bouallègue, "Comparative study of Q-Learning and SARSA algorithms for UAV path planning in 3D environments," in *Proc. IEEE INES 2024*, pp. 245–250, July 2024.

11 S. Shah, D. Dey, C. Lovett, and A. Kapoor, “AirSim: High-fidelity visual and physical simulation for autonomous vehicles,” in *Field and Service Robotics: Results of the 11th International Conference*, Springer, pp. 621–635, 2018.

12 E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

13 S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.
H. Choset, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

14 P. Yao, H. Wang, and Z. Su, “Real-time path planning of unmanned aerial vehicle for target tracking and obstacle avoidance in complex dynamic environment,” *Aerospace Science and Technology*, vol. 47, pp. 269–279, 2015.

15 Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O., “Proximal Policy Optimization Algorithms,” arXiv preprint arXiv:1707.06347, 2017.

16 S. Koenig and M. Likhachev, “D* Lite,” in *Proc. Eighteenth National Conference on Artificial Intelligence*, pp. 476-483, 2002