Reinforcement Learning Based Motion Planner and Trajectory Tracker for Unmanned Aerial Systems

by

Shaswat Garg

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2025

© Shaswat Garg 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Unmanned Aerial Vehicles (UAVs) are increasingly used for autonomous tasks such as inspection, maintenance, and search-and-rescue. However, effective trajectory tracking and obstacle avoidance in dynamic environments remain challenging. Traditional optimization-based methods lack adaptability and computational efficiency, motivating the use of reinforcement learning (RL) for UAV control. This thesis explores RL-based UAV trajectory tracking through three key phases: benchmarking RL algorithms, developing a dual-agent RL framework, and leveraging these insights to design a model-free RL approach for Aerial Continuum Manipulators (ACMs).

First, off-policy RL algorithms including DDPG, TD3, SAC, and SoftQ, are benchmarked to evaluate their generalization from simulation to real-world UAV path planning. Results showed DDPG excelling in reward maximization while TD3 provided superior collision avoidance. These insights guided the development of a dual-agent RL framework for UAV trajectory tracking in cluttered environments. The system used two RL agents: one for velocity prediction and another for real-time collision avoidance, leveraging 3D point cloud data to eliminate memory-intensive obstacle representations. Simulated and real-world experiments demonstrated improved trajectory tracking, obstacle avoidance, and adaptability over single-agent and optimization-based approaches.

Building on this, a model-free RL framework for ACMs is introduced, which integrate UAV mobility with continuum robotic arms for dexterous aerial manipulation. Traditional RL struggles with constraint enforcement, leading to unsafe behaviors. To address this, a health-driven RL architecture is proposed that implicitly incorporates constraints through a secondary health reward, ensuring safe and stable operation. Using 3D point cloud data for navigation and a curriculum learning paradigm for scalability, the framework demonstrated superior performance over state-of-the-art RL and optimization techniques in trajectory tracking and constraint adherence.

This thesis benchmarks RL-based aerial control to evaluate performance and limitations. The insights gained are then used to develop dual-agent learning and safe RL algorithms for complex aerial systems. The findings lay the groundwork for future research in refining metrics, exploring additional algorithms, and incorporating vision-based RL for enhanced perception and decision-making.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Farrokh Janabi Sharifi and Baris Fidan, for their invaluable guidance and unwavering support throughout my MASc journey. Their insights and encouragement have been instrumental in shaping this thesis.

I am also immensely grateful to my mentor, Houman Masnavi, for his consistent support and willingness to assist me whenever I encountered challenges. His mentorship has been a cornerstone of my academic growth.

A heartfelt thank you to my parents, whose unconditional love and unwavering support have provided me with the strength and solace to persevere. Lastly, I want to extend my sincere appreciation to my girlfriend for her constant encouragement and assistance, which have been a source of motivation and comfort.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

List of Figures ix

List of Tables xii

1 Introduction 1

2 Background 5

2.1 Motion Planning and Trajectory Tracking 5

2.1.1 Search-Based Methods 6

2.1.2 Sampling-Based Methods 7

2.1.3 Optimization-Based Methods 8

2.1.4 Learning-Based Methods 9

2.2 Reinforcement Learning Preliminaries 11

2.2.1 Markov Decision Process 11

2.2.2 Policy 12

2.2.3 Value Functions 12

2.2.4 Bellman Equations . . . 12
2.2.5 Optimal Policy and Value Functions . . . 13
2.3 Reinforcement Learning Algorithms . . . 15
2.3.1 Deep Deterministic Policy Gradient . . . 15
2.3.2 Twin Delayed Deep Deterministic Policy Gradient . . . 16
2.3.3 Soft Q-learning . . . 18
2.3.4 Soft Actor-Critic . . . 19
2.4 Safe Reinforcement Learning Algorithms . . . 20
2.4.1 Safe Adversarially Guided Actor-Critic . . . 20
2.4.2 Safety Editor Policy . . . 22
2.4.3 Reachability Constrained Reinforcement Learning . . . 23
2.5 Aerial Continuum Manipulator Modelling . . . 24

3 Benchmarking Off-Policy Deep Reinforcement Learning Algorithms for UAV Path Planning 32
3.1 Path Planning Benchmark Development . . . 33
3.1.1 Simulation Setup and UAV Model . . . 33
3.1.2 State Space and Action Space Representation . . . 33
3.1.3 Reward Function Design . . . 34
3.1.4 Training Environment and Obstacle Configuration . . . 35
3.1.5 Generalization to Complex Environments . . . 36
3.2 Performance Evaluation of Reinforcement Learning Algorithms . . . 37
3.2.1 Neural Network Architecture and Training . . . 37
3.2.2 Evaluation Metrics . . . 37
3.2.3 Generalization to Unseen Environments . . . 39
3.3 Experiments . . . 40
3.3.1 Experimental Setup . . . 40
3.3.2 Deployment and Path Planning using TD3 . . . 43
3.3.3 Observations and Insights . . . 44
3.4 Summary . . . 45

4 Dual Agent Learning Based Aerial Trajectory Tracking **46**
4.1 Proposed Method . . . 47
4.1.1 Dual-Agent Architecture for Trajectory Tracking . . . 48
4.1.2 Trajectory Tracking Policy . . . 50
4.1.3 Collision Avoidance Policy . . . 51
4.1.4 Mechanism . . . 55
4.2 Methodology . . . 57
4.2.1 Simulation Pipeline . . . 57
4.2.2 UAV and Sensor Setup . . . 57
4.2.3 Curriculum Learning for Policy Training . . . 57
4.3 Simulated Results . . . 60
4.3.1 Static Environment Case Study . . . 63
4.3.2 Warehouse Case Study . . . 65
4.3.3 Drone Race Case Study . . . 67
4.3.4 Dynamic Obstacle Case Study . . . 67
4.3.5 Discussion . . . 68
4.4 Experiments . . . 69
4.5 Summary . . . 70

5 Safe Reinforcement Learning for Trajectory Tracking of Aerial Continuum Manipulation in Unknown Constrained Environments **72**
5.1 Health Driven Reinforcement Learning . . . 73
5.2 Simulation Setup and Results . . . 79
5.2.1 Random obstacle scenario . . . 83
5.2.2 Two Aerial Continuum Manipulators together . . . 83
5.2.3 Dynamic Obstacles . . . 87
5.3 Real World Experiments . . . 88
5.4 Summary . . . 89

6 Conclusion **90**

6.1 Limitations 91

6.2 Future Work 91

References **93**

List of Figures

2.1 Coordinate frames in aerial continuum manipulator. 25

3.1 Parrot bebop 2 UAV 35
3.2 The gazebo environment used for training RL algorithms. The environment contains two cylindrical obstacles and a UAV attached with LiDAR sensor. 36
3.3 Neural networks representing (a) state value function, (b) Q-function and (c) policy function used for training the RL algorithms. 38
3.4 Comparison of rewards achieved by TD3, SAC, DDPG, and SoftQ algorithms after sampling 50K episodes. 38
3.5 The obstacle worlds used to test different RL algorithms. (a) 4 obstacles, (b) 5 obstacles, (c) 6 obstacles, (d) 7 obstacles, (e) 9 obstacles and (f) 22 obstacles. 41
3.6 Resulting paths after testing the RL algorithms on environments having (a) 4 obstacles, (b) 5 obstacles, (c) 6 obstacles (d) 7 obstacles, (e) 9 obstacles, and (d) 22 obstacles. 42
3.7 Experimental setup consisting of three obstacles, two triangular prisms and one cuboid. 43
3.8 The path traversed by the UAV while navigating through the obstacles. The external camera view, UAV camera view and RViZ simulation are shown together. 44

4.1 The pipeline for our dual-agent architecture. 49
4.2 Schematic diagram of UAV trying to maneuver through obstacles. There is a UAV in the middle (green circle), with a LiDAR scan around it (blue circle). O_1, O_2 and O_3 are the obstacles, with different scenarios where the UAV is moving have either velocity v_1, v_2, v_3 or v_4. 54

4.3 An illustration of collision set S_{col} (green) and trajectory set S_{trt} (orange) with two policies π_{cva} and π_{trt}. On the left, with the naive approach there is a constant back and forth between the collision avoidance policy π_{cva} and trajectory tracking policy π_{trt}. On the right, by applying the appropriate switching mechanism the collision avoidance agent pushes the UAV away from obstacle and switches to trajectory tracking policy only when safe. . . 55

4.4 Curriculum training for trajectory tracking policy π_{trt} and collision avoidance policy π_{cva}. (a) The environment used to train $\pi)trt$ seen from X-Y plane (b) For π_{cva} it starts with a simple two obstacle world containing two cylinders with radius 0.1 m. Following that the policy is trained in an environment where the radius of the cylinder is 0.3 m. Finally it is trained in an environment containing cuboids. 59

4.5 Effect of network architecture on performance of the proposed algorithm. . 61

4.6 Comparison of trajectories generated in different obstacle worlds by RL algorithms, EP and OTT. (a) 5 obstacles, (b) 9 obstacles, (c) Random world with 22 obstacles. In all three scenarios, the proposed dual-agent algorithm tracks the trajectory more precisely, whereas OTT exhibits significant deviations along the Z-axis and produces a suboptimal trajectory, with EP failing to track the trajectory optimally. 64

4.7 Performance of different algorithms in a complex warehouse. The OTT algorithm fails to find a path in the warehouse's confined space, while the proposed method successfully navigates to the goal pose, following the trajectory and avoiding obstacles. The EP algorithm initially follows the trajectory well but gets stuck at the obstacle while attempting to make the turn. After maneuvering around the obstacle, it fails to rejoin the trajectory 66

4.8 Drone racing environment. Our proposed algorithm closely follows the trajectory, successfully passing through the hoops. In contrast, OTT and EP fails to generate an optimal trajectory, leading to sub-optimal navigation. . 67

4.9 Holybro X500 V2 UAV. 69

4.10 Comparison of trajectories generated in a real world experiments by the proposed algorithm. UAV tracking (a) straight line trajectory in a single obstacle scenario, (b) straight line trajectory in two obstacle environment, (c) curve trajectory in a three obstacle scenario and (d) snapshot of third person view of the real world. 70

5.1 Aerial continuum manipulator. 79

5.2 Comparison of rewards achieved by proposed algorithm, RCRL, SAAC, and SEditor algorithms after sampling 10K episodes. 82

5.3 Four different navigation scenarios where an ACM navigates through an obstacle-filled environment. The reference trajectory is shown in green, while the UAV and continuum arm trajectories are represented in red and blue, respectively. (a) and (b) depict spline-based trajectories, demonstrating smooth navigation through obstacles. (c) involves a complex trajectory requiring intricate maneuvers, while (d) follows a square-shaped path, testing the ability to maintain structured navigation. The start and goal positions are marked in yellow and brown, respectively, with gray objects representing obstacles in the environment. 84

5.4 Three navigation scenarios where two ACMs, collaboratively transport an object through an obstacle-filled environment. The reference trajectory is shown in green, while the trajectories of the two UAVs are represented in red and purple. The continuum arm trajectories are depicted in blue, demonstrating their motion as they adapt to the object's movement. Each scenario presents different trajectory complexities, including smooth paths, sharp turns, and structured geometric patterns. 85

5.5 Two distinct dynamic obstacle scenarios that the ACM must navigate. The reference trajectory is indicated in green, while the UAV and continuum arm trajectories are shown in red and blue, respectively. In both cases, the ACM is tasked with following a straight-line trajectory. Scenario (a) features two cuboid obstacles moving at a constant velocity, whereas scenario (b) introduces two additional cylindrical obstacles moving diagonally. 86

5.6 Real-world experiment setup for ACM obstacle avoidance. (a) The UAV-mounted continuum manipulator navigates around an obstacle while tracking a predefined trajectory. The VICON Motion Capture system ensures precise localization, while simulated LiDAR scans enhance situational awareness, (b) RViz visualization of trajectory taken by the ACM. 89

List of Tables

3.1 Performance of different RL algorithms during training 39

4.1 Performance of Dual Agent algorithm, EP and OTT in different environments. 62

4.2 Effect of UAV Velocity on Success Rate. Each velocity column is divided into three categories: Proposed, EP, and OTT. 63

5.1 Parameters of continuum arm. 80

Chapter 1

Introduction

Unmanned aerial system (UAS), including unmanned aerial vehicles (UAVs) and aerial continuum manipulators (ACMs), have gained increasing popularity in a variety of fields such as cinematography [1], remote sensing, agriculture [2], wildlife conservation [3], and precision farming [4] and with ACMs being used for inspection [5], maintenance [6], and structural assembly [7]. The ability of UAS to navigate complex, cluttered environments is crucial for performing these tasks effectively. However, motion planning and trajectory tracking in such environments remains a challenging problem due to factors like dynamic surroundings, computational constraints, and the need for collision avoidance.

Traditional methods for motion planning and trajectory tracking typically rely on deterministic algorithms, such as the A* search algorithm [8], rapidly-exploring random trees (RRT) [9], or model predictive control (MPC) [10]. These approaches often assume a static or predictable environment and rely heavily on accurate models of the robot's dynamics. While these methods have been successful in structured settings, they face significant challenges in dynamic, uncertain, and unstructured environments. In such scenarios, the predefined models may not fully capture the complexities of the real world, leading to suboptimal performance or even failure in critical situations.

In recent years, the field of reinforcement learning (RL) has emerged as a powerful tool for developing intelligent and adaptive systems. RL is a branch of machine learning where an agent learns to make decisions by interacting with an environment, receiving feedback in the form of rewards or penalties [11]. Over time, the agent learns to maximize cumulative rewards by optimizing its actions, leading to the emergence of complex behaviors without explicit programming. This learning-based approach offers a promising alternative to traditional methods for motion planning and trajectory tracking, particularly in environ-

ments where uncertainty and complexity are inherent. RL algorithms are typically trained to control UAVs in simulated environments due to the benefits of (1) controlled and varied training scenarios, (2) faster and more efficient training processes, (3) data efficiency, and (4) cost-effectiveness and safety [12].

The application of RL to motion planning involves training the agent to explore the environment and discover optimal paths that balance efficiency and safety. Unlike traditional methods that require complete knowledge of the environment, RL can operate in partially observable settings, where the agent must make decisions based on limited or noisy sensory information. This capability makes RL particularly well-suited for real-world robotics applications, where robots often encounter unexpected obstacles or dynamic changes in the environment. Through trial and error, the RL agent can learn to anticipate and react to such changes, improving its robustness and adaptability.

In the context of trajectory tracking, RL enables the development of controllers that can adjust a robot's actions in real-time to minimize tracking errors. Traditional control methods, such as proportional-integral-derivative (PID) controllers [13] or linear quadratic regulators (LQR) [14], are often designed based on linear approximations of the robot's dynamics. These methods may struggle to maintain performance when faced with nonlinearities or external disturbances. RL-based controllers, however, can learn to compensate for such challenges by directly optimizing the control policy through interaction with the environment. This results in more accurate and reliable trajectory tracking, even in the presence of complex dynamics and uncertainties.

The integration of RL into motion planning and trajectory tracking marks a significant departure from traditional methods, offering advantages like continuous learning, adaptability, and the ability to manage high-dimensional spaces. RL also allows for prioritizing safety, making it ideal for complex, dynamic environments. However, RL presents challenges such as the need for extensive data, computational resources, and the learned policies may not always generalize well to new environments. Despite these hurdles, advancements in deep reinforcement learning (DRL), transfer learning, and sim-to-real techniques are making RL increasingly viable for broader adoption in robotics.

By addressing these objectives, this thesis aims to benchmark and advance the state of the art in autonomous navigation and control, contributing to the development of more intelligent, adaptable, and reliable robotic systems. This thesis specifically aims to tackle the challenges inherent in reinforcement learning-based motion planning and trajectory tracking by proposing innovative approaches that can generalize in unseen environments and handle complex constraints, such as visibility, while also ensuring efficient and real-time execution on resource-constrained devices.

Contribution and Scope of The Thesis: The thesis focuses on developing innovative RL based motion planning and trajectory tracking algorithms in complex, cluttered, and dynamic environments. The thesis places a strong emphasis on benchmarking and developing algorithms that prioritize generalization and sim-to-real capabilities. The focus on generalization ensures that the algorithms are robust and effective across diverse scenarios, not limited to the specific conditions they are trained on. This is crucial in the context of motion planning and trajectory tracking, where real-world environments can vary significantly in terms of obstacles, dynamic elements, and visibility constraints. By developing algorithms that generalize well, the thesis aims to create solutions that can adapt to these variations without the need for extensive retraining or fine-tuning. Additionally, the focus on sim-to-real capabilities ensures that the algorithms developed and tested in simulated environments can be successfully deployed in real-world scenarios. This involves addressing the common challenges of transferring learned behaviors from simulation to reality, such as differences in sensor noise, actuation, and environmental dynamics, to ensure that the algorithms perform effectively in practical applications. The key contributions of the thesis that are aligned with these objectives are as follows:

- In Chapter 3, off-policy deep RL algorithms is benchmarked for UAV path planning, focusing on simulation-to-reality transfer. An optimal simulation environment is designed and compared DDPG, TD3, SAC, and SoftQ in terms of reward maximization, constraint violations, and execution time. Our results highlighted DDPG's superior reward performance and TD3's effective collision avoidance, while real-world experiments validated TD3's ability to generate collision-free paths in structured environments.

- In Chapter 4, a dual-agent RL framework is introduced for UAV trajectory tracking in cluttered environments. Unlike prior approaches that treat tracking and obstacle avoidance as a single policy, two separate agents are used: one for trajectory tracking and another for collision avoidance, leveraging 3D point cloud data. Our results demonstrated improved performance over state-of-the-art RL and optimization-based methods, with successful real-world deployment validating the system's ability to handle complex navigation tasks.

- In Chapter 5, a model-free safe reinforcement learning (SRL) framework for trajectory tracking is developed of ACMs in unknown, constrained environments. Our approach introduced a health policy that penalizes unsafe states, ensuring safety without requiring hand-crafted constraints. By minimizing divergence from the health policy, the primary policy naturally incorporates safety considerations. Through curriculum

learning and extensive real-world validation, superior performance is demonstrated in both trajectory tracking and constraint adherence compared to existing SRL methods.

Chapter 2

Background

2.1 Motion Planning and Trajectory Tracking

Motion planning and trajectory tracking are critical components of robotic autonomy, involving the navigation of robots or autonomous vehicles through complex environments. Motion planning is the process of determining a sequence of actions or configurations that a robot must take to move from an initial state to a desired goal while avoiding obstacles. This is often done in a mathematical space known as the configuration space (C-space), which represents all possible positions and orientations of the robot. A point in this space, denoted as $\mathbf{q} \in \mathcal{C}$, where $\mathcal{C}$ is the configuration space, defines a specific configuration of the robot. The goal of motion planning is to find a continuous path $\mathbf{q}(t)$ from the initial configuration $\mathbf{q}_{\text{start}}$ to the goal configuration $\mathbf{q}_{\text{goal}}$ such that $\mathbf{q}(t) \subseteq \mathcal{C}_{\text{free}}$, where $\mathcal{C}_{\text{free}}$ represents the set of configurations that do not result in collisions with obstacles.

Once a path is determined, trajectory tracking comes into play. Trajectory tracking involves controlling the robot so that it follows the planned path $\mathbf{q}(t)$ over time. The objective is to minimize the tracking error $\mathbf{e}(t) = \mathbf{q}(t) - \mathbf{q}_{\text{ref}}(t)$, where $\mathbf{q}_{\text{ref}}(t)$ is the reference trajectory. Control algorithms, such as Proportional-Integral-Derivative (PID) controllers or Model Predictive Control (MPC), are employed to adjust the robot's inputs to ensure that $\mathbf{e}(t)$ remains small. The dynamic model of the robot can be described by the equation

$$\dot{\mathbf{q}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)),$$

where $\dot{\mathbf{q}}(t)$ is the rate of change of the configuration, $\mathbf{f}(\cdot)$ represents the robot's dynamics, and $\mathbf{u}(t)$ denotes the control inputs. The goal of trajectory tracking is to design $\mathbf{u}(t)$ such

that the robot's configuration $\mathbf{q}(t)$ closely follows the reference trajectory $\mathbf{q}_{\text{ref}}(t)$.

Motion planning and trajectory tracking are thus deeply interconnected, with the former providing a feasible path through the configuration space and the latter ensuring the robot follows that path with high accuracy. Together, they enable robots to navigate autonomously and reliably, even in complex and dynamic environments.

2.1.1 Search-Based Methods

Search-based methods are essential for motion planning in UAVs, particularly when the configuration space (C-space) can be discretized or represented explicitly. These methods use graph search algorithms to explore possible paths from the start to the goal configuration. Two prominent search-based algorithms are Dijkstra's algorithm and A*, each offering different advantages and limitations.

Dijkstra's algorithm [15] finds the shortest path in a weighted graph by expanding nodes based on their distance from the start node. It guarantees the optimal path but can be slow in large or complex environments due to the extensive number of nodes and edges that need to be explored.

A* improves on Dijkstra's algorithm by incorporating heuristics to estimate the cost from each node to the goal. This heuristic, combined with the cost from the start node, helps guide the search more efficiently towards the goal. A* is more efficient than Dijkstra's in many cases because it expands fewer nodes, but its performance heavily depends on the quality of the heuristic used. The evaluation function for A* is $f(q) = g(q) + h(q)$, where $g(q)$ is the cost from the start to node q, and $h(q)$ is the heuristic estimate to the goal.

To address limitations like large search spaces and path quality, several extensions have been developed. Weighted A* [16] modifies A* by using a weight on the heuristic to balance between path optimality and search speed. Dynamic A* (D* Lite) [17] is designed for environments that change over time, allowing the algorithm to update paths dynamically as new information becomes available. Theta* improves path quality by considering direct connections between nodes if they are visible to each other, leading to smoother paths.

Field D* [18] refines paths by interpolating between grid points, creating more natural trajectories. ARA$*$ (Anytime Repairing A*) [19] provides a way to find an initial feasible path quickly and then refine it as more time becomes available. Hierarchical A* [20] breaks the problem into multiple levels of abstraction, improving efficiency by simplifying the search at each level.

2.1.2 Sampling-Based Methods

Sampling-based methods have emerged as a fundamental approach for motion planning in UAVs, offering an effective means to navigate complex, high-dimensional configuration spaces (C-space) without requiring explicit representation. These methods, particularly Probabilistic Roadmaps (PRM) [21] and RRT, are well-suited for UAVs because they can efficiently generate feasible paths in environments where obstacles, constraints, and the high degree of freedom make traditional planning methods impractical.

The core idea behind these methods is to randomly sample points within the C-space and use these points to construct a graph or tree that connects the start and goal configurations. PRM constructs a roadmap by randomly sampling the C-space and connecting these samples with feasible paths using a local planner. The resulting graph is then used to find a path from start to goal using standard graph search algorithms. RRT, on the other hand, incrementally grows a tree from the start configuration by exploring the C-space in the direction of randomly sampled points, eventually reaching the goal configuration.

Despite their effectiveness, traditional PRM and RRT face challenges such as sampling inefficiency, suboptimal path quality, and difficulties in adapting to dynamic environments. To address these issues, several extensions have been developed. RRT* and PRM* are notable for their ability to achieve asymptotic optimality, meaning they refine the path towards the optimal solution as more samples are generated [22]. RRT* achieves this by continuously rewiring the tree to find lower-cost paths, while PRM* improves the connection strategy to enhance path quality over time.

Bi-directional RRT (BiRRT) [23] and RRT-Connect [24] are extensions that enhance the search efficiency by simultaneously growing trees from both the start and goal configurations (BiRRT) or by attempting to connect the tree to new samples as aggressively as possible (RRT-Connect). These methods significantly reduce the time required to find a feasible path, making them particularly useful in large or complex environments.

Informed RRT* [25] and Batch Informed Trees (BIT*) [26] focus the search on the most promising regions of the C-space to accelerate convergence to an optimal path. Informed RRT* uses an ellipsoidal region around the start and goal configurations to bias the sampling process toward regions that are more likely to improve the current best path. BIT* combines batch processing with informed search to efficiently refine the solution by iteratively sampling and improving the path.

RRT*-Smart [27] introduces additional path optimization techniques, such as node deletion and dynamic domain shrinking, to further improve path quality after an initial solution is found. This method smooths the path by eliminating unnecessary nodes and

focusing the search on a smaller region around the current best path, resulting in more efficient and practical paths for UAV navigation.

The use of k-d Trees in PRM improves the efficiency of the neighbor-search process, enabling faster construction of the roadmap, especially in high-dimensional spaces where the number of samples is large [28]. This optimization significantly reduces the computational complexity, making PRM more viable for real-time applications.

Motion Planning Networks (MPNet) [29] represent a cutting-edge integration of deep learning with sampling-based planning. MPNet uses a neural network trained on various environments to predict feasible paths directly, bypassing the need for random sampling. The predicted paths are then refined using traditional planning algorithms like RRT or PRM, ensuring both feasibility and optimality. This approach leverages the generalization capability of neural networks to handle diverse and complex environments, making it particularly promising for real-time UAV applications where rapid decision-making is critical.

2.1.3 Optimization-Based Methods

Optimization-based methods have gained significant attention in the field of Unmanned Aerial Vehicle (UAV) motion planning and trajectory planning due to their ability to handle complex constraints and generate feasible, efficient paths. These methods formulate the planning problem as an optimization problem, where the objective is to minimize a cost function subject to a set of constraints.

The motion planning problem for UAVs can be generally formulated as follows: The objective function typically represents the cost of the trajectory and can include terms for energy consumption, time, distance traveled, smoothness of the path, and safety. Mathematically, it is expressed as $J(\mathbf{x}(t), \mathbf{u}(t)) = \int_0^T L(\mathbf{x}(t), \mathbf{u}(t))\, dt$, where $\mathbf{x}(t)$ is the state vector and $\mathbf{u}(t)$ is the control input. The constraints ensure the feasibility of the trajectory and can include dynamic constraints (e.g., UAV dynamics), kinematic constraints (e.g., velocity and acceleration limits), and environmental constraints (e.g., obstacle avoidance). These are typically expressed as $\mathbf{x}(0) = \mathbf{x}_0, \quad \mathbf{x}(T) = \mathbf{x}_f, \quad \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)) \leq 0$, where $\mathbf{x}_0$ and $\mathbf{x}_f$ are the initial and final states, respectively, and $\mathbf{g}$ represents the constraint functions.

Several optimization techniques are utilized for UAV motion and trajectory planning: Linear and Quadratic Programming (LP/QP) are used when the cost function and constraints are linear or quadratic. These methods are computationally efficient and can be solved using standard solvers. For more complex problems involving nonlinear dynamics and constraints, Nonlinear Programming (NLP) [30] is used. Methods like Sequential

Quadratic Programming (SQP) [31] and Interior Point Methods (IPM) [32] are commonly applied. Convex optimization techniques are employed when the problem can be formulated in a convex form. This ensures global optimality and can be efficiently solved using solvers like CVXGEN [33], CasADi [34] and ECOS [35]. MPC is a popular approach where an optimization problem is solved at each time step to generate a control input that drives the UAV along a desired trajectory. MPC accounts for future states and can handle dynamic and environmental constraints effectively.

Optimization-based methods for UAV motion planning and trajectory planning provide a robust framework for generating feasible, efficient, and safe trajectories. By formulating the problem as an optimization problem, these methods can handle complex constraints and dynamic environments, making them suitable for a wide range of UAV applications. For instance, [36] used kinodynamic path searching and B-spline optimization for smooth 3D quadrotor trajectories, requiring prior maps. [37] combined global and local planners for trajectory refinement in unknown environments, leveraging online point clouds for dynamic perception. Similarly, [38] employed fast marching and frontier detection for trajectory and yaw optimization, while [39] focused on fast trajectory generation and real-time replanning in unknown environments. However, these methods rely on discretized maps such as OctoMaps, which are computationally intensive and struggle with dynamic environments. In contrast, our approach directly encodes obstacle information from 3D point clouds into the RL policy, eliminating the need for pre-built maps, which can limit performance in unknown or partially known environments. [40, 41, 42] also rely on pre-existing maps, which restricts their use in unknown environments. Other methods, like [43], use B-splines with derivative constraints for smooth trajectories but assume prior obstacle knowledge. [44] explored linear and non-linear MPC, while [45] developed MPC for collaborative and real-time planning. However, MPC relies on predefined models and constraints, limiting adaptability. Conversely, our RL-based algorithm generates flexible, adaptive trajectories in uncertain, unseen environments

2.1.4 Learning-Based Methods

Learning-based methods have emerged as a powerful approach for UAV motion planning and trajectory planning, leveraging advancements in machine learning and artificial intelligence. These methods use data-driven techniques to enable UAVs to learn optimal strategies from past experiences or simulated environments, allowing for adaptability and robustness in complex and dynamic scenarios.

One of the fundamental approaches in learning-based methods is RL. In RL, an agent

(the UAV) learns to make decisions by interacting with its environment and receiving feedback in the form of rewards or penalties. The objective is to learn a policy that maximizes the cumulative reward over time. Techniques such as Q-learning [46], Deep Q-Networks (DQN) [47], and Policy Gradient methods [48, 49, 50, 51] have been successfully applied to UAV motion planning. For example, the work in [52] demonstrated the effectiveness of Double-DQN in enabling UAVs to navigate through cluttered environments by learning from simulated flight experiences. For instance, [53] presents a UAV learning algorithm that enhances tracking performance using data from UAVs with different dynamics, adding a learning signal to the feed-forward loop without affecting closed-loop stability. [54] introduces a Physics-inspired temporal convolutional network (PI-TCN) for modeling quadrotor dynamics from robot experience, using sparse temporal convolutions and physics constraints for better generalization. However, these approaches do not account for obstacles

Supervised Learning is another prominent technique where a model is trained on a labeled dataset containing example trajectories. The model learns to predict optimal trajectories based on the input features, such as the UAV's current state and the environment map. This approach is particularly useful when a large amount of expert data is available. For instance, [55] utilized convolutional neural networks (CNNs) to predict collision-free paths for UAVs in complex environments, showing significant improvements in planning efficiency and safety.

Imitation Learning combines elements of supervised learning and reinforcement learning, where the UAV learns by mimicking expert demonstrations. This approach can be more sample-efficient than pure RL, as it leverages expert knowledge to guide the learning process. Techniques such as Behavior Cloning [56] and Generative Adversarial Imitation Learning (GAIL) [57] have been applied to UAV motion planning. In [58] models were trained to mimic the behavior of expert pilots by leveraging data collected from other urban vehicles.

Deep Learning techniques, particularly those involving recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, have been employed to handle sequential decision-making tasks in UAV trajectory planning [59, 60, 61]. These methods can capture temporal dependencies in the UAV's motion and the environment, enabling more accurate and reliable trajectory predictions. Learning-based methods have also been integrated with MPC [62, 63] to combine the strengths of both approaches. In this hybrid framework, a learning-based model predicts the future states and rewards, which are then used by MPC to optimize the control inputs. This integration allows for real-time adaptation to changing environments while maintaining the robustness of optimization-based planning.

[64] proposed a meta-learning approach for UAV trajectory tracking under faults and disturbances, using a runtime-adaptable model with data pruning. However, it requires extensive samples, is task-dependent, and sensitive to hyperparameters, increasing computational complexity and risk of overfitting. [65] introduced quadrotor control using a limited field-of-view camera during aggressive flight, leveraging differential flatness, B-Splines, and Sequential quadratic programming (SQP) with a Receding horizon control (RHC) strategy. Despite enabling real-time adjustments, the computational demands of SQP and RHC and the restricted camera view limit rapid responsiveness and environmental perception

Learning-based methods for UAV motion planning and trajectory planning offer a flexible and powerful framework for tackling complex and dynamic scenarios. By leveraging data-driven techniques, these methods enable UAVs to learn optimal strategies from past experiences, adapt to new environments, and improve their performance over time, making them suitable for a wide range of applications.

2.2 Reinforcement Learning Preliminaries

In this section, background of RL such as Markov Decision Process (MDP), value functions, Bellman equations and optimality conditions are covered.

2.2.1 Markov Decision Process

Reinforcement Learning (RL) problems are typically formulated as Markov Decision Processes (MDPs), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$:

- **State Space ($\mathcal{S}$)**: The set of all possible states the UAV can be in. Each state $s \in \mathcal{S}$ represents a unique configuration of the UAV and its environment.
- **Action Space ($\mathcal{A}$)**: The set of all possible actions the UAV can take. Each action $a \in \mathcal{A}$ corresponds to a control input or maneuver.
- **Transition Probability (P)**: The probability of transitioning from one state to another, given an action. $P(s'|s, a)$ represents the probability of moving to state s' from state s after taking action a.
- **Reward Function (R)**: The immediate reward received after transitioning from state s to state s' due to action a. $R(s, a, s')$ quantifies the desirability of this transition.

- **Discount Factor (γ)**: A factor $\gamma \in [0, 1]$ that determines the importance of future rewards. A lower value of γ prioritizes immediate rewards, while a higher value emphasizes long-term gains.

2.2.2 Policy

A policy π is a strategy that the UAV uses to decide its actions based on the current state. It can be deterministic or stochastic:

- **Deterministic Policy ($\pi(s)$)**: A function that maps each state to a specific action.
- **Stochastic Policy ($\pi(a|s)$)**: A probability distribution over actions, given the current state.

2.2.3 Value Functions

Value functions estimate the expected return (cumulative reward) of states or state-action pairs under a given policy.

- **State Value Function ($V^\pi(s)$)**: The expected return starting from state s and following policy π:

$$V^\pi(s) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s \right] \tag{2.1}$$

- **Action Value Function ($Q^\pi(s, a)$)**: The expected return starting from state s, taking action a, and then following policy π:

$$Q^\pi(s, a) = \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a \right] \tag{2.2}$$

2.2.4 Bellman Equations

The Bellman equations are foundational in RL and dynamic programming. They provide recursive relationships for the value functions, which represent the expected return from a given state or state-action pair. These equations are central to the development of algorithms such as value iteration, policy iteration, and Q-learning.

- **Bellman Expectation Equation for $V^\pi(s)$:**

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')] \tag{2.3}$$

  This equation represents the expected value of being in state s and following policy π. The value function $V^\pi(s)$ gives the expected return starting from state s, taking action a, and then continuing according to the policy. The transition probability $P(s'|s,a)$ defines the likelihood of reaching state s' from state s after taking action a. The term $R(s,a,s')$ is the immediate reward, and γ is the discount factor, determining the importance of future rewards. The equation is a recursive formulation where the value of a state depends on the values of the subsequent states.

- **Bellman Expectation Equation for $Q^\pi(s,a)$:**

$$Q^\pi(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s',a')] \tag{2.4}$$

  The Q-function, $Q^\pi(s,a)$, represents the expected return starting from state s, taking action a, and following policy π thereafter. It can be seen as the state-action value, where the agent is considering both the immediate reward and the future expected value from subsequent actions. Similar to the Bellman equation for $V^\pi(s)$, the Bellman equation for $Q^\pi(s,a)$ expresses a recursive relationship, where the value of a state-action pair depends on the sum of future values weighted by the policy's action probabilities.

These equations are the basis for many RL algorithms, including value iteration and policy iteration, and they provide the foundation for deriving optimal policies in both model-based and model-free settings.

2.2.5 Optimal Policy and Value Functions

The goal of RL is to find the optimal policy π^* that maximizes the expected return. This optimal policy leads to the highest possible value for each state or state-action pair, which is represented by the optimal value functions. The corresponding optimal value functions are defined as follows:

- **Optimal State Value Function ($V^*(s)$):**

$$V^*(s) = \max_\pi V^\pi(s) \tag{2.5}$$

The optimal state value function $V^*(s)$ represents the maximum expected return that can be achieved starting from state s by following the optimal policy π^*. It captures the best possible value achievable for each state under any policy.

- **Optimal Action Value Function ($Q^*(s, a)$)**:

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) \tag{2.6}$$

Similarly, the optimal action value function $Q^*(s, a)$ represents the maximum expected return for taking action a in state s and then following the optimal policy π^*. It evaluates the best possible action that can be taken in each state.

The Bellman optimality equations describe the recursive relationships for these optimal value functions. These equations are key to understanding how the optimal policy can be derived and are used in algorithms like Q-learning and value iteration.

- **Bellman Optimality Equation for $V^*(s)$**:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s, a)[R(s, a, s') + \gamma V^*(s')] \tag{2.7}$$

The Bellman optimality equation for $V^*(s)$ expresses the optimal value of a state as the maximum expected return achievable by taking the best action a from state s and following the optimal policy thereafter. It involves calculating the expected value for all possible next states s', weighted by the transition probabilities, and choosing the action that maximizes this expected return.

- **Bellman Optimality Equation for $Q^*(s, a)$**:

$$Q^*(s, a) = \sum_{s' \in \mathcal{S}} P(s'|s, a)[R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a')] \tag{2.8}$$

The Bellman optimality equation for $Q^*(s, a)$ defines the optimal action value for a state-action pair (s, a). It states that the optimal action value is the expected reward from taking action a in state s, plus the discounted maximum value of the next state s', considering all possible actions a' in that state. This recursive relationship enables the identification of the optimal action at each state.

2.3 Reinforcement Learning Algorithms

In this section, four state-of-the-art (SOTA) algorithms will be reviewed. These algorithms will then be utilized in the forthcoming published work.

2.3.1 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [48] is a model-free, off-policy actor-critic algorithm designed for continuous action spaces, making it particularly suitable for robotics and control tasks. DDPG combines the benefits of Q-learning and policy gradients to enable efficient learning in high-dimensional action spaces.

The actor network, $\mu(s|\theta^\mu)$, deterministically maps states s to specific actions a. The critic network, $Q(s, a|\theta^Q)$, evaluates the action taken by estimating the Q-value, which represents the expected cumulative reward from taking action a in state s. The Q-value is updated using the Bellman Equation

$$Q(s_t, a_t) = r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta^{\mu'})|\theta^{Q'}), \tag{2.9}$$

where s_t and a_t represent the state and action at time t, r_t is the reward received, and γ is the discount factor. $\theta^{\mu'}$ and $\theta^{Q'}$ are the parameters of the target actor and critic networks, respectively. DDPG employs two sets of actor and critic networks: primary networks for learning and target networks for stability. The target networks are updated using the soft update mechanism

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta', \tag{2.10}$$

where θ are the parameters of the primary network, θ' are the parameters of the target network, and τ is the soft update coefficient, typically a small value (e.g., 0.001). The deterministic policy gradient theorem is used to update the actor network. The policy gradient is given by

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s\sim\mathcal{D}} \left[\nabla_a Q(s, a|\theta^Q)|_{a=\mu(s|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu) \right], \tag{2.11}$$

where $\mathcal{D}$ represents the experience replay buffer. The critic network is updated by minimizing the loss function:

$$L(\theta^Q) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(Q(s_t,a_t|\theta^Q) - y_t\right)^2\right] \tag{2.12}$$

with the target value y_t defined as

$$y_t = r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta^{\mu'})|\theta^{Q'}). \tag{2.13}$$

DDPG also utilizes an experience replay buffer to store transitions (state, action, reward, next state) and samples mini-batches for learning. This helps break the correlation between consecutive experiences and reduces variance, thereby stabilizing the training process. By combining these techniques, DDPG effectively learns policies in complex environments with high-dimensional continuous action spaces.

2.3.2 Twin Delayed Deep Deterministic Policy Gradient

Twin Delayed Deep Deterministic Policy Gradient (TD3) [66] is an advanced RL algorithm that builds upon the Deep Deterministic Policy Gradient (DDPG) framework to address some of its limitations and improve stability and performance in continuous action spaces. TD3 introduces several key innovations aimed at enhancing learning efficiency and policy robustness.

TD3 employs an actor-critic architecture, consisting of an actor network $\mu(s|\theta^\mu)$ and two critic networks $Q_1(s,a|\theta^{Q_1})$ and $Q_2(s,a|\theta^{Q_2})$. The actor network outputs deterministic actions given a state, while the critic networks evaluate these actions by estimating the Q-values. The Bellman equation used to update the Q-values is given by

$$Q_i(s_t,a_t) = r_t + \gamma \min_{j=1,2} Q'_j(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'_j}), \tag{2.14}$$

where s_t and a_t are the state and action at time t, r_t is the reward, γ is the discount factor, and Q'_j represent the target critic networks. By using two critic networks and taking the minimum value for the target, TD3 mitigates the overestimation bias that can occur in Q-value estimation.

A notable feature of TD3 is its approach to policy and target network updates. The algorithm delays the updates to the actor network and target networks compared to the critic networks. Specifically, the actor network is updated every d steps, allowing the critic networks to stabilize and provide more accurate value estimates. This delay helps prevent

the policy from oscillating due to noisy value estimates. The update rules for the target networks are implemented with updates

$$\begin{aligned} \theta^{Q'_j} &\leftarrow \tau\theta^{Q_j} + (1-\tau)\theta^{Q'_j}, \\ \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}, \end{aligned} \tag{2.15}$$

where τ is a small coefficient (e.g., 0.001) that controls the update rate of the target networks.

In addition to these updates, TD3 incorporates target policy smoothing to improve exploration and robustness. Noise is added to the target policy to avoid overfitting and encourage exploration. The target action $\tilde{a}_{t+1}$ is computed as

$$\tilde{a}_{t+1} = \text{clip}(\mu'(s_{t+1}|\theta^{\mu'}) + \mathcal{N}(0,\sigma), -\text{clip}, \text{clip}), \tag{2.16}$$

where $\mathcal{N}(0,\sigma)$ represents noise sampled from a Gaussian distribution with mean 0 and standard deviation σ, and clip is used to bound the action within a feasible range. The critic networks are updated by minimizing the loss function

$$L(\theta^{Q_j}) = \frac{1}{N}\sum_{i=1}^{N}\left[Q_j(s_i, a_i|\theta^{Q_j}) - y_i\right]^2, \tag{2.17}$$

where the target value y_i is computed using the previously mentioned Bellman equation. The actor network is updated by maximizing the expected return

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q_1(s_i, a|\theta^{Q_1})\big|_{a=\mu(s_i|\theta^\mu)} \nabla_{\theta^\mu}\mu(s_i|\theta^\mu). \tag{2.18}$$

TD3's enhancements, including the use of twin critics, delayed policy updates, target policy smoothing, and soft target updates, collectively contribute to its improved performance and stability over DDPG. These innovations make TD3 a robust and effective choice for continuous control problems.

2.3.3 Soft Q-learning

Soft Q-Learning (SoftQ) [67] is an advanced RL algorithm designed to improve the stability and exploration capabilities of traditional Q-learning methods. Unlike conventional Q-learning, which focuses solely on maximizing rewards, Soft Q-Learning incorporates an entropy term into the value function to encourage exploration and robustness. The core of Soft Q-Learning lies in the soft Bellman equation

$$Q(s_t, a_t) = r_t + \gamma \mathbb{E}_{s_{t+1}} \left[V(s_{t+1}) \right], \tag{2.19}$$

where $V(s)$, the value function, is defined as

$$V(s) = \mathbb{E}_{a \sim \pi} \left[Q(s, a) - \alpha \log \pi(a|s) \right]. \tag{2.20}$$

Here, α is a temperature parameter that controls the balance between reward maximization and policy entropy. This entropy term, $\alpha \log \pi(a|s)$, promotes exploration by encouraging the policy to be more stochastic and less deterministic. The policy in SoftQ is updated to maximize the expected soft Q-value. The policy gradient is computed as

$$\nabla_{\theta^\pi} J(\pi) \approx \mathbb{E}_{s_t \sim \mathcal{D}} \left[\nabla_a Q(s_t, a|\theta^Q)|_{a=\pi(s_t|\theta^\pi)} \nabla_{\theta^\pi} \pi(a|s_t|\theta^\pi) \right], \tag{2.21}$$

where θ^π represents the parameters of the policy network, and $\mathcal{D}$ denotes the experience replay buffer. This gradient ensures that the policy improves in a direction that increases the soft Q-value. The Q-function is updated by minimizing the mean squared error between the current Q-values and the target values. The loss function is defined as

$$\begin{aligned} L(\theta^Q) = \frac{1}{N} \sum_{i=1}^{N} \big[& Q(s_i, a_i|\theta^Q) - \\ & \left(r_i + \gamma \mathbb{E}_{s_{i+1}} \left[\mathbb{E}_{a_{i+1} \sim \pi} \left[Q(s_{i+1}, a_{i+1}|\theta^Q) - \alpha \log \pi(a_{i+1}|s_{i+1}) \right] \right] \right) \big]^2 . \end{aligned} \tag{2.22}$$

In this formulation, N is the size of the mini-batch sampled from the replay buffer, r_i is the reward at time step i, and γ is the discount factor.

SoftQ utilizes an experience replay buffer to store transitions (state, action, reward, next state) and samples mini-batches for training. This helps to break the correlation between consecutive experiences and enhances the stability of the learning process. By

integrating the entropy term and using a soft Bellman equation, Soft Q-Learning balances exploration and exploitation, leading to more effective and robust policy learning in complex environments.

2.3.4 Soft Actor-Critic

Soft Actor-Critic (SAC) [51] is a powerful RL algorithm designed for continuous action spaces, known for its robustness and efficiency. SAC operates by optimizing a stochastic policy that not only aims to maximize the expected return but also incorporates an entropy term to promote exploration. This dual objective helps the policy learn more effectively and avoid premature convergence to suboptimal solutions.

The core idea behind SAC is to balance exploitation and exploration through entropy maximization. The policy π is parameterized by a neural network that outputs a probability distribution over actions. The objective is to maximize the expected return $\mathbb{E}[R]$ plus the entropy of the policy $\mathbb{E}[\mathcal{H}(\pi(\cdot|s))]$, where $\mathcal{H}(\pi(\cdot|s))$ denotes the entropy of the policy at state s. This leads to the following optimization objective

$$J_{\text{policy}} = \mathbb{E}_{s_t \sim \rho_{\text{env}}} \left[\mathbb{E}_{a_t \sim \pi(\cdot|s_t)} \left[\alpha \log \pi(a_t|s_t) - Q(s_t, a_t) \right] \right], \tag{2.23}$$

where α is the entropy weight that controls the trade-off between exploration and exploitation, and $Q(s_t, a_t)$ is the action-value function. To estimate the action-value function, SAC utilizes two Q-functions, $Q_1(s, a)$ and $Q_2(s, a)$, to mitigate the problem of overestimation bias. The value of a state s is estimated using a value function $V(s)$, and SAC aims to minimize the following loss function for the Q-functions

$$L_Q = \mathbb{E}_{s_t, a_t \sim \rho_{\text{exp}}} \left[\left(Q_i(s_t, a_t) - \left(r_t + \gamma \mathbb{E}_{s_{t+1} \sim \rho_{\text{env}}} \left[V(s_{t+1}) \right] \right) \right)^2 \right], \tag{2.24}$$

where r_t is the reward at time t, γ is the discount factor, and ρ_{exp} denotes the experience replay distribution. SAC also maintains target networks for both the Q-functions and the value function to stabilize training. The target networks are updated periodically using a slow-moving average of the current networks. The value function is updated using

$$L_V = \mathbb{E}_{s_t \sim \rho_{\text{exp}}} \left[\left(V(s_t) - \mathbb{E}_{a_t \sim \pi(\cdot|s_t)} \left[Q(s_t, a_t) - \alpha \log \pi(a_t|s_t) \right] \right)^2 \right]. \tag{2.25}$$

Finally, SAC employs an experience replay buffer to store past interactions and sample from this buffer to break correlations between consecutive samples, further enhancing stability and efficiency during training.

2.4 Safe Reinforcement Learning Algorithms

In this section, three state-of-the-art SRL algorithms will be reviewed. These algorithms will then be utilized in the forthcoming published work.

2.4.1 Safe Adversarially Guided Actor-Critic

The Safe Adversarially Guided Actor-Critic (SAAC) algorithm extends the Soft Actor-Critic (SAC) framework by incorporating an adversarial component that models safety constraints. The agent and adversary interact in a non-zero-sum game, where the adversary seeks to maximize constraint violations while the agent optimizes its policy while adhering to safety constraints.

The objective of the agent is to maximize the expected return while minimizing the influence of the adversary

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\pi_\theta}[Q_\theta(s,a)] + \alpha_0(1+\beta_0)H_{\pi_\theta}(a|s) - \beta_0\mathbb{E}_{\pi_\theta}[Q_\omega(s,a)], \tag{2.26}$$

where $Q_\theta(s,a)$ is the agent's soft Q-function, $Q_\omega(s,a)$ is the adversary's soft Q-function, α_0 is the entropy regularization coefficient, and β_0 is the adversarial weight. The adversary aims to maximize the probability of constraint violations by learning a policy π_ω that maximizes its expected return

$$\omega^* = \arg\max_{\omega} \mathbb{E}_{\pi_\omega}[Q_\omega(s,a)] + \alpha_0 H_{\pi_\omega}(a|s). \tag{2.27}$$

This leads to a KL-divergence formulation in policy space

$$\pi^*_\omega = \arg\min_{\omega} D_{KL}\left(\pi_\omega(\cdot|s) \| \frac{\exp(Q_\omega(s,\cdot)/\alpha_0)}{Z_\omega(s)}\right). \tag{2.28}$$

For the agent, the optimal policy update minimizes the KL-divergence between its policy and an entropy-regularized Q-function while introducing a repulsion term from the adversary's policy

$$\pi^*_\theta = \arg\min_{\theta} D_{KL}\left(\pi_\theta(\cdot|s) \| \frac{\exp(Q_\theta(s,\cdot)/\alpha)}{Z_\theta(s)}\right) - \beta D_{KL}(\pi_\theta(\cdot|s) \| \pi^*_\omega(\cdot|s)), \tag{2.29}$$

where $\alpha = \alpha_0(1 + \beta_0)$ and $\beta = \alpha_0\beta_0$. The SAAC algorithm alternates between collecting experience using the current policy and updating the function approximators. The critic networks are updated using the soft Bellman residual

$$J(Q_\phi) = \mathbb{E}_{(s,a)\sim D}\left[\frac{1}{2}\left(Q_\phi(s,a) - (r(s,a) + \gamma \mathbb{E}_{s'\sim P} V_\phi(s'))\right)^2\right], \tag{2.30}$$

where $V_\phi(s)$ is the soft value function defined as

$$V_\phi(s) = \mathbb{E}_{a\sim\pi_\theta}\left[Q_\phi(s,a) - \alpha \log \pi_\theta(a|s)\right]. \tag{2.31}$$

The policy updates follow

$$J(\pi_\theta) = \mathbb{E}_{s\sim D}\left[\mathbb{E}_{a\sim\pi_\theta}\left[\alpha \log \pi_\theta(a|s) - Q_\phi(s,a) - \beta(\log \pi_\theta(a|s) - \log \pi_\omega(a|s))\right]\right]. \tag{2.32}$$

The adversary follows a similar update

$$J(\pi_\omega) = \mathbb{E}_{s\sim D}\left[\mathbb{E}_{a\sim\pi_\omega}\left[\alpha \log \pi_\omega(a|s) - Q_\psi(s,a)\right]\right], \tag{2.33}$$

where $Q_\psi(s,a)$ estimates the probability of constraint violations. To ensure stability, the adversarial weight β is dynamically adjusted via gradient descent

$$J(\beta) = \mathbb{E}_{s\sim D}\left[\log \beta \cdot (D_{KL}(\pi_\theta||\pi_\omega) - A)\right], \tag{2.34}$$

where A is a predefined adversarial bonus target. The entropy coefficient α is also tuned to minimize

$$J(\alpha) = \mathbb{E}_{s\sim D}\left[\log \alpha \cdot (-\log \pi_\theta(a|s) - H)\right], \tag{2.35}$$

where H is the target entropy. By integrating adversarial training into the SAC framework, SAAC ensures risk-sensitive learning while maintaining exploration efficiency. The repulsion term prevents the agent from imitating unsafe actions, leading to safer and more robust policy learning.

2.4.2 Safety Editor Policy

The Safety Editor Policy (SEditor) algorithm decomposes SRL into two cooperative policies: a Utility Maximizer (UM) policy π_ϕ and a Safety Editor (SE) policy π_ψ. The UM generates a preliminary action $\hat{a}$ based on the current state s, while the SE modifies this action to ensure safety, producing a final action a.

The objective of the UM policy is to maximize the expected return

$$\max_{\phi} \mathbb{E}_{s\sim D, \hat{a}\sim\pi_\phi(\cdot|s)}[Q(s,\hat{a})], \tag{2.36}$$

where $Q(s,a)$ is the utility state-action value function. The SE policy is trained to minimize constraint violations while preserving utility

$$\max_{\psi} \mathbb{E}_{s\sim D, \hat{a}\sim\pi_\phi(\cdot|s), \Delta a\sim\pi_\psi(\cdot|s,\hat{a})} \left[-d(a,\hat{a}) + \lambda Q_c(s,a)\right], \tag{2.37}$$

where $d(a,\hat{a})$ is a distance function measuring the modification magnitude, $Q_c(s,a)$ is the constraint state-action value function, and λ is a Lagrange multiplier dynamically updated via

$$\min_{\lambda\geq 0} \lambda \left(\mathbb{E}_{s\sim D}[V_c^\pi(s)] + C\right), \tag{2.38}$$

where C is the constraint violation budget. The action editing function is defined as

$$a = h(\hat{a}, \Delta a) = \min(\max(\hat{a} + 2\Delta a, -A), A), \tag{2.39}$$

ensuring that the SE has full control over the final action. The distance function $d(a,\hat{a})$ is formulated using hinge loss

$$d(a,\hat{a}) = \max(0, Q(s,\hat{a}) - Q(s,a)), \tag{2.40}$$

which prioritizes maintaining high utility while enforcing safety. The critic networks are updated using the Bellman backup operator

$$\begin{aligned} Q(s,a) &= r(s,a) + \gamma \mathbb{E}_{s'\sim p}[V(s')], \\ Q_c(s,a) &= r_c(s,a) + \gamma \mathbb{E}_{s'\sim p}[V_c(s')], \end{aligned} \tag{2.41}$$

where $V(s)$ and $V_c(s)$ are the value functions corresponding to the utility and constraint rewards, respectively. The policies π_ϕ and π_ψ are trained using stochastic gradient descent. The policy loss for π_ϕ is given by

$$J(\pi_\phi) = \mathbb{E}_{s\sim D, \hat{a}\sim\pi_\phi(\cdot|s)} \left[\log \pi_\phi(\hat{a}|s) Q(s, \hat{a})\right], \tag{2.42}$$

and for π_ψ by

$$J(\pi_\psi) = \mathbb{E}_{s\sim D, \hat{a}\sim\pi_\phi(\cdot|s), \Delta a\sim\pi_\psi(\cdot|s,\hat{a})} \left[-d(a, \hat{a}) + \lambda Q_c(s, a)\right]. \tag{2.43}$$

By integrating an independent safety mechanism, SEditor effectively reduces constraint violations while maintaining high utility performance, making it suitable for SRL applications.

2.4.3 Reachability Constrained Reinforcement Learning

The Reachability Constrained Reinforcement Learning (RCRL) algorithm is a method designed to enforce persistent safety constraints in RL by leveraging reachability analysis. Unlike standard constrained reinforcement learning (CRL) approaches, which rely on expected discounted cumulative costs, RCRL defines a feasible set of states from which an agent can always remain safe. This feasible set is characterized using a safety value function.

Given a Markov Decision Process (MDP) $M = \langle S, A, P, r, h, c, \gamma\rangle$, where $h(s)$ represents the state constraint, the safety value function is defined as

$$V_h^\pi(s) = \max_{t\in\mathbb{N}} h(s_t^\pi | s_0 = s), \tag{2.44}$$

which captures the worst constraint violation over an infinite trajectory following policy π. The largest feasible set is then defined as

$$S_f = \{s \in S \mid \exists \pi, V_h^\pi(s) \leq 0\}. \tag{2.45}$$

RCRL aims to solve the following optimization problem

$$\max_\pi \mathbb{E}_{s\sim d_0}[V^\pi(s) \cdot \mathbb{1}_{s\in S_f} - V_h^\pi(s) \cdot \mathbb{1}_{s\notin S_f}], \tag{2.46}$$

subject to the constraint

$$V_h^\pi(s) \leq 0, \quad \forall s \in S_f \cap S_0. \tag{2.47}$$

The key theoretical foundation of RCRL is the Safety Bellman Equation (SBE)

$$V_h^*(s) = \max\left\{h(s), \min_{a\in A} V_h^*(s')\right\}, \tag{2.48}$$

which can be generalized to any policy π

$$V_h^\pi(s) = \max\{h(s), V_h^\pi(s')\}, \tag{2.49}$$

where $s' \sim P(s'|s,a)$ under π. This condition ensures that the agent remains in the feasible set throughout its trajectory. To solve the RCRL problem, a Lagrangian-based approach is used

$$L(\pi,\lambda) = \mathbb{E}_{s\sim d_0}[-V^\pi(s) + \lambda(s)V_h^\pi(s)], \tag{2.50}$$

where $\lambda(s)$ is a state-dependent Lagrange multiplier that penalizes constraint violations. The learning algorithm follows a multi-timescale stochastic approximation scheme, updating the critic $Q(s,a)$ to approximate $V^\pi(s)$, safety critic $Q_h(s,a)$ to approximate $V_h^\pi(s)$, actor $\pi(s)$ to optimize the policy and multipliers $\lambda(s)$ to enforce the safety constraint. The policy gradient update follows

$$\nabla_\theta J^\pi(\theta) = \mathbb{E}_{s\sim d^\pi}\left[\nabla_a Q(s,a)|_{a=\pi(s)} - \lambda(s)\nabla_a Q_h(s,a)|_{a=\pi(s)}\right]\nabla_\theta \pi(s;\theta). \tag{2.51}$$

2.5 Aerial Continuum Manipulator Modelling

Recent studies have introduced the concept of ACMs as a promising solution to the limitations of traditional aerial manipulators, such as rigid-link arms. ACMs, incorporating continuum robots (CRs), provide enhanced dexterity, compliance, and flexibility, allowing manipulation in unstructured and confined environments. [68] proposed dual-arm ACMs to overcome limitations in payload, dexterity, and workspace, utilizing the Euler–Lagrange approach for dynamic modeling and presenting a hybrid control framework for pre-grasp

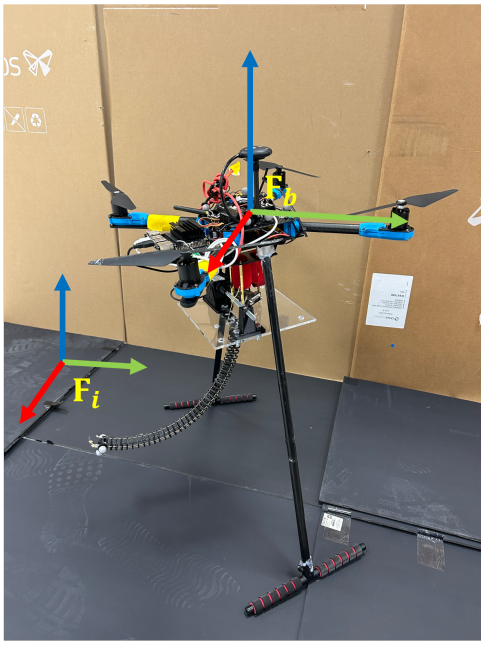


Figure 2.1: Coordinate frames in aerial continuum manipulator.

planning. Similarly, [69] utilized decoupled dynamics for control design, addressing the challenges posed by the coupling between the arm and the UAV, while [70] developed a vision-based reduced-order adaptive control approach to handle uncertainties in dynamic control without requiring precise system models. Furthermore, the dynamic modeling of ACMs has been explored by [71], highlighting the limitations of rigid manipulators and emphasizing the potential of CRs for aerial manipulation.

Building on this approach, the dynamic modeling of an ACM employs an Euler-Lagrange formulation to facilitate simulations of the proposed control methodology [72]. The tendon-driven continuum arm is mounted at the center of a multirotor UAV, and the rotation matrix R_B is defined to describe the transformation between the inertial reference frame $\mathcal{F}_I$, centered at O_I, and the UAV's body frame $\mathcal{F}_B$, centered at O_B as shown in Figure 2.1. This transformation is given as

$$R_B = \begin{bmatrix} C_\psi C_\theta & -S_\psi C_\phi + C_\psi S_\theta S_\phi & S_\psi S_\phi + C_\psi S_\theta C_\phi \\ S_\psi C_\theta & C_\psi C_\phi + S_\psi S_\theta S_\phi & -C_\psi S_\phi + S_\psi S_\theta C_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix}, \tag{2.52}$$

where C and S denote the cosine and sine of the respective angles, while ψ, θ, ϕ represent

the yaw, pitch, and roll angles of the UAV, collectively referred to as the attitude vector $\mathbf{\Phi}_B = [\phi, \theta, \psi]^\top$.

The UAV's linear velocity in the inertial frame, $\dot{\mathbf{p}}_B$, is related to its body frame velocity $\dot{\mathbf{p}}_B^B$ by

$$\dot{\mathbf{p}}_B = R_B \dot{\mathbf{p}}_B^B. \tag{2.53}$$

Similarly, the angular velocity of the UAV in $\mathcal{F}_I$, $\boldsymbol{\omega}_B$ is expressed in terms of the Euler angle derivatives as

$$\boldsymbol{\omega}_B = T_B \dot{\mathbf{\Phi}}_B, \tag{2.54}$$

where T_B is defined as

$$T_B = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & S_\phi C_\theta \\ 0 & -S_\phi & C_\phi C_\theta \end{bmatrix}. \tag{2.55}$$

The angular velocity in the body frame is then given by

$$\begin{aligned} \boldsymbol{\omega}_B^B &= R_B \boldsymbol{\omega}_B \\ &= R_B T_B \dot{\mathbf{\Phi}}_B. \end{aligned} \tag{2.56}$$

Assuming a constant-curvature model for the arm, the end-effector coordinates can be represented by two angles [73]. These angles, denoted as q_1 and q_2, correspond to the vertical and horizontal bending angles of the continuum arm, respectively. The generalized state vector is then represented $\mathbf{X} = [\mathbf{p}_B^\top, \mathbf{\Phi}_B^\top, \mathbf{q}_1, \mathbf{q}_2]^\top$.

The position of any point on the arm in the inertial frame can be described as

$$\mathbf{p}_s = \mathbf{p}_B + R_B \mathbf{p}_{Bs}^B, \tag{2.57}$$

where $\mathbf{p}_{Bs}^B$ represents the relative position of a section of the arm in the body frame, expressed as

$$\mathbf{p}_s^B = \begin{bmatrix} \frac{l}{q_1}\left(1 - C_{q_s}\right) C_{q_2} \\ \frac{l}{q_1}\left(1 - C_{q_s}\right) S_{q_2} \\ \frac{l}{q_1} S_{q_s} \end{bmatrix}. \tag{2.58}$$

Here, l represents the length of the arm, and s is a normalized parameter ranging from 0 to 1 along the arm's length. The linear velocity of the arm in the body frame is computed as

$$\dot{\mathbf{p}}_{Bs}^{B} = J_p(s)\dot{\mathbf{q}}, \tag{2.59}$$

where $J_p(s)$ is the position Jacobian matrix for linear velocity of the arm in the UAV body frame. The rotation of the arm segment with respect to the UAV body frame is represented by the matrix R_s^B, defined as

$$R_s^B = \begin{bmatrix} C_{q_2}^2 C_{q_s} + S_{q_2}^2 & C_{q_2} S_{q_2} C_{q_s} - C_{q_2} S_{q_2} & C_{q_2} S_{q_s} \\ C_{q_2} S_{q_2} C_{q_s} - S_{q_2} C_{q_2} & S_{q_2}^2 C_{q_s} + C_{q_2}^2 & S_{q_2} S_{q_s} \\ -C_{q_2} S_{q_s} & -S_{q_2} S_{q_s} & C_{q_s} \end{bmatrix}. \tag{2.60}$$

where, $q_1 s/l = q_s$. Finally, the angular velocity of the arm segment is determined as

$$\boldsymbol{\omega}_s^B = \mathbf{Skew}(t_s)\dot{\mathbf{q}}, \tag{2.61}$$

where $\mathbf{Skew}(t_s)$ is the skew-symmetric matrix of the vector t_s, which is typically the unit vector along the axis of rotation or the direction of the arm segment. The skew-symmetric matrix, $\mathbf{Skew}(t_s)$, is defined as

$$\mathbf{Skew}(t_s) = \begin{pmatrix} 0 & -t_{s3} & t_{s2} \\ t_{s3} & 0 & -t_{s1} \\ -t_{s2} & t_{s1} & 0 \end{pmatrix},$$

where $t_s = [t_{s1}, t_{s2}, t_{s3}]^\top$ is a 3D vector. This matrix operation creates a cross-product representation, which is used to compute the angular velocity $\boldsymbol{\omega}_s^B$, essentially applying the rotational dynamics to the time derivative of the joint angles $\dot{\mathbf{q}}$.

To compute the linear velocity of the continuum arm in the inertial frame $\mathcal{F}_I$, the derivative of the arm's position is derived as

$$\dot{\mathbf{p}}_s = \dot{\mathbf{p}}_B - \mathrm{Skew}(R_B \mathbf{p}_{Bs}^B)\boldsymbol{\omega}_B + R_B J_p(s)\dot{\mathbf{q}}, \tag{2.62}$$

where $\dot{\mathbf{p}}_B$ is the UAV's linear velocity in the inertial frame and $\boldsymbol{\omega}_B$ is the angular velocity of the UAV in its body frame. Similarly, the angular velocity of each section of the arm, expressed in the inertial frame, is determined as

$$\boldsymbol{\omega}_s = \boldsymbol{\omega}_B + R_B J_o(s)\dot{\mathbf{q}}, \tag{2.63}$$

where $J_o(s)$ is the Jacobian matrix for angular velocity of the arm in the body frame. The Jacobian matrices $J_p(s)$ and $J_o(s)$ are derived as follows

$$J_p(s) = \begin{bmatrix} lC_{q_2}\left(\frac{C_{q_s}-1}{q_1^2}\right) + sS_{q_s}\frac{C_{q_2}}{q_1} & lS_{q_2}\left(\frac{C_{q_s}-1}{q_1}\right) \\ lS_{q_2}\left(\frac{C_{q_s}-1}{q_1^2}\right) + sS_{q_s}\frac{S_{q_2}}{q_1} & -lC_{q_2}\left(\frac{C_{q_s}-1}{q_1}\right) \\ -l\left(\frac{S_{q_s}}{q_1^2}\right) + s\frac{C_{q_s}}{q_1} & 0 \end{bmatrix}, \tag{2.64}$$

$$J_o(s) = \begin{bmatrix} -\frac{s}{l}S_{q_2} & -C_{q_s}S_{q_s}C_{q_2} \\ \frac{s}{l}C_{q_2} & -C_{q_s}S_{q_s}S_{q_2} \\ 0 & S_{q_s}^2 \end{bmatrix}. \tag{2.65}$$

After formulating the kinematics of ACMs, the dynamic model is derived using the Euler-Lagrange formulation. The Lagrangian $\mathcal{L}$ is defined as

$$\mathcal{L} = \mathcal{K} - \mathcal{U}, \tag{2.66}$$

where $\mathcal{K}$ is the total kinetic energy, and $\mathcal{U}$ is the total potential energy of the system. The Euler-Lagrange equation is employed to obtain the equations of motion for the ACMs

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{X}}_i}\right) - \frac{\partial \mathcal{L}}{\partial \mathbf{X}_i} + \frac{\partial \mathcal{D}}{\partial \dot{\mathbf{X}}_i} = u_i, \quad i = 1, \ldots, n_q, \tag{2.67}$$

where $\mathcal{D}$ is the dissipation function accounting for energy losses, u_i represents external control inputs, $\mathbf{X}_i$ denotes the generalized coordinates of the system, and n_q is the total number of generalized coordinates. The Euler-Lagrange equation is utilized to describe the dynamics of the ACMs as

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{X}}_i}\right) - \frac{\partial \mathcal{L}}{\partial \mathbf{X}_i} + \frac{\partial \mathcal{D}}{\partial \dot{\mathbf{X}}_i} = u_i, \quad i = 1, \ldots, n_q, \tag{2.68}$$

where $\mathbf{X}_i$ represents the i-th generalized coordinate, u_i denotes the corresponding control input, and $\mathcal{D}$ accounts for linear viscous damping effects.

The total kinetic energy $\mathcal{K}$ consists of contributions from the UAV's kinetic energy (K_B), the continuum arm's main backbone kinetic energy (K_{mb}), and the kinetic energy of its disks (K_d).

The kinetic energy of the UAV is determined as

$$K_B = \frac{1}{2} m_B \dot{\mathbf{p}}_B^T \dot{\mathbf{p}}_B + \frac{1}{2} \boldsymbol{\omega}_B^T R_B I_B R_B^T \boldsymbol{\omega}_B, \tag{2.69}$$

where m_B is the UAV's mass and I_B is the UAV's moment of inertia matrix in the body frame. For the continuum arm's backbone, the kinetic energy is given by

$$K_{mb}(s) = \frac{1}{2} \rho_s A_s \dot{\mathbf{p}}_s^T \dot{\mathbf{p}}_s + \frac{1}{2} \boldsymbol{\omega}_s^T R_B R_S^B I_s R_S^{BT} R_B^T \boldsymbol{\omega}_s, \tag{2.70}$$

where ρ_s is the mass density, A_s is the cross-sectional area, I_s is the moment of inertia for each section, and s denotes the length along the continuum arm. The kinetic energy of a single disk is expressed as

$$K_d(j) = \frac{1}{2} m_d \dot{\mathbf{p}}_d^T \dot{\mathbf{p}}_d + \frac{1}{2} \boldsymbol{\omega}_d^T R_B R_d^B I_d R_d^{BT} R_B^T \boldsymbol{\omega}_d, \tag{2.71}$$

where m_d and I_d are the mass and moment of inertia for each disk, $\mathbf{p}_d$ represents its position, and $\boldsymbol{\omega}_d$ is its angular velocity. The total kinetic energy is obtained as

$$\mathcal{K} = K_B + \int_0^l K_{mb}(s)\, ds + \sum_{j=1}^{n} K_d(j), \tag{2.72}$$

or equivalently

$$\mathcal{K} = \frac{1}{2} \dot{\mathbf{X}}^T M \dot{\mathbf{X}}, \tag{2.73}$$

where M is the system's mass matrix. The potential energy includes contributions from the UAV (U_B), the continuum arm backbone (U_{mb}), and its disks (U_d). The potential energy of the UAV is

$$U_B = m_B g \mathbf{e}_3^T \mathbf{p}_B, \tag{2.74}$$

where g is the gravitational acceleration and $\mathbf{e}_3 = [0, 0, 1]^T$. For the backbone

$$U_{mb} = \int_0^l \left(\rho_s A_s g \mathbf{e}_3^T (\mathbf{p}_B + R_B \mathbf{p}_{Bs}^B)\right) ds + \frac{2E_s I_s}{l} q_1^2, \tag{2.75}$$

where E_s is the Young's modulus, and q_1 is a generalized coordinate. For the disks

$$U_d = \sum_{j=1}^{n} \left(m_d(j) g \mathbf{e}_3^T (\mathbf{p}_B + R_B \mathbf{p}_{Bs}^B)\right)_{s=jh}. \tag{2.76}$$

The total potential energy is

$$\mathcal{U} = U_B + U_{mb} + U_d. \tag{2.77}$$

Consequently, the complete dynamic model of the ACMs is expressed as

$$M(\mathbf{X})\ddot{\mathbf{X}} + C(\mathbf{X}, \dot{\mathbf{X}})\dot{\mathbf{X}} + \mathbf{G}(\mathbf{X}) = \mathbf{u} + \mathbf{u}_{ext}, \tag{2.78}$$

where $M(\mathbf{X})$ is the mass matrix, $C(\mathbf{X}, \dot{\mathbf{X}})$ accounts for Coriolis and centrifugal effects, $\mathbf{G}(\mathbf{X})$ represents the gravity vector, $\mathbf{u}$ is the control input, and $\mathbf{u}_{ext}$ denotes external forces. Since ACMs have complex coupled dynamics, these dynamics must be considered while training RL algorithms. The interdependencies between various degrees of freedom and the non-linearities in their behavior necessitate accurate modeling to ensure the RL agent learns effective control strategies for manipulation tasks. The gravity vector $\mathbf{G}(\mathbf{X})$ is determined as

$$\mathbf{G}(\mathbf{X}) = \left(\frac{\partial \mathcal{U}(\mathbf{X})}{\partial \mathbf{X}}\right)^T, \tag{2.79}$$

where $\mathcal{U}(\mathbf{X})$ is the total potential energy of the system. The elements of the Coriolis matrix C_{ij} are calculated as

$$\begin{aligned} C_{ij} &= \frac{1}{2} \sum_{k=1}^{8} \left(\frac{\partial M_{ij}}{\partial X_k} + \frac{\partial M_{ik}}{\partial X_j} - \frac{\partial M_{jk}}{\partial X_i}\right) \dot{X}_k, \quad i, j \\ &= 1, \ldots, 8, \end{aligned} \tag{2.80}$$

where M_{ij} are the elements of the mass matrix. The control input $\mathbf{u}$ is defined as

$$\mathbf{u} = \text{diag}(R_B, \mathbf{Q}^T, I_2)\text{diag}(\mathbf{\Omega}, I_2) \begin{bmatrix} \mathbf{f}_B \\ \mathbf{f}_{ten} \end{bmatrix}, \tag{2.81}$$

where, $\mathbf{f}_B$ is the input vector of forces exerted by the UAV motors, $\mathbf{f}_{ten}$ represents the generalized force for the continuum arm, based on tendon actuation, I_2 is the identity matrix of size 2 and Ω accounts for motor configurations and, for an underactuated platform, is defined as

$$\Omega = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & -d \\ -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ -c & -c & -c & -c \end{bmatrix}, \tag{2.82}$$

where d is the distance from the motors to the UAV's center, and c is drag coefficient.

Chapter 3

Benchmarking Off-Policy Deep Reinforcement Learning Algorithms for UAV Path Planning

This chapter studies the benchmarking of off-policy RL algorithms specifically tailored for Unmanned Aerial Vehicle UAV path planning. The primary objective is to assess how well these algorithms generalize from simulation environments to real-world deployment scenarios. The chapter provides a comprehensive analysis that includes the design of optimal simulation environments, a comparative study of various off-policy RL algorithms—namely DDPG, TD3, SAC, and SoftQ and the implementation of a simulation-to-reality transfer procedure.

The contributions of this chapter are as follows -

- This chapter introduces a carefully designed simulation environment tailored for benchmarking RL algorithms in UAV path planning. Various state and action space representations are explored, and their effectiveness is compared based on the maximum rewards achieved. Additionally, the accuracy with which the RL algorithms can plan to reach a desired state is thoroughly evaluated, ensuring that the environment provides a robust testing ground for performance assessments.
- A detailed comparison of different off-policy RL algorithms, including DDPG, TD3, SAC, and SoftQ, is conducted. This analysis highlights the strengths and weaknesses of each algorithm, providing valuable insights into their performance across various

metrics such as reward optimization, collision avoidance, and overall stability in diverse UAV path planning scenarios.

- A systematic procedure is proposed to address the challenge of transferring RL models from simulation to real-world deployment. This transfer method focuses on mitigating the discrepancies between simulated and real-world environments, ensuring that the RL algorithms can effectively generalize and perform reliably in practical applications. The procedure serves as a crucial step towards bridging the gap between theoretical research and real-world implementation in UAV path planning.

3.1 Path Planning Benchmark Development

To develop the proposed benchmark for UAV path planning, a Gazebo simulation environment is employed, focusing on a quadrotor model—the Parrot Bebop 2. The primary objective is to navigate the UAV from a designated start pose to a specified goal pose while avoiding obstacles. The simulation environment is designed to closely replicate real-world conditions, providing a robust platform for training and evaluating various RL algorithms.

3.1.1 Simulation Setup and UAV Model

The UAV utilized in this benchmark is equipped with a 2D LiDAR sensor mounted on top of the quadrotor. This sensor plays a crucial role in integrating obstacle information into the UAV's state space, allowing the RL algorithms to effectively navigate through environments with obstacles. The LiDAR sensor features a minimum range of 0.12 m and a maximum range of 5 m, with a resolution of 1°. This setup enables the sensor to emit 360 rays, covering a full 360° view around the UAV. The information gathered from these rays provides a detailed understanding of the surrounding obstacles, which is vital for safe and efficient path planning.

3.1.2 State Space and Action Space Representation

The state space S for the RL algorithms is defined by two key components:

- **Error Vector** e_t: This represents the difference between the UAV's current position p_t and the goal position g_t in the UAV's base frame. Mathematically, it is expressed

as $e_t = p_t - g_t$. The inclusion of this error term in the state space ensures that the RL algorithm is directly informed of the UAV's progress toward its target. By continuously minimizing this error, the algorithm can focus on effectively guiding the UAV to the desired pose, which is crucial for generalizing the learned behavior across different goal positions and varying environmental conditions.

- **LiDAR Range Data**: The state space also includes the range data from the 360 rays emitted by the LiDAR sensor. This data provides real-time information about the distances to obstacles surrounding the UAV. By incorporating the LiDAR data into the state space, the RL algorithm gains a comprehensive understanding of the UAV's environment, enabling it to make informed decisions to avoid collisions. The ability to generalize across different environments is enhanced by this representation, as the algorithm can learn to navigate a wide variety of obstacle configurations based on the LiDAR feedback.

The action space of the RL agent consists of changes in velocity $\Delta v = [v_x, v_y, v_z]$ along the X, Y, and Z axes, respectively. These velocities dictate the movement of the UAV in three-dimensional space. In this research, the Parrot Bebop 2 UAV is used. It is a lightweight and agile quadcopter, known for its robust flight capabilities, providing an ideal platform for testing and validating the RL-based control strategies. As shown in Figure 3.1, the UAV's design supports the precise control required for complex tasks in three-dimensional environments, making it suitable for real-world reinforcement learning applications. The Parrot Bebop 2's maximum velocity is capped at 1.5 m/s, and its maximum acceleration is limited to 2 m/s^2, ensuring that the UAV operates within safe and realistic physical constraints. The predicted action from the RL algorithm is executed over a time interval $\Delta t = 0.07$ seconds, allowing the UAV to adjust its course dynamically as it progresses toward the goal.

3.1.3 Reward Function Design

The reward function is a critical component of the RL training process, guiding the learning algorithm by assigning rewards or penalties based on the UAV's actions. The reward function R_t is defined as follows:

- **Stable UAV Condition**: If the UAV remains stable at time t, the reward is given by $-10e_t$, where e_t is the error vector between the current position and the goal position. The weight -10 was determined through trial and error. This negative reward encourages the UAV to minimize the distance to the goal.



Figure 3.1: Parrot bebop 2 UAV

- **Goal Region**: If the UAV successfully reaches the goal region at time t, a positive reward of +10 is awarded. This incentivizes the RL agent to reach the target location.
- **Collision Penalty**: If the UAV collides with an obstacle or behaves erratically, a penalty of -50 is imposed. This significant penalty discourages unsafe maneuvers and promotes collision-free navigation.

3.1.4 Training Environment and Obstacle Configuration

The RL algorithms are initially trained in a relatively simple environment, featuring two cylindrical obstacles with a radius of 0.1 m and a length of 5 m. These obstacles are randomly placed within the environment to introduce variability in the training scenarios. The goal positions are chosen from a small bounding box defined by the coordinates $x \in [-1, 1]$ m, $y \in [-1, 1]$ m, and $z \in [1, 3]$ m. This restricted goal region provides a focused area for training, allowing the algorithms to learn effective navigation strategies within a controlled space as shown in Figure 3.2.

During the training phase, the RL algorithms are given a maximum of 10 timesteps to find a collision-free path to the goal pose. A collision is assumed to occur if the UAV comes within 0.2 m of an obstacle. This stringent criterion ensures that the RL algorithms are trained to prioritize safety and avoid close encounters with obstacles.

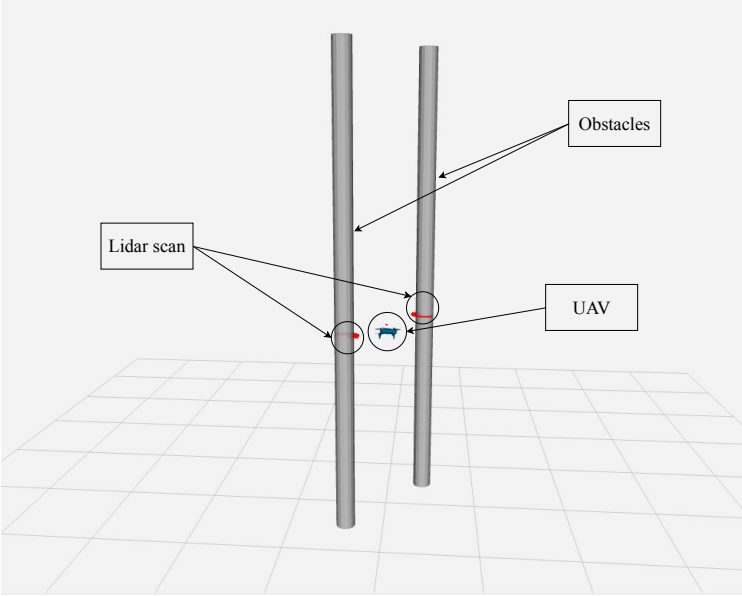


Figure 3.2: The gazebo environment used for training RL algorithms. The environment contains two cylindrical obstacles and a UAV attached with LiDAR sensor.

3.1.5 Generalization to Complex Environments

After completing the training in the simple environment, the RL algorithms are tested in more complex environments without any additional training. This step is crucial for evaluating the generalization capabilities of the algorithms—specifically, their ability to transfer the learned behaviors from the training environment to more challenging and diverse scenarios. The performance of each algorithm in these complex environments provides valuable insights into their robustness and adaptability, which are essential for real-world UAV path planning applications.

The benchmark developed in this chapter thus serves as a comprehensive framework for evaluating the effectiveness of off-policy RL algorithms in UAV path planning. By systematically testing the algorithms in both simple and complex environments, the study offers a detailed analysis of their strengths and weaknesses, contributing to the ongoing advancement of autonomous UAV navigation technologies.

3.2 Performance Evaluation of Reinforcement Learning Algorithms

In this section, the performance of different RL algorithms are evaluated for UAV path planning, specifically focusing on their ability to navigate from a start pose to a goal pose in environments with varying obstacle configurations. The algorithms considered include DDPG, TD3, SAC, and SoftQ.

3.2.1 Neural Network Architecture and Training

All the RL algorithms employ variants of a deep neural network architecture, as depicted in Figure 3.3, to represent the policy, state-value, and Q-value functions. The neural network serves as the backbone for these algorithms, facilitating decision-making by approximating the complex relationships between the state inputs and the optimal actions to be taken. The training process for each algorithm involved running $50,000$ episodes, as this is determined to be sufficient for the algorithms to reach stable, asymptotic behavior within the simulated environment. Training is conducted on a GeForce GTX TITAN X GPU, ensuring that the computations are efficient and that the algorithms could learn from extensive simulations in a reasonable timeframe.

Figure 3.4 illustrates the average rewards attained by the different algorithms during training, with the results averaged over five replicas to ensure robustness. The rewards serve as a key indicator of how effectively each algorithm learns to navigate the UAV through the environment while maximizing performance and minimizing collisions.

3.2.2 Evaluation Metrics

To assess the viability of employing RL for UAV control, several key performance metrics are considered:

- **Maximum Reward Attained During Training** (R_{max}): This metric measures the highest reward achieved by an algorithm during the training phase, indicating its ability to optimize the navigation task.
- **Frequency of Constraint Violations** ($const_{vio}$): This metric quantifies how often the algorithm violated constraints, such as colliding with obstacles or failing to adhere to safe navigation paths. Lower values indicate better adherence to safety and robustness criteria.

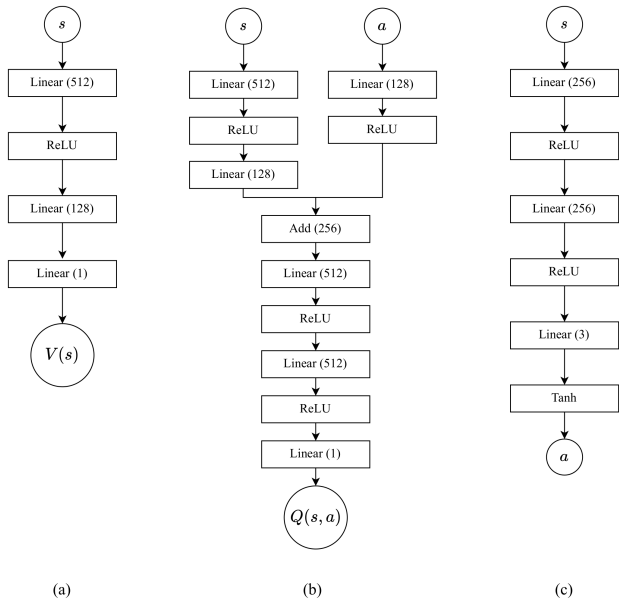


Figure 3.3: Neural networks representing (a) state value function, (b) Q-function and (c) policy function used for training the RL algorithms.

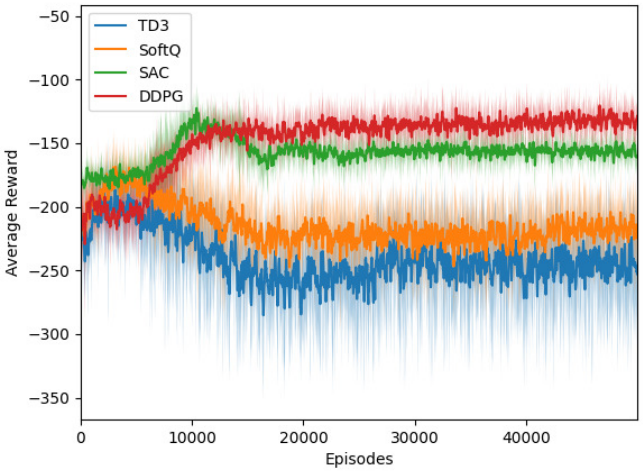


Figure 3.4: Comparison of rewards achieved by TD3, SAC, DDPG, and SoftQ algorithms after sampling 50K episodes.

- **Execution Time** (T_{exec}): This metric measures the time required by the RL algorithm to generate waypoints based on the input state, which is critical for real-time deployment. Faster execution times are preferred for timely decision-making in dynamic environments.

Table 3.1 presents the performance of the different algorithms across these metrics. The DDPG algorithm demonstrated superior performance in terms of maximum reward, outperforming SAC, TD3, and SoftQ by 89%, 128%, and 16.7%, respectively. This indicates that DDPG is more effective in optimizing the UAV's path to achieve higher rewards during training. Furthermore, all algorithms exhibited execution times within acceptable thresholds for real-time deployment, with DDPG again leading with the fastest execution time of 0.46 milliseconds, making it highly suitable for real-world applications where quick responses are essential.

Table 3.1: Performance of different RL algorithms during training

Algorithm	R_{max}	$const_{vio}$ (%)	T_{exec} (ms)
DDPG	-50.23	4.8	0.46
TD3	-115.48	8.26	0.51
SAC	-95.12	3.21	0.98
SoftQ	-60.35	8.09	0.56

3.2.3 Generalization to Unseen Environments

Generalization is a critical aspect of deploying RL algorithms in real-world scenarios, as it determines the algorithm's ability to perform well in environments that differ from those seen during training. To facilitate this generalization, the environment is simplified during the training phase, and the state space is expressed in terms of error and LiDAR data. By focusing on the error between the current and goal positions and incorporating LiDAR data to represent the obstacle space, the RL algorithms are equipped to navigate various environments effectively, regardless of the starting position or obstacle configuration.

The generalization performance of the algorithms is tested in diverse environments with varying obstacle layouts of same height, as shown in Figure 3.5. Figures 3.6a to 3.6f illustrate the paths generated by the algorithms in these different environments. In environments with four (Figure 3.6a), five (Figure 3.6b), and six (Figure 3.6c) obstacles, TD3 consistently managed to reach the goal region without collisions. TD3 initially avoided

obstacles by moving away from the goal, but upon recognizing this deviation, it corrected its path and successfully reached the goal. DDPG showed similar behavior but collided with an obstacle while maneuvering around the final obstacle.

In more complex scenarios, with an increased number of obstacles and greater distances between the start and goal positions (Figures 3.6d to 3.6f), the performance of all algorithms deteriorated when tested over 100 different runs. None of the algorithms could generate a collision-free path, and all collided with obstacles while attempting to navigate through the environment. In the most complex environment featuring 22 obstacles, both TD3 and DDPG initially moved in the wrong direction before colliding with obstacles, while SAC and SoftQ, despite moving towards the goal, are unable to avoid collisions.

These results highlight the challenges of generalization in RL-based UAV path planning, particularly in complex and dynamic environments. While some algorithms, like TD3, show promise in simpler scenarios, further research and refinement are needed to enhance the robustness and adaptability of these algorithms in more challenging environments. This study underscores the importance of optimizing state space representations, reward functions, and algorithmic strategies to improve generalization and ultimately achieve reliable UAV navigation in real-world settings.

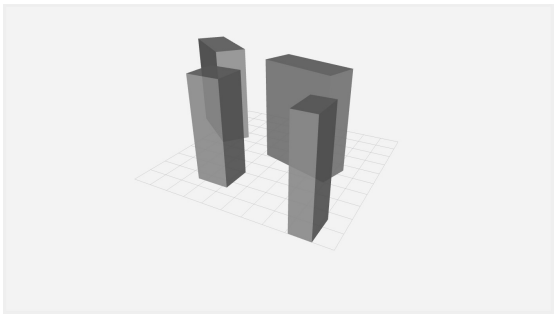
3.3 Experiments

This section presents the results of the real-world experimentation of the best-performing RL algorithm, TD3, deployed on a Parrot Bebop 2 quadrotor. The experiment aimed to validate the algorithm's performance in navigating a UAV through a controlled environment populated with obstacles, simulating real-world conditions.

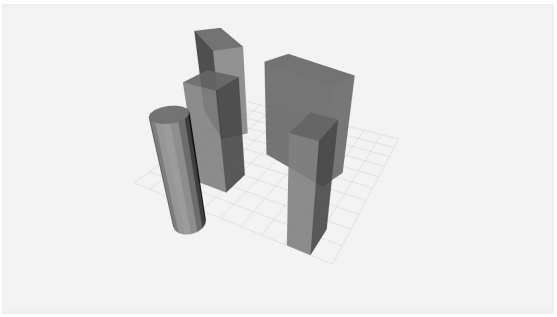
3.3.1 Experimental Setup

The test environment is constructed with three obstacles, specifically designed to challenge the UAV's path-planning capabilities. The obstacles included two triangular prisms, each with an approximate length of 0.7 m and a height of 1.7 m, and a cuboid with dimensions of 0.5 m by 0.5 m by 1.45 m. The UAV is required to traverse this environment, maneuvering through the narrow spaces between the obstacles to reach a designated target position on the opposite side as shown in Figure 3.7.

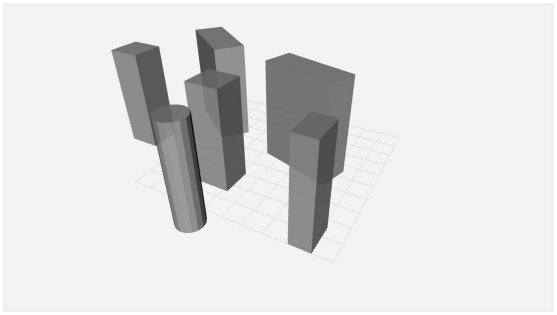
To accurately capture the obstacle positions and ensure precise interaction with the UAV, a VICON Motion Capture system is employed. This system provided real-time



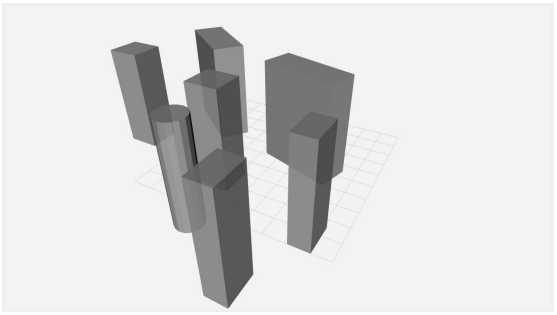
(a)



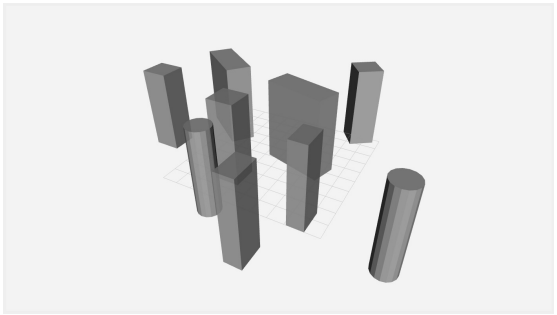
(b)



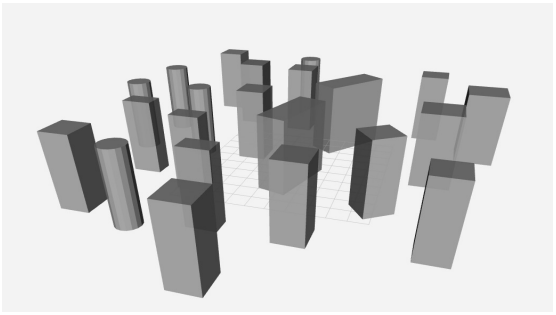
(c)



(d)



(e)



(f)

Figure 3.5: The obstacle worlds used to test different RL algorithms. (a) 4 obstacles, (b) 5 obstacles, (c) 6 obstacles, (d) 7 obstacles, (e) 9 obstacles and (f) 22 obstacles.

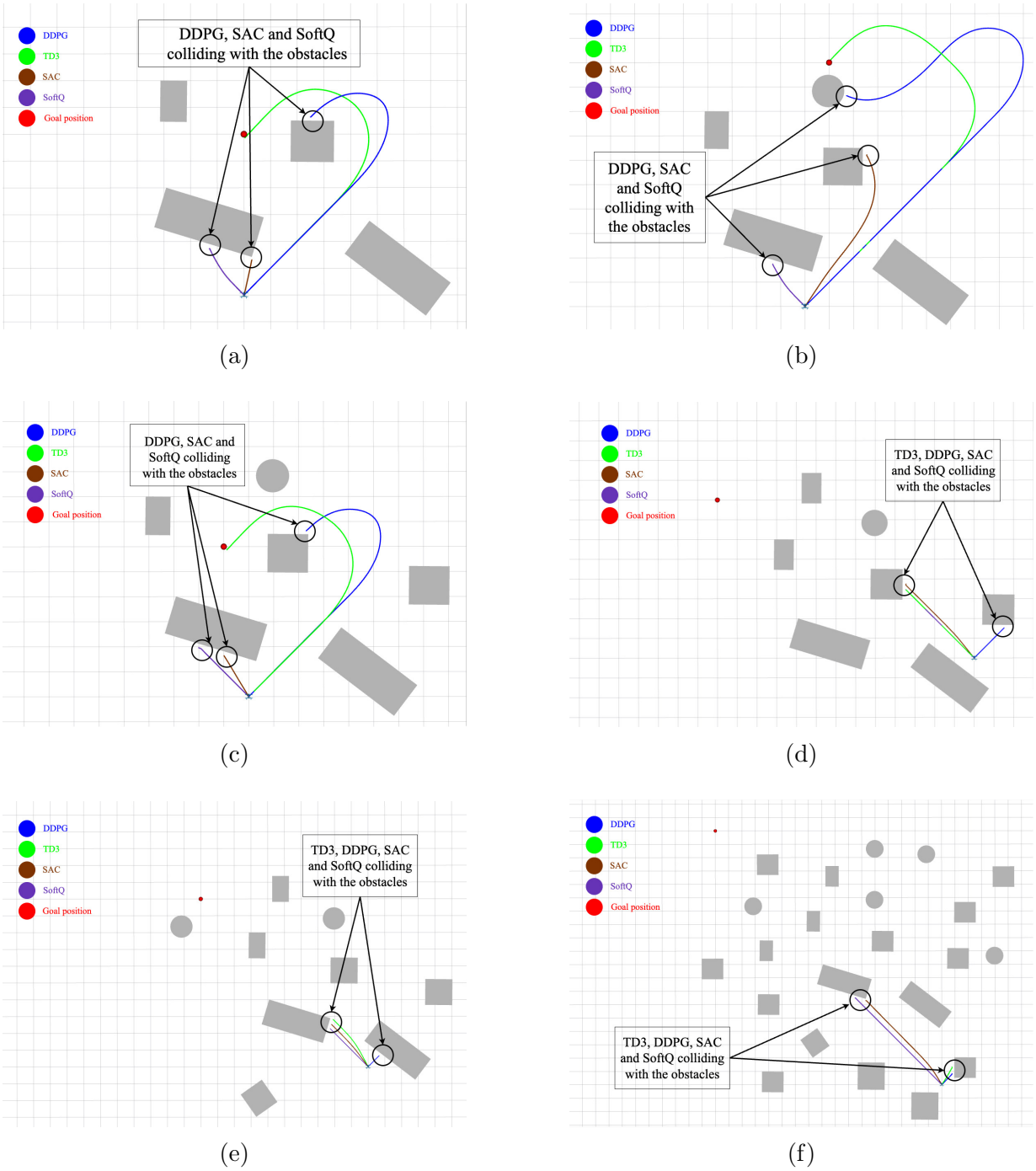


Figure 3.6: Resulting paths after testing the RL algorithms on environments having (a) 4 obstacles, (b) 5 obstacles, (c) 6 obstacles (d) 7 obstacles, (e) 9 obstacles, and (d) 22 obstacles.

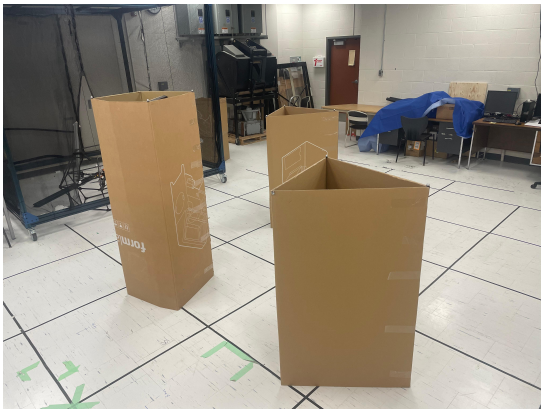


Figure 3.7: Experimental setup consisting of three obstacles, two triangular prisms and one cuboid.

updates on the center pose of each obstacle, which are then used to generate a LiDAR scan simulating the presence of the obstacles. The LiDAR data, crucial for obstacle detection and avoidance, is derived based on the obstacle shapes and their poses provided by the VICON system.

3.3.2 Deployment and Path Planning using TD3

Given its superior performance in environments with fewer obstacles during the simulation phase, TD3 is chosen for the experimental validation. The algorithm is tasked with generating a collision-free path for the UAV in the real-world environment described above.

As shown in Figure 3.8, the path generated by TD3 is visualized using RViZ, an open-source 3D visualization tool often used in robotics. The figure also includes snapshots from the external camera, UAV camera, and RViZ simulation, providing a comprehensive view of the UAV's navigation process.

During the experiment, TD3 successfully generated a path that avoided collisions, even when the UAV came dangerously close to the first triangular obstacle. The UAV, equipped with a 2D LiDAR sensor, detected the obstacle as it approached and adjusted its trajectory accordingly. This real-time adjustment highlights the effectiveness of the LiDAR data integration and the robustness of the TD3 algorithm in real-world scenarios.

The UAV's ability to sense and respond to obstacles in its environment is a critical aspect of safe navigation, particularly in cluttered or dynamic environments. In this ex-



Figure 3.8: The path traversed by the UAV while navigating through the obstacles. The external camera view, UAV camera view and RViZ simulation are shown together.

periment, the UAV's path is continuously adjusted to avoid obstacles, demonstrating the algorithm's capacity to generalize from simulated environments to the real world effectively.

3.3.3 Observations and Insights

The real-world experiment provided valuable insights into the practical application of the TD3 algorithm for UAV path planning. While the algorithm performed well in a relatively simple environment with fewer obstacles, the experiment also highlighted areas for potential improvement. For instance, while TD3 managed to avoid collisions, the UAV's path is not always optimal in terms of distance or smoothness. The UAV's close encounter with the first triangular obstacle, although successfully mitigated, suggests that further refinement of the obstacle avoidance mechanism could enhance the algorithm's overall efficiency.

Moreover, the reliance on LiDAR data for obstacle detection and path planning proved effective in this controlled environment. However, in more complex or dynamic real-world scenarios, additional sensors or more advanced perception algorithms may be required to ensure consistent performance. The experiment also underscored the importance of precise obstacle detection and pose estimation, which are facilitated by the VICON system in this case. In the absence of such a system, the UAV's ability to navigate accurately could be compromised, highlighting the need for robust sensing and perception capabilities in real-world deployments.

3.4 Summary

This work proposed benchmarking of off-policy deep RL algorithms applied to path planning of the UAVs. The benchmarking is performed in a simulation environment that closely mimics real-world scenarios, focusing on velocity level planning of UAVs in the presence of obstacles. The selected algorithms for benchmarking include DDPG, TD3, SAC, and SoftQ.

The results of the benchmarking process reveal notable differences in the performance of these algorithms. DDPG achieves the highest maximum reward during training, but its performance degrades in more complex environments. TD3 demonstrates robustness by successfully navigating through environments with varying obstacle configurations. SAC and SoftQ, while achieving competitive results during training, struggle in handling more cluttered scenarios. All the algorithms fail to generate a collision-free path in very complex environments, where the start and goal positions are far away. The benchmarking evaluates the algorithms based on metrics such as maximum reward, constraint violations, and execution time. Additionally, the generalization capabilities of the trained algorithms are tested in environments with different obstacle configurations. The benchmarking process provides insights into the strengths and weaknesses of each algorithm, aiding researchers and developers in selecting the most suitable algorithm for specific UAV planning tasks.

Chapter 4

Dual Agent Learning Based Aerial Trajectory Tracking

This chapter extends the work of Chapter 4, extending the RL framework for trajectory tracking of UAV in cluttered environments. While Chapter 4 introduced the core principles of using RL for UAV motion planning, focusing on single-agent architectures, this chapter advances the methodology by integrating a dual-agent architecture to address the challenges of dynamic environments.

Traditional optimization techniques discussed earlier have been shown to struggle with computational efficiency and robustness, particularly in the face of environmental uncertainties. To address these limitations, a novel DRL based approach is introduced that utilizes 3D point cloud data for environmental perception. Unlike memory-intensive methods such as occupancy grids, our approach enables real-time processing and decision-making.

The dual-agent system is designed to handle the complexities of trajectory tracking and collision avoidance simultaneously. One agent is responsible for predicting the UAV's velocities to ensure accurate following of a reference trajectory, while the second agent is dedicated to real-time obstacle management. This division of responsibilities allows for a more adaptive and resilient system capable of operating in both static and dynamic environments.

Building on the experiments and simulations presented in Chapter 4, this chapter showcases the enhanced performance of our dual-agent system, comparing it against both state-of-the-art RL models and traditional optimization-based methods. The use of a curriculum learning paradigm is further explored, allowing the system to scale effectively to increasingly complex scenarios, thereby ensuring robust and reliable UAV operation in real-world

applications.

The contributions of this chapter are as follows -

- This chapter presents a novel RL framework specifically designed for trajectory tracking in unknown and cluttered environments. By leveraging 3D point cloud data as the sole sensor input, the framework eliminates the need for memory-intensive obstacle representations like OctoMaps, thus enhancing efficiency and scalability.
- A key contribution is the proposal of a two-agent RL architecture that ensures near-optimal trajectory tracking in both static and dynamic environments. This architecture facilitates real-time performance and adaptability to environmental uncertainties, balancing computational demands and improving overall robustness.
- Extensive experiments conducted in simulated and real-world settings demonstrate the effectiveness of the RL framework in navigating unknown and cluttered environments. The chapter includes a comparative analysis with existing state-of-the-art RL and optimization-based algorithms, highlighting the framework's advantages in terms of efficiency, accuracy, and adaptability.
- To scale RL algorithms to more complex environments, a curriculum learning paradigm is introduced. This approach systematically increases task complexity, allowing the RL model to progressively improve and effectively handle intricate and demanding scenarios. This contribution supports the generalizability and practical application of the RL framework.

4.1 Proposed Method

Trajectory tracking is a critical task in autonomous UAV navigation, where the UAV must closely follow a predefined reference trajectory $Q_{\text{ref}}(t) = [x_{\text{ref}}(t), y_{\text{ref}}(t), z_{\text{ref}}(t)]$. The UAV's position at time t is denoted by $Q(t) = [x(t), y(t), z(t)]$, and the objective is to minimize the tracking error $e(t)$, defined as the Euclidean distance between the UAV's current position and the reference trajectory

$$e(t) = \|Q_{\text{ref}}(t) - Q(t)\|_2. \tag{4.1}$$

The goal is to ensure that this tracking error remains within a predefined threshold $\epsilon_K > 0$, such that

$$e(t) \leq \epsilon_K. \tag{4.2}$$

This condition ensures that the UAV closely adheres to the reference trajectory during the flight episode. However, when the UAV detects an obstacle along its path, it must temporarily deviate from the reference trajectory to avoid a collision, and then rejoin the trajectory once the obstacle is cleared.

To address the dual challenge of trajectory tracking and collision avoidance, a two-agent RL architecture is proposed. This architecture is designed to switch between different policies depending on the UAV's configuration and sensor data, thereby ensuring robust navigation in complex environments. The dual-agent architecture is built upon TD3 based on results observed in Chapter 4.

4.1.1 Dual-Agent Architecture for Trajectory Tracking

The proposed architecture comprises two agents: one dedicated to trajectory tracking and the other to collision avoidance as shown in Figure 4.1. The trajectory tracking agent (π_{trt}) is responsible for following the reference trajectory in obstacle-free scenarios, ensuring that $e(t)$ remains within the allowable threshold ϵ_K. The collision avoidance agent (π_{cva}), on the other hand, is triggered when the UAV enters a state belonging to the collision trigger set $S_{\text{col}} \subseteq S$. This set comprises states where an imminent collision is detected, necessitating a deviation from the trajectory to avoid obstacles.

The set of states that do not belong to the collision trigger set is defined as the trajectory set $S_{trt} = S \setminus S_{\text{col}}$. When the UAV is in a state belonging to S_{trt}, the trajectory tracking agent π_{trt} is activated, guiding the UAV along the reference path.

By dynamically switching between these two agents based on real-time sensor data, the UAV can maintain a balance between accurate trajectory tracking and effective collision avoidance. This dual-agent RL architecture is crucial for navigating complex environments where obstacles are present, ensuring that the UAV can safely and efficiently reach its destination.

This approach provides a robust solution for UAV trajectory tracking in unknown, cluttered environments, leveraging the strengths of various off-policy RL algorithms to handle both trajectory adherence and obstacle avoidance effectively.

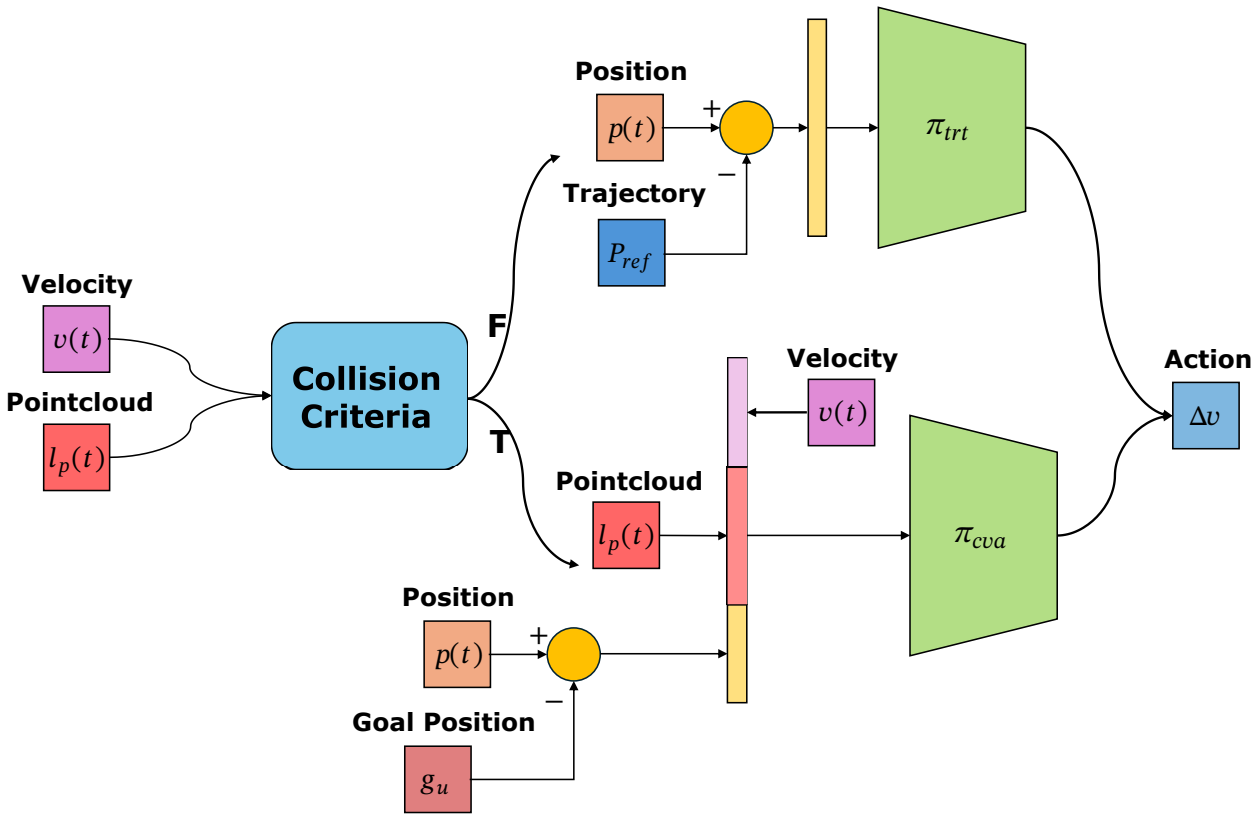


Figure 4.1: The pipeline for our dual-agent architecture.

4.1.2 Trajectory Tracking Policy

The trajectory tracking policy π_{trt} is designed to control a UAV in such a way that it closely follows a predefined reference trajectory $Q_{ref} = [q(0), q(t_1), q(t_2), \ldots, q(t_n)]$, where each $q(t_i)$ represents a target position that the UAV needs to reach at time t_i for $i = 1, 2, \ldots, n$. The effectiveness of this policy hinges on the UAV's ability to minimize the error between its current position $q(t)$ and the reference points along the trajectory, both at the current time and up to a certain number of steps into the future.

The state space S for the trajectory tracking policy is crucial for accurately capturing the UAV's positioning relative to the reference trajectory. Specifically, S includes the errors between the UAV's current position $q(t)$ and the reference points from the current time t up to $t + m$ steps into the future. Formally, the state space S is defined as

$$S = \{e(t), e(t+1), \ldots, e(t+m)\}, \tag{4.3}$$

where the error at time $t + k$ is given by

$$e(t+k) = q(t) - q(t+k) \quad \text{for } k = 0, 1, \ldots, m. \tag{4.4}$$

This formulation allows the policy to consider not only the immediate error but also the anticipated future errors, enabling more proactive and effective control decisions. The action space for the policy π_{trt} involves the control of the UAV's velocity in three dimensions—$\Delta v = [\Delta v_x, \Delta v_y, \Delta v_z]$, representing changes in velocity along the X, Y, and Z axes, respectively. The predicted action (i.e., the computed change in velocity) is executed on the UAV for a short time interval Δt. This process continues iteratively, with the policy continuously generating control actions to drive the UAV towards the reference trajectory until the UAV reaches the designated goal position.

The performance of the RL algorithm used to train π_{trt} is guided by a carefully designed reward function. The reward function is structured to incentivize the UAV to stay on the reference trajectory while minimizing deviations and errors. The reward at any time t is defined as

$$\mathrm{R}(t) = \begin{cases} \alpha_4 & \text{if UAV follow trajectory} \\ r_1(t) & \text{otherwise} \end{cases} \tag{4.5}$$

Here, the reward $r_1(t)$ is calculated as

$$r_1(t) = -\alpha_1 \|e(t)\|_2^2 - \alpha_2 \|e_g(t)\|_2^2 - \alpha_3 d(t), \tag{4.6}$$

where, $\|e(t)\|_2^2$ is the l_2 norm of the error between the UAV's current position and the current reference point $q(t)$, $\|e_g(t)\|_2^2$ is the l_2 norm of the global error between the UAV's current position and the ultimate goal position g_u and $d(t)$ represents the deviation of the UAV from the reference trajectory, calculated as the perpendicular distance from the UAV's current position to the segment of the trajectory extending from t to $t + m$. The global error term $\|\mathbf{e_g}(t)\|_2^2$ is included to prompt the UAV to move towards the goal. It is observed that using only the tracking error $\|\mathbf{e}(t)\|_2^2$, the UAV would either learn not to move or in some cases, exhibit back-and-forth motion without reaching the goal position.

The constants α_1, α_2, α_3, and α_4 are positive scalar parameters that weight the different components of the reward function, balancing the UAV's proximity to the reference trajectory, its progression towards the goal, and its overall stability. An episode is considered successful if the UAV consistently follows the reference trajectory within a permissible error bound. Specifically, the episode is deemed successful if the tracking error $\|e(t)\|_2^2$ remains below the threshold ϵ_K throughout the flight

$$\|e(t)\|_2^2 \leq \epsilon_K. \tag{4.7}$$

This criterion ensures that the UAV maintains close adherence to the reference trajectory, only deviating when necessary, such as to avoid obstacles. The integration of the state space, action space, and reward function into the RL framework allows for the training of a robust trajectory tracking policy that can navigate complex environments while achieving precise control and goal attainment.

4.1.3 Collision Avoidance Policy

In complex environments where obstacles may appear unexpectedly, the UAV's control must dynamically shift from the trajectory tracking policy π_{trt} to the collision avoidance policy π_{cva}. The collision avoidance policy π_{cva} is responsible for maneuvering the UAV around obstacles and ensuring it remains safe until it can return to its original trajectory. This policy is crucial for maintaining the UAV's safety and mission success, particularly in environments where obstacles are not static or are difficult to predict.

The state space S for the collision avoidance policy includes several key components that inform the UAV's decision-making process. These components are:

- **3D Point Cloud Data** $l_p(t)$: Provided by the UAV's LiDAR sensor, this data represents the surrounding environment, including the positions of potential obstacles.
- **Global Error** $e_g(t)$: The squared l_2 norm of the error between the UAV's current position $q(t)$ and the goal position g_u.
- **Current Velocity** $v(t)$: The UAV's velocity vector, which includes components $[v_x(t), v_y(t), v_z(t)]$ in the X, Y, and Z axes.

Formally, the state space S for the collision avoidance policy can be expressed as

$$S = \{l_p(t), e_g(t), v(t)\}. \tag{4.8}$$

This comprehensive state representation allows the policy π_{cva} to assess the UAV's environment, its current progress toward the goal, and its current motion dynamics, thereby enabling informed decisions for obstacle avoidance. Similar to the trajectory tracking policy, the action space for π_{cva} involves controlling the UAV's velocity in three dimensions. The policy outputs a change in velocity $\Delta v = [\Delta v_x, \Delta v_y, \Delta v_z]$, representing adjustments in the UAV's speed and direction along the X, Y, and Z axes, respectively. These velocity changes are executed on the UAV for a short time interval Δt, with the policy continuously generating new actions until the UAV successfully navigates around the obstacle and returns to the reference trajectory.

The reward function for π_{cva} is designed to incentivize safe navigation, penalize proximity to obstacles, and reward progress towards the goal. The reward at any time t is defined as

$$\mathrm{R}(t) = \begin{cases} r_2(t) & \text{if UAV is stable} \\ \alpha_8 & \text{if UAV reaches goal position} \\ -\alpha_9 & \text{otherwise} \end{cases} \tag{4.9}$$

where,

$$r_2(t) = -\alpha_5 \|e_g(t)\|_2^2 + \alpha_6 l_m(t) + \alpha_7 c(t). \tag{4.10}$$

In this formulation, $\|e_g(t)\|_2^2$ is the squared l_2 norm of the global error between the UAV's current position and the goal position g_u, $l_m(t)$ represents the minimum distance between the UAV and the nearest obstacle at time t and $c(t)$ accounts for any collision-related factors or constraints that the UAV needs to respect.

The reward $r_2(t)$ balances the UAV's progress toward the goal, its safety concerning nearby obstacles, and adherence to operational constraints. A significant penalty $-\alpha_9$ is imposed if the UAV violates any constraints, with the primary constraint being to avoid collisions with obstacles.

To quantify and manage the UAV's risk of collision, a collision heuristic based on the UAV's velocity and the 3D point cloud data from the LiDAR sensor is introduced. The collision heuristic assesses the likelihood of the UAV encountering an obstacle by calculating the UAV's trajectory relative to the obstacles detected in the environment. The heuristic is presented in Algorithm 1. The region l_r in Algorithm 1 is determined by analyzing the LiDAR scan in 0.1° increments. If a LiDAR value is less than 1 m, it is included in l_r until a value exceeds 1 m, marking the region's end. As the scan spans 0° to 360°, it would not suffer from angle wrapping.

Algorithm 1 Collision heuristic

Require: Velocity $v(t)$, 3D pointcloud $l_p(t)$
Ensure: *reward*
$reward \leftarrow 0$
$\theta_v \leftarrow \arctan(v_y(t), v_x(t))$
$l_r \leftarrow$ **calculate_regions**$(l_p(t))$
for $r \in l_r$ **do**
  /* Orientation of 1^{st} & end point in r */
  $\theta_s \leftarrow \arctan(r[0]_y, r[0]_x)$
  $\theta_l \leftarrow \arctan(r[\text{end}]_y, r[\text{end}]_x)$
  if $\theta_s \leq \theta_v \leq \theta_l$ **then**
    $reward = -\alpha_{10}\|v(t)\|$
    return *reward*
  else
    if N is odd **then**
      $val \leftarrow \min(\text{abs}(\theta_v - \theta_s), \text{abs}(\theta_v - \theta_l))$
      $reward \leftarrow \max(reward, val)$
    end if
  end if
end for
return *reward*

The collision heuristic calculates the reward based on the UAV's velocity vector $v(t)$ and the relative positions of detected obstacles. If the UAV's velocity vector aligns with

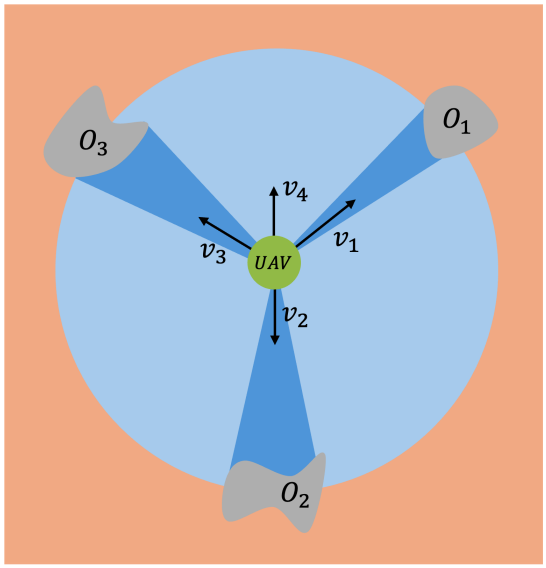


Figure 4.2: Schematic diagram of UAV trying to maneuver through obstacles. There is a UAV in the middle (green circle), with a LiDAR scan around it (blue circle). O_1, O_2 and O_3 are the obstacles, with different scenarios where the UAV is moving have either velocity v_1, v_2, v_3 or v_4.

the orientation of a detected obstacle region, a negative reward is assigned, indicating a higher risk of collision. Conversely, if the UAV is moving away from obstacles, the reward is more favorable.

Consider the scenario illustrated in Figure 4.2, where the UAV is surrounded by three obstacles labeled O_1, O_2, and O_3. The UAV's velocity vectors v_1, v_2, and v_3 represent movement toward the obstacles, resulting in negative rewards as these directions increase the risk of collision. On the other hand, velocity vector v_4 represents movement away from the obstacles, leading to a positive reward.

In this context, the reward function and collision heuristic work together to ensure that the UAV minimizes the risk of collision while continuing to make progress towards its goal. An episode is deemed successful if the UAV reaches a goal region, defined as a sphere of radius r_g centered around the goal position g_u, without violating any constraints such as colliding with obstacles.

The dual-agent RL framework, with its sophisticated collision avoidance policy π_{cva}, allows the UAV to navigate safely through complex environments, maintaining mission success even in the presence of dynamic obstacles.

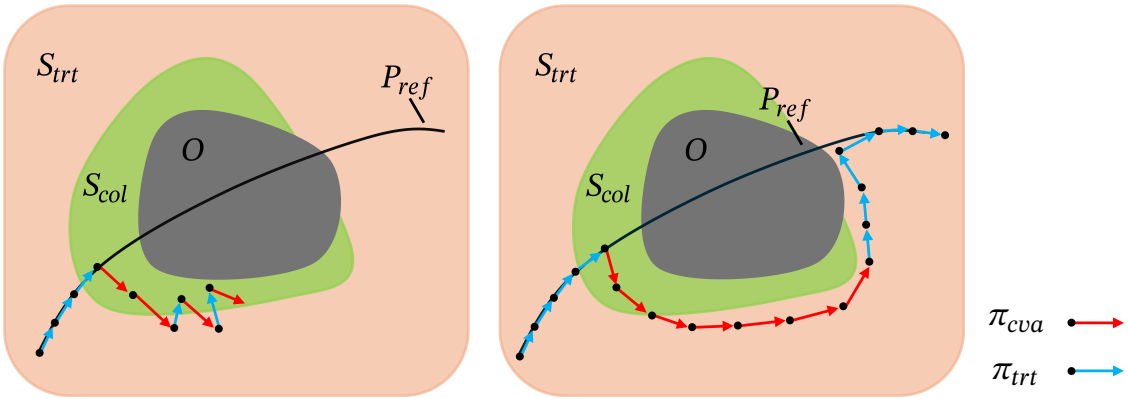


Figure 4.3: An illustration of collision set S_{col} (green) and trajectory set S_{trt} (orange) with two policies π_{cva} and π_{trt}. On the left, with the naive approach there is a constant back and forth between the collision avoidance policy π_{cva} and trajectory tracking policy π_{trt}. On the right, by applying the appropriate switching mechanism the collision avoidance agent pushes the UAV away from obstacle and switches to trajectory tracking policy only when safe.

4.1.4 Mechanism

This section elaborates on the decision-making mechanism for switching between the trajectory tracking policy π_{trt} and the collision avoidance policy π_{cva} during UAV navigation. The UAV is designed to follow a reference trajectory using π_{trt} but must avoid obstacles by temporarily switching to π_{cva}. However, a naive switching mechanism, as depicted on the left side of Figure 4.3, can cause the UAV to frequently switch between π_{trt} and π_{cva}, potentially trapping it near obstacles due to constant back-and-forth control.

The simple approach to policy selection is described by Equation 4.11

$$a_{simple} = \begin{cases} \pi_{cva}(\mathbf{s}) & \text{if } \mathbf{s} \in C_{col} \\ \pi_{trt} & \text{otherwise} \end{cases} \tag{4.11}$$

Here, $\mathbf{s}$ represents the current state of the UAV. If $\mathbf{s}$ lies within the collision set S_{col}, the collision avoidance policy π_{cva} takes control; otherwise, the trajectory tracking policy π_{trt} governs the UAV's actions. While this method theoretically ensures that the UAV tracks its trajectory and avoids collisions, it can lead to frequent, undesirable switching between the two policies.

To address the problem of excessive switching, a more sophisticated mechanism is

proposed. When the UAV enters the collision set S_{col}, π_{cva} takes over. However, control is only transferred back to π_{trt} when it is safe to do so, ensuring that no further switching will occur for a specified future horizon. This reduces the reliance on π_{cva} and minimizes unnecessary oscillations between policies.

The improved action selection mechanism which follows a structure similar to [74] is defined as follows

$$a_t = \begin{cases} \pi_{cva}(\mathbf{s}) & \text{if } (\mathbf{s}(t) \in S_{col}) \vee \\ & \qquad ((a_{t-1} \in \pi_{cva}) \wedge \\ & (\nexists \{a(\tau)\}_{\tau=t}^{w+t-1} \in \pi_{trt} \text{ s.t.} \\ & \qquad \{s(\tau)\}_{\tau=t+1}^{w+t} \notin S_{col})) \\ \pi_{trt} & \text{otherwise.} \end{cases} \tag{4.12}$$

In this approach, two primary conditions govern the switching:

- **Current State Check**: If the UAV's current state $\mathbf{s}(t)$ is within the collision trigger set S_{col}, π_{cva} takes control.
- **History-Dependent Switching**: If the previous action a_{t-1} is governed by π_{cva}, the UAV continues to follow π_{cva} unless there exists a sequence of future actions $\{a(\tau)\}_{\tau=t}^{w+t-1}$ from π_{trt} that would lead to states $\{s(\tau)\}_{\tau=t+1}^{w+t}$ outside S_{col}.

This heuristic ensures that the UAV only switches back to π_{trt} when it is certain that no further collisions will occur within the specified horizon, as demonstrated in the right side of Figure 4.3.

To implement this switching strategy, the UAV must meet the following collision criteria before switching back to π_{trt}:

- **Current State Evaluation**: The UAV's current state $\mathbf{s}(t)$ is assessed against the collision set S_{col} using a heuristic. If the heuristic determines that the UAV is on a collision course, the state is considered within S_{col}; otherwise, it is in S_{trt}.
- **Avoiding Excessive Switching**: To prevent the UAV from getting stuck in narrow spaces, the UAV should not switch frequently between π_{cva} and π_{trt}. This is ensured by checking that if the previous action is dictated by π_{cva}, there should not be any sequence of future actions from π_{trt} leading to states outside S_{col}. If such a sequence does not exist, the UAV continues with π_{cva}.

This methodology ensures safer and more efficient UAV navigation, balancing trajectory tracking with robust collision avoidance.

4.2 Methodology

4.2.1 Simulation Pipeline

The simulation pipeline for this research is built on the Robot Operating System (ROS) [75], a widely-used framework in robotics that facilitates the development, simulation, and deployment of robotic systems. For this study, all policies are trained within the Gazebo simulator, which is integrated with ROS to provide a realistic 3D environment for testing and training robotic algorithms.

4.2.2 UAV and Sensor Setup

The training process involves a simple quadrotor, specifically the Parrot Bebop 2 UAV, equipped with a 3D LiDAR sensor. The LiDAR sensor is critical for integrating obstacle information into the UAV's environment, thereby enriching the state space of the RL algorithm. The sensor features a minimum range of 0.12 m and a maximum range of 4 m, with a horizontal resolution of 1° that covers the entire circumference. In the vertical plane, the LiDAR point cloud spans from -30° to 30° with a 1° resolution, emitting a total of $21,600$ rays. This configuration allows the UAV to detect and react to obstacles effectively, which is crucial for safe navigation.

The UAV's dynamic constraints are also factored into the simulation: it has a maximum velocity of 1.5 m/s and a maximum acceleration of 2 m/s^2. The predicted action generated by the RL policies is executed on the UAV at a time step of $\Delta t = 0.04$ seconds. To ensure the computational efficiency of the LiDAR data processing, the point cloud received from the sensor is voxel downsampled with a voxel size of 0.08, resulting in approximately 200 filtered points.

4.2.3 Curriculum Learning for Policy Training

To effectively train the policies in increasingly complex environments, a curriculum learning strategy is employed. Curriculum learning involves starting with simpler tasks and

progressively advancing to more complex ones as the agent (in this case, the UAV) masters each level of difficulty [76]. Direct deployment in complex tasks resulted in sub-optimal behavior, with the UAV struggling in long-horizon planning in challenging environments.

For the training of the policy π_{trt}, the UAV initially operates in an empty world, tasked with following a simple straight-line trajectory. The starting position P_{start} is set at $[0, 0, 2]$ m, with goal positions chosen from a small bounding box defined by $x \in [-1, 1]$ m, $y \in [-1, 1]$ m, and $z \in [1, 3]$ m. After mastering this simple scenario, the UAV is then spawned at random positions within a defined workspace C_{rdm} and trained to follow the previously generated trajectory as shown in Figure 4.4a. This progressive approach helps the UAV to develop the necessary skills to navigate more complex environments [76].

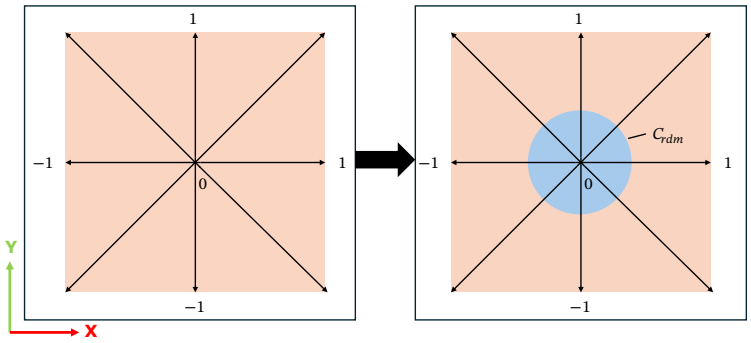
The policy π_{cva} is trained in several simulated environments, each designed to incrementally increase the difficulty of the trajectory tracking task as shown in Figure 4.4b.

- **Environment 1**: The first environment consists of two random cylindrical obstacles, each with a radius of 0.1 m and a length of 5 m. The goal positions in this scenario are within a bounding box of $x \in [-1, 1]$ m, $y \in [-1, 1]$ m, and $z \in [1, 3]$ m. The UAV starts each episode at the position $[0, 0, 2]$ m and must find a collision-free path to the goal within 20 timesteps. A collision is defined as the UAV coming within 0.35 m of an obstacle.

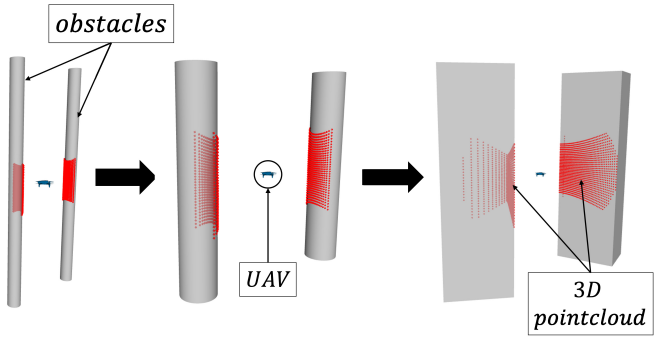
- **Environment 2**: The second environment increases the complexity by introducing two cylindrical obstacles with a larger radius of 0.25 m but the same length of 5 m. The goal positions are expanded to a bounding box of $x \in [-2, 2]$ m, $y \in [-2, 2]$ m, and $z \in [1, 3]$ m.

- **Environment 3**: The final environment presents the most challenging scenario, featuring cuboid obstacles with dimensions of $[1.5, 1, 5]$ m. The goal positions are selected from a bounding box of $x \in [-4, 4]$ m, $y \in [-4, 4]$ m, and $z \in [1, 3]$ m.

The parameter values for the algorithm (α_i for $i = 1, \ldots, 10$ are determined through trial and error with values ranging from $[2.0, 300.0]$. These values are fine-tuned to balance the trade-off between exploration and exploitation during the training process, ensuring that the UAV could effectively learn to navigate complex environments while minimizing collisions.



(a)



(b)

Figure 4.4: Curriculum training for trajectory tracking policy π_{trt} and collision avoidance policy π_{cva}. (a) The environment used to train $\pi)trt$ seen from X-Y plane (b) For π_{cva} it starts with a simple two obstacle world containing two cylinders with radius 0.1 m. Following that the policy is trained in an environment where the radius of the cylinder is 0.3 m. Finally it is trained in an environment containing cuboids.

4.3 Simulated Results

In this section, the results from our previous work [77] are used, where TD3 was employed as the base algorithm for each policy in our dual-agent RL framework. In our study, the effect of network architecture on the success rate is systematically evaluated through an ablation study as shown in Figure 4.5. The success rate is defined based on 100 random iterations with random starting and goal positions, where the algorithm is considered successful if it reaches the goal without colliding and following the reference trajectory with a maximum deviation of less than 2 m; otherwise, it is deemed a failure. This analysis examined various configurations of hidden layers and neurons per layer to determine the optimal network design for the TD3 algorithm. The results indicated that a network with three hidden layers and 256 neurons per layer yielded the highest success rate, achieving up to 83.3% in simulation. Specifically, the collision avoidance policy π_{cva} is trained for $50,000$ episodes, while the trajectory tracking policy π_{trt} is trained for 10,000 episodes. These durations are selected as TD3 demonstrated stable and asymptotic behavior within the training environment by these points. Training is conducted using a GeForce GTX TITAN X GPU. Following training, the policies are tested in both simulation and physical experiments using on an Intel Nuc Core i5-8259U with 16GB RAM. The choice of neural network architecture is critical, as it directly influenced the algorithm's performance across the environments.

To evaluate the effectiveness of the proposed approach, extensive simulation studies are conducted across various scenarios:

- **Static Obstacles Environment (Scenario I)**: In this scenario, the UAV navigates through an environment with fixed obstacles.
- **Warehouse Scenario (Scenario II)**: This static obstacle scenario simulates a typical warehouse environment with shelving and other structural obstacles.
- **Drone Racing Scenario (Scenario III)**: The UAV must navigate through a series of static gates in a drone racing context, emphasizing speed and precision.
- **Dynamic Obstacles Environment (Scenario IV)**: In this more complex scenario, the UAV must avoid obstacles that move within the environment, requiring real-time decision-making.

Table 4.1 presents the comparative results of the proposed dual-agent RL architecture, where TD3 is utilized for both the trajectory tracking policy π_{trt} and the collision avoidance policy π_{cva}. The dual-agent approach is benchmarked against three alternatives:

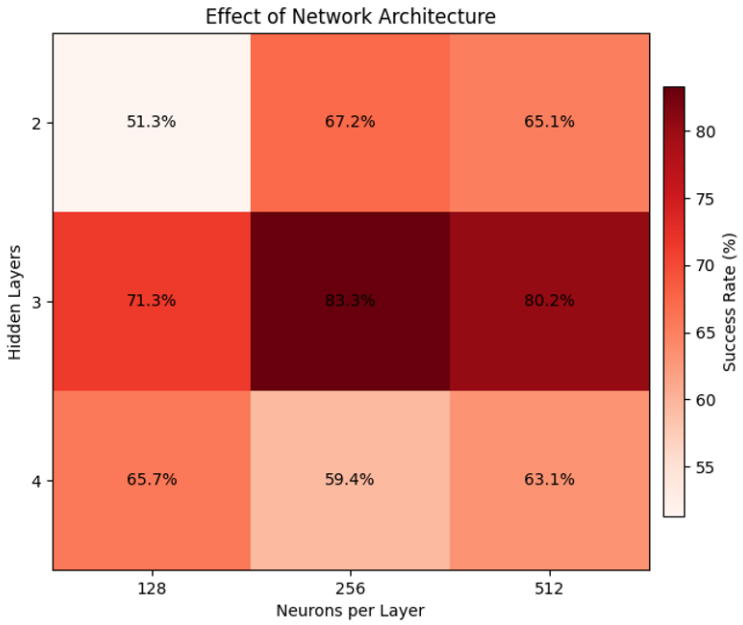


Figure 4.5: Effect of network architecture on performance of the proposed algorithm.

- **Single-Agent TD3**: This is a baseline where a single TD3 agent is responsible for both trajectory tracking and collision avoidance tasks.
- **Ego planner (EP)**: A SOTA motion planning algorithm that leverages gradient-based optimization to generate smooth and dynamically feasible trajectories while ensuring real-time obstacle avoidance [39].
- **Optimization-Based Algorithm (OTT)**: Another SOTA algorithm in trajectory tracking and obstacle avoidance tasks, as referenced in [37].

The algorithms are compared based on the following performance metrics:

- **Trajectory Length (L_{traj})**: The total length of the path taken by the UAV from start to goal.
- **Execution Time (T_{exec})**: The total time taken by the UAV to complete the trajectory.
- **Maximum Deviation from Reference Trajectory (M_{dev})**: The largest distance between the UAV's actual trajectory and the reference trajectory.

Additionally, the algorithms' computation time is evaulated, defined as the duration required to generate a path at each timestep. The proposed dual-agent framework achieved a computation time of 0.0002s, significantly outperforming OTT (0.006s) and EP (0.001s) in terms of efficiency. Additionally, Table 4.2 presents an ablation study on the impact of UAV velocity on success rates across different environments. This analysis evaluates how variations in UAV speed affect the system's ability to maintain accurate trajectory tracking and collision avoidance. The results highlight the trade-offs between faster maneuverability and the increased risk of constraint violations, providing insights into the optimal velocity range for safe and efficient operation.

Table 4.1: Performance of Dual Agent algorithm, EP and OTT in different environments.

Algorithm	L_{traj} (m)	T_{exec} (s)	M_{dev} (m)
5 Obstacle World			
Dual Agent	17.55	30	1.68
EP	20.23	34	2.13
OTT	24.78	63	4.21
9 Obstacle World			
Dual Agent	24.53	35	1.61
EP	25.87	42	2.87
OTT	27.24	78	4.81
Random Obstacle World			
Dual Agent	48.39	78	2.87
EP	56.712	82	12.35
OTT	57.88	90	4.93
Warehouse Obstacle World			
Dual Agent	20.94	32	3.87
EP	31.08	40	6.67
OTT	-	-	-
Drone Racing Obstacle World			
Dual Agent	59.60	95	2.95
EP	60.18	96	5.04
OTT	79.01	109	14.79
Dynamic Obstacle World			
Dual Agent	11.33	9.8	2.65
EP	-	-	-
OTT	13.23	15	3.12

Table 4.2: Effect of UAV Velocity on Success Rate. Each velocity column is divided into three categories: Proposed, EP, and OTT.

Environment	Velocity: 0.5 m/s			Velocity: 1.0 m/s			Velocity: 1.5 m/s			Velocity: 2.0 m/s		
	Prop.	EP	OTT	Prop.	EP	OTT	Prop.	EP	OTT	Prop.	EP	OTT
Rnd. Obs. World	81.3%	80.2%	80.9%	76.2%	74.3%	75.5%	72.9%	70.8%	71.1%	67.4%	64.2%	65.8%
Warehouse World	85.3%	83.6%	83.9%	77.9%	75.4%	76.1%	72.3%	70.0%	71.2%	63.4%	60.4%	61.6%
Drone Racing World	87.1%	86.2%	86.7%	72.8%	70.1%	70.9%	70.1%	67.9%	68.3%	62.9%	60.3%	61.6%

4.3.1 Static Environment Case Study

In this case study, the UAV is assigned the task of navigating predefined trajectories in three distinct environments containing randomly placed static obstacles. The first two environments featured five and nine obstacles, where the UAV followed a straight-line trajectory while ensuring obstacle avoidance. In the third and most complex environment, which contained 22 obstacles, the UAV navigated using a third-order spline trajectory, passing through multiple intermediate waypoints. Figure 4.6c presents qualitative results from this benchmark, illustrating the UAV's trajectory in the 22-obstacle environment.

In the environments with five and nine obstacles, our proposed dual-agent system, EP, and OTT successfully generated collision-free trajectories. However, our method demonstrated superior efficiency and accuracy in terms of trajectory length and deviation from the reference path. Specifically, in the five-obstacle environment as shown in Figure 4.6a, our algorithm produced a trajectory 15% shorter than EP and 40% shorter than OTT, with a maximum deviation of 1.68 meters, less than half of that observed with OTT. Furthermore, our method is computationally efficient, executing in less than half the time of OTT and requiring only 13% of EP's computation time. Similarly, in the nine-obstacle environment, our dual-agent framework generated a trajectory 11% shorter than OTT and 5% shorter than EP, while maintaining a 3.5 times lower deviation than OTT and 1.8 times lower than EP as shown in Figure 4.6b.

In the 22-obstacle environment, all three approaches—our proposed method, EP, and OTT—successfully reached the goal while following the reference trajectory. However, OTT exhibited a maximum deviation 41% greater than our algorithm, with a trajectory length 16% longer, and also displayed erratic movements along the Z-axis, often struggling to maintain stable trajectory tracking. Additionally, OTT halted before reaching the goal, highlighting its difficulty in handling highly cluttered spaces. Similarly, EP demonstrated a maximum deviation three times higher than our method and produced a trajectory 17% longer. While EP initially followed the trajectory with high accuracy, it significantly deviated midway through, struggling to avoid obstacles and ultimately failing to complete the trajectory. In this random obstacle environment, our proposed algorithm achieved a success rate of 81.3% at 0.5 m/s, outperforming EP by 1.4% and OTT by 0.5%. At a higher

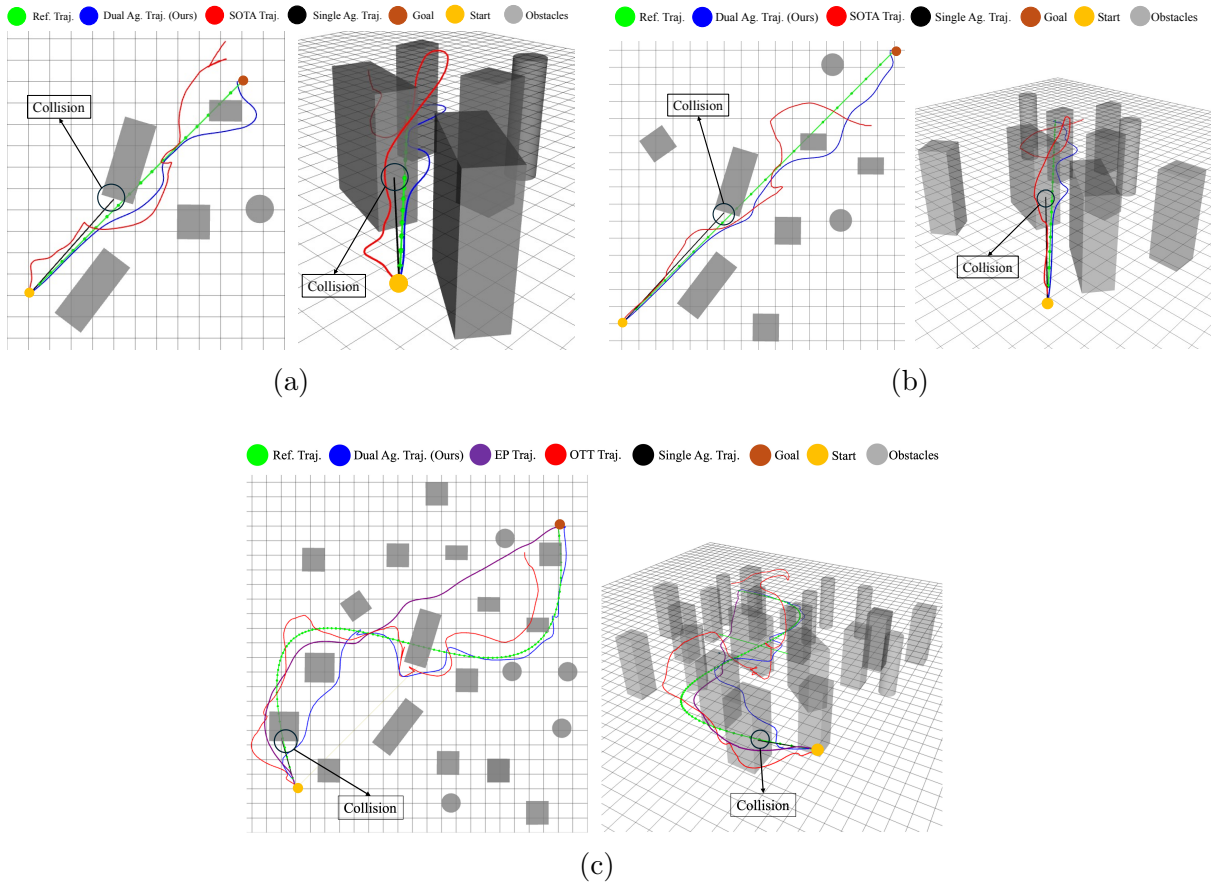


Figure 4.6: Comparison of trajectories generated in different obstacle worlds by RL algorithms, EP and OTT. (a) 5 obstacles, (b) 9 obstacles, (c) Random world with 22 obstacles. In all three scenarios, the proposed dual-agent algorithm tracks the trajectory more precisely, whereas OTT exhibits significant deviations along the Z-axis and produces a suboptimal trajectory, with EP failing to track the trajectory optimally.

speed of 2.0 m/s, our method maintained a success rate of 67.4%, showing an improvement of 5.0% over EP and 2.4% over OTT. The single-agent baseline, however, failed to complete any of the three environments, consistently colliding with obstacles, indicating its inability to adapt to dynamically constrained spaces.

An analysis is conducted to evaluate the robustness of the proposed algorithm in environments with varying obstacle densities. With five obstacles, the algorithm achieved a success rate of 89.1%. As the number of obstacles increased to nine, the success rate slightly dropped to 87.9%, and further declined to 83.9% with 20 obstacles. In highly cluttered environments, the success rate continued to decrease, reaching 78.3% with 100 obstacles and 73.1% with 150 obstacles. In the most challenging case, where the environment contained 200 obstacles, the algorithm maintained a success rate of 72.9%. These results indicate that while performance naturally declines as complexity increases, the proposed approach remains highly robust and capable of navigating dense, cluttered spaces with a consistently high success rate.

4.3.2 Warehouse Case Study

The RL algorithm is evaluated in a complex warehouse environment, where the UAV is required to follow an acyclic trajectory while avoiding obstacles and navigating between storage racks. The qualitative results for this benchmark are presented in Figure 4.7. Both OTT and the single-agent algorithm failed to generate a collision-free trajectory in this environment while attempting to follow the reference trajectory. The single-agent algorithm initially followed the straight-line path of the reference trajectory but collided with an obstacle early in the task execution, as highlighted by the black circle in Figure 4.7. OTT, on the other hand, exhibited erratic behavior, spiraling between the racks without successfully aligning with the reference trajectory, ultimately failing to complete the task.

EP initially demonstrated accurate trajectory following, successfully avoiding obstacles in the early stages of navigation. However, while making a turn between the shelves, it stalled at an obstacle, struggling to find a clear avoidance path. After resuming, EP failed to realign with the reference trajectory and ultimately missed the goal point. In contrast, our proposed algorithm successfully generated a collision-free trajectory, effectively maneuvering through the racks and obstacles while maintaining adherence to the reference path. Our approach reached the goal position with a total trajectory length 48% shorter than EP, while achieving a 67% lower maximum deviation and a 25% reduction in execution time. These results highlight the robustness of the proposed dual-agent system in handling complex, cluttered environments where traditional planning methods struggle to maintain efficiency and obstacle avoidance.

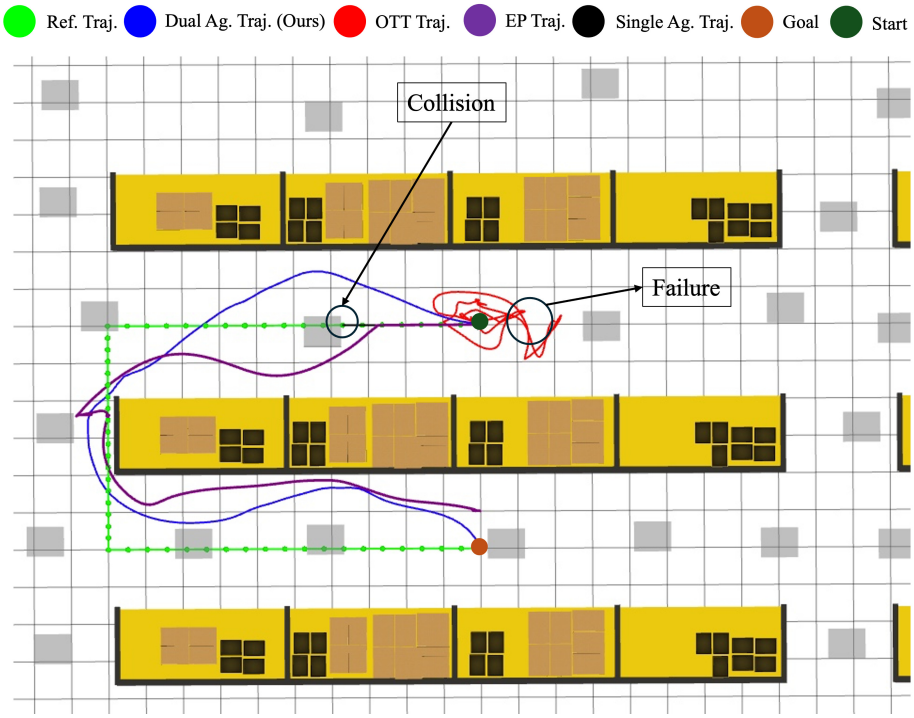


Figure 4.7: Performance of different algorithms in a complex warehouse. The OTT algorithm fails to find a path in the warehouse's confined space, while the proposed method successfully navigates to the goal pose, following the trajectory and avoiding obstacles. The EP algorithm initially follows the trajectory well but gets stuck at the obstacle while attempting to make the turn. After maneuvering around the obstacle, it fails to rejoin the trajectory

.

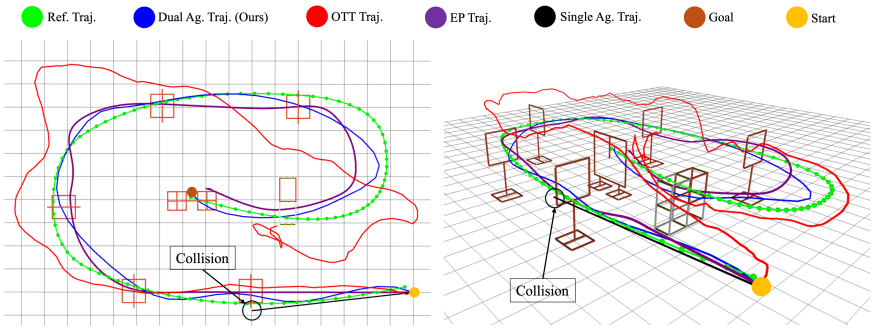


Figure 4.8: Drone racing environment. Our proposed algorithm closely follows the trajectory, successfully passing through the hoops. In contrast, OTT and EP fails to generate an optimal trajectory, leading to sub-optimal navigation.

4.3.3 Drone Race Case Study

In the drone race case study, the UAV is tasked with following a trajectory through a series of hoops in a static environment, simulating a competitive race. The course featured sharp turns and varying altitudes, challenging the UAV's trajectory tracking capabilities. The hoops are placed at different heights and distances, requiring the UAV to dynamically adjust its speed and direction to accurately pass through each checkpoint.

The proposed dual-agent system demonstrated superior accuracy and consistency compared to EP and OTT, closely following the intended path and successfully passing through the hoops. In contrast, OTT is unable to generate a trajectory that passed through the hoops due to the narrow spaces, resulting in large deviations from the reference trajectory. EP performed better than OTT in following the reference path but struggled with tight hoop placements, often failing to find a valid path through them or passing underneath instead. The dual-agent system successfully reached the goal with a trajectory length of 59.6 m and an execution time of 95 s. It achieved a success rate of 87.1% at 0.5 m/s, outperforming EP by 1.0% and OTT by 0.5%. At a higher speed of 2.0 m/s, the success rate remained at 62.9%, exceeding EP by 4.3% and OTT by 2.1%. These results highlight the adaptability of the proposed framework in precise trajectory tracking and maneuvering through constrained spaces.

4.3.4 Dynamic Obstacle Case Study

In the final case study, dynamic obstacles are introduced into the environment, moving toward the UAV at a velocity of 0.1 m/s. The RL algorithms operated without any

explicit knowledge of the obstacles' state or velocity, relying solely on point cloud data for real-time decision-making. The proposed dual-agent system successfully generated a safe, collision-free trajectory, dynamically adjusting its path based on environmental feedback. In contrast, the single-agent algorithm failed to effectively avoid the moving obstacle and resulted in a collision.

OTT is able to reach the target position while successfully avoiding obstacles, demonstrating its capability in dynamic environments. However, EP, which relies on static maps for trajectory planning, is not designed for such scenarios and, therefore, failed to adapt to the moving obstacles. These results highlight the effectiveness of our dual-agent framework in reactive motion planning, ensuring robust obstacle avoidance and trajectory tracking in unpredictable environments.

4.3.5 Discussion

The results from these simulations demonstrate that the dual-agent RL approach with TD3 outperforms both the single-agent TD3 and the SOTA optimization-based algorithm across all tested scenarios. The dual-agent approach effectively balances the UAV's need to track the trajectory while avoiding obstacles, leading to shorter trajectory lengths and lower execution times. Moreover, the reduced computation time per timestep highlights the efficiency of the proposed method, making it well-suited for real-time UAV navigation tasks.

The detailed results for each scenario are summarized in Table 4.1, which underscore the robustness and efficiency of our dual-agent RL approach in handling dynamic and static environments, ensuring reliable trajectory tracking and obstacle avoidance. Our findings contribute significantly to advancing UAV navigation systems, paving the way for more efficient and safer autonomous operations in complex environments. Despite the superior performance of our algorithm, there is an added complexity from using separate policies for trajectory tracking and collision avoidance, which may increase the computational burden and complicate real-time decision-making. Additionally, the system's heavy reliance on 3D pointcloud data makes it sensitive to sensor noise and inaccuracies, which could degrade performance in real-world scenarios with environmental interference. Additionally, the performance of the dual-agent system could improve if it is trained for several iterations in dynamic obstacle environments or in static environments with complex obstacles.



Figure 4.9: Holybro X500 V2 UAV.

4.4 Experiments

To validate the algorithm in real-world conditions, experiments were conducted using the Holybro X500 V2 UAV. The Holybro X500 V2 is a robust and highly flexible quadcopter designed for professional applications, offering a reliable platform for testing autonomous flight systems as shown in Figure 4.9. Equipped with powerful motors, high-precision GPS, and an advanced flight controller, the UAV is capable of performing in a variety of environments. This makes it ideal for testing the UAV-agnostic nature of our algorithm. The experiments were carried out across three progressively complex scenarios to assess the algorithm's adaptability and performance across different real-world conditions. In the first case, the UAV followed a straight-line trajectory while avoiding a single triangular obstacle, testing basic collision avoidance. The second scenario involved navigating between two closely spaced obstacles, challenging trajectory adherence and maneuvering precision. The third scenario combined three irregularly placed obstacles with a curved reference trajectory, requiring the UAV to balance obstacle avoidance and realignment. Obstacle positions are tracked using the VICON Motion Capture system, with simulated LiDAR scans ensuring accurate situational awareness.

Figures 5.9a–5.9c illustrate the UAV's trajectories across different scenarios, where the green line represents the reference trajectory and the blue line shows the actual UAV path. In the first scenario, the UAV followed the trajectory closely, with only minor deviations as it smoothly avoided the obstacle. In the second scenario, navigating between two obstacles introduced slightly larger deviations, reflecting the increased complexity of

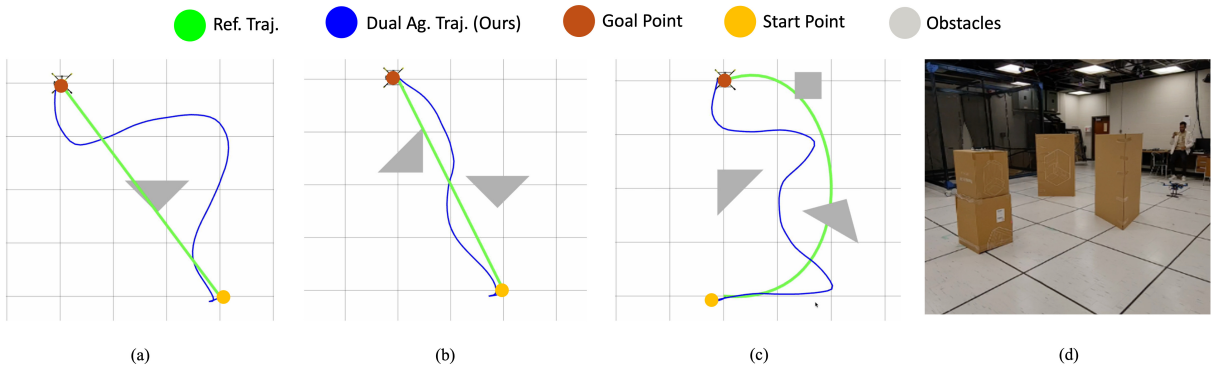


Figure 4.10: Comparison of trajectories generated in a real world experiments by the proposed algorithm. UAV tracking (a) straight line trajectory in a single obstacle scenario, (b) straight line trajectory in two obstacle environment, (c) curve trajectory in a three obstacle scenario and (d) snapshot of third person view of the real world.

maintaining trajectory accuracy while ensuring collision avoidance. In the third scenario, the UAV encountered a highly cluttered environment, requiring more significant trajectory adjustments to prioritize obstacle avoidance while still attempting to realign with the curved reference path. These results demonstrate the proposed algorithm's ability to handle varying levels of complexity, effectively balancing trajectory tracking and collision avoidance.

4.5 Summary

In this work, a RL-based method is presented for aerial trajectory tracking in cluttered environments. Our proposed dual-agent reinforcement learning algorithm has demonstrated superior performance in trajectory tracking and collision avoidance across a variety of environments, outperforming both the state-of-the-art optimization algorithms and a single-agent approach.

The results underscore the robustness and efficiency of our dual-agent RL approach in handling dynamic and static environments, ensuring reliable trajectory tracking and obstacle avoidance. Our findings contribute significantly to advancing UAV navigation systems, paving the way for more efficient and safer autonomous operations in complex environments. Despite the superior performance of our algorithm, there is an added complexity from using separate policies for trajectory tracking and collision avoidance, which may in-

crease the computational burden and complicate real-time decision-making. Additionally, the system's heavy reliance on 3D pointcloud data makes it sensitive to sensor noise and inaccuracies, which could degrade performance in real-world scenarios with environmental interference.

Chapter 5

Safe Reinforcement Learning for Trajectory Tracking of Aerial Continuum Manipulation in Unknown Constrained Environments

UAVs are increasingly being used for tasks such as construction, pick-and-place operations, and other complex tasks that require precise manipulation. To achieve this, UAVs need to be equipped with a manipulation system. In this work, we specifically chose aerial continuum manipulators (ACMs) due to their compliant nature, which makes them ideal for interacting with fragile or unpredictable objects. Thus, a novel model-free safe reinforcement learning (SRL) framework is presented for trajectory tracking of ACMs in unknown, constrained environments. The key innovation is the introduction of a health policy network, which ensures safety by penalizing unsafe states (e.g., collisions, excessive tendon tension). Unlike traditional reinforcement learning approaches that rely on manually designed constraint penalties, this method enforces safe behavior implicitly through a health-driven reward function and Kullback–Leibler (KL) divergence minimization between the task policy and the health policy. The work also employs curriculum learning to gradually increase training complexity, ensuring better generalization and convergence. The proposed framework is validated through simulations and real-world experiments, demonstrating superior trajectory tracking and safety compliance compared to state-of-the-art SRL algorithms.

The key contributions are -

- Introduces a health policy network that enforces safety implicitly without manual constraint penalties.
- Utilizes KL divergence minimization to align the task policy with safety constraints.
- Employs curriculum learning for stable training and improved generalization.
- Demonstrates superior trajectory tracking and safety compliance over existing SRL methods.
- Enables safe and adaptive obstacle avoidance in both static and dynamic environments.
- Validated through simulations and real-world experiments on ACM trajectory tracking.

5.1 Health Driven Reinforcement Learning

An off-policy, health-driven algorithm derived from a KL divergence variant of the policy iteration framework is proposed. Firstly, the details of derivation process are shown followed by demonstrating that the resulting algorithm converges to the optimal policy within its density class, and subsequently introduce a practical deep RL algorithm grounded in this theoretical foundation.

A health reward term is introduced to quantify the health of the system, capturing critical aspects such as collision avoidance for robots or entanglement prevention for continuum arms. This term allows us to define and train a health policy, which is optimized using an off-policy approach similar to DDPG, where the primary objective is to model and preserve the system's health, implicitly incorporating the constraints in the system.

Once the health policy π_h is trained, it is leveraged to constrain our baseline policy. Specifically, the baseline policy is trained not only to maximize the task reward but also to reduce the divergence between itself and the health policy. This ensures that the constraints represented by the health policy are implicitly incorporated into the baseline policy.

In the policy evaluation step of our proposed health-constrained algorithm, the aim is to compute the value of a policy π under the combined objectives of maximizing task rewards and adhering to system health constraints. For a fixed policy, the health-constrained Q-value can be iteratively computed starting from any initial function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ by repeatedly applying a modified Bellman backup operator $\mathcal{H}^{\pi}$, defined as

$$\mathcal{H}^{\pi}Q(s_t, a_t) \triangleq r(s_t, a_t) - \alpha D_{KL}\left(\pi(a_t|s_t)||\pi_h(a_t|s_t)\right), \tag{5.1}$$

where $r(s_t, a_t)$ is the task reward, α is a weighting factor for the KL divergence term, and π_h represents the health policy.

This operator $\mathcal{H}^{\pi}$ integrates health-driven constraints directly into the Bellman update, ensuring that the resulting policy adheres to both the task and health objectives.

Lemma 1 (Health-Driven Policy Evaluation) *Consider the health Bellman backup operator $\mathcal{H}^{\pi}$ defined in Equation 4. Given a mapping $Q : S \times A \to \mathbb{R}$ with a finite action space $|A|$, the sequence $Q^{k+1} = \mathcal{H}^{\pi}Q^{k}$ is defined. This sequence converges to the Q-value function of policy π as k approaches infinity.*

Proof:

The health-augmented reward function is defined as follows

$$r_h(s, a) \triangleq r(s, a) - \mathbb{E}_{s'\sim p}\left[D_{KL}\left(\pi(\cdot|s)||\pi_h(\cdot|s)\right)\right]. \tag{5.2}$$

Rewriting the update rule

$$Q(s, a) \leftarrow r_h(s, a) + \gamma \mathbb{E}_{s'\sim p, a'\sim\pi}[Q(s', a')]. \tag{5.3}$$

Then the standard convergence results for policy evaluation can be applied [78].

Following that it is demonstrated that the new, updated policy achieves a higher value compared to the old policy and the result is formalized in Appendix.

Lemma 2 (Health Policy Improvement) *Let $\pi_{old} \in \Pi$ and let π_{new} be the optimizer of the minimization problem. Then, the health Q-value satisfies $Q^{\pi_{new}}(s_t, a_t) \geq Q^{\pi_{old}}(s_t, a_t)$ for all $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$, where $|\mathcal{A}| < \infty$.*

Proof:

Let $\pi_{\text{old}} \in \Pi$, and define the state-action value $Q^{\pi_{\text{old}}}$. The new policy, π_{new}, is defined as

$$\pi_{\text{new}}(\cdot|s_t) = \arg\min_{\pi'\in\Pi}\left[Q^{\pi_{\text{old}}}(s_t, \cdot) - |D_{KL}\left(\pi(\cdot|s)||\pi_h(\cdot|s)\right)|\right], \tag{5.4}$$

which is equivalent to

$$\pi_{\text{new}}(\cdot|s_t) = \arg\min_{\pi'\in\Pi} J_{\pi_{\text{old}}}(\pi'(\cdot|s_t)). \tag{5.5}$$

Since π_{old} is a valid candidate policy, the cost function satisfies

$$J_{\pi_{\text{old}}}(\pi_{\text{new}}(\cdot|s_t)) \leq J_{\pi_{\text{old}}}(\pi_{\text{old}}(\cdot|s_t)). \tag{5.6}$$

This implies that

$$\begin{aligned} &Q^{\pi_{\text{old}}}\left(s_t|\pi_{\text{new}}(\cdot|s_t)\right) - \left|D_{KL}\left(\pi_{\text{new}}(\cdot|s)||\pi_h(\cdot|s)\right)\right| \geq Q^{\pi_{\text{old}}}\left(s_t|\pi_{\text{old}}(\cdot|s_t)\right) - \\ &\left|D_{KL}\left(\pi_{\text{old}}(\cdot|s)||\pi_h(\cdot|s)\right)\right|. \end{aligned} \tag{5.7}$$

Now, consider the health Bellman equation

$$Q^{\pi_{\text{old}}}(s_0, a_0) = r(s_0, a_0) + \gamma\left(\mathbb{E}_{s_1\sim p, a_1\sim\pi_{\text{old}}}\left[-\left|D_{KL}\left(\pi_{\text{old}}(\cdot|s)||\pi_h(\cdot|s)\right)\right| + Q(s_1, a_1)\right]\right). \tag{5.8}$$

Using the bound from the inequality above, $Q^{\pi_{\text{old}}}(s_t, a_t)$ can be expanded recursively

$$Q^{\pi_{\text{old}}}(s_0, a_0) \leq r(s_0, a_0) + \gamma\left(\mathbb{E}_{s_1\sim p, a_1\sim\pi_{\text{new}}}\left[-\left|D_{KL}\left(\pi_{\text{new}}(\cdot|s)||\pi_h(\cdot|s)\right)\right| + Q(s_1, a_1)\right]\right). \tag{5.9}$$

By repeatedly applying the health Bellman equation and the above inequality

$$Q^{\pi_{\text{old}}}(s_t, a_t) \leq Q^{\pi_{\text{new}}}(s_t, a_t). \tag{5.10}$$

Finally, $Q^{\pi_{\text{new}}}$ converges.

Having established that the new policy π_{new} achieves a higher Q-value than the old policy π_{old}, the deviation in value between the optimal policy π^* and an arbitrary policy π in terms of KL divergence is analyzed. Specifically, the aim is to bound the Q-value deviation using the KL divergence constraint.

Lemma 3 (Bounding the Policy Deviation) *Let Q^* be the optimal Q-value function corresponding to the optimal policy π^*, and let Q be the Q-value function of any other policy π. If the KL divergence between the optimal and arbitrary policy satisfies*

$$D_{KL}(\pi^*(a|s)\|\pi(a|s)) = \Delta,$$

then the deviation in the Q-value function is bounded as

$$|Q - Q^*|_\infty \leq \frac{\gamma\Delta}{1-\gamma}. \tag{5.11}$$

Proof:

The deviation in the value function can be expressed as

$$|Q - Q^*|_\infty = \left| \sum_{t=0}^{\infty} \gamma^t (R_t - R_t^*) + \sum_{t=0}^{\infty} \gamma^t D_{KL}(\pi^*(a|s) \| \pi(a|s)) \right|. \tag{5.12}$$

Using the given KL constraint

$$\begin{aligned} |Q - Q^*|_\infty &\leq \left| \sum_{t=0}^{\infty} \gamma^t (R_t - R_t^*) \right| + \left| \sum_{t=0}^{\infty} \gamma^t D_{KL}(\pi^*(a|s) \| \pi(a|s)) \right|, \\ |Q - Q^*|_\infty &\leq \left| \sum_{t=0}^{\infty} \gamma^t (R_t - R_t^*) \right| + \left| \sum_{t=0}^{\infty} \gamma^t \Delta \right|. \end{aligned} \tag{5.13}$$

Since Δ is constant over time, the summation can be simplified

$$\sum_{t=0}^{\infty} \gamma^t \Delta = \frac{\gamma \Delta}{1 - \gamma}. \tag{5.14}$$

Hence, the desired bound is obtained

$$|Q - Q^*|_\infty \leq \frac{\gamma \Delta}{1 - \gamma}. \tag{5.15}$$

In this framework, π_ϕ,$\pi_{\phi'}$ represent the normal policy and target policy network, Q_θ,$Q_{\theta'}$ represent the critic and target critic network. π_h is the health policy and Q_h is the health critic network, trained using the health reward value R_h.

The health policy, π_h, is trained to maximize a health reward signal, R_h, which encapsulates the desired health in the system. Using the Deep Deterministic Policy Gradient (DDPG) algorithm [48], π_h is optimized to map states to actions in continuous action spaces effectively. DDPG combines actor-critic architecture with deterministic policy updates, making it particularly suitable for tasks with high-dimensional and continuous state-action domains.

The health critic network, $Q_h(s, a)$, approximates the expected cumulative reward when executing action a in state s under policy π_h. This network is trained using the temporal difference (TD) loss function

$$\mathcal{L}_{critic} = \mathbb{E}_{s,a,r,s'}\left[\left(Q_h(s,a) - (r_h + \gamma Q_h^{'}(s', \pi_h(s')))\right)^2\right], \tag{5.16}$$

where r_h is the health reward obtained for transitioning to the next state s', γ is the discount factor, and $Q_h^{'}$ is the target critic network, which helps stabilize training by providing a slowly updated target for the TD learning process.

The health actor network, π_h, is trained to maximize the critic's evaluation by updating the policy parameters to improve the expected return as estimated by Q_h. The loss function for the actor is defined as

$$\mathcal{L}_{actor} = -\mathbb{E}_s\left[Q_h(s, \pi_h(s))\right]. \tag{5.17}$$

The gradient of this loss is used to adjust the parameters of π_h via backpropagation. To improve stability and convergence, DDPG employs techniques such as soft updates for the target networks and exploration noise (e.g., Ornstein-Uhlenbeck process) added to the policy during training. Soft updates ensure that the target networks Q_h' and π_h' are updated gradually with a mixing parameter τ, avoiding abrupt shifts in the learning process

$$\begin{aligned} \theta_h' &\leftarrow \tau\theta_h + (1-\tau)\theta_h', \\ \phi_h' &\leftarrow \tau\phi_h + (1-\tau)\phi_h', \end{aligned} \tag{5.18}$$

where θ_h and ϕ_h represent the parameters of the critic and actor networks, respectively.

Through this training process, π_h learns to prioritize actions that maintain system health or adhere to predefined safety standards while balancing the exploration-exploitation trade-off inherent to RL.

Once trained, the health actor network is used to learn the standard policy. The standard policy π_ϕ is trained to achieve a balance between maximizing the task reward and ensuring the system's health by integrating the health policy π_h. This approach allows the standard policy to inherit the benefits of the health policy while maintaining its primary focus on task performance, thereby satisfying both the task objectives and the safety or health constraints.

The critic network Q_θ is trained using an updated Temporal Difference (TD) error, incorporating a penalty term based on the health Bellman equation to enforce alignment with the health policy π_h. The loss function is defined as

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[(Q_\theta(s,a) - y)^2\right], \tag{5.19}$$

where the target value y is given by

$$y = r + \gamma Q_{\theta'}(s', \pi_{\phi'}(\cdot|s')) - D_{KL}\left(\pi_{\text{new}}(\cdot|s) \| \pi_h(\cdot|s)\right). \tag{5.20}$$

In this formulation, r is the task reward obtained for transitioning to the next state s', and γ is the discount factor that weights the future expected return. The term $Q_{\theta'}(s', \pi_{\phi'}(\cdot|s'))$ is the target critic network's estimate of the value of the next state-action pair, providing a stable target for the TD update. The KL divergence term, $D_{KL}\left(\pi_{\text{new}}(\cdot|s) \| \pi_h(\cdot|s)\right)$, penalizes deviations of the learned policy π_{new} from the health policy π_h. This divergence regularization encourages the policy to adhere to the constraints and safety considerations encoded in the health policy while optimizing task performance. By integrating this penalty, the critic learns to evaluate actions not only in terms of task success but also with respect to their compliance with health or safety requirements, thus achieving a robust balance between maximizing cumulative task rewards and satisfying system constraints.

The policy network π_ϕ is optimized to achieve a balance between maximizing the task reward and ensuring adherence to health or safety constraints encoded in the health critic Q_h. The policy loss function incorporates both the expected task value and a penalty term that emphasizes alignment with the health policy. Specifically, the policy is trained to minimize the following loss

$$L(\phi) = \mathbb{E}_{s\sim D}\left[Q_\theta(s, \pi_\phi(s)) + \left(Q_\theta(s, \pi_\phi(s)) - Q_h(s, \pi_\phi(s))\right)^2\right]. \tag{5.21}$$

In this formulation, the first term, $Q_\theta(s, \pi_\phi(s))$, represents the expected cumulative reward as evaluated by the standard critic Q_θ, driving the policy to maximize task performance. The second term, $\left(Q_\theta(s, \pi_\phi(s)) - Q_h(s, \pi_\phi(s))\right)^2$, penalizes discrepancies between the task value and the health value, encouraging the policy to favor actions that align with health or safety objectives. By squaring this difference, the penalty amplifies deviations, effectively guiding the policy toward maintaining system health while pursuing task goals. This loss function ensures that the learned policy simultaneously maximizes task rewards and respects the constraints imposed by the health policy, resulting in a robust and balanced policy suitable for complex, safety-critical environments.

To stabilize training, the target networks $Q_{\theta'}$ and $\pi_{\phi'}$ are updated using soft updates

$$\begin{aligned} \theta' &\leftarrow \tau\theta + (1-\tau)\theta', \\ \phi' &\leftarrow \tau\phi + (1-\tau)\phi', \end{aligned} \tag{5.22}$$



Figure 5.1: Aerial continuum manipulator.

By leveraging the health policy π_h during the training process, the standard policy π_ϕ achieves the best of both worlds—maximizing the task reward while ensuring the health and safety of the system. This integration enables the standard policy to satisfy the health constraints, fostering robust performance in dynamic and safety-critical environments.

5.2 Simulation Setup and Results

Our simulation pipeline is built on robot operating system (ROS) [75]. The simulation environment employs a Holybro X500 UAV equipped with a continuum arm comprising four disks as shown in Figure 5.1. The dimensions and physical properties of the ACM are detailed in Table 5.1. To simulate sensor feedback, an artificial point cloud is created with respect to the world frame. At each time step, the algorithm receives point cloud information constrained within a 1-meter radial range from the ACM. The point cloud is computed from the ACM's base frame within this range, ensuring a localized representation of the environment. Additionally, the vertical range of the perceived point cloud is limited to angles between $[-30°, 30°]$.

The normal policy π_ϕ is trained to control the ACM end effector to follow a reference trajectory $\mathbf{q_{ref}} = (\mathbf{q}(0), \mathbf{q}(t_1), \ldots, \mathbf{q}(t_n))$, where $\mathbf{q}(t_i)$ represents the target positions. The state space $\mathbf{S}$ include the 3D LiDAR point cloud $\mathbf{l_p}(t)$, the tendon tensions $\mathbf{t_e}(t)$, the ACM's current velocity $\mathbf{v}(t)$ and positional errors between the end effectors current position $\mathbf{q_e}(t)$ and the reference points up to m steps ahead: $\mathbf{S} = \{\mathbf{e}(t), \mathbf{e}(t+1), \ldots, \mathbf{e}(t+m)\}$, with

Table 5.1: Parameters of continuum arm.

Parameter	Value
No of disk	39
Disk diameter	25 mm
Length of arm	38 cm
Backbone material	Nitinol

$\mathbf{e}(t+k) = \mathbf{q_e}(t) - \mathbf{q}(t+k)$. The policy outputs velocity changes $\mathbf{\Delta v} = [\Delta v_x, \Delta v_y, \Delta v_z]$ and change in tendon tensions $\mathbf{\Delta t_e} = [\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4]$, executed over Δt. The ACM continuously adjusts its control until reaching the goal. The reward function for training the RL algorithms is detailed below

$$\mathrm{R}(t) = \begin{cases} \alpha_4 & \text{if ACM follows trajectory} \\ o_1(t) & \text{otherwise} \end{cases} \tag{5.23}$$

$$o_1(t) = -\alpha_1 \|\mathbf{e}(t)\|_2^2 - \alpha_2 \|\mathbf{e_g}(t)\|_2^2 - \alpha_3 d(t). \tag{5.24}$$

The squared l_2 norm of the tracking error, $\|\mathbf{e}(t)\|_2^2$, represents the difference between the ACMs's end effector current position and the reference point $\mathbf{q}(t)$. Similarly, $\|\mathbf{e_g}(t)\|_2^2$ denotes the squared l_2 norm of the global error, measuring the distance between the end effector current position and the goal position $\mathbf{g_u}$. The deviation from the reference trajectory, $d(t)$, is defined as the perpendicular distance from the ACM to the trajectory segment spanning from t to $t+m$. The episode is considered successful if the UAV follows the trajectory, i.e., $\|\mathbf{e}(t)\|_2^2 \leq \epsilon_K$. The health policy is trained to maintain the ACM in a safe and stable state, preventing damage and ensuring operational integrity. Its state space encompasses the UAV's current position $\mathbf{q_a}(t)$, the end effector position of the continuum arm $\mathbf{q_e}(t)$, tendon tensions $\mathbf{t_e}(t)$, the velocity of the ACM $\mathbf{v}(t)$, and the surrounding 3D LiDAR point cloud data. Similar to the normal policy, the health policy outputs control actions comprising velocity adjustments $\mathbf{\Delta v} = [\Delta v_x, \Delta v_y, \Delta v_z]$ and changes in tendon tensions $\mathbf{\Delta t_e} = [\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4]$, executed over a time step Δt. The reward function encourages safe operation by penalizing unsafe states and rewarding stable behavior, defined as

$$\mathrm{R_h}(t) = \begin{cases} o_2(t) & \text{if ACM is safe} \\ \alpha_5 & \text{otherwise} \end{cases} \tag{5.25}$$

$$o_2(t) = \alpha_6 l_m(t) + \alpha_7 \|\mathbf{d_{eu}}(t)\|_2^2 - \alpha_3 d_c(t). \tag{5.26}$$

The goal is to eliminate two potential unfavorable behaviors: (1) collision between the arm and the UAV and collision of the ACM with obstacles, and (2) the entanglement of the continuum arm. If any of these conditions are violated, the health policy receives a large negative reward to penalize such behaviors. On the other hand, when the ACM is unlikely to enter an undesirable configuration, the policy rewards positive values, reflecting the system's stability and safety. The reward function, therefore, incorporates these considerations to promote the avoidance of hazardous states and encourage safer, more reliable operation of the ACM. Here, $l_m(t)$ is the minimum range of the ACM from the obstacles at time t, $d_{eu}(t)$ is the height difference between the UAV and end effector. $d_c(t)$ quantifies the entanglement of the continuum arm. It is designed to penalize self-intersections and excessive deformation while rewarding configurations that maintain clearance between segments. The formulation integrates multiple factors, including self-intersection, knot complexity and spatial clearance. The continuum arm is discretized to a set of N points, $r_i = (x_i, y_i, z_i), i = 1, .., N$. Self-intersection is penalized by detecting pairs of points along the backbone whose distances fall below a predefined threshold, contributing a negative term to the reward

$$E_{\text{self}} = \sum_{i \neq j} \sigma(\epsilon - \|r_i - r_j\|), \tag{5.27}$$

where $\sigma(x)$ is a step function that returns 1 if $x > 0$ (indicating intersection) and 0 otherwise, and ϵ is the collision threshold. Additionally, knot complexity is quantified using the Gauss linking number, ensuring that topological loops are discouraged

$$\begin{aligned} E_{\text{knot}} &= |Lk|, \\ Lk &= \left| \frac{1}{4\pi} \sum_{i \neq j} \frac{(r_{i+1} - r_i) \times (r_{j+1} - r_j) \cdot (r_i - r_j)}{\|r_i - r_j\|^3} \right|, \end{aligned} \tag{5.28}$$

where Lk represents the Gauss linking number. Conversely, to encourage configurations that maintain spatial separation, a clearance term is introduced, summing the inverse distance between all pairs of backbone points with an exponent p to emphasize short-range interactions

$$E_{\text{clearance}} = \sum_{i \neq j} \frac{1}{\|r_i - r_j\|^p}, \quad p > 2. \tag{5.29}$$

The final entanglement function is expressed as a weighted sum of these terms

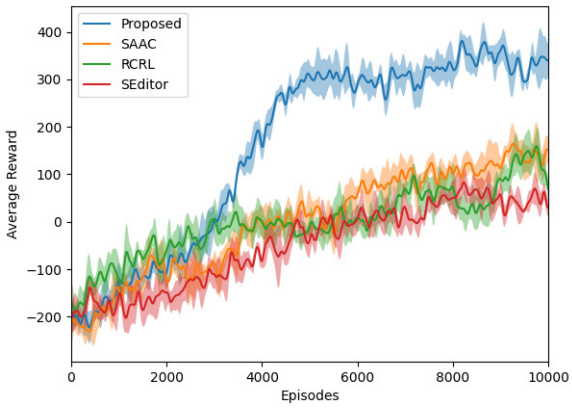


Figure 5.2: Comparison of rewards achieved by proposed algorithm, RCRL, SAAC, and SEditor algorithms after sampling 10K episodes.

$$d_c(t) = -w_1 E_{\text{self}} - w_2 E_{\text{knot}} + w_3 E_{\text{clearance}}. \tag{5.30}$$

For entanglement, self-intersections occur, meaning $E_{\text{self}} > 0$ and $E_{\text{knot}} > 0$, which results in a negative contribution to $d(t)$. Conversely, when the arm maintains a non-entangled state, $E_{\text{self}} = 0$ and $E_{\text{knot}} = 0$, and the clearance term $E_{\text{clearance}}$ dominates, leading to a higher reward. Hence, $d(t)$ is minimized in an entangled state and maximized when the arm is fully untangled. Curriculum learning is employed to train both policies effectively, starting with simpler tasks and progressing to complex ones [76]. Direct deployment in complex tasks resulted in sub-optimal behavior, with the ACM struggling in long-horizon planning in challenging environments. An ablation study is also performed to analyze the effect of network architecture.

Based on the ablation study conducted with curriculum learning, CL4 yielded the best performance with a network consisting of 128 neurons, 3 layers and trained with a learning rate of 0.0001. Consequently, the best configuration is trained for 10,000 iterations. The results, shown in the Figure 5.2, demonstrate that the proposed algorithm outperformed other SRL algorithms, achieving 31% higher rewards than RCRL, 3 and 4 times higher rewards than SAAC and SEditor respectively . To further test the performance of our proposed algorithm, it is deployed in different complex scenarios, including (1) random obstacles, (2) two ACMs holding an object together, and (3) dynamic obstacles.

5.2.1 Random obstacle scenario

In this experiment, the ACM is tasked with navigating predefined trajectories in a random static obstacle environment. In the first two scenarios, it followed cubic spline trajectories while successfully avoiding obstacles. The third scenario involved a more complex trajectory, forming two triangles with a shared vertex. In the final scenario, the ACM navigated a square trajectory while avoiding a highly cluttered environment containing 22 obstacles of varying shapes and heights. Figure 5.4 illustrates the qualitative results of the random environment benchmark with 22 obstacles.

Across all scenarios, the ACM initially follows the planned path. As obstacles are encountered, the control strategy dynamically adapts. For small obstacles, the algorithm modifies the ACM's end-effector position without repositioning the UAV, thereby conserving energy. For larger obstacles, the UAV is repositioned while maintaining constant tendon tensions in the ACM. This strategy ensures an optimal balance between ACM adjustments and UAV repositioning for efficient navigation. The trajectories vary in shape, with some following a more constrained path while others adopt a wider and smoother cubic spline, as illustrated in Figures 5.3a and 5.3b.

For scenarios following cubic spline trajectories, the ACM exhibited maximum deviations of 2.24 m and 3.61 m, with total trajectory lengths of 38.972 m and 55.04 m, respectively. In the third scenario, where the trajectory forms two triangles with a shared vertex, the total length is 83.53 m with a maximum deviation of 1.20 m. The final scenario, featuring a square trajectory, had a maximum deviation of 0.78 m, with a reference trajectory exceeding the total length by 1.1 times.

5.2.2 Two Aerial Continuum Manipulators together

In this simulation, two ACMs cooperate to transport an object through an environment with five stationary obstacles, demonstrating their ability to navigate complex trajectories while maintaining stability and coordination. The simulation consists of three distinct scenarios designed to evaluate their performance in trajectory tracking, obstacle avoidance, and cooperative manipulation. In the first scenario, the ACMs follow a straight-line trajectory from the starting point to the goal. This scenario serves as a baseline, assessing their ability to maintain formation and stability while counteracting external disturbances such as wind or minor control delays. The ACMs achieved a maximum deviation of 1.14 m from the reference trajectory, with a total trajectory length of 15.71 m. The second scenario introduces a Z-shaped trajectory, which requires the ACMs to execute sharp turns and

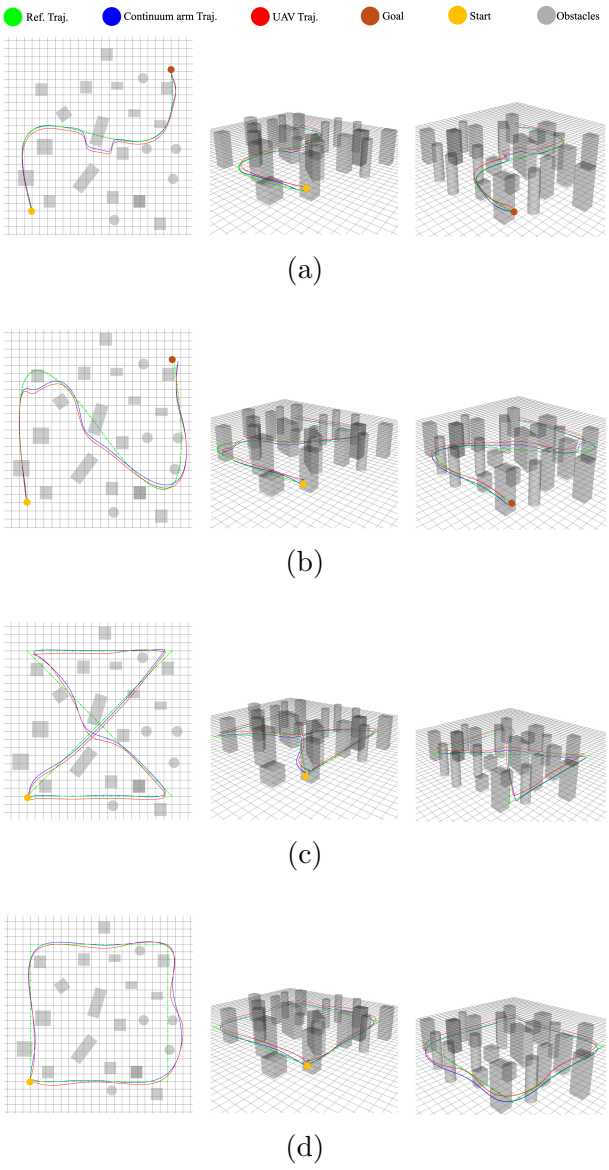


Figure 5.3: Four different navigation scenarios where an ACM navigates through an obstacle-filled environment. The reference trajectory is shown in green, while the UAV and continuum arm trajectories are represented in red and blue, respectively. (a) and (b) depict spline-based trajectories, demonstrating smooth navigation through obstacles. (c) involves a complex trajectory requiring intricate maneuvers, while (d) follows a square-shaped path, testing the ability to maintain structured navigation. The start and goal positions are marked in yellow and brown, respectively, with gray objects representing obstacles in the environment.

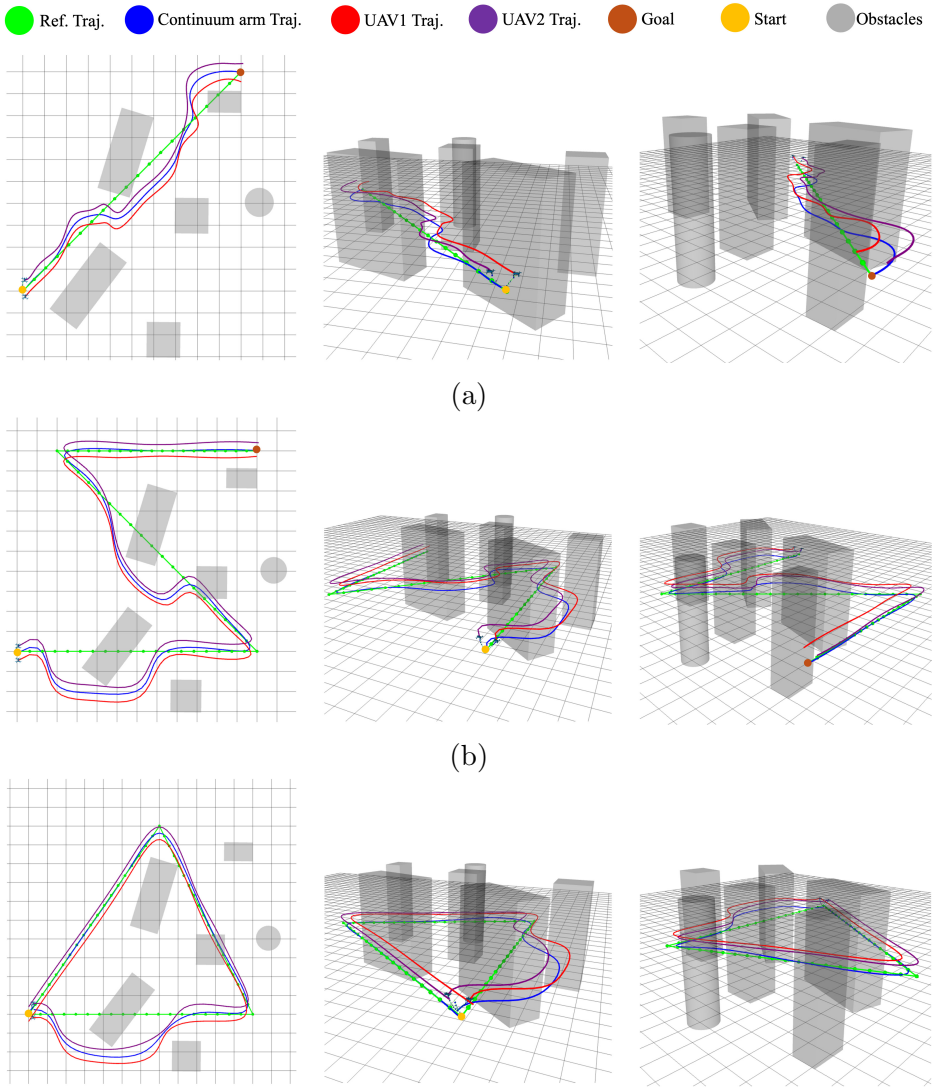


Figure 5.4: Three navigation scenarios where two ACMs, collaboratively transport an object through an obstacle-filled environment. The reference trajectory is shown in green, while the trajectories of the two UAVs are represented in red and purple. The continuum arm trajectories are depicted in blue, demonstrating their motion as they adapt to the object's movement. Each scenario presents different trajectory complexities, including smooth paths, sharp turns, and structured geometric patterns.

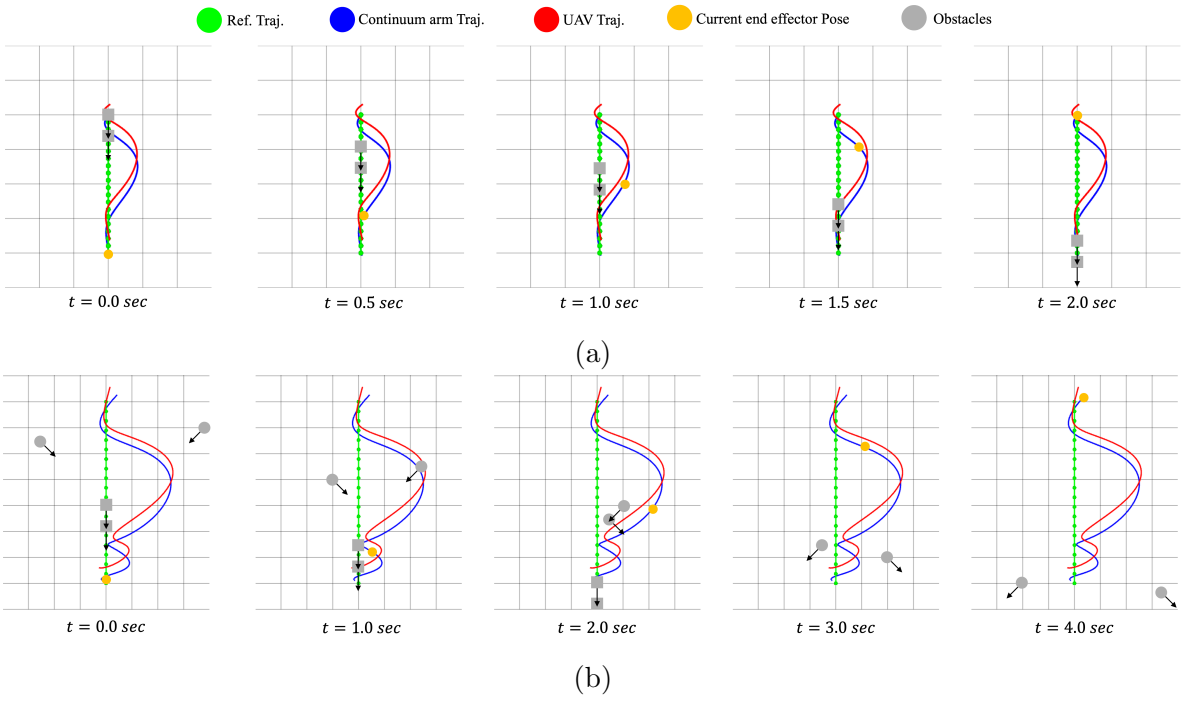


Figure 5.5: Two distinct dynamic obstacle scenarios that the ACM must navigate. The reference trajectory is indicated in green, while the UAV and continuum arm trajectories are shown in red and blue, respectively. In both cases, the ACM is tasked with following a straight-line trajectory. Scenario (a) features two cuboid obstacles moving at a constant velocity, whereas scenario (b) introduces two additional cylindrical obstacles moving diagonally.

adjust their relative positions to navigate corners efficiently. This scenario highlights their responsiveness, path-following precision, and ability to coordinate dynamic movements. The Z-shaped trajectory resulted in a total path length of 39.14 m , with a maximum deviation of 2.81 m occurring at the sharpest corner. The third scenario involves a triangular trajectory, where the ACMs must perform multiple turns and maintain consistent object orientation while navigating the triangular path. For the triangular trajectory, the total path length is 35.84 m, with a maximum deviation of 4.80 m observed at the apex of the triangle. This scenario further tests their cooperative control strategies and ability to handle complex maneuvers in confined spaces.

5.2.3 Dynamic Obstacles

In real-world robotics applications, particularly in aerial manipulation and autonomous navigation, environments are rarely static. Dynamic obstacles, such as moving objects, pedestrians, or other robots, pose significant challenges that demand real-time decision-making, precise trajectory adjustments, and adaptive control strategies. Successfully navigating such environments is crucial for ACMs, which combine the agility of UAVs with the dexterity of continuum arms to perform complex tasks like inspection, rescue operations, and object transportation. Unlike static obstacle avoidance, where a path can be precomputed, dynamic scenarios require continuous perception, prediction, and correction to ensure smooth and safe navigation. Testing ACMs in such environments evaluates their ability to react to unexpected changes, maintain trajectory stability, and execute precise maneuvers while avoiding collisions.

Figure 5.5 illustrates a dynamic obstacle navigation scenario where an ACM must traverse an environment with moving obstacles while following a predefined reference trajectory. The ACM encounters multiple dynamic obstacles, represented by gray objects, which move at a speed of 0.1 m/s. As the obstacles move unpredictably, the ACM must continuously adjust its trajectory, ensuring that both the UAV and the continuum arm maintain a smooth and coordinated path. The images in the figure provide a time-sequenced visualization of the ACM's navigation, highlighting critical moments where it deviates from the reference trajectory to avoid obstacles before realigning itself.

Figure 5.5 provide a time-sequenced visualization of the ACM's response to dynamic obstacles. In the first scenario as shown in Figure 5.5a, the UAV and continuum arm begin by closely following the reference path but are required to make slight deviations as they encounter obstacles along the way. The maximum deviation recorded for this scenario is 0.86 m, occurring when the UAV adjusts its trajectory to avoid an obstacle while ensuring that the continuum arm maintains a stable grip. The second scenario in Figure 5.5b presents a more complex navigation challenge, where the ACM must navigate through a winding trajectory while reacting to moving obstacles. Here, the UAV and continuum arm demonstrate a more pronounced deviation from the reference trajectory, with a maximum recorded deviation of 2.4 m, occurring at a sharp turn where an obstacle moves directly into the UAV's path. Despite these deviations, the ACM successfully reaches the goal position while maintaining a coordinated and smooth trajectory.

One of the key aspects of this experiment is the ACM's ability to navigate dynamic obstacles without prior training in a similar scenario. Instead of relying on pre-learned obstacle motion patterns, the algorithm learns to predict obstacle behavior using only point cloud data, without any explicit information on obstacle velocity or heading. This

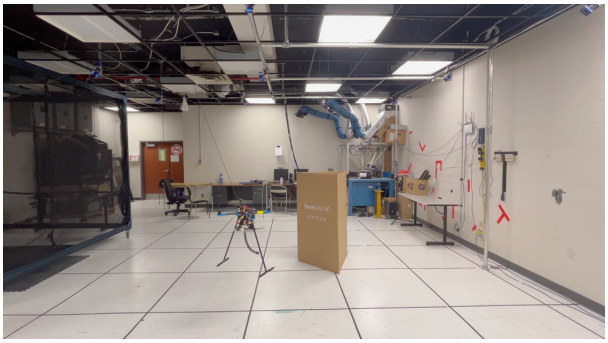
is achieved through an adaptive motion-planning strategy that extracts temporal features from real-time LiDAR point cloud updates. By analyzing changes in the detected obstacle's spatial distribution over successive frames, the algorithm infers potential movement trends and adjusts the ACM's trajectory accordingly. This predictive approach allows the UAV and continuum arm to dynamically replan their motion, ensuring safe navigation even in unpredictable environments.

This experiment highlights the ACM's ability to dynamically adapt to unseen dynamic obstacles, showcasing its obstacle avoidance strategies, real-time trajectory adjustments, and overall robustness. The results demonstrate that while minor deviations from the reference trajectory are inevitable in dynamic scenarios, the ACM is capable of effectively navigating complex environments while minimizing instability and ensuring safe passage. Such capabilities are critical for deploying ACMs in real-world applications, including autonomous inspection, search and rescue missions, and transportation in cluttered, dynamic spaces.

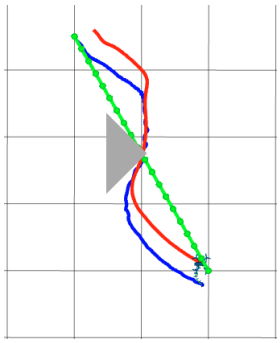
5.3 Real World Experiments

After the simulations, the algorithm is deployed in a real-world scenario where the ACM must avoid an obstacle while following a straight-line trajectory, as illustrated in Figure 5.6a. The algorithm runs on a Jetson Nano, equipped with a Quad-core ARM Cortex-A57 MPCore processor and 4 GB of 64-bit LPDDR4 memory, onboard the ACM. Obstacle positions are tracked using the VICON Motion Capture system, with simulated LiDAR scans ensuring precise situational awareness.

Figure 5.6b depicts the ACM's trajectory in the obstacle avoidance scenario. The green line represents the reference trajectory, the blue line shows the actual path taken by the end effector, and the red line represents the UAV's path. Initially, the ACM follows the designated trajectory. Midway, upon encountering the obstacle, the UAV deviates while the continuum arm adjusts its shape accordingly. Once the obstacle is successfully avoided, the UAV rejoins the trajectory, with the continuum arm accurately tracking the path. ACM had a max deviation of 0.5 m and total trajectory length 1.45 times longer than the reference trajectory.



(a)



(b)

Figure 5.6: Real-world experiment setup for ACM obstacle avoidance. (a) The UAV-mounted continuum manipulator navigates around an obstacle while tracking a predefined trajectory. The VICON Motion Capture system ensures precise localization, while simulated LiDAR scans enhance situational awareness, (b) RViz visualization of trajectory taken by the ACM.

5.4 Summary

In this chapter, a novel model-free SRL framework has been introduced for trajectory tracking of ACMs in unknown, constrained environments. Our approach integrates a health-driven policy that incorporates system constraints through a dedicated health reward signal, ensuring safety without the need for hand-crafted penalties. By employing a KL divergence constraint between the standard and health policies, our method achieves a balance between task performance and safety.

Through comprehensive simulations and real-world experiments, it is demonstrated that our framework surpasses state-of-the-art SRL methods, delivering superior trajectory tracking accuracy, reduced collision rates, and enhanced safety in dynamic, cluttered environments. Additionally, our ablation studies and curriculum learning experiments underscored the framework's robustness and adaptability to varying task complexities. This research advances the field of aerial manipulation by providing a scalable, effective solution for safe ACM trajectory tracking. In future work, the aim is to integrate vision-based perception for enhanced obstacle avoidance and extend our framework to multi-ACM collaborative tasks, tackling complex real-world scenarios and further broadening its applicability.

Chapter 6

Conclusion

The thesis addressed the challenge of RL-based trajectory tracking for unmanned aerial systems including UAVs and ACMs in cluttered and dynamic environments. Traditional trajectory tracking methods, including optimization-based approaches and model predictive control, suffer from computational complexity, reliance on explicit environmental models, and limited adaptability to real-world uncertainties. To overcome these limitations, this thesis proposed novel deep RL frameworks that integrate adaptive control, safety-aware policies, and benchmarking methodologies to enhance the robustness and efficiency of UAV trajectory tracking.

The first contribution of this thesis focused on the benchmarking of off-policy RL algorithms for UAV path planning, systematically evaluating algorithms such as DDPG, TD3, SAC, and SoftQ across multiple simulated and real-world environments. This study provided valuable insights into algorithmic trade-offs, particularly demonstrating DDPG's superiority in reward maximization and TD3's effectiveness in collision avoidance. The benchmarking framework established a robust evaluation pipeline, addressing the sim2real gap and paving the way for future improvements in UAV autonomy.

The second contribution focused on development of dual-agent RL framework for UAV trajectory tracking in unknown environments. By leveraging 3D point cloud perception instead of memory-intensive occupancy maps, this framework introduced two specialized RL agents: one dedicated to tracking the reference trajectory and another for dynamic collision avoidance. The architecture ensured real-time adaptability, outperforming conventional RL and optimization-based methods in both simulated and real-world experiments. Additionally, the integration of curriculum learning allowed for scalable training in increasingly complex environments, ensuring generalization to real-world UAV deployments.

The final contribution introduced a model-free SRL framework for trajectory tracking of ACMs in constrained environments. Unlike conventional RL methods that rely on manually crafted penalty functions for safety constraints, this work introduced a health policy that implicitly encoded safety requirements, including collision avoidance and tension limits. By minimizing the divergence between the primary tracking policy and the health policy, this framework ensured safe operation without the need for explicit constraint definitions, significantly improving safety and stability compared to existing SRL approaches.

6.1 Limitations

While this work has demonstrated promising results, several limitations remain that warrant further investigation. The proposed reinforcement learning frameworks, although effective in controlled simulations and selected real-world scenarios, face challenges in adapting to highly unstructured or adversarial environments. Training these deep RL policies requires substantial data and computational resources, which can limit real-time applicability, and the simulation-to-reality transfer still encounters issues due to discrepancies in sensor fidelity, actuation delays, and unexpected environmental disturbances. Moreover, while the introduction of a health policy has enhanced safety during trajectory tracking, it does not offer formal guarantees of stability or strict constraint enforcement. Additionally, the current framework primarily focuses on single-agent scenarios, leaving multi-agent coordination and swarm behavior largely unaddressed, which is critical for many practical UAV applications.

6.2 Future Work

Future research should explore hybrid learning architectures that combine model-based control and RL to improve sample efficiency and real-time adaptability. Integrating MPC with RL could ensure trajectory corrections while maintaining optimality and stability. To address the reality gap, domain adaptation and few-shot learning methods could be leveraged to enhance the transferability of learned policies across different real-world scenarios. Additionally, extending the framework to multi-agent systems could enable collaborative UAS navigation and trajectory tracking, particularly in swarm robotics applications. The incorporation of vision-based navigation, particularly with deep learning models such as transformers, could further enhance UAS perception in GPS-denied or visually complex

environments. Lastly, formal verification methods, such as Lyapunov-based stability analysis and constrained policy optimization, could be integrated into reinforcement learning training to provide stronger theoretical guarantees for safety and robustness. These advancements would expand the applicability of the proposed methodologies to more challenging and diverse real-world UAS tasks, including urban air mobility, search-and-rescue missions, and autonomous inspection in complex environments.

Further exploration into mathematical robustness and performance properties of these algorithms could be a valuable focus in future works. Specifically, addressing the reliability and stability of the proposed frameworks, ensuring they perform effectively under a wider range of environmental conditions and disturbances, would strengthen their real-world applicability. Lastly, Control Barrier Functions (CBFs) could be integrated with the RL algorithms developed in this thesis. CBFs can be used to enforce safety constraints rigorously, adding an additional layer of robustness by ensuring the system remains within safe operational limits. Investigating the integration of CBFs into the dual-agent RL framework could significantly improve the safety and stability of trajectory tracking, especially in environments with complex constraints. Formal verification methods, such as Lyapunov-based stability analysis and constrained policy optimization, could be integrated into reinforcement learning training to provide stronger theoretical guarantees for safety and robustness. These advancements would expand the applicability of the proposed methodologies to more challenging and diverse real-world UAS tasks, including urban air mobility, search-and-rescue missions, and autonomous inspection in complex environments.

References

[1] Quentin Galvane, Julien Fleureau, Francois-Louis Tariolle, and Philippe Guillotel. Automated cinematography with unmanned aerial vehicles. *arXiv preprint arXiv:1712.04353*, 2017.

[2] Guoqing Zhou, Vince Ambrosia, Albin J Gasiewski, and Geoff Bland. Foreword to the special issue on unmanned airborne vehicle (uav) sensing systems for earth observations. *IEEE Transactions on Geoscience and Remote Sensing*, 47(3):687–689, 2009.

[3] Julie Linchant, Jonathan Lisein, Jean Semeki, Philippe Lejeune, and Cédric Vermeulen. Are unmanned aircraft systems (uas s) the future of wildlife monitoring? a review of accomplishments and challenges. *Mammal Review*, 45(4):239–252, 2015.

[4] Panagiotis Radoglou-Grammatikis, Panagiotis Sarigiannidis, Thomas Lagkas, and Ioannis Moscholios. A compilation of uav applications for precision agriculture. *Computer Networks*, 172:107148, 2020.

[5] Miguel Angel Trujillo, José Ramiro Martínez-de Dios, Carlos Martín, Antidio Viguria, and Aníbal Ollero. Novel aerial manipulator for accurate and robust industrial ndt contact inspection: A new tool for the oil and gas inspection industry. *Sensors*, 19(6):1305, 2019.

[6] Anibal Ollero, Guillermo Heredia, Antonio Franchi, Gianluca Antonelli, Konstantin Kondak, Alberto Sanfeliu, Antidio Viguria, J Ramiro Martinez-de Dios, Francesco Pierri, Juan Cortés, et al. The Aeroarms project: Aerial robots with advanced manipulation capabilities for inspection and maintenance. *IEEE Robotics & Automation Magazine*, 25(4):12–23, 2018.

[7] Anibal Ollero, Marco Tognon, Alejandro Suarez, Dongjun Lee, and Antonio Franchi. Past, present, and future of aerial robotic manipulators. *IEEE Transactions on Robotics*, 38(1):626–645, 2021.

[8] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[9] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[10] Manfred Morari and Jay H Lee. Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23(4-5):667–682, 1999.

[11] Ian H Witten. An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34(4):286–295, 1977.

[12] Jeroen Van Baar, Alan Sullivan, Radu Cordorel, Devesh Jha, Diego Romeres, and Daniel Nikovski. Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics. In *IEEE International Conference on Robotics and Automation*, pages 6001–6007, 2019.

[13] Stuart Bennett. *A History of Control Engineering 1930-1955*. Peter Peregrinus, GBR, 1st edition, 1993.

[14] Norman Lehtomaki, NJAM Sandell, and Michael Athans. Robustness results in linear-quadratic gaussian based multivariable control designs. *IEEE Transactions on Automatic Control*, 26(1):75–93, 1981.

[15] Edsger W Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pages 287–290. 2022.

[16] Rüdiger Ebendt and Rolf Drechsler. Weighted A* search–unifying view and application. *Artificial Intelligence*, 173(14):1310–1342, 2009.

[17] Sven Koenig and Maxim Likhachev. D* lite. In *18th National Conference on Artificial Intelligence*, pages 476–483, 2002.

[18] Dave Ferguson and Anthony Stentz. Field D*: An interpolation-based path planner and replanner. In *Robotics Research: Results of the 12th International Symposium ISRR*, pages 239–253. Springer, 2007.

[19] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems*, 16, 2003.

[20] Robert C Holte, Maria B Perez, Robert M Zimmer, and Alan J MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*, pages 530–535, 1996.

[21] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[22] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

[23] Ahmed Hussain Qureshi and Yasar Ayaz. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robotics and Autonomous Systems*, 68:1–11, 2015.

[24] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation. Symposia Proceedings*, pages 995–1001, 2000.

[25] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, 2014.

[26] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *IEEE International Conference on Robotics and Automation*, pages 3067–3074, 2015.

[27] Jauwairia Nasir, Fahad Islam, Usman Malik, Yasar Ayaz, Osman Hasan, Mushtaq Khan, and Mannan Saeed Muhammad. RRT*-smart: A rapid convergence implementation of rrt. *International Journal of Advanced Robotic Systems*, 10(7):299, 2013.

[28] Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *2015 IEEE International Conference on Robotics and automation*, pages 2951–2957, 2015.

[29] Ahmed H Qureshi, Anthony Simeonov, Mayur J Bency, and Michael C Yip. Motion planning networks. In *IEEE International Conference on Robotics and Automation*, pages 2118–2124, 2019.

[30] Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

[31] Paul T Boggs and Jon W Tolle. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1995.

[32] Florian A Potra and Stephen J Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1-2):281–302, 2000.

[33] Jacob Mattingley and Stephen Boyd. Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*, 13:1–27, 2012.

[34] Joel AE Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11:1–36, 2019.

[35] Alexander Domahidi, Eric Chu, and Stephen Boyd. Ecos: An SOCP solver for embedded systems. In *European Control conference*, pages 3071–3076, 2013.

[36] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 4(4):3529–3536, 2019.

[37] Geesara Kulathunga, Hany Hamed, Dmitry Devitt, and Alexandr Klimchik. Optimization-based trajectory tracking approach for multi-rotor aerial vehicles in unknown environments. *IEEE Robotics and Automation Letters*, 7(2):4598–4605, 2022.

[38] Ping Zhong, Bolei Chen, Siyi Lu, Xiaoxi Meng, and Yixiong Liang. Information-driven fast marching autonomous exploration with aerial robots. *IEEE Robotics and Automation Letters*, 7(2):810–817, 2021.

[39] Xin Zhou, Zhepei Wang, Hongkai Ye, Chao Xu, and Fei Gao. Ego-planner: An esdf-free gradient-based local planner for quadrotors. *IEEE Robotics and Automation Letters*, 6(2):478–485, 2020.

[40] Mouhyemen Khan, Munzir Zafar, and Abhijit Chatterjee. Barrier functions in cascaded controller: Safe quadrotor control. In *American Control Conference)*, pages 1737–1742, 2020.

[41] Philipp Foehn, Dario Brescianini, Elia Kaufmann, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. Alphapilot: Autonomous drone racing. *Autonomous Robots*, 46(1):307–320, 2022.

[42] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

[43] Lihui Wang and Yilin Guo. Speed adaptive robot trajectory generation based on derivative property of b-spline curve. *IEEE Robotics and Automation Letters*, 8(4):1905–1911, 2023.

[44] I Kagan Erunsal, Jianhao Zheng, Rodrigo Ventura, and Alcherio Martinoli. Linear and nonlinear model predictive control strategies for trajectory tracking micro aerial vehicles: A comparative study. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 12106–12113, 2022.

[45] Jad Wehbeh, Shatil Rahman, and Inna Sharf. Distributed model predictive control for uavs collaborative payload transport. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 11666–11672, 2020.

[46] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[47] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[48] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[49] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897. PMLR, 2015.

[50] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[51] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[52] Hu Zijian, Gao Xiaoguang, Wan Kaifang, Zhai Yiwei, and Wang Qianglong. Relevant experience learning: A deep reinforcement learning method for uav autonomous motion planning in complex unknown environments. *Chinese Journal of Aeronautics*, 34(12):187–204, 2021.

[53] Zhu Chen, Xiao Liang, and Minghui Zheng. Knowledge transfer between different uavs for trajectory tracking. *IEEE Robotics and Automation Letters*, 5(3):4939–4946, 2020.

[54] Alessandro Saviolo, Guanrui Li, and Giuseppe Loianno. Physics-inspired temporal learning of quadrotor dynamics for accurate model predictive trajectory tracking. *IEEE Robotics and Automation Letters*, 7(4):10256–10263, 2022.

[55] Karim Amer, Mohamed Samy, Mahmoud Shaker, and Mohamed ElHelw. Deep convolutional neural network based autonomous drone navigation. In *Thirteenth International Conference on Machine Vision*, volume 11605, pages 16–24. SPIE, 2021.

[56] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[57] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[58] Antonio Loquercio, Ana I Maqueda, Carlos R Del-Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 3(2):1088–1095, 2018.

[59] Ramya S Nair and P Supriya. Robotic path planning using recurrent neural networks. In *2020 11th International Conference on Computing, Communication and Networking Technologies*, pages 1–5, 2020.

[60] Yu Chen, Qi Dong, Xiaozhou Shang, Zhenyu Wu, and Jinyu Wang. Multi-uav autonomous path planning in reconnaissance missions considering incomplete information: A reinforcement learning method. *Drones*, 7(1):10, 2022.

[61] Bohao Li and Yunjie Wu. Path planning for uav ground target tracking via deep reinforcement learning. *IEEE Access*, 8:29064–29074, 2020.

[62] Junxiao Xue, Xiangyan Kong, Bowei Dong, and Mingliang Xu. Multi-agent path planning based on MPC and DDPG. *arXiv preprint arXiv:2102.13283*, 2021.

[63] Zijia Niu, Xiaohu Jia, and Wang Yao. Communication-free mpc-based neighbors trajectory prediction for distributed multi-uav motion planning. *IEEE Access*, 10:13481–13489, 2022.

[64] Esen Yel and Nicola Bezzo. A meta-learning-based trajectory tracking framework for uavs under degraded conditions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6884–6890, 2021.

[65] Bryan Penin, Riccardo Spica, Paolo Robuffo Giordano, and François Chaumette. Vision-based minimum-time trajectory generation for a quadrotor uav. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6199–6206, 2017.

[66] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[67] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR, 2017.

[68] Shahab Ghorbani, Zahra Samadikhoshkho, and Farrokh Janabi-Sharifi. Dual-arm aerial continuum manipulation systems: modeling, pre-grasp planning, and control. *Nonlinear Dynamics*, 111(8):7339–7355, 2023.

[69] Seyed Hamed Hashemi, Farrokh Janabi-Sharifi, and Amir Jalali. Robust global stabilization of aerial continuum manipulation systems via hybrid feedback. *ISA transactions*, 138:160–167, 2023.

[70] Zahra Samadikhoshkho, Shahab Ghorbani, and Farrokh Janabi-Sharifi. Vision-based reduced-order adaptive control of aerial continuum manipulation systems. *Aerospace Science and Technology*, 121:107322, 2022.

[71] Anuraj Uthayasooriyan, Fernando Vanegas, Amir Jalali, Krishna Manaswi Digumarti, Farrokh Janabi-Sharifi, and Felipe Gonzalez. Tendon-driven continuum robots for aerial manipulation—a survey of fabrication methods. *Drones*, 8(6):269, 2024.

[72] Zahra Samadikhoshkho, Shahab Ghorbani, and Farrokh Janabi-Sharifi. Modeling and control of aerial continuum manipulation systems: A flying continuum robot paradigm. *IEEE Access*, 8:176883–176894, 2020.

[73] Ammar Amouri, Abdelouahab Zaatri, and Chawki Mahfoudi. Dynamic modeling of a class of continuum manipulators in fixed orientation. *Journal of Intelligent & Robotic Systems*, 91(3):413–424, 2018.

[74] Tsung-Yen Yang, Tingnan Zhang, Linda Luu, Sehoon Ha, Jie Tan, and Wenhao Yu. Safe reinforcement learning for legged locomotion. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2454–2461, 2022.

[75] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):60–74, 2022.

[76] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *26th Annual International Conference on Machine Learning*, page 41–48, 2009.

[77] Shaswat Garg, Houman Masnavi, Baris Fidan, Farrokh Janabi-Sharifi, and Iraj Mantegh. Benchmarking off-policy deep reinforcement learning algorithms for UAV path planning. In *International Conference on Unmanned Aircraft Systems*, pages 317–323, 2024.

[78] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[79] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[80] Seyed Mostafa Safdarnejad, John D Hedengren, Nicholas R Lewis, and Eric L Haseltine. Initialization strategies for optimization of dynamic systems. *Computers & Chemical Engineering*, 78:39–50, 2015.

[81] Erdem Acar, Gamze Bayrak, Yongsu Jung, Ikjin Lee, Palaniappan Ramu, and Suja Shree Ravichandran. Modeling, analysis, and optimization under uncertainties: a review. *Structural and Multidisciplinary Optimization*, 64(5):2909–2945, 2021.