Path Planning and Control of UAV using Machine Learning and Deep Reinforcement Learning Techniques

Yintao Zhang

A Thesis

in The Department

of

Mechanical, Industrial and Aerospace Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Mechanical Engineering) at

Concordia University

Montréal, Québec, Canada

March 2023

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: **Yintao Zhang**

Entitled: **Path Planning and Control of UAV using Machine Learning and Deep Reinforcement Learning Techniques**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Mechanical Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Xiupu Zhang

_____ External Examiner
Dr. Yang Shi

_____ Examiner
Dr. Jun Yan

_____ Examiner
Dr. Javad Dargahi

_____ Examiner
Dr. Wen-Fang Xie

_____ Supervisor
Dr. Youmin Zhang

Approved by _____
Dr. Muthukumaran Packirisamy, Graduate Program Director

January 10, 2023
Date of Defense

_____
Dr. Mourad Debbabi, Dean, Gina Cody School of Engineering and Computer Science

Abstract

Path Planning and Control of UAV using Machine Learning and Deep Reinforcement Learning Techniques

Yintao Zhang, Ph.D.

Concordia University, 2023

Uncrewed Aerial Vehicles (UAVs) are playing an increasingly significant role in modern life. In the past decades, lots of commercial and scientific communities all over the world have been developing autonomous techniques of UAV for a broad range of applications, such as forest fire monitoring, parcel delivery, disaster rescue, natural resource exploration, and surveillance. This brings a large number of opportunities and challenges for UAVs to improve their abilities in path planning, motion control and fault-tolerant control (FTC) directions. Meanwhile, due to the powerful decision-making, adaptive learning and pattern recognition capabilities of machine learning (ML) and deep reinforcement learning (DRL), the use of ML and DRL have been developing rapidly and obtain major achievement in a variety of applications.

However, there is not many researches on the ML and DRl in the field of motion control and real-time path planning of UAVs. This thesis focuses on the development of ML and DRL in the path planning, motion control and FTC of UAVs. A number of contributions pertaining to the state space definition, reward function design and training method improvement have been made in this thesis, which improve the effectiveness and efficiency of applying DRL in UAV motion control problems. In addition to the control problems, this thesis also presents real-time path planning contributions, including relative state space

definition and human pedestrian inspired reward function, which provide a reliable and effective solution of the real-time path planning in a complex environment.

Acknowledgments

This thesis is submitted in partial fulfillment of the requirements for the degree of philosophy doctor (Ph.D.) at Concordia University (CU). The presented research has been conducted under the supervision of Prof. Youmin Zhang in the Diagnosis, Flight Control and Simulation (DFCS) lab and Networked Autonomous Vehicles (NAV) lab at the Department of Mechanical, Industrial and Aerospace Engineering, Concordia University.

I have been very fortunate to be able to collaborate with so many outstanding researchers during my Ph.D. studies. Particularly, my deepest gratitude goes to my supervisor, Prof. Youmin Zhang, for offering me this valuable and unique opportunity to pursue my Ph.D. degree and learn how to do research. Beyond the studies, he also teaches me how to deal with different important phases in my life. Without his consistent support and encouragement, as well as valuable and insightful guidance, my research progress would probably be significantly slowed and unsatisfactory. His positive attitude to both life and work, great passion and diligence to research, organizational skills, and keen insight on solving problems have always been a relentless source of inspiration to me. It is my great privilege to work and study under his guidance.

I would like to thank Prof. Jun Yan, Prof. Wen-Fang Xie, Prof. Javad Dargahi, and Prof. Yang Shi for joining my examining committee and providing useful feedback and brilliant comments during my comprehensive exam and research proposal. I am also grateful to the graduated and current group members who have helped me a lot, not only on the discussion of challenging issues, experimental platform construction and tests, paper writing, but also

problems met in my daily life. It is my great pleasure to do research and share opinions with them and learn from them. I have been benefited a lot from them.

Finally, I would like to thank my parents and my girlfriend, for their love, understanding, consistent support, and countless sacrifices, which have played an indispensable role in my life. It can never be enough for me to thank them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

1.1.1 Uncrewed Aerial Vehicles (UAVs)

During the past decades, uncrewed aerial vehicles (UAVs) have been broadly deployed in wide range of civil applications and military missions such as forest fire detection and monitoring [1, 2, 3], regional surveillance [4, 5, 6, 7], parcel delivering [8, 9, 10], disaster rescue [11, 12, 13] and entertainment [14, 15]. Especially in the recent years, with the fast-developing techniques in guidance, navigation and control (GNC), small-scale UAVs have received increasing attention of both researchers and engineers. In general, the UAVs can be categorized into three main types according to their flying power mechanism and configuration design [16].

(1) *Fixed-wing UAV*. As shown in Fig. 1.1, it has fixed wings and the flight capability is achieved by using wings to generate lift caused by the UAV's forward speed. According to the flying mechanism, fixed-wing UAVs have to maintain a certain forward speed and need a long runway to take off and land. Compared to multi-rotor UAV, fixed-wing UAVs are more energy-efficient and can carry more payload.

Figure 1.1: A sample of fixed-wing UAV.

(2) *Multi-rotor UAV*. As shown in Fig. 1.2, multi-rotor UAVs are equipped with several propellers (three or more). The flying lift is directly generated by the downward thrust caused by propellers.

Figure 1.2: A sample of multi-rotor UAV.

The most popular multi-rotor UAV is quadrotor UAV, which is shown in Fig.1.3. Unlike the fixed-wing UAV, multi-rotor UAVs do not have many actuators such as elevator, aileron and rudder, the motion and attitude are controlled by changing the angular speed of each propeller. Compared to fixed-wing UAV, multi-rotor UAV is capable of vertical taking-off and landing (VTOL) and high maneuverability.

(3) *Compound UAV*. A compound UAV combines the fixed-wing and multi-rotor UAVs. As shown in Fig.1.4, they have both wings and multiple propellers, which enables the UAV both abilities of fast forward speed and VTOL. Although the compound UAVs have extra abilities, their complex structure reduces the safety and stability of flight and incurs high maintenance cost.

Figure 1.3: A sample of quadrotor UAV.

Figure 1.4: A sample of compound UAV.

In summary, multi-rotor UAV has attracting more and more attention in both industrial and academic communities due to its high maneuverability, small size, capability of VTOL, and both indoor and outdoor flight.

1.1.2 Machine Learning and Deep Reinforcement Learning

Machine learning (ML) techniques have been deeply investigated and used in a wide applications in the autonomous agents and unmanned systems because of its adaptive learning ability [17, 18]. Reinforcement learning (RL) is one of the branches of ML, which can optimize the action policy based on the interactions between agent and environment. Especially for navigation in complex environment, RL has showed its powerful learning ability [19, 20].

However, most of the existing RL path planning algorithms are heavily relied on the

pre-known information of the map. These algorithms cannot deal with unknown environment information because it is impossible to train a new Q-table at each time step. It is a quite challenging issue for RL algorithm if there are dynamic obstacles or unknown static obstacles in the map. Therefore, this thesis concentrates on the study of application of RL algorithm in real-time path planning.

During the execution of ML and RL for continuous control problem, several problems remain unsolved, for instance, the continuous nature of system input and output, the correlation of data experience and the convergence speed of learning [21]. These issues have been limiting the application of ML and RL in control systems over the past several years. However, the deep Q-network (DQN) has successfully combined Q-learning and neural network (NN), which can be used for continuous control problems. [22, 23]. Prior to DQN, using large, non-linear function approximator to learn value functions is believed to be difficult and unstable [24]. DQN has unveiled a new page of possibility due to the following two innovations:

(1) The design of replay buffer. In order to reduce the correlations between data, the network of DQN is trained off-policy while the training data comes from replay buffer, an experience sets.

(2) The design of target network. Aiming at breaking the correlation of two networks, both networks will be trained by two independent target networks in order to maintain a consistent objective.

Although DQN has been able to solve problem with continuous action space, a disadvantage could not be ignored that DQN requires observation in high dimension while solving problems in low-dimensional action spaces [25]. Aiming at overcoming this shortcoming, based on the idea of DQN, a new approach deep deterministic policy gradient (DDPG) is proposed in [26]. Compared to DQN, an important feature of DDPG is its

simplicity, DDPG requires only an actor-critic framework, making the algorithm easy to implement and applicable for some complex problems and larger networks [27, 28].

Although DDPG has good performance, it has inevitable drawbacks because DDPG may overestimate the Q-values in the critic network [29]. These estimation errors may accumulate and eventually lead to a non-optimal action. Aiming at mitigating the overestimation, a twin-delayed DDPG (TD3) was proposed in [27, 30]. The TD3 algorithm introduces a double deep Q-learning in the DDPG algorithm. Compared to DDPG, TD3 algorithm has better action performance and faster learning speed due to the following improvements [30, 27, 31].

(1) TD3 has two Q-networks with the same structure instead of one Q-network. TD3 uses the smaller one as the Q-value. As a result, the overestimation problem will be alleviated.

(2) TD3 reduces the update frequency in order to solve the coupling problem of unchanged actor-critic networks.

(3) TD3 adds noise to the target action which makes the algorithm more difficult to exploit Q-function errors and makes the critic smoother.

Currently, the proof of stability is the biggest challenge of DRL while it is used as controller. Due to the characteristics of NN, it is impossible to give a strict mathematical proof. However, according to [32], the DRL can ensure a certain gradient descent within a certain range.

1.2 Literature Review

1.2.1 Path Planning of UAV

Path planning is one of the essential technologies of UAVs, which designs a safe, collision-free and least-cost path based on one or more criteria. It has become an indispensable technique in a wide range of problems such as disaster rescue [33, 34, 35], fire detection [36, 37], self-driving [38, 39], parcel delivery [40, 41] and surveillance [42, 43].

Path Planning Method Classification

According to the mechanism of used algorithm, path planning can be categorized into five aspects: field-based algorithms, grid-based algorithms, node-based algorithms, sampling-based algorithms and intelligent algorithms.

A typical example of field-based algorithm is artificial potential field (APF) method [44, 45]. A sample of APF method is shown in Fig. 1.5. The basic concept behind APF is to generate several potential fields in the environment. There are two forces caused by the potential field, obstacles are repelled by repulsive force and the UAV is drawn toward the objective by attractive force. In the potential field environment, UAV will move from high-potential area towards low potential area.

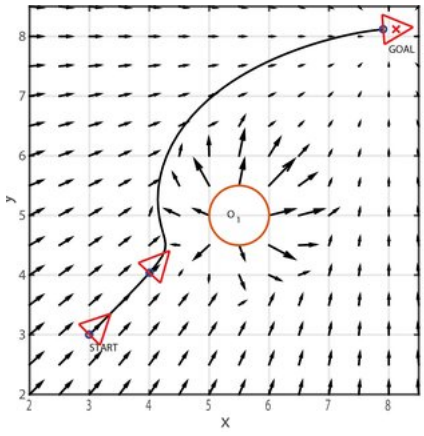


Figure 1.5: A sample of artificial potential field path planning method.

Grid-based algorithm [46, 47] divides the entire map into a number of grid units or specific areas which reduce the complexity and difficulty to find the optimal path. A sample of grid-based path planning algorithm is shown in Fig. 1.6. UAV moves from the current grid to the next one. The objective of grid-based algorithm is to find a continuous sequence of these grids connecting the start position and goal position.

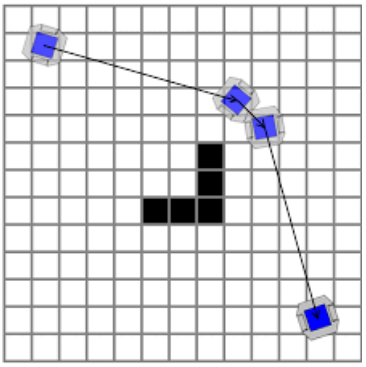


Figure 1.6: A sample of grid-based path planning method.

Node-based algorithms [48, 49] is usually considered as the simplest technique to find a path for UAV. It is regarded as an effective and efficient strategy for selecting a proper path that requires less time and computing complexity. A sample of node-based path planning algorithm is shown in Fig. 1.7. The main idea of node-based algorithm is to identify a number of special nodes in the environment, the UAV moves from one node to another until reaching the target. The path can be easily obtained by connecting the nodes.

Sampling-based algorithms are usually used to solve path planning problem with high-dimensional systems, especially for those have state and action constraints. A sample of sampling based path planning algorithm is shown in Fig. 1.8. Rapidly-exploring random trees (RRT) [50, 51] and probabilistic roadmaps (PRM) [52, 53] are two typical sampling-based algorithms. The RRT algorithm grows a tree from the initial configuration and then selecting random samples in the search space. While a sample is drawn, a link is made. Feasible links will be added to the tree. Repeat the sampling process and expand the existing state until the goal is reached. The main idea behind PRM is to pick random samples

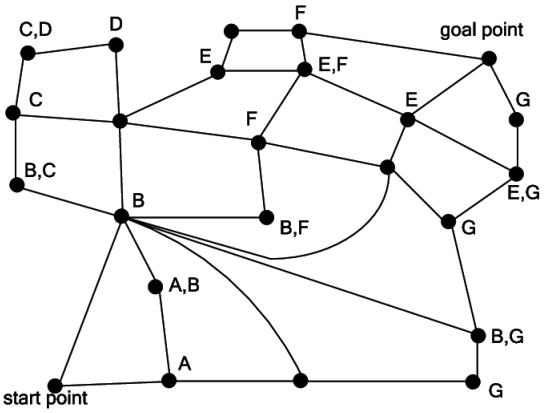


Figure 1.7: A sample of node-based path planning method.

from the UAV's configuration space, check whether they are in the open space, then connect them to adjacent configurations until the target. There are two phases of PRM, construction and query phase. In the first phase, a roadmap is generated to approximate the possible actions that can be taken in the environment. In query phase, the path can be obtained by connecting the start and goal configurations to the graph.

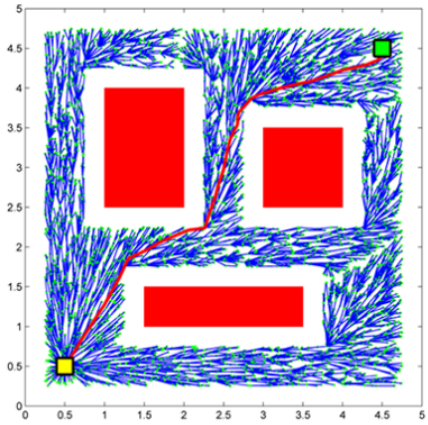


Figure 1.8: A sample of sampling-based path planning method.

Intelligent algorithms consists of genetic algorithm (GA) [54, 55] and ant colony optimization algorithm (ACO) [56, 57]. A sample of ACO path planning algorithm is shown in Fig. 1.9. The solution evolves in GA by using the selecting, crossover and mutating calculators to simulate the process of evolution. The best solution will be obtained after a large

number of iterations. ACO is inspired by the behaviour of ants and insects, the pheromone of communication will be enhanced in good choices. The optimal path will be generated by following the higher level of pheromone.

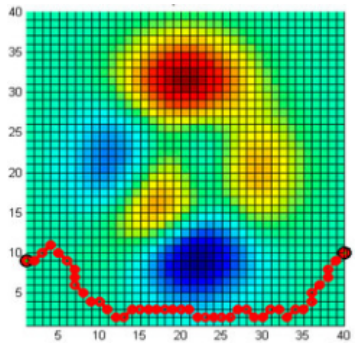


Figure 1.9: A sample of ACO path planning method.

Path Planning Problem Classification

According to the types of solved problem, path planning can also be divided into two main categories, one is off-line path planning and the other is on-line path planning. The main difference is whether the path is generated in advance or during the flight. The off-line path planning comprehensively considers the global information and then generates an optimal path from the start position to the terminal prior to the flight. In other words, the path is fixed and cannot be re-planned during the flight. Since the off-line path planning method takes consideration of the global information, it is also regarded as global path planning. On the contrary, on-line path planning can generate the trajectory during the flight based on local information with a certain range. As a result, on-line path planning is also named as local path planning. It is obviously that off-line path planning cannot deal with the problem with dynamic environment. However, along with the fast-developing artificial intelligence and image processing techniques in recent years, self-navigation and autonomous driving has become the hot topic. On-line path planning meets lots of new challenges and reveals a new page. In local path planning, it is necessary to perceives

the surrounding environment, a combination of LiDAR, GPS, inertial navigation system and cameras is a powerful positioning tool which can provide precise information of local environment. This positioning technique is generally called simultaneous localization and mapping (SLAM), which is shown in Fig. 1.10. With the help of SLAM, UAV will have a strong perception of the surrounding environment which is crucial to on-line path planning. Therefore, on-line path planning is a promising research direction and a hot topic in recent future.

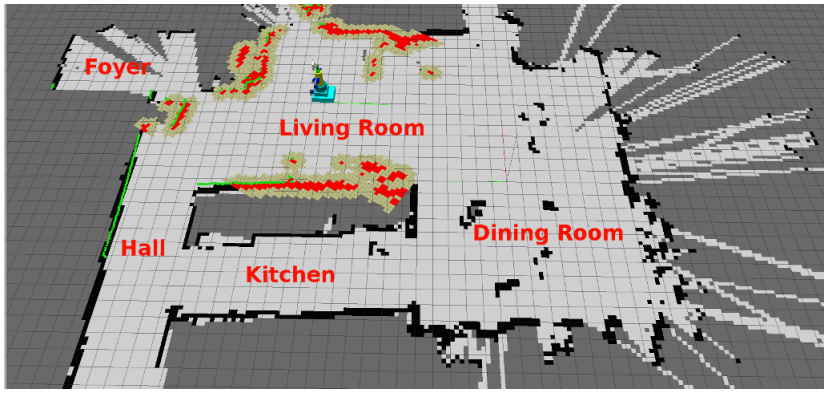


Figure 1.10: An example of SLAM method.

Reinforcement Learning based Path Planning Method

In recent years, RL is becoming a promising approach for solving path planning problem. Q-learning was first proposed in [58], it can be learned from delayed rewards and punishments. After that, a number of researchers focus on solving the path planning problem using RL. In [59, 60], the authors proposed a Q-learning based navigation system for obstacle avoidance problem. The successful rate is considered to the high and the overall performance is good. Some literature improved the Q-learning algorithm for navigation system design. In [61], an extended Q-learning algorithm is investigated, only the best action of a state will be stored in the Q-table. This design greatly increases the learning

speed but has poor performance when facing complicated environment. In [62], an improved Q-learning is investigated, which can be considered as the future work of [61]. The main idea of this algorithm is the definition of locked state. When a state is locked, there will be no exploration in this state. This mechanism guarantees the fast convergence speed and good performance at the same time. Some of the researchers have extended RL to a more complicated environment such as 3-D environment, Aranibar et al. [63] use RL to solve path planning problem in 3-D environment, both Q-learning and neural Q-learning are applied in the greedy search algorithm. It should be noted that the reward function used in Q-learning is the same one used in [59]. Simulation results showed that Q-learning is more suitable for small and medium size of environment while the neural Q-learning works better on the large size of environment. Some algorithms also try to combine RL with other intelligent methods such as FLS and neural network (NN). A hybrid method combining RL and fuzzy logic is proposed in [64], the navigation system has two tasks, one is obstacle avoidance and the other is goal seeking. Compared to other RL algorithm, the major difference of this method is that two Q-learning algorithm is constructed independently. The fuzzy rules are designed with the assistant of RL algorithms. Another hybrid approach is proposed in [65], the designed navigation system has three main functions, the first one is obstacle avoidance, the second one is goal-seeking and the last one is supervising fuzzy logic system. The fuzzy supervisor mechanism generate the best action based on the information from obstacle avoidance module and goal-seeking module. For the problem of continuous action and state space, Yang et al. [66] proposed a method combining deep neural network and RL. A multiple layers neural network is utilized to estimate Q-values. This algorithm can generate path from start position to goal position in a continuous environment. It can be inferred that using RL as solution of path planning problem is becoming a tendency due to the excellent performance of Q-learning algorithm.

1.2.2 Motion Control of UAV

Motion control of UAV is a broad concept which contains lots of issues such as path following control [67, 68], formation control [69, 70] and auto-landing control [71, 72]. In general, all the control problems which enables the system to control the precise speed and location of UAV can be regarded as motion control.

In this thesis, the motion control of UAV is studied in the following three direction, first one is path following control of fixed-wing UAV, the second one focuses on the formation control of a group of fixed-wing UAVs, the last one will study the auto-landing problem of quadrotor UAV.

Path following control

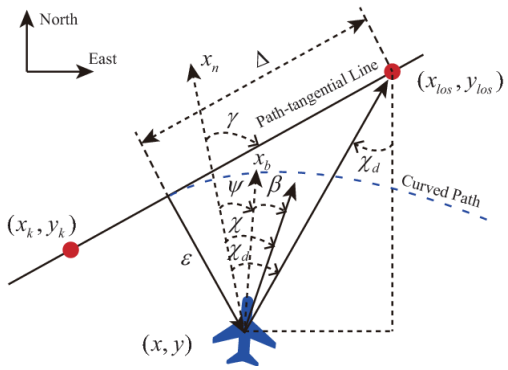


Figure 1.11: An example of path following control problem.

Traditionally, there is a wide variety of approaches solving path following problem. Most of these techniques can be considered as a direct application of classic control theories on a geometry model. The most commonly used methods are pure pursuit [73, 74], line-of-sight [75, 76], and constant bearing guidance [77].

However, traditional methods have their distinct limitations, such as weak robustness of model uncertainties, instability under external disturbances and low efficiency in computing. Under this circumstance, ML techniques, as well as RL, has become a promising

solution since it can overcome most of these limitations. In [78], an actor-critic-based reinforcement learning algorithm is proposed, which is adapted to capture the experience during the path following trial. Reference [79] investigates the path following control problem in 3-D environment. To ensure the control adaptability without dependence on an accurate model, the Q-learning algorithm is directly adopted for learning the action policy.

Formation Control

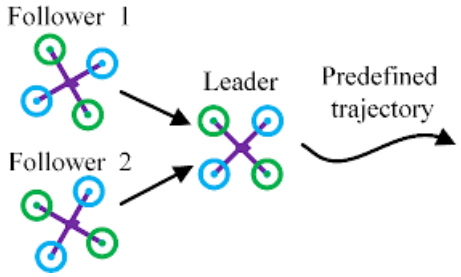


Figure 1.12: An example of formation control problem.

During the past couple of years, cooperative control of multiple unmanned aerial vehicles (multi-UAVs) has been extensively studied in a number of applications such as disaster rescue, forest fire detection and surveillance [80, 81, 82, 16]. For the above applications, multi-UAVs have distinct advantage over an individual UAV [83, 84, 85]. In the cooperative control of multi-UAVs, formation control is one of the most challenging and interesting topics. Generally, the formation control is to design an appropriate protocol or an algorithm such that the UAV group can maintain a geometric shape while moving [86].

Among published literature, the following approaches have received considerable attentions in the study of formation control, such as virtual structure [87, 88], behavior-based method [89, 90], leader-follower structure [91, 92] and potential function-based method [93, 94]. Leader-follower method is the most popular approach because of its simple structure. The main idea of leader-follower method is considering the leader as reference of the followers. Thus, the group behavior can be easily specified by controlling the motion of

leader.

However, the traditional methods have limitations such as low efficiency in computing, instability when facing disturbances and weak robustness in model uncertainties [95]. In order to overcome the above difficulties, ML and RL techniques, have attracted extensive attention and develops rapidly in recent years. In [96], a distributed optimal control method using reinforcement learning is proposed to address the UAV formation tracking problem. The proposed control law contains a distributed observer and a model-free off-policy reinforcement learning protocol. The reinforcement learning algorithm is designed to obtain the optimal control input without any knowledge of the follower's dynamics.

Auto-Landing Control

Figure 1.13: An example of auto-landing control problem.

The autonomous landing is one of the key topics of UAV applications [97, 98, 99, 100], especially in a long-term missions because of the short duration of battery [101]. In addition, the use of multiple vehicles has obvious benefit compared to a single one, such as a combination of UAV. Therefore, there are a plenty of researches focused on the topic of auto-landing on a moving platform [98, 102, 103, 104, 105]. Compared to landing on a static platform, landing on a moving one is more complex and difficult [106].

Generally, the approaches solving autonomous landing problems can be divided into two main categories, one is based on pose prediction and pose control [98, 102, 107], the other is based on trajectory generation and motion control [108, 104, 105, 109]. However, most of the existing researches are model-based approaches, which requires an accurate model of the landing system. To overcome the limitation of traditional methods, data-based approaches have attracted the attention of researchers. Reference [110] proposes a UAV forced landing site detection system which is based on machine learning approaches including the Gaussian Mixture Model and the Support Vector Machine (SVM). The proposed algorithm has significant improvement in detecting a safe landing area and is more reliable. Muvva [111] investigates the problem that a UAV is auto-landing on another UAV. The single shot multi-box detector (SSD) network is trained to help the controller land on another flying UAV. In [112], a new autonomous tracking and landing approach based on a deep reinforcement learning strategy is proposed with the objective of dealing with the UAV motion control problem in an unpredictable and harsh environment. The proposed method combines the DDPG algorithm and heuristic rules and helps the UAV automatically learns the landing maneuver by an end-to-end neural network.

1.2.3 Fault-Tolerant Control of UAV

UAVs are prone to faulty conditions such as malfunctions, actuator degradation, sensor errors, external disturbances and other faulty issues [113, 114, 115]. These faulty issues may lead to degraded performance, system instability and even catastrophic results. According to [116], 61% of the flight accidents were caused by the loss of control. Motivated by the essential requirement of safe flight, researchers have done extensive works on fault-tolerant control (FTC) of UAVs [117, 118, 115, 119, 120].

FTC is an efficient and effective strategy that improves system robustness or helps system to recover when facing faulty situations. The algorithms of FTC can be divided into

two main directions, one is passive FTC and the other is active FTC [121]. The idea of passive FTC is to design a reliable controller which is robust enough to maintain system performance against faults [122]. There is no controller reconfiguration in passive FTC so these controllers must accommodate both normal and faulty situations [123]. In contrast to passive FTC, with the help of fault diagnosis and identification (FDI) technique, active FTC algorithms react to the faulty situations actively [124]. As soon as the fault is detected by the FDI mechanism, the controllers of active FTC can reconfigure themselves to compensate the negative effect of fault. Then the system performance will be recovered to healthy conditions.

1.3 Objectives of This Thesis

This thesis aims to design and develop the path planning, motion control and fault-tolerant control schemes with application to both fixed-wing and multi-rotor UAVs. Particularly, this thesis is organized based on the following research objectives:

(1) Designing and developing a path planning algorithm based on Q-learning that can be used in real-time 3-D collision-free path planning mission.

(2) Designing and developing a path following control method that can counteract the adverse effects from side-slip angle and environment disturbance while successfully operate the UAV to follow the desired path.

(3) Designing and developing a formation control scheme for a group of fixed-wing to maintain the formation shape in the operation.

(4) Designing and developing an auto-landing control strategy for multi-rotor UAVs, which can drive the UAV land on the moving ground platform safely and smoothly.

(5) Designing and developing a fault-tolerant control method for fixed-wing UAV without an explicit model while considering actuator faults in order to enhance the reliability and safety of the flight.

In summary, the conducted research works in this thesis are primarily expected to synthesize advanced levels of path planning, collision avoidance capability, motion control and fault tolerant control in UAVs, these in turn can guarantee the satisfactory, reliable and safe performance in both individual and multiple UAVs levels. Finally, the proposed algorithms and strategies are verified by a series of simulations on commonly used UAV models and experimental tests on the QDrone and QCar platform.

1.4 Contributions of This Thesis

The main contributions of this thesis can be summarized as follows:

(1) Real-time path planning.

Design a state space which is based on relative location. This new state space definition can map the dynamic environment precisely and timely.

Design a new reward function which is inspired by the human pedestrian behavior. With the help of this new reward function, the trained algorithm can avoid obstacles effectively while maintaining the shortest path in a complex environment.

(2) Path following control for fixed-wing UAV.

Design a new framework of DDPG for UAV path following problem.

Design a new training technique, named the double experience replay buffer (DERB) which improves the learning ability.

(3) Formation control for a group of fixed-wing UAVs.

Design a new definition of state space which is suitable for the DRL algorithm to learn and control the UAV group for formation maintenance mission.

Design a new reward function to sensitively and accurately represents the performance of an action in the formation control problem.

Design a replay buffer with flexible capacity, which aims at improving the learning efficiency of the algorithm. In addition, the prioritized sampling methods can also increase the learning efficiency by replaying the better samples more frequently.

(4) Auto-landing control for multi-rotor UAV.

Design a transfer-imitation learning training (TILT) approach, which can help TD3 algorithm learning safely and effectively in the real-world applications.

(5) Fault-tolerant control for fixed-wing UAV in the presence of actuator faults.

Develop a model-free fault-tolerant controller based on the state-of-the-art SAC algorithm. The main advancements of SAC are entropy and double Q-learning. The concept of entropy helps the algorithm to have a better exploration of the data. And Double Q-learning also avoids the overestimation of NN.

Design a robust term in the state space, which provides an extra knowledge of system status. An additional dimension of information can help the algorithm to generate better decisions, especially in faulty situations.

Design a new reward function for performance of an action in the FTC control problem, especially under system fault conditions. The design of penalty and bonus is an accelerator for training process.

1.5 Organization of This Thesis

The rest of this thesis is organized as follows:

• Chapter 2 provides an overview of some preliminary knowledge used in this thesis.

• Chapter 3 illustrates the problem of 3-D real-time Q-learning path planning problem in the presence of obstacles. Two Q-learning algorithms are studied, the first one builds a new environment model that allows machine learning algorithms to be effectively and efficiently applied in real-time path planning. The second one introduces the idea of human pedestrian behavior, the algorithm performance in complex environment has been significantly improved by learning from pedestrian behavior.

• Chapter 4 introduces the motion control of UAVs in three main directions. First one is path following control. A deep deterministic gradient policy algorithm is proposed to drive the UAV follow the desired path with disturbances. The second one refers to the formation control of a group of fixed-wing UAVs. A leader-follower formation control methodology is designed based on the twin-delayed deep deterministic gradient policy. Compared to the original DDPG, the proposed TD3 method has fast response time and reliable performance. The third one is the auto-landing control of multi-rotor UAVs. A training method is proposed for the real-world control issues.

• Chapter 5 addresses the FTC problem of fixed-wing UAV. Based on the actor-critic framework, the SAC algorithm is constructed by introducing two main techniques, the concept of entropy and clipped double Q-learning. Then, a robust term is defined to help the algorithm for a better awareness of the system status, which makes it easier to maintain system performance under system faults. After that, a new reward function is designed to evaluate the value of each experience even under faulty conditions.

• Chapter 6 presents conclusions of the conducted research works and important findings, and summarizes several predominant ideas for the future developments of the thesis's outcomes.

1.6 Publications During the Thesis Work

Journal Papers

(1) Yintao Zhang, Youmin Zhang, and Ziquan Yu (2022), Fault-tolerant control for fixed-wing UAV using soft actor-critic deep reinforcement learning. International Journal of Aerospace Engineering, 2022.

(2) Yintao Zhang, Youmin Zhang, and Ziquan Yu (2021), Path following control for UAV using deep reinforcement learning approach. Guidance, Navigation and Control 1.01 (2021): 2150005.

(3) Yintao Zhang, Youmin Zhang, and Ziquan Yu (2019), Wild forest fire monitoring using multiple UAVs based on reinforcement learning framework. Science China Information Sciences, 2019.

(4) Yaohong Qu, Yintao Zhang, and Youmin Zhang (2017), A global path planning algorithm for fixed-wing UAVs. Journal of Intelligent & Robotic Systems 91(3): 691-707, 2018.

Conference Papers

(1) **(Best Paper Award)** Yintao Zhang, Youmin Zhang, and Ziquan Yu (2020), A deep reinforcement learning strategy for UAV path following control under sensor fault. in *Proceedings of International Conference on Guidance Navigation and Control (ICGNC)*, 2020, Tianjin, China.

(2) Yintao Zhang, Youmin Zhang, and Ziquan Yu (2019), A solution for searching and monitoring forest fires based on multiple UAVs. in *Proceedings of International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019, Atlanta, USA.

(3) Yintao Zhang, Youmin Zhang, and Ziquan Yu (2018), Line-of-sight path following control on UAV with sideslip estimation and compensation. in *Proceedings of 37th Chinese Control Conference (CCC)*, 2018, Wuhan, China.

(4) Yintao Zhang and Youmin Zhang (2017), Hovering quadrotor control in a windy environment using neural network. in *Proceedings of the Canadian Congress of Applied Mechanics (CANCAM)*, 2017, Victoria, Canada.

Chapter 2

Preliminaries

2.1 Modelling of a Quadrotor UAV

The quardrotor UAV, which is shown in Fig. 2.1, is a small-scale UAV with four propellers evenly placed around the body. The main body contains battery, on-board sensors and micro-controller. The pitch, yaw and roll motion can be achieved by independently controlling the rotational speed of each propeller. The front and rear propellers rotate clockwise, while the left and right ones rotate counter-clockwise (shown in Fig. 2.1 and Fig. 2.2). Using this configuration, each pair of propellers contributes to the motion along x-axis and y-axis, respectively.

Before modelling of the quadrotor UAV, the coordinate systems need to be defined in advance. Two coordinate systems will be used, one is the earth coordinate system $S_e - O_e x_e y_e z_e$ and the other is the body coordinate system $S_b - O_b x_b y_b z_b$. The directions of x_e, y_e and z_e are North, East and downward (vertical to the North-East surface). While the directions of x_b, y_b and z_b are pointing to the UAV's head, UAV's right wing and downward (vertical to the $x_b y_b-$ surface). The position $X^e = [x_e, y_e, z_e]^T$ and attitude $\Theta^e = [\phi, \theta, \psi]^T$ of quadrotor UAV are variables in the earth coordinate system which is also regarded as the inertial reference frame. The transnational velocity $V^B = [u, v, w]^T$ and rotational velocity

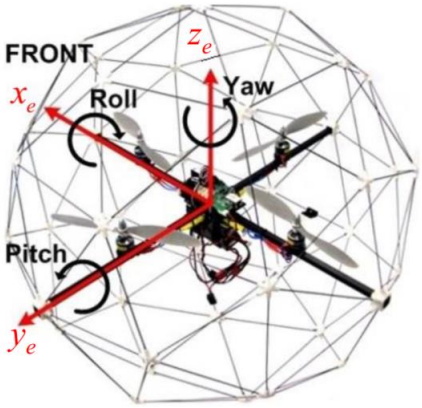


Figure 2.1: A typical quadrotor UAV.

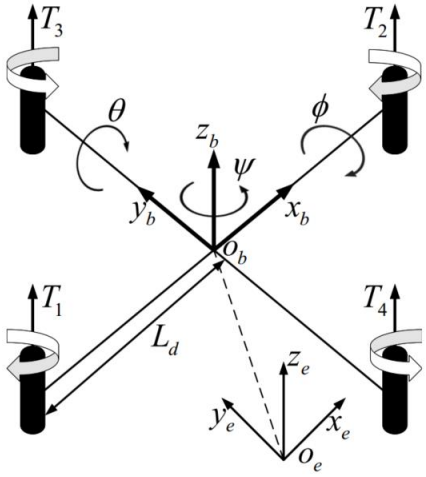


Figure 2.2: Configuration of the quadrotor UAV.

$\omega^B = [p, q, r]^T$ are defined in the body frame.

In order to derive the equations of motion, the Newton-Euler formulation needs to be employed. However, the Newton's equation of motion is usually derived in the inertial reference frame, whereas the motion of object is more convenient if described in the body coordinate system. In addition, the data collected by on-board sensors is respect to the body-fixed frame. As a result, a transform matrix from body-fixed frame to the inertial reference frame is necessary. The transform matrix $R_{B\to e}$ can be obtained by multiplying

three rotation matrices of roll, pitch and yaw motion.

$$
\begin{aligned}
R_{B\to e} =& R_\psi R_\theta R_\phi \\
=& \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \\
=& \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\psi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}
\end{aligned} \tag{2.1}
$$

Another transform matrix $T_{B\to e}$ can be obtained by resolving the Euler angle rates into rotational velocities in the body-fixed frame as follows:

$$
\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot\phi \\ 0 \\ 0 \end{bmatrix} + R_\phi \begin{bmatrix} 0 \\ \dot\theta \\ 0 \end{bmatrix} + R_\phi R_\theta \begin{bmatrix} 0 \\ 0 \\ \dot\psi \end{bmatrix} = (T_{B\to e})^{-1} \begin{bmatrix} \dot\phi \\ \dot\theta \\ \dot\psi \end{bmatrix} \tag{2.2}
$$

Therefore, the transform matrix is:

$$
T_{B\to e} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \tag{2.3}
$$

According to the above transform matrices, the kinematic equations can be described as:

$$
\begin{bmatrix} \dot X^e \\ \dot\Theta^e \end{bmatrix} = \begin{bmatrix} R_{B\to e} & 0_{3\times3} \\ 0_{3\times3} & T_{B\to e} \end{bmatrix} \begin{bmatrix} V^B \\ \omega^B \end{bmatrix} \tag{2.4}
$$

2.2 Markov Decision Process Model

Markov decision process (MDP) is a discrete-time stochastic control process. It is also a modelling method for decision-making problems where the result can be controlled by the decision-maker. A typical MDP is a four-element tuple (S, A, P, R).

- S is a set of states, which is generally called state space.
- A refers to a set of actions, normally called action space. A_s stands for the available actions at state s.
- P represents the transition probability, which is defined as $P(s'|s, a)$, where s' is the next state if agent takes action a at current state s.
- $R(s, a)$ is the immediate reward, it will be received if agent takes action a at state s.

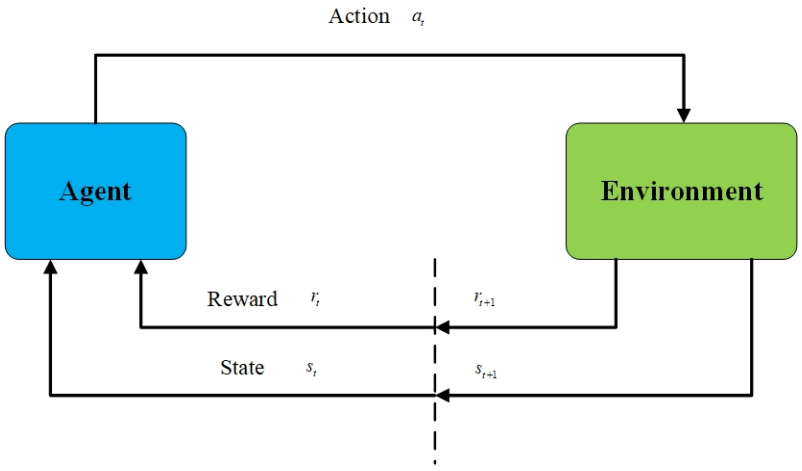


Figure 2.3: A typical MDP model.

As shown in Fig. 2.3, at each time instant, the agent perceives the environment and obtains the current state s_t. After that, the decision-making policy π will generate the best action a_t. Then, the agent executes the selected action and receives an immediate reward $r_t = R(s_t, a_t)$. Finally, the agent will enter the next state s_{t+1}.

The goal of MDP is to find a certain good policy π for decision-making. The policy π

will maximize the value function which is defined below:

$$V(s) = \mathbb{E}[R] = \mathbb{E}[\sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)] \tag{2.5}$$

where $\gamma \in (0, 1]$ is the discount rate and R refers to the discount accumulated reward $R = \sum_0^\infty \gamma^i r_i$.

2.3 Basics of Reinforcement Learning

For the reinforcement learning (RL) algorithm used in this thesis, the objective is trying to maximize the following Q-function:

$$Q_\pi(s_t, a_t) = R_t + \gamma \mathbb{E}_{s_{t+1}}[V_\pi(s_{t+1})] \tag{2.6}$$

where $\mathbb{E}_{s_{t+1}}[V_\pi(s_{t+1})]$ is the weighted sum which is defined below:

$$\mathbb{E}_{s_{t+1}}[V_\pi(s_{t+1})] = \sum_{s_{t+1}} P_{t \to t+1} V_\pi(s_{t+1}) \tag{2.7}$$

where $V_\pi(s_t)$ is the value function defined in (2.5).

$\pi(a_t|s_t)$ is the policy to generate the best action a_t at state s_t. To sum up, the goal of learning in RL is to find the optimal policy π^*.

$$\pi^* = \arg\max Q_\pi(s_t, a_t) \tag{2.8}$$

2.4 Deep Neural Network

The deep neural network (DNN) used in this thesis consists of a fully connected network, activation function and regularization.

As shown in Fig. 2.4, the DNN is constructed with neural units and connections between neurons. A fully connected network includes an input layer, hidden layers and an output layer.

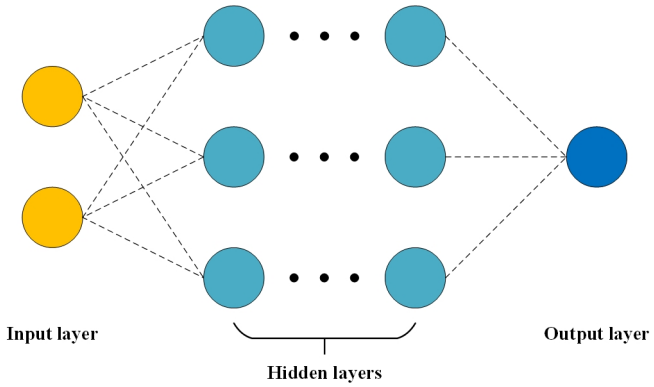


Figure 2.4: The structure of deep neural network.

The activation function is attached to every neural unit, and determines whether this unit is activated or not. If there is no activation function, the neural network is equal to a linear function and the output is a combination of linear calculations of each input. Therefore, activation function is essential to the non-linearity of a neural unit. The activation function can be represented as

$$Y = \text{Activation}(\sum(\text{weight} \times \text{input}) + \text{bias}) \tag{2.9}$$

However, neural network (NN) is easily to be overfitted. In order to eliminate the overfitting, regularization is necessary to be adopted. Regularization has two main directions, one is L_1 regularization and the other is L_2 regularization, which are described as

$$\begin{aligned} L_1 :\ & J_1(\theta) = [y_\theta(x) - y]^2 + \lambda_1 \sum |\theta_i| \\ L_2 :\ & J_2(\theta) = [y_\theta(x) - y]^2 + \lambda_2 \sum \theta_i^2 \end{aligned} \tag{2.10}$$

where $J_{1,2}(\theta)$ stands for the optimization function, θ_i is the parameter of NN and $\lambda_{1,2}$

represents the weight of regularization.

The main difference between L_1 and L_2 regularization is that the L_1 regularization tries to estimate the median value of the data while L_2 regularization tries to estimate the average value of the data.

Chapter 3

UAV Path Planning based on Reinforcement Learning

This chapter studies the topics of path planning of multi-rotor UAV based on reinforcement learning algorithms. UAV is trying to find an optimal path from the start position to destination in the presence of obstacles. Two types of path planning methods are investigated in the subsequence, namely the 3-D real-time path planning and human pedestrian behavior based path planning method.

3.1 3-D Real-Time Q-Learning Path Planning

Most of the existing RL path planning algorithms are focusing on the off-line path planning problem. That is because the state space of the algorithm is generally defined as the absolute location in the environment map. If there are any changes in the environment, the algorithm needs to be re-trained. As a result, there is a limitation that these algorithms cannot deal with real-time path planning problem. Therefore, the path planning problem in dynamic or unknown environment is a quite challenging issue for RL algorithms.

In this section, a new Q-learning algorithm is proposed as the solution of real-time

path planning. Different from the traditional state definition, the proposed algorithm uses a relative location to represent the surrounding environment. The main contribution of the proposed method is a new definition of state space, which can accurately and uniquely represent the surrounding environment no matter how it changes.

3.1.1 Problem Formulation

The real-time path planning problem considered in this study is shown in Fig. 3.1. A UAV is flying from the start position S to the destination T in the presence of obstacles. There are two kinds of obstacles considered, one is fixed obstacle and the other is moving obstacle. It is assumed that the on-board sensors have a detection range, and the state information obstacles within the detection range can be obtained. The environment within detection range is the local environment needs to be considered. The goal of the proposed path planning algorithm is to plan a safe and short path without collision.

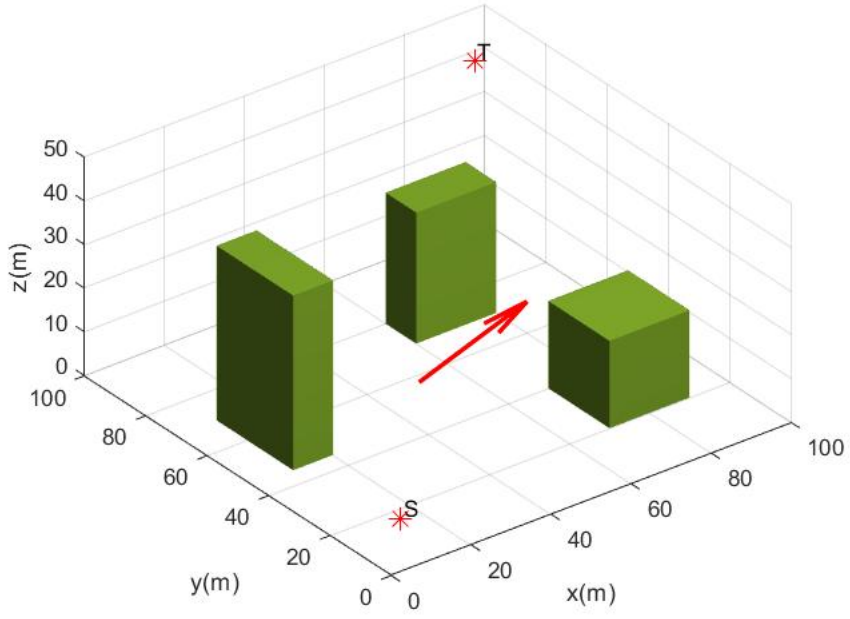


Figure 3.1: The investigated path planning problem.

3.1.2 Algorithm Design

Environment Model and State Space Definition

As stated in the previous section, the traditional state definition of Q-learning algorithm is using the absolute location in the environment. However, the 3-D real-time Q-learning path planning algorithm studied in this section uses the relative location instead of absolute location in order to mapping the environment dynamically. At each time step, the UAV is located in the center of the sensor area. The radius of the sensor range can be evenly divided into 3 sections, the close section, the middle section and the ranged section. Define the length of each section as the side length of a cube. Then, a unit cube is obtained. The local environment within sensor range can be constructed by lining up the unit cubes. As shown in Fig. 3.2, the sensor range consists of $(7 \times 7 \times 7 = 343)$ unit cubes and each unit has a unique relative location.

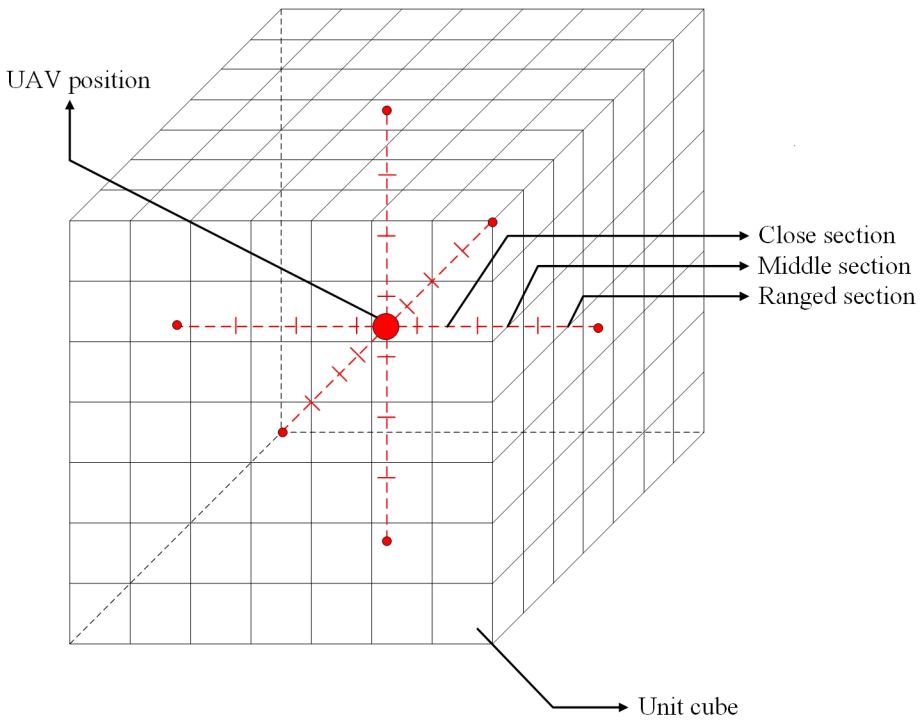


Figure 3.2: The sensor range of UAV.

The state space includes three elements. The first element is the relative location of obstacle. As shown in Fig. 3.2, there are 343 unit cubes in the detection range. The UAV location, which is the center of the detection range, should be subtracted. All 342 unit cubes are labelled as $R_{cube,1}, R_{cube,2}, ..., R_{cube,342}$. If there is no obstacle detected, use label $R_{cube,0}$.

The second element is the heading direction of moving obstacle. Similar to the action space definition, there are 26 moving directions considered. They are combinations of forward, backward, leftward, rightward, upward and downward, which are labelled as $D_{ob,1}, D_{ob,2}, ..., D_{ob,26}$.

The third element is the destination cube, which illustrates the direction of destination. Connect the UAV location and destination location, as shown in Fig, 3.3. The line will inevitably pass through several unit cubes within the sensor range. Where the outermost unit cube, that is, the unit cube closest to the destination, will be regarded as the destination cube. It must be one of $R_{cube,1}, R_{cube,2}, ..., R_{cube,342}$.

Action Space

After the state space has been defined, the possible actions at each state should be addressed. The UAV has 26 moving directions which are labelled as $A_{ac,1}, A_{ac,2}, ..., A_{ac,26}$. They are combinations of six primary directions, which are forward, backward, leftward, rightward, upward and downward.

Specifically, the 26 possible actions can be divided into three parts. First one is up-moving actions, which is shown in Fig. 3.4. It contains 9 directions, which are upward, up-forward, up-leftward, up-rightward, up-backward, up-left-forward, up-right-forward, up-left-backward and up-right-backward.

Second one is the horizontal-moving actions, which is defined in Fig.3.5. The directions are forward, leftward, rightward, backward, left-forward, right-forward, left-backward and

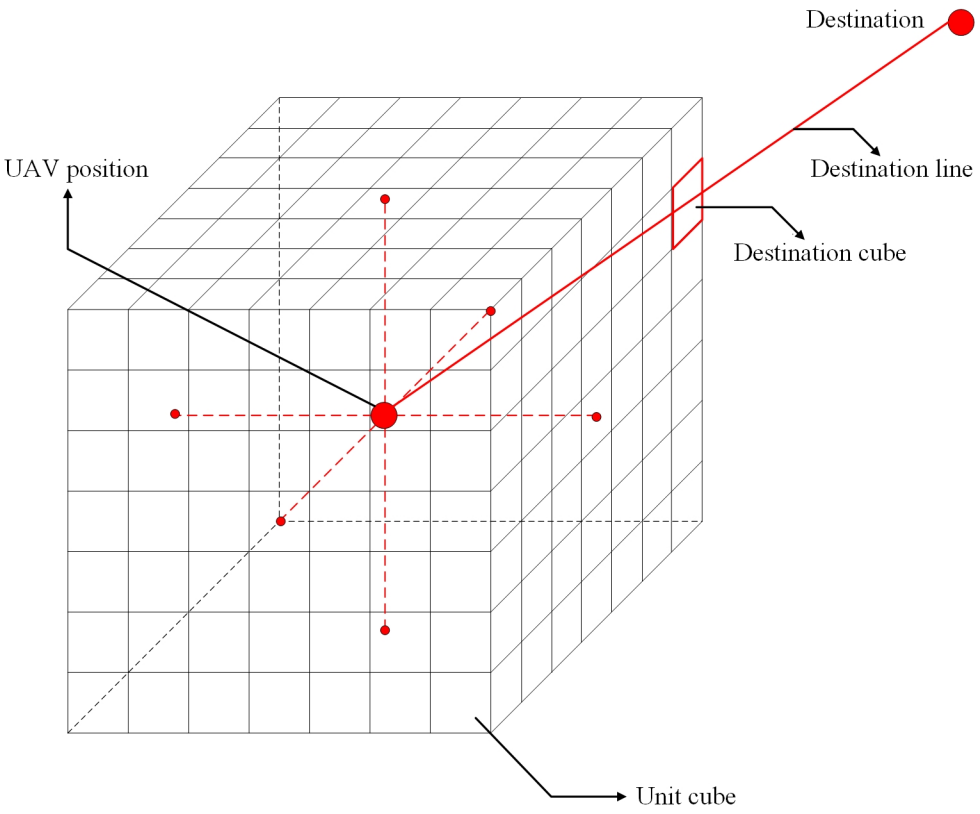


Figure 3.3: The destination cube.

right-backward.

The lats one is the down-moving directions, which is illustrated in Fig .3.6. There are 9 directions in this part, downward, down-forward, down-leftward, down-rightward, down-backward, down-left-forward, down-right-forward, down-left-backward and down-right-backward.

Reward Function

Reward function can be regarded as an evaluation which indicates the performance of the selected action at current state. Generally, the design of reward function leads the direction of learning. Therefore, a good design can accelerate the learning speed and improve the overall performance. The reward function used in the proposed algorithm is designed

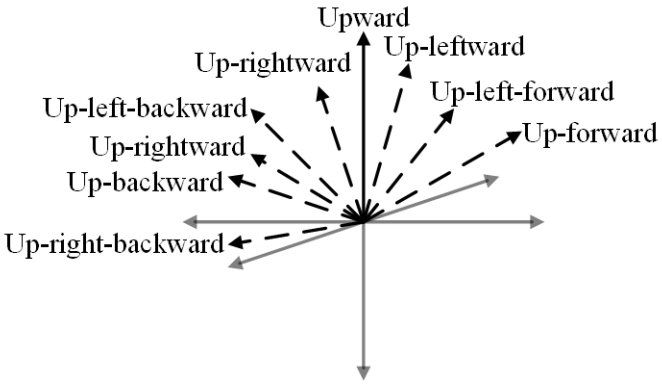


Figure 3.4: The up-moving actions.

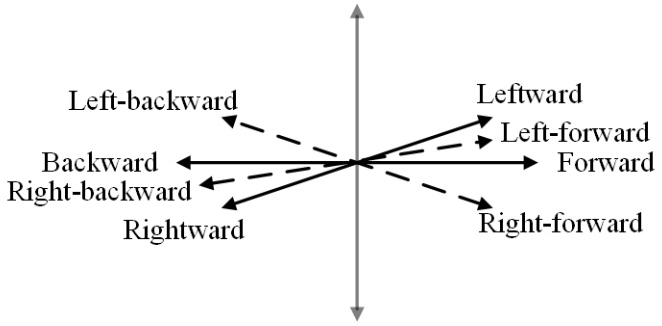


Figure 3.5: The horizontal-moving actions.

as:

$$\begin{aligned} R_{3d}(s,a) =& \beta_1(d_{u\to t}(s-1) - d_{u\to t}(s)) \\ &+ e^{\beta_2(d_{u\to o}(s-1) - d_{u\to o}(s))} \end{aligned} \tag{3.1}$$

where β_1 and β_2 are the weight coefficients. $d_{u\to t}(s)$ is the distance from location at current state s to the destination cube of current state s, which is expressed as follows:

$$d_{u\to t}(s) = \sqrt{(x_u(s) - x_t(s))^2 + (y_u(s) - y_t(s))^2 + (z_u(s) - z_t(s))^2} \tag{3.2}$$

where $P_{uav}(s) = [x_u(s), y_u(s), z_u(s)]$ is the location of UAV at state s. $P_{des-cube}(s) = [x_t(s), y_t(s), z_t(s)]$ is the location of UAV at state s. $d_{u\to t}(s-1)$ refers to the distance from

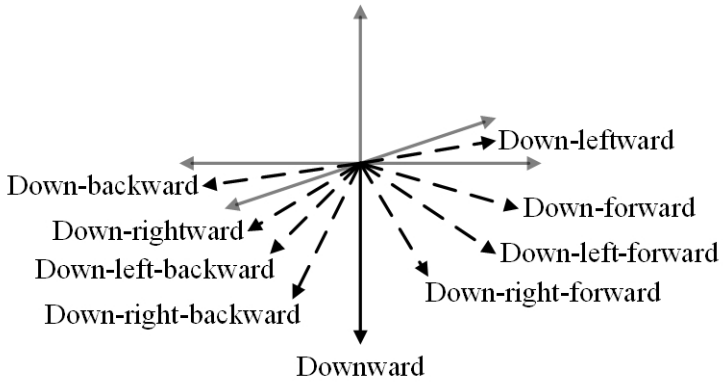


Figure 3.6: The down-moving actions.

previous state $s-1$ to the destination cube of current state s:

$$d_{u\to t}(s-1) = \sqrt{(x_u(s-1)-x_t(s))^2 + (y_u(s-1)-y_t(s))^2 + (z_u(s-1)-z_t(s))^2} \tag{3.3}$$

$d_{u\to o}(s)$ is the distance between detected obstacle and the UAV.

$$d_{u\to o}(s) = \sqrt{(x_u(s)-x_o(s))^2 + (y_u(s)-y_o(s))^2 + (z_u(s)-z_o(s))^2} \tag{3.4}$$

where $P_{obstacle}(s) = [x_o(s), y_o(s), z_o(s)]$ is the location of obstacle at state s.

Updating Q-table

The initial value of Q-table is set to be zero. With the training phase, Q-table will be full-filled with Q-values updated by the following equation:

$$Q_{3d}(s,a) = R_{3d}(s,a) + \gamma \text{Max}(Q_{3d}(s_{t+1}, a_{t+1})) \tag{3.5}$$

3.1.3 Algorithm Training and Executing

The training process will be realized by exploring the map and then reach the destination. Each success trial will be considered as an episode. The structure of the proposed

Q-learning training process is shown in Fig. 3.7.

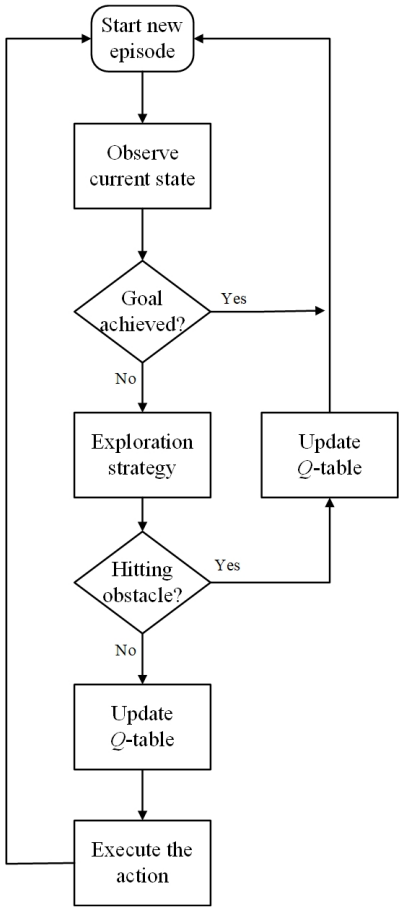


Figure 3.7: The flowchart of training the proposed Q-learning algorithm.

After the proposed Q-learning algorithm has been well trained, the collision-free path can be generated by the updated Q-table. Specific steps are shown in Algorithm 1.

3.1.4 Simulation Results

Success Rate

Success rate or miss rate is considered to be an important evaluation factor of the training effect in learning algorithm design. During training phase, if the UAV hits obstacle or the UAV cannot reach the terminal point within the maximum iterations, this episode is considered to be unsuccessful. Generally, for all the learning algorithm, it is expected

Algorithm 1: 3-D path planning algorithm.

Initialization

Starting state $S_{3d,start}$ and destination state T_{3d}

The well trained Q-table Q_{3d}

Planning the path

Define the UAV current state: $S_{3d,current} = S_{3d,start}$

While ($S_{3d,current} \neq T_{3d}$)

If(There is only one obstacle or no obstacle detected)

Find the best action at current state using Q_{3d}

$Q(S_{3d,current}, a_{best})$ = Max[$Q_{3d}(S_{3d,current}, a)$]

Execute the best action and obtain the next state

End If

If(There is more than one obstacle in sensor range)

Label the obstacles from OB_1 to OB_n (n is the number of obstacles)

Generate a new composite Q-table $Q_{3d,com}$

For each possible action a_i at current state $S_{3d,current}$,

$Q_{3d,com}(S_{3d,current}, a_i) = \sum_{j=1}^{n} Q_{3d}(S_{3d,current}, a_i)(OB_j)$

Find the max value of $Q_{3d,com}$,the corresponding action a_i is the best action

Execute the best action and obtain the next state

End If

Define the current state as the executed next state

End While

Table 3.1: Training effect on successful rate

Training episodes	Successful episodes	Successful rate
100	65	65%
200	142	71%
300	231	77%
400	325	81.3%
500	422	84.4%
600	518	86.3%
700	616	88%
800	715	89.4%

that with more training episodes, the performance will be better and the successful rate is greater. The successful rate of the training process is sampled every 100 episodes. During the training phase, the number of successful episodes is counted and then successful rate can be calculated. The training effect on successful rate is presented in Table 3.1 and Fig. 3.8. In Table 3.1, the successful rate increases as the episodes goes up. When the training just started, there is only 100 training episodes, the successful rate is 65%. After that, the algorithm has been trained hundreds of times until 800 training episodes was reached, where the successful rate is increased to almost 90%, which is considered to be a high successful rate.

Fig. 3.8 shows the successful rate of every 100 episodes. The successful rate increases fast at the beginning and maintains in a high level after 500 episodes.

Performance Testing in Simulation

Since the investigated UAV is a multi-rotor UAV, the operating area of it is small and the altitude is low. The simulation test environment in this section has $100(m)$ width, $100(m)$ length and $50(m)$ height. The detection range is 10 meters. S and T denote start point and the terminal point.

The following three figures show the performance in three different scenarios with both fixed and moving obstacles. In Fig. 3.9, the environment is the least complex. The planned

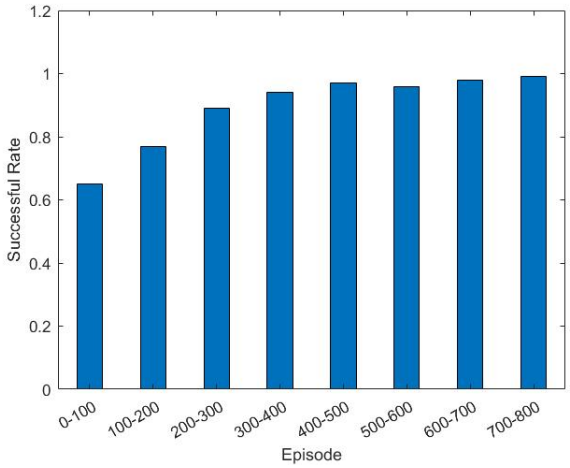


Figure 3.8: The successful rate of every 100 episodes.

path can effectively avoid obstacles and reach the destination. More moving obstacles are considered in Fig. 3.10, the proposed algorithm finds a safe path when passing through all the obstacles. In Fig. 3.11, the environment is obviously more complicated, the algorithm follows the strategy of staying away from all obstacles. Although the total length is longer, it is guaranteed a collision-free path.

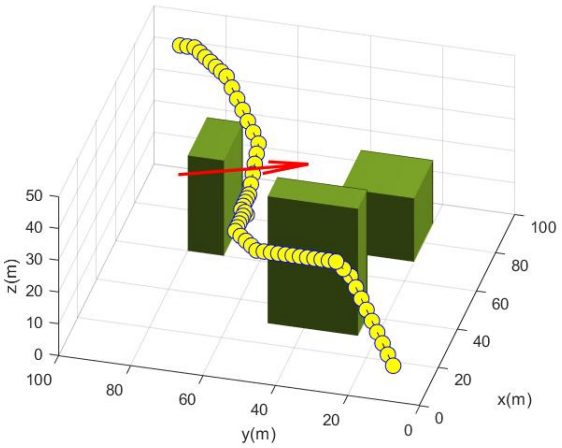


Figure 3.9: The testing scenario 1.

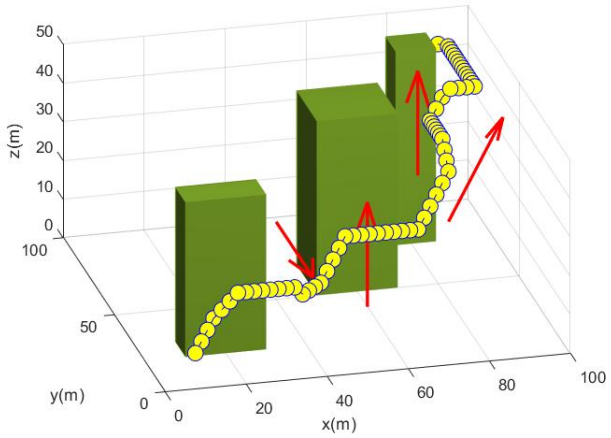


Figure 3.10: The testing scenario 2.

3.2 3-D Real-Time Path Planning based on Human Pedestrian Behavior

The path planning algorithm proposed in this paper is inspired by the human pedestrian behavior. It can be observed that pedestrians seldom collide each other, even in crowded places, such as metro stations and famous tourist sites. If the behavior of pedestrians can be applied to uncrewed robots (UGV and UAV), the obstacle avoidance ability of robots will be greatly improved.

Let us consider how a pedestrian tries to avoid collision in a crowded place. First, pedestrian will focus on the surrounding environment and take less care about the environment which is far away until it comes closer. Second, pedestrian will never be too close to another pedestrian, instead, he or she will keep distance from other pedestrians in advance. Third, for fixed obstacles, such as walls and benches, pedestrians will walk as close as possible to these objects. Because doing like this can reduce the collision risk from the side of fixed objects. This is a wise and safe strategy. Fourth, if the walking directions of two pedestrians will not cross, then these two pedestrians can be very close since there is no risk of collision. For example, two friends are walking along the street, they can be

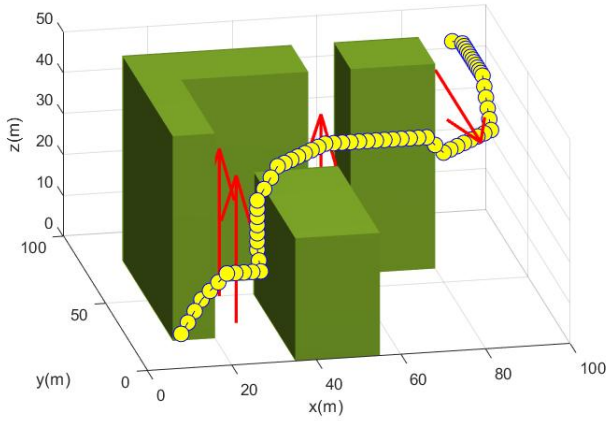


Figure 3.11: The testing scenario 3.

shoulder to shoulder. Therefore, moving direction is a very important factor in pedestrian behavior. In fact, researchers have noticed the behavior of pedestrians as early as the 1990s. Helbing [125] investigates the pedestrian dynamics and established a mathematics model of pedestrian. Inspired by the above ideas, the specific steps of the proposed path planning method is described below.

It should be noted that the state and action spaces definition remain the same with the 3-D real-time Q-learning path planning algorithm.

3.2.1 Reward Function

Reward function can be considered as the most significant component in a Q-learning algorithm. Because it gives feedback for the action and leads the learning direction of the algorithm. In other words, the reward function indicates how good or how bad an action is. A good design of reward function can result in a fast-learning speed and an outstanding overall performance.

In this study, the reward function is inspired by human pedestrian behavior. Due to [125], the pedestrian behavior can be formulated as a social force model. The model can

be simplified as follows:

$$F(t) = f^D(t) + f_{\alpha\beta}(t) + f_{\alpha B}(t) \tag{3.6}$$

where $F(t)$ is the total force at time instant t. There are 3 main factors that influences the motion of a pedestrian. The first one is destination factor $f^D(t)$, pedestrian is always willing to reach destination as comfortable as possible. Therefore, they usually choose the shortest way if there is no obstacles.

$$f_D(t) = \frac{\vec{e_\alpha}(t) - \vec{d_\alpha}(t)}{\tau(t)} \tag{3.7}$$

where $\vec{e_\alpha}(t)$ is the destination direction, $\vec{d_\alpha}(t)$ refers to the actual moving direction of pedestrian α and $\tau(t)$ is a parameter related to the moving speed.

The second factor is caused by other pedestrians $f_{\alpha\beta}(t)$, a walking person α will keep a certain safe distance from other pedestrians β and this distance depends on the moving direction and speed of other pedestrians.

$$f_{\alpha\beta} = V_{\alpha\beta}(b) \tag{3.8}$$

where $V_{\alpha\beta}(b)$ is a repulsive potential field with time-varying parameter b.

The third one is stationary object factor $f_{\alpha B}$, a pedestrian will naturally keep a certain distance from fixed obstacles to avoid collisions. If the fixed obstacle is quite close, the force will be considerable strong and if the obstacle is far away, the force will be quite weak.

$$f_{\alpha B} = U_{\alpha B} \tag{3.9}$$

where $U_{\alpha B}$ is a repulsive and monotonic decreasing potential field.

Besides the above three main factors, in the algorithm design, bonus and penalty strategies also needs to be considered in the reward function. Once the goal is reached, a positive reward value will be given to reinforce good actions. However, if a collision occurs, a huge negative value will be assigned. Therefore, the reward function R_{path} in this paper consists of five terms in total.

$$R_{path} = r_{attract} + r_{moving} + r_{fixed} + r_{bonus} + r_{penalty} \tag{3.10}$$

The first term $r_{attract}$ is an evaluation of the current flying direction. $r_{attract}$ will receive a positive value while UAV is flying closer to the destination. And a negative value is assigned if the flying direction is opposite to the destination direction.

$$r_{attract} = \omega_1 \cos \gamma_{c \to d} \tag{3.11}$$

where ω_1 is the weight parameter and $\gamma_{c \to d}$ stands for the angle between current moving direction and the destination direction.

The second term r_{moving} is defined below:

$$r_{moving} = -\omega_2 \left(\frac{1}{d_{t,c \to o}} + \frac{1}{3 \cdot d_{t+1,c \to o}} + \frac{1}{9 \cdot d_{t+2,c \to o}}\right) \tag{3.12}$$

where ω_2 is the weight parameter. $d_{t,c \to o}$ represents the distance from UAV to moving obstacle at current time instant t and $d_{t+1,c \to o}$ is the distance at next time instant $t+1$ if UAV and obstacle continues their movement along the current direction. The design of r_{moving} is imitating the pedestrian behavior, a pedestrian always make predictions to avoid obstacles in advance. Therefore, r_{moving} not only considers the current situation, but also takes the future risks into account.

Although UAV needs to avoid both fixed and moving obstacles, the reward functions,

r_{fixed} and r_{moving}, are quite different. The fixed obstacle is not necessarily to be considered if the UAV is far from it. An activation range is applied in the r_{fixed} design. Due to the distance definition, the activation range contains close zone, middle zone and ranged zone. If the fixed obstacle is located within the activation range, UAV will receive negative feedback. While the fixed obstacle is located beyond the activation range, a small positive value is rewarded to the UAV. Therefore, the reward function is defined as:

$$r_{fixed} = \begin{cases} -\omega_3(\ln d_{t,c\to f} + \ln d_{t+1,c\to f}), & \text{Obstacle located in the activation range.} \\ 1, & \text{Otherwise.} \end{cases} \tag{3.13}$$

where ω_3 is the weight parameter and $d_{t,c\to f}$ stands for the distance between UAV and fixed obstacle at current time instant t.

The last two term r_{bonus} and $r_{penalty}$ are designed as follows:

$$r_{bonus} = \begin{cases} 100, & \text{If UAV reaches destination.} \\ 0, & \text{Otherwise.} \end{cases} \tag{3.14}$$

$$r_{penalty} = \begin{cases} -1000, & \text{If UAV collides with an obstacle.} \\ 0, & \text{Otherwise.} \end{cases} \tag{3.15}$$

3.2.2 Q-table Updating and Algorithm Training

Initially, the values in Q-table will be set to zero. With the exploration and exploitation in the training phase, the values in Q-table will be updated by the following equation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha_{learning} R_{path,t} + \gamma \text{Max}(Q(s_{t+1}, a_{all})) \tag{3.16}$$

where $\alpha_{learning}$ is the learning rate and γ refers to the discount rate. a_{all} represents all the actions at a state. $\text{Max}(Q(s_{t+1}, a_{all}))$ stands for the maximum Q-value among all the

possible actions at the next time instant.

In order to ensure the diversity of training data, the obstacle position and speed are randomly distributed in each episode. A training episode will end and start a new episode when the following conditions occurred. First, the UAV reaches destination. Second, the UAV collides with obstacles. Third, the training steps exceed the maximum limit. The design of maximum limit aims at preventing internal dead loops, which means the UAV keeps going back and forth along a certain loop. If this condition occurs, the UAV is unable to reach the destination and trapped in a local optimal solution.

The algorithm train phase will be divided into three training steps: phase 1, only fixed obstacles are considered. phase 2, only moving obstacles are considered. phase 3, both fixed and moving obstacles are considered. The mechanism of the first two steps aims at reinforcing the training effect and the last step is to guarantee a good performance in a comprehensive environment.

3.2.3 Planning the Path

After the training phase, the collision-free path can be generated with the help of the well-trained algorithm. As stated in the section 4.3, the state definition only considers one obstacle within the detection range. If there are more than one obstacles, the action with the highest $Q_{total}(s,a)$ will be selected. The $Q_{total}(s,a)$ is the sum of Q-values of different obstacle cases.

$$Q_{total}(s,a) = Q_{ob_1}(s,a) + Q_{ob_2}(s,a) + \cdots + Q_{ob_n}(s,a), \{ob_1, ob_2, \cdots, ob_n\} \in OB \quad (3.17)$$

where OB is the collection of obstacles within the detection range.

The pseudo-code of the proposed path planning algorithm is shown in Algorithm 2.

Algorithm 2: Human Pedestrian Path Planning Algorithm

Initialization:
UAV starting position and destination
The unknown environment and trained Q-table
Real-time planning:
while Destination is not reached
  Observe the surrounding environment within the detection range
  Obtain the state information s_{cur} and the number of obstacles n
  for action $i = [1, 26]$ **do**
    $Q_{total}(s_{cur}, a_i) = \sum_{j=1}^{n} Q_{ob_j}(s_{cur}, a_i)$
  end for
  Find the best action a_{max} who has the maximum value of $Q_{total}(s_{cur}, a_{max})$
  Execute the selected action a_{max}
End While

3.2.4 Simulation and Experiment Results

Training Results and Analysis

The miss rate and average reward are two key factors of evaluating training effect. Miss rate, as the name indicates, is the ratio of failed and total training episodes. A failed episode means the UAV collides with an obstacle or the total training steps during one episode exceeded the maximum number. Due to the stochastic exploration strategy, UAV will inevitably collide with obstacles as the number of training episode grows. And the UAV may be trapped in local loops since the algorithm is not well trained. As shown in Table 2, the miss rate is a little high at the beginning. As the number of training episode increases, the miss rate drops rapidly and finally converges to a small value.

Simulation Results

Since the investigated UAV is a multi-rotor UAV, the operating area of it is small and the altitude is low. The simulation test environment in this section has $100(m)$ width, $100(m)$ length and $50(m)$ height. The detection range is 10 meters. Fig. 3.12, Fig. 3.13 and Fig. 3.14 illustrate the performance in three different scenarios with both fixed and moving

	Total episode	Failed episode	Miss rate(%)
Fixed	First1×10^4	3073	15.4%
	Second 1×10^4	1092	5.5%
	Third 1×10^4	556	2.8%
Moving	First 1×10^4	2285	11.4%
	Second 1×10^4	1170	5.8%
	Third 1×10^4	464	2.3%
Mixed	First 1×10^4	651	3.3%
	Second 1×10^4	318	1.6%
	Third 1×10^4	107	0.5%

Table 3.2: The miss rate during different training phases.

obstacles.

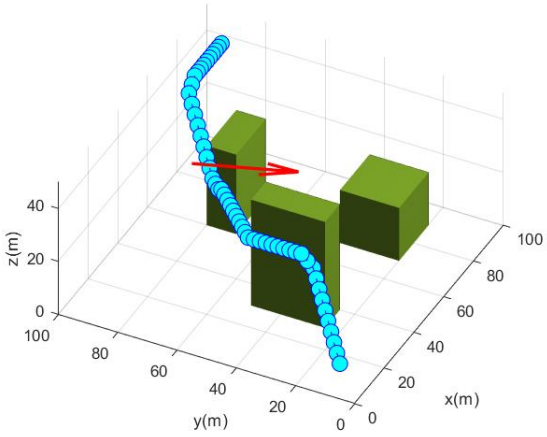


Figure 3.12: Simulation testing scenario 1.

In Fig. 3.12, the environment is less complicated compared to the other two scenarios. The UAV can effectively avoid all obstacles and reach the destination. In the simulation environment of Fig. 3.13, more moving obstacles are considered. The planned path does not collide moving obstacle, nor keep far away from them. It finds a safe and short path when passing through obstacles. From Fig. 3.14, the simulation environment of scenario 3 is obviously more complicated, the moving obstacles are closer to the fixed ones, which increases the difficulty to plan a safe and short path. With the mechanism of imitating

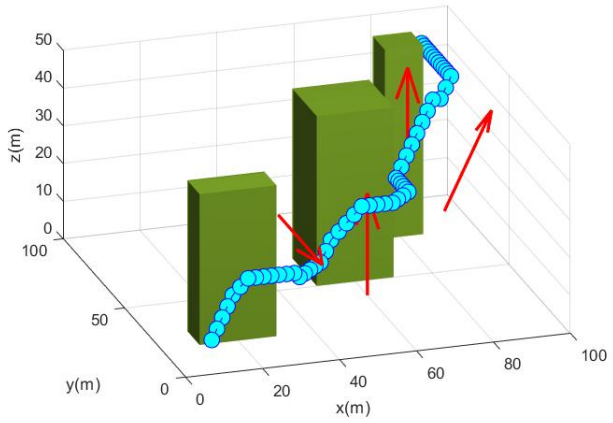


Figure 3.13: Simulation testing scenario 2.

pedestrian behavior, the UAV does not avoid the multi-obstacle zone flies along the edge of fixed obstacles and successfully pass through the multi-obstacle zone without collision. In summary, the path generated by the proposed algorithm leads the UAV to destination with a safe and shortest trajectory.

Comparison of Simulation Results

The comparison of the 3-D real-time Q-learning path planning algorithm and the human pedestrian behavior based real-time path planning algorithm are illustrated in the following three figures. Although both two algorithms can reach the destination without collision, however, the path length is not the same. As shown in Fig. 3.15, Fig. 3.16, and Fig. 3.17, yellow line stands for the human pedestrian behavior-based algorithm and the cyan line refers to the 3-D real-time Q-learning path planning algorithm. The length of two paths is shown in Table 3.2.4. It obviously that the pedestrian behavior-based algorithm path length is always shorter than the one of Q-learning path planning algorithm. In addition, from the errors between two paths, the more complicated the environment is, the bigger the errors is.

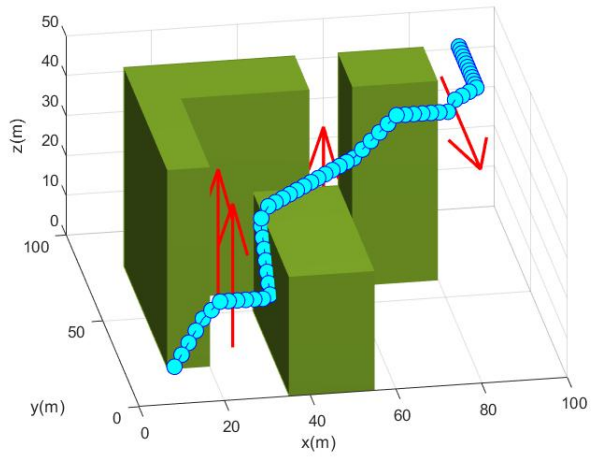


Figure 3.14: Simulation testing scenario 3.

	Q-learning (m)	Pedestrian behavior (m)	Errors (m)
Scenario 1	146.2	140.7	5.5
Scenario 2	158.4	148.2	10.2
Scenario 3	165.1	150.8	14.3

Table 3.3: The comparison of path lengths between two path planning algorithms.

Real Flight Tests

The real flight test is carried out on the QDrone and Qcar platform in the NAVL Lab at Concordia University. Testing video has been uploaded to the YouTube, which is available at *https://www.youtube.com/watch?v=FGp2LhPuVk0&t=1s*. The following figures (Fig. 35) illustrate the effectiveness of the proposed algorithm in real flight test.

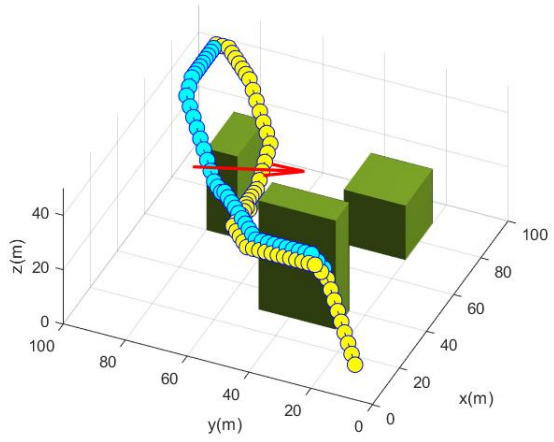


Figure 3.15: Comparison of two path planning algorithm in simulation testing scenario 1.

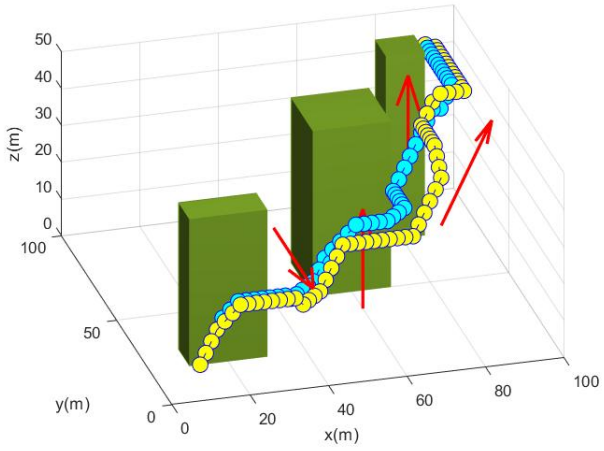


Figure 3.16: Comparison of two path planning algorithm in simulation testing scenario 2.

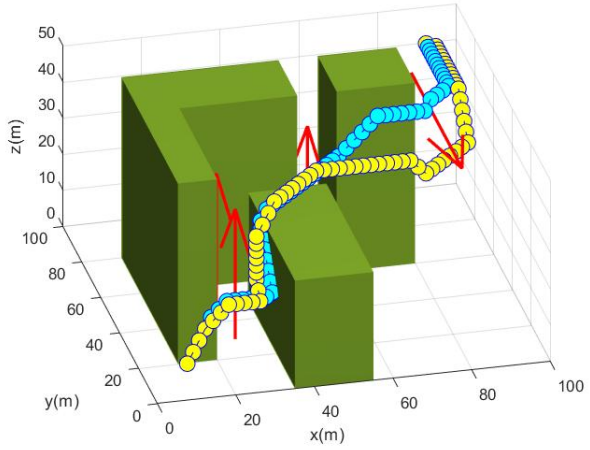


Figure 3.17: Comparison of two path planning algorithm in simulation testing scenario 3.

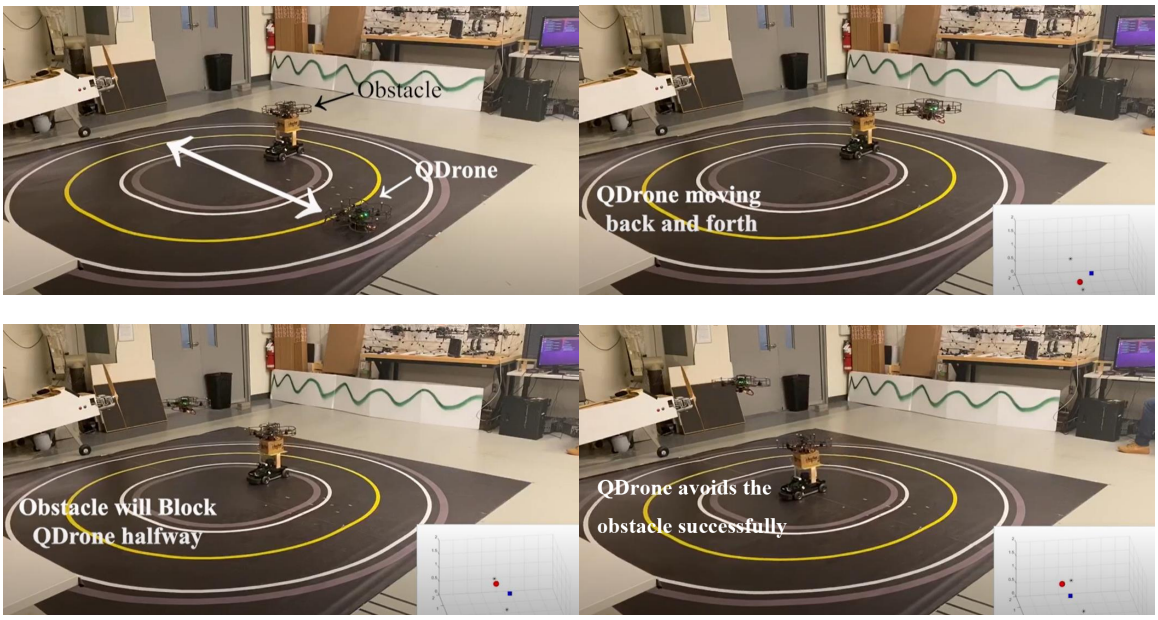


Figure 3.18: The proposed algorithm is validated in real flight test. (Video link: https://www.youtube.com/watch?v=FGp2LhPuVk0&t=1s)

Chapter 4

Motion Control of UAV

In this chapter, the investigation of using DRL in UAV motion control will be conducted in three specific directions: path following control, formation control and auto-landing control.

4.1 Path Following Control of Fixed-Wing UAV

This section proposes an improved deep deterministic policy gradient (DDPG) algorithm for path following control problem of fixed-wing UAV. A specific reward function is designed for minimizing the cross-track error of the path following problem. In the training phase, a double experience replay buffer (DERB) is used to increase the learning efficiency and accelerate the convergence speed. Simulation results are carried out to show the effectiveness of the proposed DERB-DDPG method.

In DDPG algorithm, the experience replay buffer is an important component mechanism because it can break the data correlation and accelerate the convergence speed in the training phase [22]. Generally, the replay buffer is used to store the experiences and a batch of samples will be extracted from replay buffer during training phase. In order to accelerate the training speed, some researchers proposed a prioritized experience replay

buffer (PERB) [126], which guarantees the samples with larger values to be selected more frequently. However, repeating a limited number of experiences may be lack of data diversity and easily lead the algorithm converging to a local optimization [127]. Based on the ideas of prioritized experience replay buffer, a double experience replay buffer (DERB) is proposed in this section, which successfully inherits the advantage of PERB but avoids the disadvantage of it. As the name indicates, DERB has two experience replay buffers. One is working as ordinary replay buffer and the other only stores the experiences with high temporal difference (TD) values. In the training process, the two replay buffers are working individually. Part of the samples are extracted from the ordinary one and the rest are from the one with high TD values. Another mechanism is that the proportion of two replay buffers is dynamically changing. If the training process is just started, more samples will be selected from the one with high TD values, which aims at speeding up training. When the algorithm has been trained and already has a good performance, more samples will be extracted from the ordinary one in order to guarantee the diversity of training data.

The main contributions of the proposed method include the following aspects: 1) a new framework of DDPG for UAV path following is established; 2) a new reward function is designed for UAV path following problem; 3) a new training technique, called the DERB, is proposed to improve the learning ability.

4.1.1 Problem Formulation

UAV Modelling

Modelling the UAV in path following problem requires two coordinate systems: one is earth-surface inertial reference frame $S_g - O_g x_g y_g z_g$ and the other is aircraft-body reference frame $S_b - O_b x_b y_b z_b$. For earth-fixed reference frame, the velocity of UAV can be defined as $[\dot{x}_g,\ \dot{y}_g,\ -\dot{h}]^T$. While in body-fixed reference frame, the velocity of UAV is $[u,\ v,\ w]^T$.

The translational kinematic equations of UAV could be defined as:

$$\begin{cases} \dot{x}_g = ucos\theta cos\psi + v(sin\phi sin\theta cos\psi - cos\phi sin\psi) + \\ \qquad w(sin\phi sin\psi + cos\psi sin\theta cos\psi) \\ \dot{y}_g = ucos\theta sin\psi + v(sin\phi sin\theta sin\psi + cos\phi cos\psi) + \\ \qquad w(-sin\phi cos\psi + cos\phi sin\theta sin\psi) \\ \dot{h} = usin\theta - vsin\phi cos\theta - wcos\phi cos\theta \end{cases} \tag{4.1}$$

where ϕ, ψ, θ, are the roll, yaw and pitch angle, respectively.

The angular kinematic equations could be expressed as:

$$\begin{cases} \dot{\phi} = p + (rcos\phi + qsin\phi)tan\theta \\ \dot{\theta} = qcos\phi - rsin\phi \\ \dot{\psi} = 1/cos\theta(rcos\phi + qsin\phi) \end{cases} \tag{4.2}$$

where p, q, r are the roll, pitch, and yaw rate, respectively.

It is assumed that the UAV has a constant speed and flies at a constant altitude. Therefore, the kinematic equations could be simplified as:

$$\begin{aligned} \dot{x}_g &= ucos\psi - vsin\psi \\ \dot{y}_g &= usin\psi + vcos\psi \\ \dot{\psi} &= r \end{aligned} \tag{4.3}$$

From (4.3), it is obvious that the yaw angle ψ is the key of motion control. Therefore, the goal of the path following algorithm is to generate a proper value of yaw angle ψ_d, which can guarantee the UAV fly on the desired path. Due to the physical limitations of a fixed-wing UAV, the yaw angle is restricted to the range from -30° to 30°.

In actual flight, the general idea for solving UAV path following problem is placing a

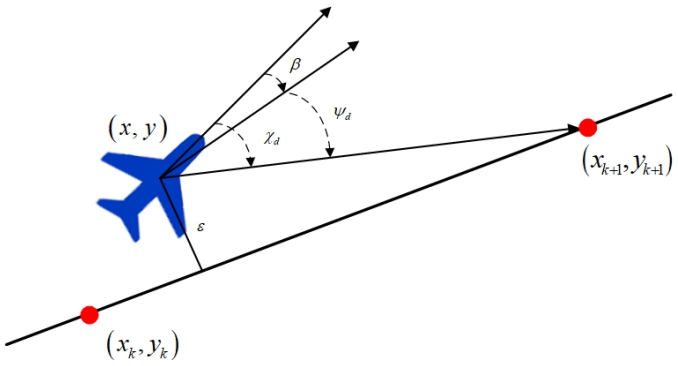


Figure 4.1: The geometry of path following problem

series of waypoints on the planned path [128]. After that, the guidance algorithm steers the UAV to follow these waypoints sequentially. The geometry of the path following problem is shown in Fig. 4.1. A 2-D path P_{path} is considered as the desired path, where a set of successive waypoints (x_k, y_k) is placed on the target path. ϵ stands for the cross-track error, which is defined as the shortest distance between the UAV and desired path. Course angle χ_d is the sum of side-slip angle β and desired yaw angle ψ_d.

$$\chi_d = \psi_d + \beta \tag{4.4}$$

where side-slip angle β refers to the influence of negative effects including internal uncertainties and external disturbances. The side-slip angle is generally considered to be unknown in the research of path following problem. In other words, path following controller should be designed to compensate for the negative effects of side-slip angle.

State and Action Specification

In the application of RL algorithm, the formulation of the problem is decisive for the convergence of algorithm. The state space of the proposed method S_{pf} is defined as:

$$S_{pf} = \{\epsilon, \chi_d, D_n, B_{pf}\} \tag{4.5}$$

where ϵ is cross track error, refers to the distance between UAV and path tangential line, which is shown in Fig. 4.1. χ_d stands for the desired course angle. D_n refers to the distance between UAV current location and the next waypoint. The last one is B_{pf}, a binary flag variable. If the UAV position has been on the desired path and UAV flying direction is pointing to the next waypoint ($\epsilon = 0$ and $\chi_d = 0$), that is perfect tracking, the value of flag variable $B_{pf} = 1$, otherwise $B_{pf} = 0$.

The action space of the proposed method A_{pf} is given by:

$$A_{pf} = \{\psi_d\} \tag{4.6}$$

where ψ_d is the desired yaw angle.

When algorithm is learning directly from the data of physical observations (for example, distances, velocities and angles), the ranges of these variables may differ a lot. This greatly decreases the effectiveness and efficiency of the algorithm. One effective approach is to scale the variable values, make them in the same range. Therefore, the state and action spaces are designed in 4 dimensions and 1 dimension respectively with normalized variables scaling from -1 to $+1$.

Reward Shaping

Generally, with respect to the RL algorithm, the reward function is considered as the most important component. A proper design of the function can result in a good performance and fast convergence speed. Therefore, the reward function $R_{pf}(t)(s_t, a_t)$ is defined by (4.7) and (4.8):

$$\begin{aligned} shaping_t = &- \sqrt{\psi_{d_t}^2 + \psi_{d_{t-1}}^2} - \sqrt{\chi_{d_t}^2 + \chi_{d_{t-1}}^2} \\ &- 10\sqrt{\epsilon_t^2 + \epsilon_{t-1}^2} + 10BD_n \end{aligned} \tag{4.7}$$

$$R_{pf}(t) = shaping_t - shaping_{t-1} \tag{4.8}$$

It can be concluded from (4.7) and (4.8) that the reward function explicitly emphasizes the importance of minimizing the cross-track error. The requirement of smooth flight is also considered in reward function. Following this idea, the UAV is able to learn minimizing the distance between desired path and generated smooth yaw angle references, which results in a less aggressive motion.

4.1.2 DERB-DDPG for UAV Path Following Control

Actor-Critic Framework

DDPG is known as a policy-based RL method which aims at solving problem with continuous space in states and actions [129]. Facing the continuous nature of actions and states, DDPG algorithm utilizes two neural networks in the actor-critic framework in order to estimate the deterministic policy and the Q-function [27].

The DDPG framework for path following control problem is shown in Fig. 4.2. The desired yaw angle ψ_d is provided by DDPG algorithm, then the control signals $[\delta_a, \delta_r]$ are generated by PID controller. With the control inputs, the UAV current state information $[\beta, p, r, \psi]$ will be given to the path following environment and PID controller. After that, the UAV current state information will be transferred to DDPG algorithm state $[\epsilon, \chi_d, D_n, B]$ for further calculation.

In the DDPG mechanism, two networks, actor and critic, are included. The actor network generates the action based on state information. While, the critic network evaluates the executed action, then updates the actor network. During the training phase, experiences are stored in the replay buffer. When updating the network, samples will be extracted from the replay buffer to break the inter-correlations between experiences and make the training

phase easier to converge. During the execution phase, the critic network and replay buffer will not be involved. The actor network works individually to calculate the action based on observed state.

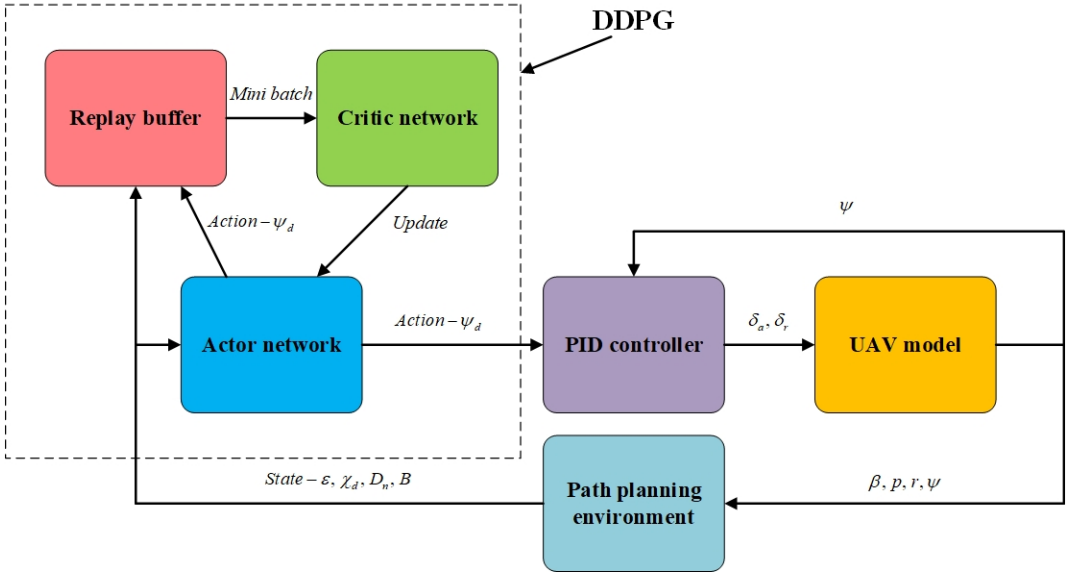


Figure 4.2: The framework of DDPG for path following control

For the actor network policy π_{pf}, the value function $V^{\pi_{pf}}$ regarding to accumulated discount reward $R_{pf}(t)$ is defined below

$$V_{pf}^{\pi}(s_t) = \mathbb{E}[R_{of}(s)|s_t, a_t = \pi_{pf}(s_t)] \tag{4.9}$$

For every action-state pair, the value function could be expressed by the recursive relationship, which is known as the Bellman equation as following

$$Q_{pf}^{\pi}(s_t, a_t) = \mathbb{E}[R_{pf}(s_t, a_t) + \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) Q_{pf}^{\pi_{pf}}(s_{t+1}, a_{t+1})] \tag{4.10}$$

The optimal policy π^* is the one with maximum value of the function V_{pf}^{π} (or Q_{pf}^{π}). The optimal policy π^* can be described as a function μ since it is a deterministic policy. The Q-function (4.10) will be learned using the reactions from environment using a stochastic

policy μ', where μ' is the exploration policy.

$$Q^{\mu}_{pf}(s_t, a_t) = \mathbb{E}[R_{pf}(s_t, a_t) + \gamma Q^{\mu}_{pf}(s_{t+1}, \mu(s_{t+1}))] \tag{4.11}$$

The function approximator, which is parameterized by $\theta^{Q_{pf}}$, is used to approximate the Q-function. The optimization could be obtained by minimizing the loss

$$L(\theta^{Q_{pf}}) = \mathbb{E}[(Q_{pf}(s_t, a_t|\theta^{Q_{pf}}) - y_{pf}(t))^2] \tag{4.12}$$

where $y_{pf}(t) = R_{pf}(s_t, a_t) + \gamma Q_{pf}(s_{t+1}, \mu(s_{t+1})|\theta^{Q_{pf}})$.

The critic Q-function is updated using the Bellman equation as in a standard RL algorithm. The actor network is learned by following and applying the chain rule to the expected return from the start distribution J with respect to the actor parameters.

$$\bigtriangledown_{\theta\mu} J \approx \mathbb{E}[\bigtriangledown_{\theta\mu} Q_{pf}(s_t, \mu(s_t|\theta^{\mu})|\theta^{Q_{pf}})] \tag{4.13}$$

There will be a challenge if using neural network for reinforcement learning: the training data are not identically distributed and might be correlated since they are generated from exploration sequentially. As a result, it is necessary to use mini-batches, which are extracted from replay buffer. During each time episode, the actor and critic networks will be learned by a mini-batch. After that, the tuple $(s_t, a_t, R_{pf}(t), s_{t+1})$ will be added into the experience replay buffer. In order to break the correlation between samples, the replay buffer should be large enough.

The two target networks in this algorithm will be updated using soft rules. Initially, target networks are assigned the same parameter values with the actor and critic networks, $\pi'_{pf}(s|\theta^{\pi'})$ and $Q'_{pf}(s, a|\theta^{Q'})$. Then, the weights of them will be updated by softly tracking

the learned networks

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \tag{4.14}$$

where $\tau \ll 1$. This may slow the learning speed due to the fact that (4.14) will slow down the propagation of value approximation. However, it greatly improves the stability of learning, target networks π'_{pf} and Q'_{pf} will be trained without divergence.

Exploration Law

The exploration in DDPG algorithm is independent from learning. The exploration action $a_{pf}(t)$ is constructed by adding a random noise N_t from an Ornstein-Uhlenbeck (OU) process to the exploration policy $\mu'(s_t)$.

$$a_t = \mu'(s_t) + N_t \tag{4.15}$$

where the exploration policy is $\mu'(s_t) = \mu(s_t|\theta^{\mu}_t)$, the noise is sampled from OU process $N_t \sim OU(\nu, \vartheta, \varsigma)$. Since (4.15) is a random exploration law, the training is easily to fail if $a_{pf}(t)$ is too small or too large. As a result, in order to reduce the failure rate of training, the exploration action $a_{pf}(t)$ should be limited within $[a_{min}, a_{max}]$. The effect of exploration varies due to the selected values of parameters $(\nu, \vartheta, \varsigma)$. For example, a larger value of ς will lead to a broaden range of actions. As a result, the noise N_t is suitable for redesigning which could be adjusted to achieve a fast learning ability.

Double Experience Replay Buffer Design

For each time step, the TD value will be calculated to partially reflect the performance of this action, which is defined as:

$$TD = R_{pf}(s_t, a_t) + \gamma Q'_{pf}(s_{t+1}, \mu(s_{t+1})) - Q_{pf}(s_t, a_t) \tag{4.16}$$

The main idea behind DERB is to repeat the experiences with greater TD values more frequently in order to accelerate the training speed. In the proposed method, two experience replay buffers RB_1 and RB_2 are designed to store the experiences, where RB_1 is the ordinary replay buffer which stores all the experiences and RB_2 is used to store the experiences with high TD values. Obviously, RB_2 has a much smaller size compared to RB_1.

While extracting samples from replay buffer, the RB_1 and RB_2 are working at the same time. The majority of extracted samples are randomly selected from RB_1 and the others are randomly selected from RB_2. When the learning process just started, the experiences with high TD values are apparently better. However, with the training process goes on, the algorithm will evolve and generate better performance. The experiences with higher TD values may not be the best choice anymore. As a result, the percentage of experiences from RB_2 should be reduced. The probability p_{RB} stands for the percentage of RB_2 during the learning process. p_{RB} is defined as follows:

$$p_{RB} = \frac{\epsilon_{avg}}{\epsilon_{con}} \tag{4.17}$$

where ϵ_{avg} denotes the average cross-track error during the current training episode. ϵ_{con} refers to a constant value which aims at balancing the percentage of RB_2.

If the average cross-track error increases, the p_{RB} will also increase. On the contrary, if the average cross-track error is relatively low, the p_{RB} will decrease. Specifically, p_{RB} will be relatively high at the starting period and eventually approaches to 0.

This mechanism is quite important for the convergence of training process because always learning from some successful experiences will lead to a weak robustness and the lack of flexibility. To sum up, the DERB is used at the beginning to speed up the learning process and eventually becomes dispensable when the algorithm approaches convergence.

The specific steps of the proposed DERB-DDPG algorithm is displayed in Algorithm

3.

Algorithm 3: DERB-DDPG algorithm

Step. 1 Initialization
Randomly initialize actor network $\pi_{pf}(s|\theta^{\pi})$ and critic network $Q_{pf}(s,a|\theta^{Q})$ with weights θ^{π} and θ^{Q}
Copy the weights to target network π'_{pf} and Q'_{pf}, $\theta^{\pi'_{pf}} \leftarrow \theta^{\pi_{pf}}$, $\theta^{Q'_{pf}} \leftarrow \theta^{Q_{pf}}$
Initialize replay buffer $\mathbf{DERB}$
Step. 2 Exploration
for episode = $[1, m]$
Initialize the noise process $N \sim OU$ for exploration
for t = $[1, T]$
Generate action $a_{pf}(t) = \mu(s_{t|\theta_{\pi_{pf}}}) + N_t$ based on the current actor policy and exploration noise
Execute action $a_{pf}(t)$, then receive reward $R_{pf}(t)$ and obtain the new state s_{t+1}
Store transition (s_t, a_t, r_t, s_{t+1}) into $\mathbf{DERB}$
Step. 3 Update
Extract a mini-batch of n transitions $(s_i, a_i, R_{pf}(i), s_{i+1})$ from replay buffer $\mathbf{DERB}$
Calculate $y_{pf}(i) = R_{pf}(s_i, a_i) + \gamma Q_{pf}(s_{i+1}, \mu(s_{i+1})|\theta^{Q_{pf}})$
Update critic network by minimizing the loss $L = \frac{1}{n}\sum_i (y_{pf}(i) - Q_{pf}(s_i, a_i|\theta^{Q_{pf}}))^2$
Update the actor policy by using the policy gradient
$\bigtriangledown_{\theta^{\mu}} J \approx \frac{1}{n}\sum_i \bigtriangledown_a Q_{pf}(s, a|\theta^{Q_{pf}})|_{s=s_i, a=\mu(s_i)} \bigtriangledown_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$
Update the target networks
$\theta^{\pi'_{pf}} \leftarrow \tau\theta^{\pi_{pf}} + (1-\tau)\theta^{\pi'_{pf}}$
$\theta^{Q'_{pf}} \leftarrow \tau\theta^{Q_{pf}} + (1-\tau)\theta^{Q'_{pf}}$
end
end

4.1.3 Simulation and Analysis

UAV Model

Since the UAV is assumed to maintain a constant altitude. Therefore, only the lateral motion needs to be considered. The linearized lateral model of fixed-wing UAV can be expressed as:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{4.18}$$

where $x = [\beta,\ p,\ r,\ \psi]^T$, $u = [\delta_r,\ \delta_a]^T$, and δ_r, δ_a are rudder and aileron deflection angles, respectively.

The matrices A, B, C and D are given by:

$$A = \begin{bmatrix} -0.732 & 0.0143 & -0.996 & 0.0706 \\ -893 & -9.059 & 2.044 & 0 \\ 101.673 & 0.0186 & -1.283 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.19}$$

$$B = \begin{bmatrix} 0 & 0.244 \\ 328.653 & 308.498 \\ 47.528 & 102.891 \\ 0 & 0 \end{bmatrix} \tag{4.20}$$

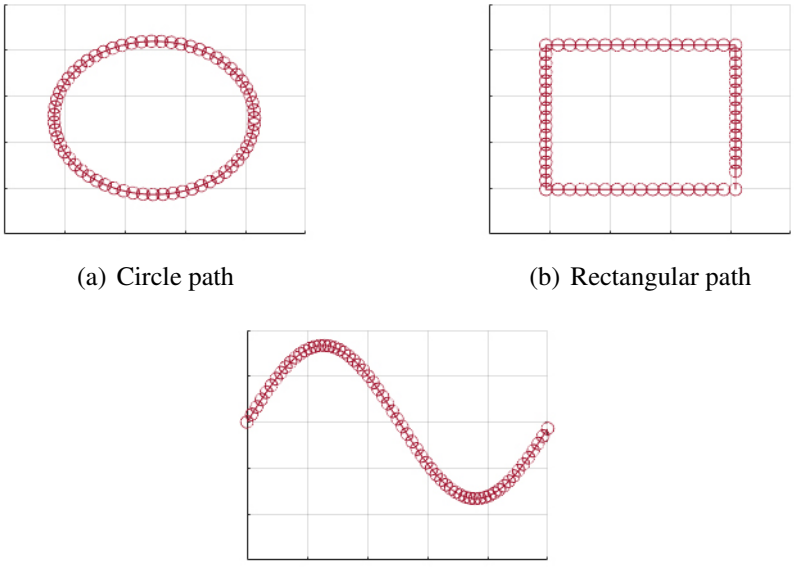
$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{4.21}$$

In the simulations, the PID controllers are applied in the inner loop to achieve the yaw stability. The control signals $u = [\delta_a\ \delta_r]^T$ are given based on ψ_d and ψ, where ψ_d is generated by the proposed DDPG algorithm.

Training Settings

Choosing suitable hyper parameters for the proposed SAC controllers can significantly improve training efficiency. Specifically, the learning rate is $1 \times 10 - 3$, the discount factor is 0.98, the smooth parameter ρ is 3×10^{-3}. The deep neural network has three hidden layers and each layer has 32 units.

During the training phase, there are three kinds of training paths, which are shown in Fig. 4.3. The first one is a circle path, which aims at training the soft turning ability. The second one is a rectangular path, which aims at training sharp turning ability. The third one is a sine or cosine curve path generated by trigonometric functions, which aims at training the comprehensive turning ability. Since the shape of training paths has been determined, in order to guarantee the diversity of the training data, the size and magnitude of training path differ in each training episode.



(a) Circle path

(b) Rectangular path

(c) Sine path

Figure 4.3: Three training paths

For each training episode, it consists of three training missions which stands for the three training paths. Training efficiency is an important measurement, which illustrates the learning ability of algorithm, which could be defined as the ratio between effective episodes and total episodes. The effective episode refers to the episode which UAV has completed all three training missions and reach the terminal of the training path. Due to the exploration law, the UAV may go too far from the reference path and could not reach the terminal. Failure of arrival or stay too far from the reference could not be counted

as effective episode. The learning algorithm reacts to the surrounding environment at a frequency of 10 Hz. The size of mini-batch is 20.

Performance Comparison

To demonstrate the effectiveness and improvement of the proposed DERB-DDPG algorithm, comparisons of three DRL algorithms have been carried out. The first one is deep Q-network (DQN). The second one is the original DDPG algorithm. The third one is the advanced DERB-DDPG algorithm. While the simulation settings, reward function and parameters of these three algorithms remain the same. To measure the performance of the above algorithms, three quantitative evaluations are carried out. They are successful rate, accumulated reward and average cross-track error, respectively.

During the entire training phase, the successful rate of the three methods are shown in Fig. 4.4. From Fig. 4.4, the DERB-DDPG uses 1,500 episodes to achieve a success rate over 80%, while the original DDPG uses 500 more episodes to achieve the same rate. However, the DQN remains low success rate within 2,000 training episodes. This is because the technique of DERB, it accelerates the learning speed to achieve fast convergence.

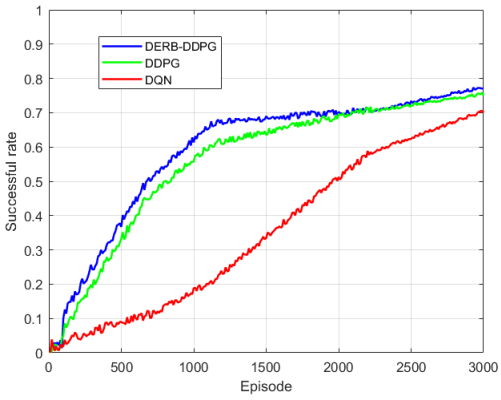


Figure 4.4: Successful rates of DQN, original DDPG and DERB-DDPG (with moving window of 10 episodes).

The accumulated rewards of the three algorithms are displayed in Fig. 4.5. The accumulated reward of all three algorithms goes up as the episode increases, and finally converges to a constant value. Although the DERB-DDPG has a lower value at the first 100 episodes, it converges faster and achieve a higher steady-state value within the least episodes.

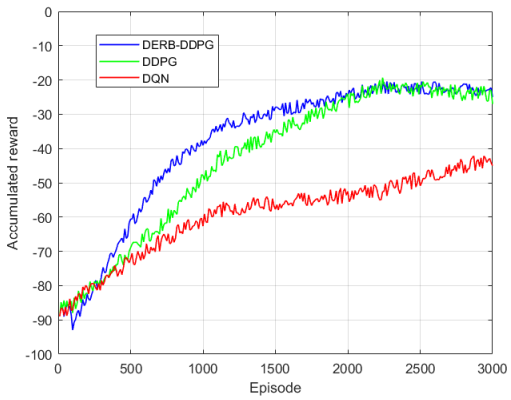


Figure 4.5: Accumulated reward of DQN, original DDPG and DERB-DDPG (with moving window of 10 episodes).

The average cross-track error is illustrated in Fig. 4.6, which is obtained by calculating the average value of the cross-track error at each time step. The cross-track errors of all three algorithms keep decreasing with training episode increasing. Such a phenomenon proves the effectiveness of the reward function. By using the designed reward function, all the algorithms can minimize the cross-track error. It is obviously that the DERB-DDPG has smaller cross-track error than the DQN and original DDPG at all times.

Comprehensively considering Figs. 4.4, 4.5 and 4.6, the DERB-DDPG has a higher successful rate, faster convergence speed and smaller cross-track error than DQN and original DDPG.

Fig. 4.7 depicts the overall performances of the DQN, original DDPG and DERB-DDPG in a specific path following mission. The black line demonstrates the reference curved path, the blue line stands for the proposed DERB-DDPG algorithm, green dotted

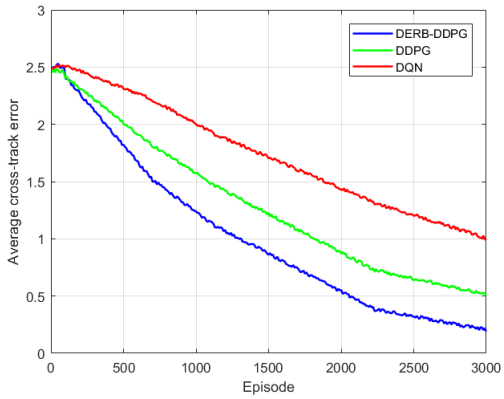


Figure 4.6: Average cross-track error of DQN, original DDPG and DERB-DDPG (with moving window of 10 episodes).

line refers to the original DDPG algorithm and red dotted line represents the DQN algorithm. From Fig. 4.7, it is obvious that the blue line, which illustrates the proposed DERB-DDPG algorithm, is closer to the reference path. Such a phenomenon proves that the proposed DERB-DDPG algorithm not only has good performance in the training paths, but also has strong adaptability when facing new paths.

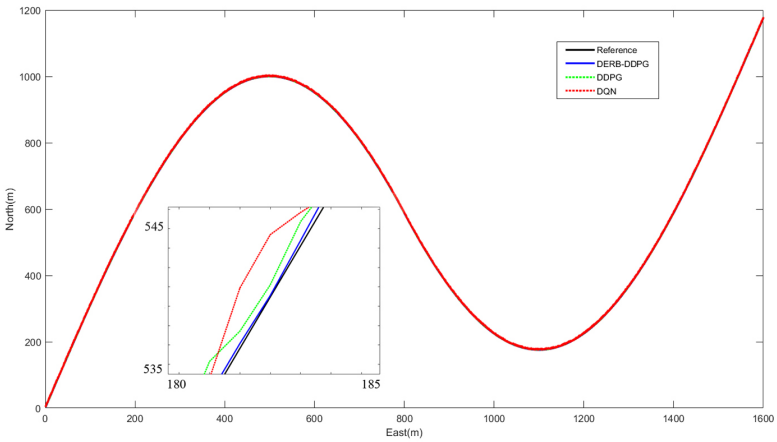


Figure 4.7: Path following performance comparison.

To further validating performance of the proposed method, a more complicated simulation path is carried out in Fig. 4.8. From this figure, the DERB-DDPG still has the best overall performance. Although the original DDPG and DQN may perform better than

DERB-DDPG in some waypoints, however, their performances are not stable, especially in a complicated case. This proves the robustness and stability of the proposed DERB-DDPG algorithm.

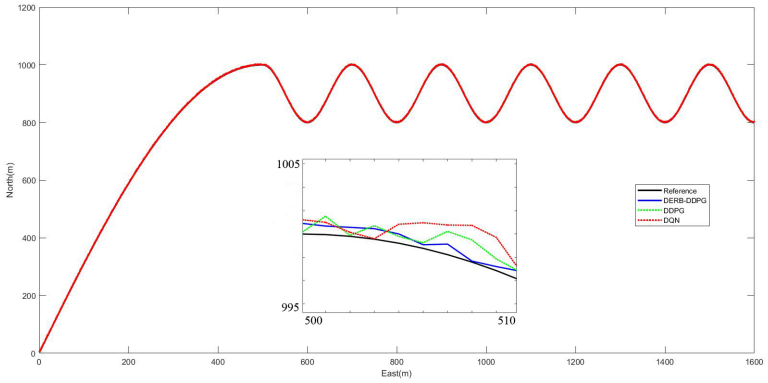


Figure 4.8: Path following performances on another example.

4.2 Formation Control of Multiple Fixed-Wing UAVs

This section investigates the formation control of multi-UAVs using a twin-delayed deep deterministic policy gradient (TD3) approach. The proposed TD3 algorithm combines the deep deterministic policy gradient (DDPG) algorithm and double Q-learning technique to construct a continuous controller for formation tracking of multi-UAVs. Different from DDPG, the TD3 has two Q-networks which explore the environment independently. The smaller Q-value will be selected to compute targets and then update the Q-functions. In addition, the target action policy is clipped to lie in valid action range and the update is delayed to avoid exploiting error experiences. At last, a prioritized experience replay buffer with flexible capacity is introduced to increase the convergence speed. Simulation results in different scenarios show the effectiveness of the proposed method.

As investigated in the previous section, the DDPG algorithm has good performance, however, it has inevitable drawbacks because DDPG may overestimate the Q-values in

the critic network [29]. These estimation errors may accumulate and eventually lead to a non-optimal action. Aiming at mitigating the overestimation, a twin-delayed DDPG (TD3) was proposed in [27, 30]. The TD3 algorithm introduces a double deep Q-learning in the DDPG algorithm. Compared to DDPG, TD3 algorithm has better action performance and faster learning speed due to the following improvements [30, 27, 31].

(1) TD3 has two Q-networks with the same structure instead of a single Q-network. TD3 uses the smaller one as the Q-value. As a result, the overestimation problem will be alleviated.

(2) TD3 reduces the update frequency in order to solve the coupling problem of unchanged actor-critic networks.

(3) TD3 adds noise to the target action which makes the algorithm harder to exploit errors from Q-function and makes the critic smoother.

Based on the idea of TD3, this section proposes an advanced TD3 controller for the multi-UAVs formation control problem. Firstly, the kinematic model of follower UAV is derived and the formation control will be realized by follower UAV tracking the leader UAV. Secondly, the actor-critic networks are constructed to build a framework of the proposed TD3 algorithm. After that, a double Q-learning mechanism is utilized to replace the single Q-learning in order to avoid overestimation of the Q-function. Then, the action policy is clipped to ensure the generated action lies in a valid range. Next, the policy and target networks are updated less frequency than the Q-functions in order to prevent the negative impact of volatility, which is quite common in DDPG. Finally, the capacity of replay buffer is flexible to better meet the requirement of data diversity.

The main contributions of the proposed TD3 algorithm are outlined as follows:

(1) *State and action space definition for formation control*. This section proposes a new

definition of state and action spaces which is suitable for the DRL algorithm to learn and control the UAV group for formation maintenance mission.

(2) *Reward function*. A new reward function is designed to sensitively and accurately represents the performance of an action in the formation control problem.

(3) *Replay buffer*. The replay buffer used in this section has a flexible capacity, which aims at improving the learning efficiency of the algorithm. In addition, the prioritized sampling methods can also increase the learning efficiency by replaying the better samples more frequently.

4.2.1 Problem Formulation and Formation Kinematic Model

The formation control problem of multi-UAVs can be considered as a leader-follower tracking problem of the relative distance and angle between followers and the leader. Following this idea, a kinematic formulation of the follower UAV is derived. After that, the state space representation of the leader and follower system will be presented.

Modelling the Follower UAV

The kinematics of a follower UAV is depicted in Fig. 4.9, d_l stands for the relative distance between leader and follower, ψ_{FL} and θ_{FL} are the relative yaw and pitch angles, respectively.

Three coordinate systems will be used in the studied formation control problem. They are inertial frame $S_I - O_I x_I y_I z_I$, follower body frame $S_B - O_B x_B y_B z_B$, and line-of-sight (LOS) frame $S_L - O_L x_L y_L z_L$.

The follower velocity vector V_f in follower body coordinate frame S_B is considered as

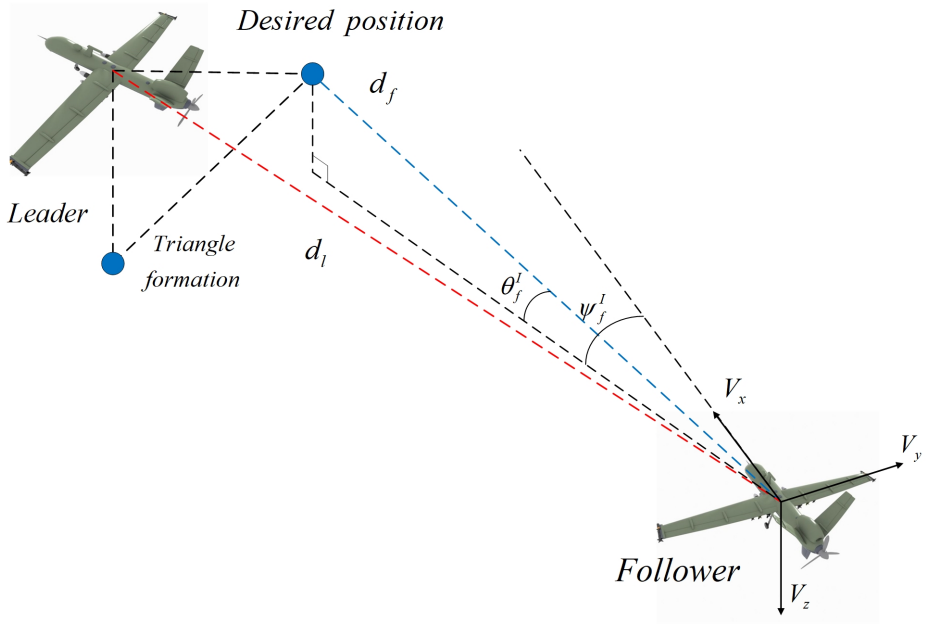


Figure 4.9: The leader-follower kinematics.

$V_f^B = [v_f\ 0\ 0]^T$. While in inertial coordinate system, the follower velocity vector is:

$$V_f^I = C_{B\to I} V_f^B = \begin{bmatrix} v_f \cos\theta_f^I \cos\psi_f^I \\ v_f \cos\theta_f^I \sin\psi_f^I \\ -v_f \sin\theta_f^I \end{bmatrix} \tag{4.22}$$

where v_f is the speed of follower, $C_{B\to I}$ stands for the transfer matrix from body frame to inertial frame. θ_f^I and ψ_f^I are the pitch and yaw angles of follower with respect to inertial reference system.

Define $\omega_B = [w_{xB}\ w_{yB}\ w_{zB}]^T$ as the angular velocity of body frame. Therefore, the

follower's acceleration in body frame $A_f^B = [a_{xB}\ a_{yB}\ a_{zB}]$ can be expressed as:

$$\begin{aligned} A_f^B = \begin{bmatrix} a_{xB} \\ a_{yB} \\ a_{zB} \end{bmatrix} &= \dot{V}_f^B + \omega_B \times V_f^B \\ &= \begin{bmatrix} \dot{v}_f \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} w_{xB} \\ w_{yB} \\ w_{zB} \end{bmatrix} \times \begin{bmatrix} v_f \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \dot{v}_f \\ w_{zB} v_f \\ -w_{yB} v_f \end{bmatrix} \end{aligned} \tag{4.23}$$

The follower's motion equations are:

$$\begin{cases} \dot{x}_f = v_f \cos\psi_f^I \cos\theta_f^I \\ \dot{y}_f = v_f \sin\psi_f^I \cos\theta_f^I \\ \dot{z}_f = -v_f \sin\theta_f^I \\ \dot{v}_f = a_{xB} \\ w_{xB} = 0 \\ w_{yB} = \dfrac{a_{yB}}{v_f} \\ w_{zB} = -\dfrac{a_{zB}}{v_f} \end{cases} \tag{4.24}$$

where (x_f, y_f, z_f) stands for the follower's position in inertial frame.

d_f is defined as the distance from the position of follower to the desired position, which is calculated based on the leader's position in order to maintain the shape of the formation.

$D_f = [d_f\ 0\ 0]^T$ is considered as the distance vector. So, the velocity vector is:

$$V_f^L = \dot{D}_f + \Omega_L \times D_f = \begin{bmatrix} \dot{d}_f \\ w_{zL} d_f \\ w_{yL} d_f \end{bmatrix} \tag{4.25}$$

The acceleration vector A_L can be obtained by taking derivative of (4.25):

$$\begin{aligned} A_f^L &= \begin{bmatrix} a_{xL} \\ a_{yL} \\ a_{zL} \end{bmatrix} = \dot{V}_f^L + \Omega_L \times V_f^L \\ &= \begin{bmatrix} \ddot{d}_f - d_f(w_{yL}^2 + w_{zL}^2) \\ 2\dot{d}_f w_{zL} + d_f \dot{w}_{zL} + d_f w_{xL} w_{yL} \\ -2\dot{d}_f w_{yL} - d_f \dot{w}_{yL} + d_f w_{xL} w_{zL} \end{bmatrix} \end{aligned} \tag{4.26}$$

According to (4.26), the kinematic equation can be described as:

$$\begin{cases} \ddot{d}_f = d_f(w_{yL}^2 + w_{zL}^2) + a_{xL} \\ \dot{w}_{zL} = \dfrac{a_{yL} - 2\dot{d}_f w_{zL} - d_f w_{xL} w_{yL}}{d_f} \\ \dot{w}_{yL} = \dfrac{-a_{zL} - 2\dot{d}_f w_{yL} + d_f w_{xL} w_{zL}}{d_f} \end{cases} \tag{4.27}$$

In addition, the angular velocity is expressed as:

$$\begin{bmatrix} w_{xL} \\ w_{yL} \\ w_{zL} \end{bmatrix} = \begin{bmatrix} w_{yB} \sin\psi_f^L \cos\theta_f^L - w_{zB} \sin\theta_f^L - \dot{\psi}_f^L \sin\theta_f^L \\ w_{yB} \cos\psi_f^L + \dot{\theta}_f^L \\ w_{yB} \sin\psi_f^L \sin\theta_f^L + w_{zB} \cos\theta_f^L + \dot{\psi}_f^L \cos\theta_f^L \end{bmatrix} \tag{4.28}$$

It should be noted that (4.28) is obtained under the assumption of $w_{xB} = 0$.

State-Space Representation

There are 7 state variables in the kinematic model of follower UAV. Specifically, d_f is the distance between the follower UAV and its corresponding desired position. $\dot{d}_f$ is the time derivative of the above distance. w_{zL} is the angular velocity along z-axis and w_{yL} is the angular velocity along y-axis. ψ_f^I and θ_f^I are the yaw angle and pitch angle. v_f is the velocity of the follower UAV.

$$\begin{cases} x_1 = d_f \\ x_2 = \dot{d}_f \\ x_3 = w_{zL} \\ x_4 = w_{yL} \\ x_5 = \psi_f^I \\ x_6 = \theta_f^I \\ x_7 = v_f \end{cases} \tag{4.29}$$

The above state variables can be either measured by on-board sensors or calculated based on sensor measurements. Combining (4.29), the state space representation of the kinematic model can be expressed as follows:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -x_1 x_3^2 - x_1 x_4^2 - a_{xL} \\ \dot{x}_3 = \dfrac{a_{yL}}{x_1} - x_4 w_{xL} \\ \dot{x}_4 = -\dfrac{a_{zL}}{x_1} + x_3 w_{xL} \\ \dot{x}_5 = \dfrac{x_3}{\cos x_6} - \dfrac{a_{yB}}{x_7} + \dfrac{a_{zB}}{x_7} \sin x_5 \tan x_6 \\ \dot{x}_6 = x_4 + \dfrac{a_{zB}}{x_7} \cos x_5 \\ \dot{x}_7 = a_{xB} \end{cases} \tag{4.30}$$

The acceleration vector of follower UAV A_f^L can be expressed as:

$$A_f^L = \begin{bmatrix} a_{xL} \\ a_{yL} \\ a_{zL} \end{bmatrix} = \begin{bmatrix} a_{xl} \\ a_{yl} \\ a_{zl} \end{bmatrix} - \begin{bmatrix} a_{xf} \\ a_{yf} \\ a_{zf} \end{bmatrix} \tag{4.31}$$

The acceleration vector of follower $A_f^L = [a_{xf}\ a_{yf}\ a_{zf}]^T$ can be obtained using the following equation:

$$\begin{bmatrix} a_{xf} \\ a_{yf} \\ a_{zf} \end{bmatrix} = \begin{bmatrix} \cos\psi_f^{BI}\cos\theta_f^{BI} & \sin\psi_f^{BI}\cos\theta_f^{BI} & -\sin\theta_f^{BI} \\ -\sin\psi_f^{BI} & \cos\psi_f^{BI} & 0 \\ \cos\psi_f^{BI}\sin\theta_f^{BI} & \sin\psi_f^{BI}\sin\theta_f^{BI} & \cos\theta_f^{BI} \end{bmatrix} + \begin{bmatrix} a_{xB} \\ a_{yB} \\ a_{zB} \end{bmatrix} \tag{4.32}$$

Based on the above equations, the state space representation can be rewrote as:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -x_1 x_3^2 - x_1 x_4^2 - a_{xl} + a_{xB}\cos x_5 \cos x_6 \\ \qquad + a_{yB}\sin x_5 \cos x_6 - a_{zB}\sin x_6 \\ \dot{x}_3 = \dfrac{a_{xB}\sin x_5}{x_1} - \dfrac{a_{yB}\cos x_4}{x_1} + \dfrac{a_{yl}}{x_1} \\ \qquad + \dfrac{x_4}{\cos x_6}\left(\dfrac{a_{zB}}{x_7}\sin x_5 + x_3 \sin x_6\right) \\ \dot{x}_4 = \dfrac{a_{xB}\cos x_5 \sin x_6}{x_1} + \dfrac{a_{yB}\sin x_5 \sin x_6}{x_1} \\ \qquad + \dfrac{a_{zB}\cos x_6}{x_1} - \dfrac{a_{zl}}{x_1} + x_3 w_{xL} \\ \dot{x}_5 = \dfrac{x_3}{\cos x_6} - \dfrac{a_{yB}}{x_7} + \dfrac{a_{zB}}{x_7}\sin x_5 \tan x_6 \\ \dot{x}_6 = x_4 + \dfrac{a_{zB}}{x_7}\cos x_5 \\ \dot{x}_7 = a_{xB} \end{cases} \tag{4.33}$$

where a_{xB}, a_{yB} and a_{zB} are considered as the control inputs while a_{xl}, a_{yl} and a_{zl} are considered to be known.

The control objective of the formation control problem is that the UAV group can form a specific shape and maintain this formation shape while moving. Based on the above equations, the formation maintenance will be achieved by giving follower UAV a proper value of the accelerations on three axes.

4.2.2 The Proposed TD3 Algorithm

Although DDPG has good performance in the continuous control problem, an obvious shortcoming can not be ignored, which is the trained Q-function sometimes overestimates the Q-values dramatically. As a result, this will lead to failure because the algorithm starts

exploiting the overestimation errors in the Q-function. Therefore, a TD3 algorithm is proposed to address the above limitation by introducing the following three major improvement.

Improvement 1: Clipped Double Q-learning

TD3 simultaneously learns two Q-functions (Q_{ϕ_1} and Q_{ϕ_2}) in the same way as DDPG learns a single one. The two Q-functions share a single target, and the smaller value will be used in the target function:

$$y_{min}(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,targ}}(s', a'(s')) \tag{4.34}$$

After that, both Q-functions will be updated by regressing to the above target $y_{min}(r, s', d)$.

$$\begin{aligned} L(\phi_1) &= E[(Q_{\phi_1}(s, a) - y_{min}(r, s', d))^2] \\ L(\phi_2) &= E[(Q_{\phi_2}(s, a) - y_{min}(r, s', d))^2] \end{aligned} \tag{4.35}$$

The main idea of improvement 1 is using the smaller Q-value in the target function and then two Q-functions regress towards that target. In this way, the overestimation in the Q-function will be decreased greatly.

Improvement 2: Delayed Policy Update

In the proposed TD3 algorithm, the policy and target networks are updated in a less frequency than the updates of Q-function. The policy is updated by maximizing the Q-function $\max_{\theta} E[Q_{\phi}(s, \mu_{\theta}(s))]$. Specifically, the proposed TD3 algorithm updates the policy only once when the Q-functions update twice. This aims at reducing the negative impact of volatility, which is normally appears in DDPG.

Improvement 3: Target Policy Smoothing

In order to avoid learning from Q-function overestimation errors, noise is added in the target action.

In DDPG, action used in the target function are generated by the target policy $\mu_1\theta_{targ}$.

$$a'(s') = \mu_1\theta_{targ}(s') \tag{4.36}$$

While in TD3, action is generated with a clipped noise added on the action.

$$a'(s') = \text{clip}(\mu_{\theta_{targ}}(s') + \text{clip}(\epsilon, -c, c), a_{min}, a_{max}) \tag{4.37}$$

where $\epsilon \sim N(0, \sigma)$.

The range of target action lies from a_{min} to a_{max}. Target policy smoothing is important because it avoids a failure mode which may happen in DDPG. If the Q-function estimator develops a sharp peak for some actions (which is considered to be incorrect estimation), the policy will exploit that peak value and then learns an incorrect behavior.

Pseudo-Code of the Proposed TD3 Algorithm

The pseudo-code of the proposed TD3 is shown in the Algorithm 4.

4.2.3 Formation Controller Design

State and Action Specification

In actual application of DRL algorithm, the definition of state space and reward function is decisive for the overall performance of the algorithm. A proper design of state and reward function can lead to a fast convergence speed and small steady-state error. This section proposes a new state space definition which is suitable for the formation control problem.

Algorithm 4: The proposed TD3 algorithm

Randomly initialize policy π and Q-functions ϕ_1 and ϕ_2
Copy the weights to target network $\theta_{targ} \leftarrow \theta$, $\phi_{targ,1} \leftarrow \phi_1$ and $\phi_{targ,2} \leftarrow \phi_2$
Empty replay buffer $\mathbf{D}$, set update counter $j = 0$ and set policy update frequency $delay\_parameter = 2$
for Episode = $[1, m]$ **do**
  for Time step = $[1, T]$ **do**
    Observe the current state s_t and generate the action
    $a_t = \text{clip}(\mu_\theta(s_t) + \epsilon, a_{min}, a_{high})$, $\epsilon \sim \mathbf{N}$
    Execute the action a_t and the agent reach the next state s_{t+1}
    Observe the next state s_{t+1} and obtain the immediate reward r_t
    Store the transition (s_t, a_t, r_t, s_{t+1}) into the replay buffer $\mathbf{D}$
    if The replay buffer has enough transitions to update **then**
      Randomly sample a batch of n transitions from replay buffer $\mathbf{D}$
      Compute target actions
      $a'(s') = \text{clip}(\mu_{\theta_{targ}}(s') + \text{clip}(\epsilon, -c, c), a_{min}, a_{max}), \epsilon \sim N(0, \sigma)$
      Compute targets
      $y_{min}(r, s') = r + \gamma \min_{i=1,2} Q_{\phi_{i,targ}}(s', a'(s'))$
      Update Q-functions by gradient descent and add 1 to the update counter
      $j = j + 1$
      $\bigtriangledown_{\phi_i} \frac{1}{n} \sum (Q_{\phi_i}(s, a) - y(r, s'))^2$ where $i = 1, 2$
      if $j \mod delay\_parameter = 0$ **then**
        Update policy by gradient ascent
        $\bigtriangledown_\theta \frac{1}{n} \sum Q_{\phi_i}(s, \mu_\theta(s))$, where $i = 1, 2$
        Update target networks
        $\phi_{targ,i} \leftarrow \rho\phi_{targ,i} + (1 - \rho)\phi_i$, where $i = 1, 2$
        $\theta_{targ} \leftarrow \rho\theta_{targ} + (1 - \rho)\theta$
      end if
    end if
  end for
end for

The action space of the proposed TD3 algorithm are the accelerations along x, y and z axis, a_{xB}, a_{yB} and a_{zB}.

$$A = \{a_{xB}, a_{yB}, a_{zB}\} \tag{4.38}$$

The definition of state space needs to uniquely determine a specific and accurate situation of agent. Whenever the agent reaches a state, the same action applied will lead to a unique consequence. It should be noted that a redundancy in state space is not acceptable because it will inevitably increase the calculation difficulty, reduce the learning efficiency and degrade the overall performance. Therefore, the definition of the state space needs to illustrate the agent's situation concisely and accurately.

In the problem of leader-follower formation control, the state information of the follower UAV includes the translational variables and the angular variables.

$$\text{State information} = \{\underbrace{d_f, v_f, a_f}_{\text{translational}}, \underbrace{\psi_f, \theta_f, w_{zL}, w_{yL}}_{\text{angular}}\} \tag{4.39}$$

where d_f represents the distance between follower UAV and desired position. v_f and a_f stand for the velocity and acceleration of the UAV. ψ_f and θ_f refer to the yaw angle and pitch angle. w_{zL} and w_{yL} are the angular velocity of yaw motion and pitch motion respectively.

Some papers directly use the above observed state information as the state space of the algorithm [130, 131, 132]. This is acceptable, however, directly using observed information as the state space in MDP may increase the learning difficulty and computing burden. As a result, the convergence speed may be slowed and the overall performance may be degraded. Therefore, the state space needs to be specialized for the learning objective, which is approximating Q-functions and generating the best actions. Since the variables in action space are sub-divided to each axis, the state space needs to do the same sub-division for the

convenience of learning. The state space is defined as:

$$S = \{d_{fx}, d_{fy}, d_{fz}, v_{fx}, v_{fy}, v_{fz}, a_{fx}, a_{fy}, a_{fz}, \psi_f, \theta_f, B_{flag}\} \tag{4.40}$$

where the distances to desired position d_{fx}, d_{fy}, d_{fz} determines the current position of the agent. The velocities v_{fx}, v_{fy}, v_{fz} and accelerations a_{fx}, a_{fy}, a_{fz} describe the current translational information. The two angles ψ_f, θ_f illustrate the current attitude of the agent. B_{flag} is a binary flag variable, which indicates whether the follower UAV is perfect-tracking. If the position of follower UAV is exactly located on the desired position, this is called perfect-tracking. And the value of flag variable $B_{flag} = 1$, otherwise $B_{flag} = 0$.

The above state and action spaces consist of physical observations (distance, angles, and velocities), however, the ranges of these variables may differ a lot. If the algorithm is learning from the original values, the effectiveness and efficiency will be greatly decreased. It is necessary to scale the values of these variables. Therefore, during the learning process, the values of state and action variables will be normalized from -1 to $+1$.

Reward Function

Generally, the reward function is considered to be one of the most important component in a RL algorithm because reward function evaluates how good or how bad the action is. Furthermore, reward function also determines the learning direction and the overall performance. The reward function is composed with five parts: distance reward $r_{t,d}$, velocity reward $r_{t,v}$, angle reward $r_{t,a}$, penalty $r_{t,p}$ and bonus $r_{t,b}$.

The distance reward $r_{t,d}$ is defined as:

$$r_{t,d} = -\mu_1 \frac{(d_{t+1,f} - d_{t,error})^2}{d_{par}} \tag{4.41}$$

where μ_1 is a positive constant parameter, $d_{t+1,f}$ denotes the distance between the current position and the estimated next position after taking action a_t. $d_{t,error}$ refers to the distance between the current position and the desired position. d_{par} is a positive constant parameter to normalized the distance reward. From (4.41), the follower UAV will receive a larger value of distance reward while it is closer to the desired position. On the contrary, the distance reward will decrease when UAV is moving far away from the desired position.

Second, $r_{t,v}$ is the velocity reward.

$$r_{t,v} = \mu_2 \frac{V_{velo}}{|V_{velo}|} \cdot \frac{1}{V_{velo} - 1} \tag{4.42}$$

where μ_2 is a positive constant parameter. V_{velo} refers to a time-varying parameter which is defined below:

$$V_{velo} = \frac{2d_{t,error}}{(v_{t,f} + v_{t+1,f}) \cdot T} + \frac{(v_{t,f} + v_{t+1,f}) \cdot T}{2d_{t,error}} \tag{4.43}$$

where $v_{t,f}$ and $v_{t+1,f}$ are the velocities of follower UAV at current time instant t and the next time instant $t+1$. T stands for the time interval during each time instant. The idea behind equation (4.42) includes two main issues. First, the distance reward will be a negative value when the velocity and the desired position are in the opposite direction. While the velocity and desired position are in the same direction, the distance reward will be positive. Second, the follower UAV will receive a larger reward if the estimated position is closer to the the desired position.

The angle reward is defined as:

$$r_{t,a} = \mu_3 \sqrt{(\psi_{t+1,f} - \psi_{t+1,L})^2 + (\theta_{t+1,f} - \theta_{t+1,L})^2} \tag{4.44}$$

where μ_3 is a positive constant parameter. $\psi_{t,f}$ and $\psi_{t,L}$ are the yaw angles of follower and leader. $\theta_{t,f}$ and $\theta_{t,L}$ are the pitch angles of follower and leader. The design of angle reward aims at eliminating the oscillations while follower UAV is close to the desired position.

The penalty is defined as $r_{t,p}$. The follower UAV will receive a huge negative value when the pitch angle or the yaw angle changes greatly, which aims at preventing sharp turning maneuver.

$$r_{t,p} = \begin{cases} -100, & |\psi_{t+1,f} - \psi_{t,f}| \geq \frac{\pi}{6} \\ -100, & |\theta_{t+1,f} - \theta_{t,f}| \geq \frac{\pi}{6} \\ 0, & |\psi_{t+1,f} - \psi_{t,f}| < \frac{\pi}{6} \\ 0, & |\theta_{t+1,f} - \theta_{t,f}| < \frac{\pi}{6} \end{cases} \tag{4.45}$$

The bonus is defined as $r_{t,b}$. Bonus can be considered as an extra reward which is not included in the expected reward. Bonus can help the agent exploiting the good experiences and accelerating convergence speed. In this section, if the UAV position is matched with the desired position (while there is little error between follower UAV and desired position), the follower UAV will receive a huge bonus to enhance this good action.

$$r_{t,b} = \begin{cases} 10, \text{ UAV matches the desired position} \\ 0, \text{ otherwise} \end{cases} \tag{4.46}$$

In summary, the reward function can be constructed by combining the above five value functions.

$$R_{fc}(t) = r_{t,d} + r_{t,v} + r_{t,a} + r_{t,p} + r_{t,b} \tag{4.47}$$

Replay Buffer Design

The experience replay buffer is used to break data correlation. However, in many applications, especially for the algorithms with advanced learning ability, the fixed capacity of replay buffer can not meet the requirement of data diversity. As a result, the learning efficiency will be reduced. In terms of the above limitation, a dynamic capacity replay buffer (DCRB) is proposed to dynamically adjust the capacity of the experience pool

during the training phase. Learning curve theory is introduced to realize the dynamic adjustment. Learning curve is a dynamic evaluation technique which describes a learning process that the learner continuously improves its performance using the accumulation experience [133]. Among the literature concerning learning curve theory [133, 134, 135], the Wright learning curve (WLC) is the most widely used method. The WLC equation can be expressed as follows.

$$TP(n) = kn^{\alpha} \tag{4.48}$$

where n represents the number of explorations, $TP(n)$ refers to the time period of the n-th episode. k is a parameter stands for the learning effect. α is the learning coefficient which is defined as follows

$$\alpha = \frac{\lg s}{\lg 2} \tag{4.49}$$

where s is the decreasing rate. According to (4.48) and (4.49), the learning efficiency equation can be constructed as

$$\eta(x) = \frac{1}{k} n^{-\beta_1} \tag{4.50}$$

where $\eta(x)$ stands for the learning efficiency of the x-th episode. β_1 is the learning coefficient which is defined as

$$\beta_1 = \frac{\lg \gamma}{\lg 2} \tag{4.51}$$

where γ is the reward discount rate ranged from 0 to 1.

In order to adjust the capacity of experience replay buffer as the number of training episodes increases, the learning efficiency equation (4.50) and (4.51) will be combined to construct the replay buffer capacity change function.

$$N = N \frac{1}{k} (i)^{-\frac{\lg \gamma}{\lg 2}} \tag{4.52}$$

where N refers to the capacity of the replay buffer and i is the number of explorations.

During the training phase, a batch of experience samples will be extracted from replay buffer. Generally, all the samples in the replay buffer have equal chance to be extracted. In other words, this sampling method ignores the significance of each experience. Aiming at addressing the importance of different experiences, a prioritized sampling method is introduced here.

The main idea of prioritized sampling method is to adjust the extracting distribution, the better experiences will be learned more frequently. For each experience in the replay buffer, the corresponding temporal difference (TD) error will be calculated to reflect the performance of this sample, which is defined as follows.

$$TD_i = r(s_t, a_t) + \gamma Q'(s_{t+1}, \mu(s_{t+1})) - Q(s_t, a_t) \tag{4.53}$$

If the value of TD error increases, this experience will have a higher probability to be replayed. The extracting probability P_i is determined by the priority value of each experience in the current replay buffer. The priority value can be obtained by

$$vp_i = (|TD_i| + \sigma)^{\beta_2} \tag{4.54}$$

where σ is a constant with small positive value which guarantees the priority value will not be zero. β_2 refers to a weight constant. After that, the extracting probability of each experience is

$$P_i = \frac{vp_i}{\sum_1^N p_N} \tag{4.55}$$

where N refers to the capacity of the current replay buffer. Since vp_i must be a positive value, as a result, P_i will not be zero. In other words, all experiences have chance to be replayed.

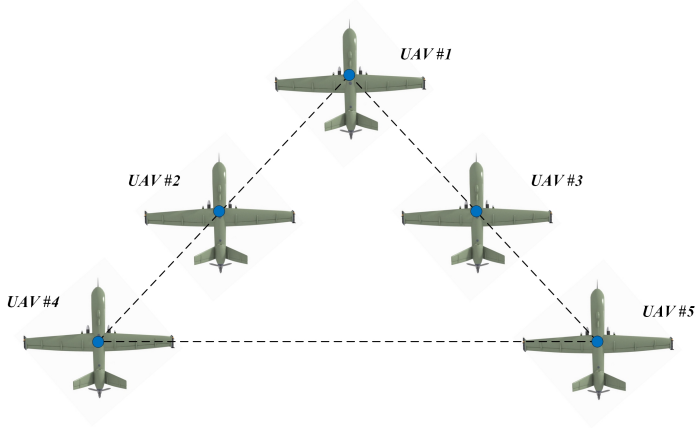


Figure 4.10: The desired formation of the UAV group.

4.2.4 Simulation and Analysis

The performance of the proposed TD3 algorithm will be evaluated in several scenarios. In the simulation, a group of five UAVs is considered. In the UAV team, UAV $\#1$ is set to be the leader and the others UAV $\#2$, UAV $\#3$, UAV $\#4$ and UAV $\#5$ are followers. Leader UAV can receive the control command and follower UAV can only receive the state information of the leader. As shown in Fig. 4.10, a triangle formation is given to the UAV team and the goal of the follower UAV is to maintain this formation.

Training Settings

The training procedure of the proposed algorithm is implemented in the virtual environment. In order to ensure the variety of training samples and reduce the negative effect of overfitting, the algorithm will be trained in three cases. In the first case, the leader will fly along a circle-shaped or sine-wave shaped trajectory. However, the size of the trajectory keeps changing between different episodes to broaden the training samples. In the second case, the leader will fly in 2-D environment and the leader is softly changing the attitude, velocity and acceleration at each time instant, which means the trajectory of the leader may not follow a specific shape. This case aims at avoiding overfitting of a circle-shape or

sine-wave shape. In the third case, the leader will fly a 3-D trajectory with softly changing attitude, velocity, altitude and acceleration at each time instant. The lateral motion has been well training in the above two cases, the third case is designed for training the longitudinal motion and reinforce the performance in lateral motion. This training method can broaden the training samples, avoid overfitting and break the coupling between different motions.

Training Analysis

Success rate or miss rate is considered as an important evaluation issue of the training effect in RL algorithm design. For the formation control problem in this section, a successful episode means the UAV can follow the leader's trajectory or stay close to it. If the follower UAV is located too far from the desired trajectory, this episode will be considered as a failure and there is no need to learn from this bad experience. Saving this experience will ruin the experience pool of the replay buffer. Therefore, this training episode will be seized immediately and start a new training episode.

The successful rates of the three training cases are shown in Table 4.1. Table 4.1 not only lists the total successful rate, but also lists the recent successful rate, which means the successful rate within the most recent 200 training episodes. The total successful rate presents the integral learning ability while the recent successful rate illustrates the real-time learning ability during the training process. Case 1 will be trained firstly and then Case 2. Case 3 will be trained lastly. Each case is assigned 2000 training episodes to guarantee the convergence of the algorithm.

From Table 4.1, the successful rate of Case 1 is comparative low because the initial parameters of the proposed algorithm are selected randomly, the algorithm needs more episodes to learn. Although the total successful rate is growing slowly, however, the recent successful rate is increasing quickly and approaches over 90%. This proves a fact that the algorithm can reach a high-effective training process in a short time. To better illustrate the

Table 4.1: The successful rates during the training process

Leader case	**Training episodes**	**Total successful episodes**	**Total successful rate**	**Recent successful episodes**	**Recent successful rate**
Case 1	200	64	32%	64	32%
	400	149	37.3%	85	42.5%
	600	271	45.2%	122	61%
	800	427	53.4%	156	78%
	1000	569	56.9%	142	71%
	1200	765	63.8%	196	98%
	1400	959	68.5%	194	97%
	1600	1143	71.4%	184	92%
	1800	1336	74.2%	193	96.5%
	2000	1531	76.6%	195	97.5%
Case 2	200	142	71%	142	71%
	400	295	73.8%	153	76.5%
	600	474	79%	179	89.5%
	800	667	83.4%	193	96.5%
	1000	855	85.5%	188	94%
	1200	1049	87.4%	194	97%
	1400	1249	89.2%	200	100%
	1600	1445	90.3%	196	98%
	1800	1642	91.2%	197	98.5%
	2000	1838	91.6%	196	98%
Case 3	200	159	79.5%	159	79.5%
	400	332	83%	173	86.5%
	600	513	85.5%	181	90.5%
	800	703	87.9%	190	95%
	1000	892	89.2%	189	94.5%
	1200	1086	90.5%	196	98%
	1400	1282	91.6%	196	98%
	1600	1474	92.1%	192	96%
	1800	1667	92.6%	193	96.5%
	2000	1862	93.1%	195	97.5%

Table 4.2: Initial conditions of scenario 1

Aircraft	**Position** $(x,y)(m)$	**Velocity** (m/s)	**Attitude** $\psi(rad)$
Leader	$(1, 3)$	14	1.2
Follower 1	$(0, 2)$	10	1
Follower 2	$(1, 1)$	8	1.4
Follower 3	$(-2, 2)$	12	1.1
Follower 4	$(1, 0)$	9	0.9

training efficiency, Fig. 4.11 draws the total successful rate and recent successful rate in one figure.

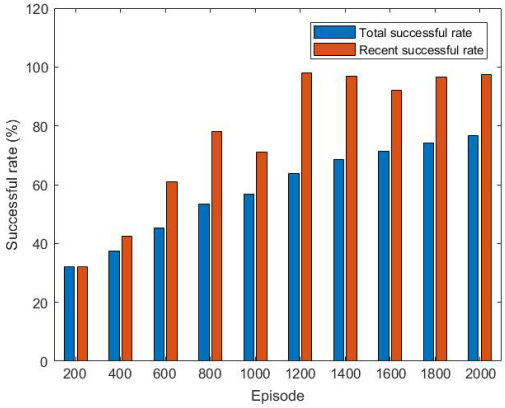
Formation Control Performance

- **Scenario 1**

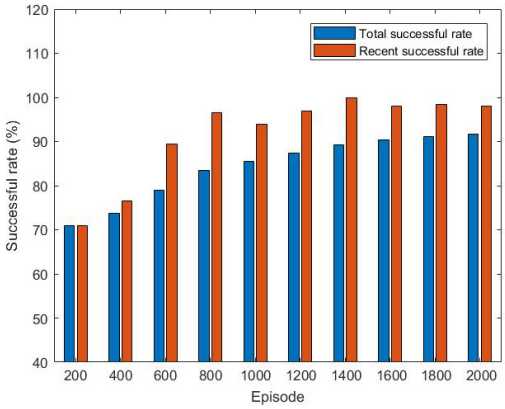
The first test example is illustrated in Fig. 4.12, the leader UAV is flying in a sine wave trajectory and maintains a constant altitude. Therefore, the pitch angle θ and the acceleration on z-axis a_z will remain zero. The initial conditions of the UAVs are listed in Table 4.2.

Fig. 4.12 shows the top view of the UAV group's trajectory. It can be observed that the desired formation configuration is achieved quickly and then the UAV team maintains this formation. Fig. 4.13 illustrates the positions of the UAV group in x and y coordinate axes. Fig. 4.14 demonstrates the steady-state errors of the follower UAVs with respect to time history. From Fig. 4.14, it can be seen that the steady-state error is approaching zero in a short time. Since the altitude remains the same in the testing scenario 1, only the yaw angle ψ needs to be studied. Fig. 4.15 depicts the yaw angles with respect to time history. The yaw angle oscillated at the beginning and then approach to the reference value from leader UAV. Fig. 4.16 shows the actions (control inputs) generated by the proposed TD3 algorithm with respect to time history. It is obviously that the control inputs have constrained within the range of $[-4, 4]$ due to the clip mechanism of TD3 algorithm.

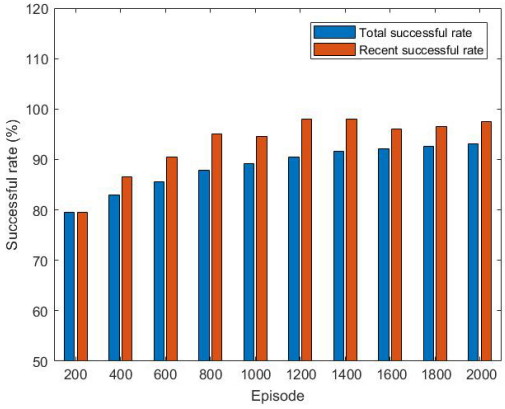
- **Scenario 2** The second testing example is shown in Fig. 4.17, the leader UAV is flying



(a) Successful rates of Case 1



(b) Successful rates of Case 2



(c) Successful rates of Case 3

Figure 4.11: The total successful rate and recent successful rate of three training cases.

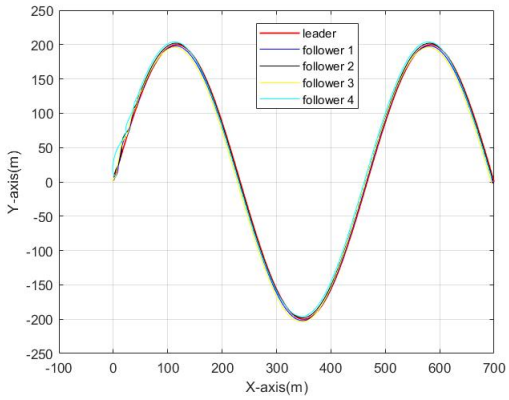
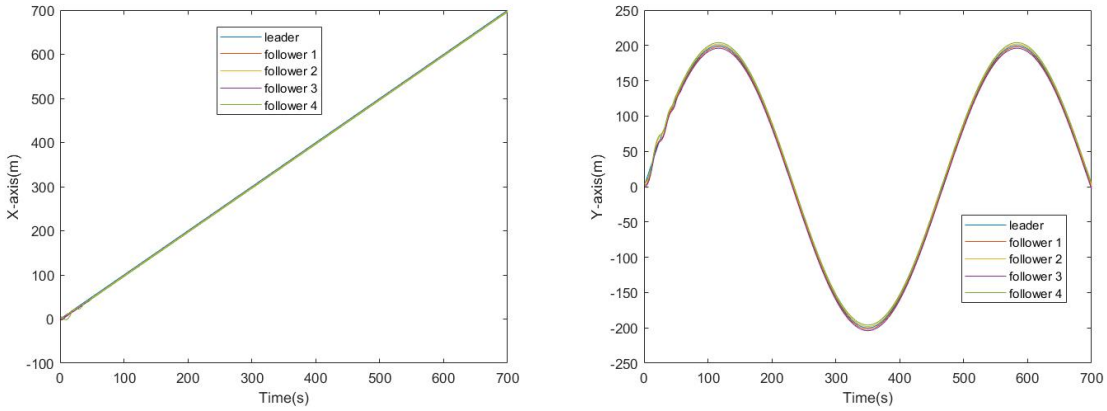


Figure 4.12: The trajectories of the UAV group in scenario 1.



(a) Positions of the UAV group in X-axis (b) Positions of the UAV group in Y-axis

Figure 4.13: The positions of the leader and followers in X-axis and Y-axis.

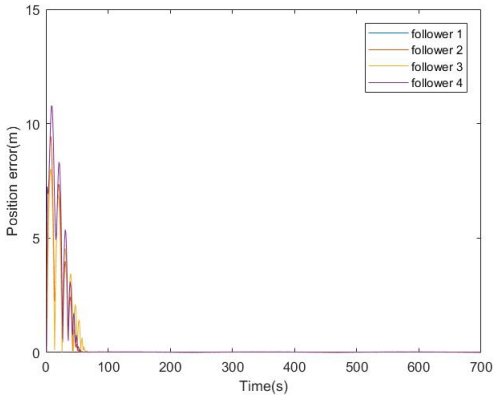


Figure 4.14: The position errors of the UAV group in scenario 1.

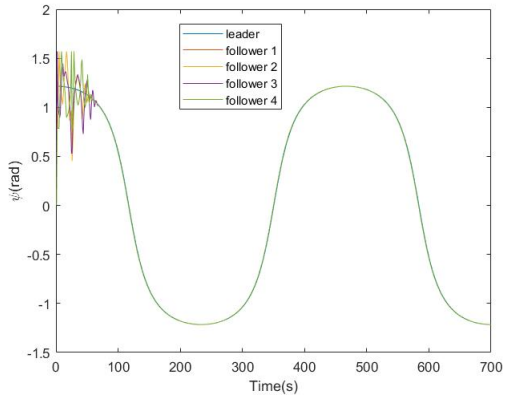
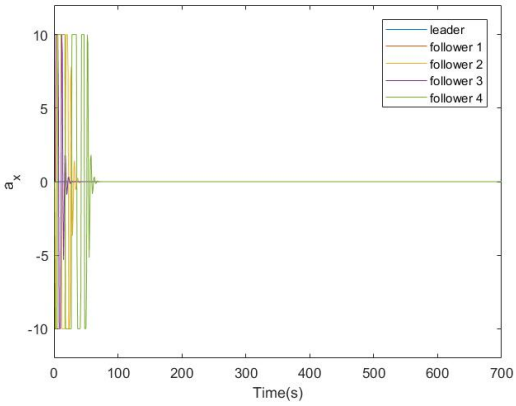
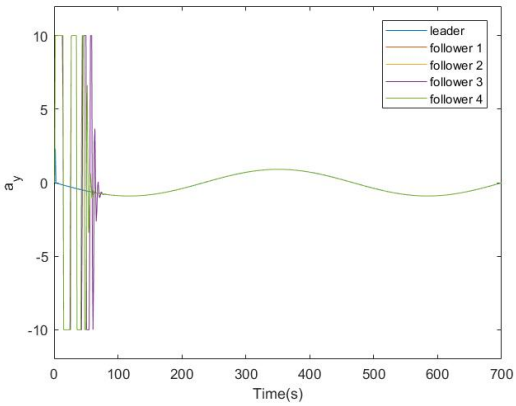


Figure 4.15: The yaw angle ψ of the leader and followers in scenario 1.



(a) The control input a_x of the UAV group



(b) The control input a_y of the UAV group

Figure 4.16: The control inputs of the leader and followers in scenario 1.

Table 4.3: Initial conditions of scenario 2

Aircraft	**Position** $(x,y,z)(m)$	**Velocity** (m/s)	**Attitude** $(\theta,\psi)(rad)$
Leader	$(0, 100, 0)$	26	(0.07,0.01)
Follower 1	$(-2, 98, -1)$	20	(0,0)
Follower 2	$(6, 96, 1)$	18	(0,0)
Follower 3	$(-4, 95, 0)$	23	(0,0)
Follower 4	$(12, 95, -2)$	22	(0,0)

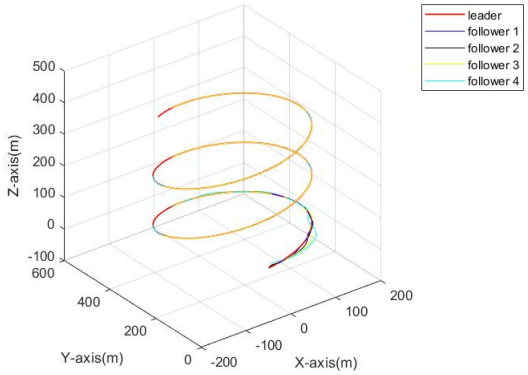
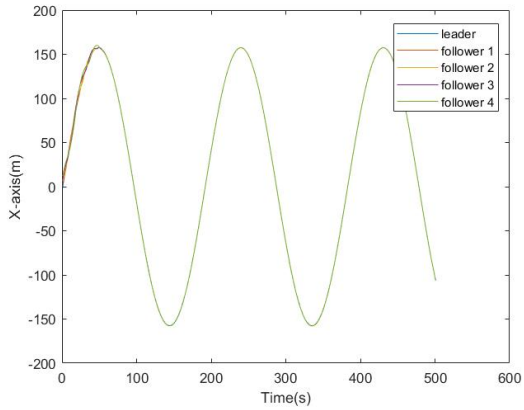


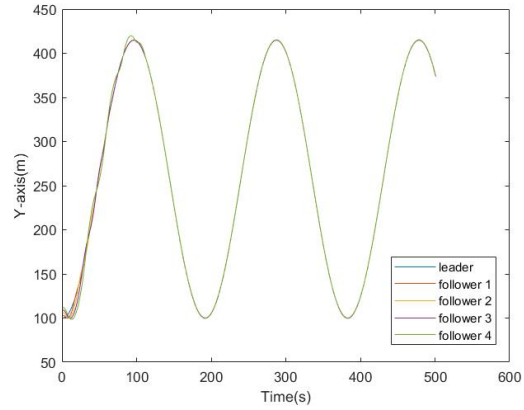
Figure 4.17: The 3-D trajectories of the UAV group in scenario 2.

along a continuous growing circle trajectory in 3-D environment. The position, velocity, attitude and accelerations of the leader UAV keeps changing during the testing. The initial conditions of the UAVs are listed in Table 4.3.

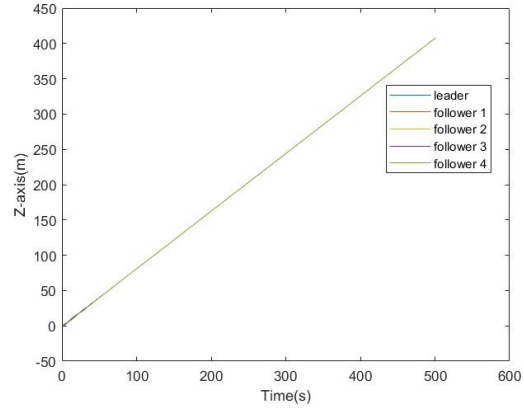
Fig. 4.17 shows the trajectories of the UAV group. The desired formation configuration is achieved within a short time and then the formation maintains. Fig. 4.18 illustrates the positions of the UAV group in x, y and z coordinate axes. Fig. 4.19 demonstrates the steady-state errors of the follower UAVs with respect to time history. From Fig. 4.19, it can be seen that the steady-state error is approaching to zero in a short time. Fig. 4.20 depicts the pitch angles and yaw angles of the UAV group with respect to time history. The yaw angle oscillated at the beginning and then approach to the reference value from leader UAV. Fig. 4.21 shows the actions (control inputs) generated by the proposed algorithm with respect to time history. It is obviously that the control inputs have constrained within



(a) Positions of the UAV group in X-axis



(b) Positions of the UAV group in Y-axis



(c) Positions of the UAV group in Z-axis

Figure 4.18: The positions of the leader and followers in X-axis, Y-axis and Z-axis.

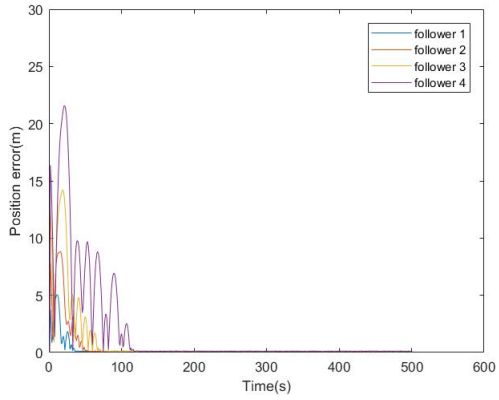
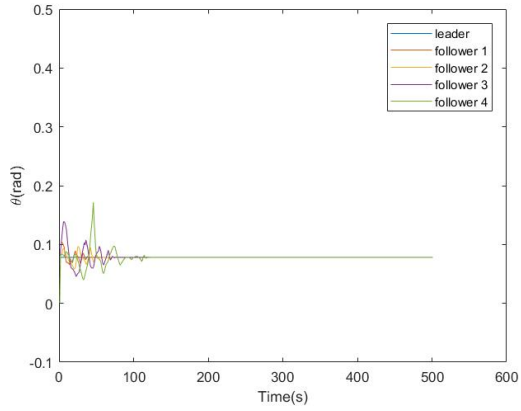
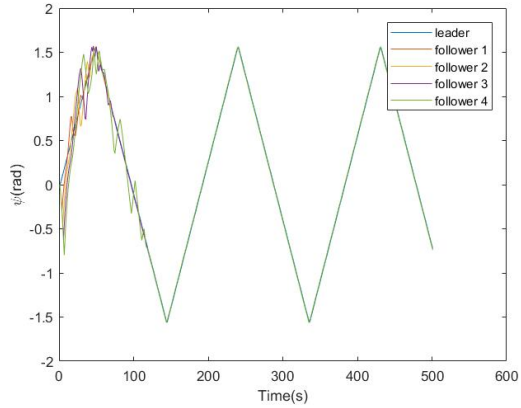


Figure 4.19: The position errors of the UAV group in scenario 2.

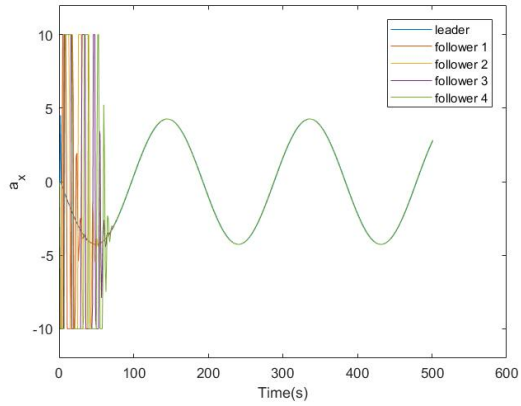


(a) The pitch angle θ of the UAV group

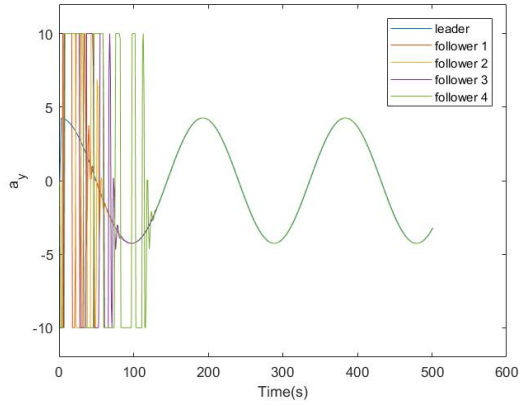


(b) The yaw angle ψ of the UAV group

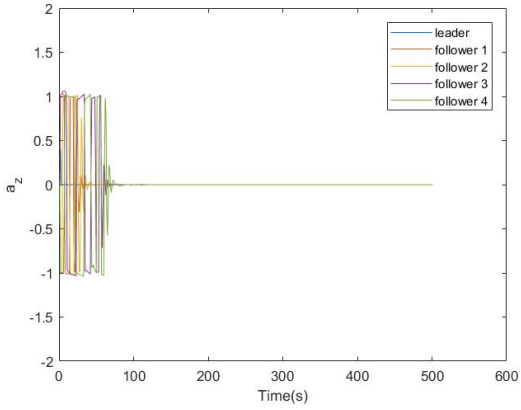
Figure 4.20: The pitch angle θ and yaw angle ψ of the leader and followers in scenario 2.



(a) The control input a_x of the UAV group



(b) The control input a_y of the UAV group



(c) The control input a_z of the UAV group

Figure 4.21: The control inputs of the leader and followers in scenario 2.

a certain range due to the clip mechanism of the proposed algorithm.

4.3 Auto-Landing Control of Quadrotor UAV

An autonomous landing problem of quadrotor UAV is studied in this section. Unlike the above investigated motion control problems, path following and formation control, the proposed auto-landing controller will be validated in real flight tests with QDrone quadrotor platform. While applying DRL algorithm to a real-world control problem, the biggest challenge is the algorithm training and learning. That is because the training cost is considerably lower in the simulation flight than in the real-world flight. In the real-world training, it is impossible to repeat the training episode millions of times. In addition, a failure trial may cause UAV crash and irreversible damage to component. Therefore, the traditional training method may not be applicable in real-world applications. Aiming at this issue, a transfer-imitation learning training (TILT) method is proposed to help DRL algorithm learning safely and effectively. The DRL algorithm used in this section is also TD3 algorithm, the main research direction focuses on the training method of real-world control problem. Therefore, the specific steps of TD3 algorithm will not be repeatedly described.

4.3.1 Problem Formulation

The UAV autonomous landing problem can be described as follows. Given a continuous moving platform on the ground, such as a truck or a car. A quadrotor UAV in the air which can detect or receive the motion and position of the moving platform. The goal of the UAV is trying to land on the moving platform accurately and safely. The autonomous landing system is shown in Fig. 4.22. It is assumed that the UAV is aware of its own location $[x_{uav}, y_{uav}, z_{uav}]$ and the location of moving platform $[x_{mp}, y_{mp}, z_{mp}]$.

Fig. 4.23 describes the autonomous landing system in a MDP model. It consists of

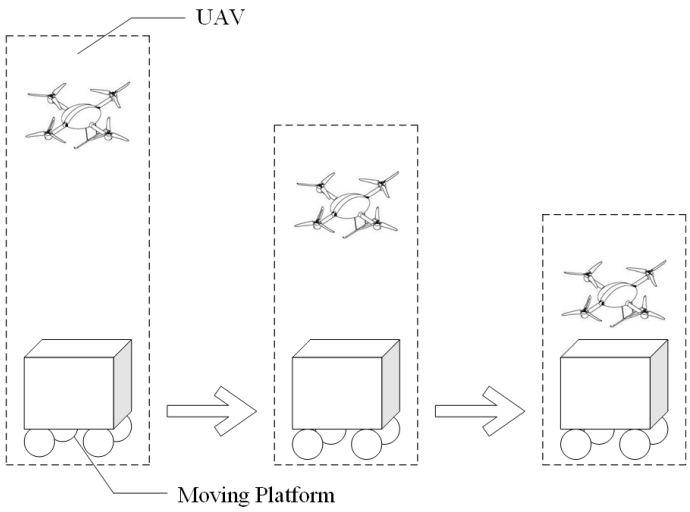


Figure 4.22: UAV autonomous landing system.

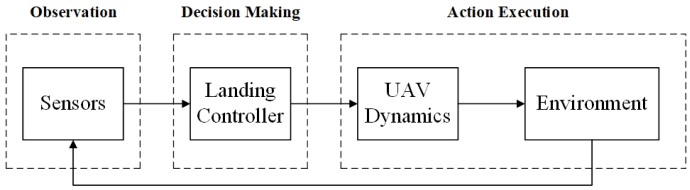


Figure 4.23: MDP model of autonomous landing system.

three parts, observation, decision making and action execution. During each time instant, the position information of UAV and moving platform is observed at the first place. Based on the observed data, the decision-making mechanism will generate an optimal action for the UAV. Then, the generated action will be executed. After that, both the UAV and moving platform will reach a new state. Repeat the above process until UAV has landed on the platform.

4.3.2 State and Action Definition for Autonomous Landing

In the context of RL, the formulation of the investigated problem plays a decisive role in the training efficiency and algorithm performance. There are plenty of possible designs of state and action spaces, as well as the definitions of reward function. Although various

designs may lead to the same result, however, the convergence speed and training effectiveness differ a lot. A good design should minimize the input information, reduce the calculation burden, improve efficiency of training data and prevent divergence. The state space is defined by:

$$S = \{p_{x,land}, p_{y,land}, p_{z,land}, v_{x,land}, v_{y,land}, v_{z,land}, e_{x,land}, e_{y,land}, e_{z,land}\} \tag{4.56}$$

where $p_{x,land}$, $p_{y,land}$ and $p_{z,land}$ are the position of UAV. $v_{x,land}$, $v_{y,land}$, and $v_{z,land}$ are the velocities of UAV along x, y, and z axis. $e_{x,land}$, $e_{y,land}$, and $e_{z,land}$ are the distance errors between UAV and the center of ground platform.

The action space consists of the accelerations along x, y and z axis, which is defined as follows:

$$A = \{a_{x,land}, a_{y,land}, a_{z,land}\} \tag{4.57}$$

where a_x, a_y, a_z are the accelerations along x-axis, y-axis and z-axis. And they need to be bounded $a_{x,land} \in [a_{xmin,land}, a_{xmax,land}]$, $a_{y,land} \in [a_{ymin,land}, a_{ymax,land}]$, $a_{z,land} \in [a_{zmin,land}, a_{zmax,land}]$ in order to meet the dynamic constraints of UAV and reduce the failure during training. The above state and action spaces also need to be normalized from -1 to $+1$ to reduce the difference in values.

4.3.3 Reward Function Design

The reward function of the auto-landing TD3 algorithm consists of parts: distance reward $r_{t,d,land}$, velocity reward $r_{t,v,land}$, penalty $r_{t,p,land}$ and bonus $r_{t,b,land}$.

The distance reward $r_{t,d,land}$ is defined as:

$$r_{t,d,land} = -\omega_{1,land}(dis_t - dis_{t-1}) \tag{4.58}$$

where $\omega_{1,land}$ is a positive constant parameter. dis_t is the distance between UAV and the center of ground platform at time instant t, which is calculated by:

$$dis_t = \sqrt{e_{t,x,land}^2 + e_{t,y,land}^2 + e_{t,z,land}^2} \tag{4.59}$$

The velocity reward $r_{t,v,land}$ is defined as:

$$\begin{aligned} r_{t,v,land} =& \omega_{2,land}\left(\frac{1}{(v_{t,x,land} - v_{t-1,x,land})^2} + \frac{1}{(v_{t,y,land} - v_{t-1,y,land})^2} + \frac{1}{(v_{t,z,land} - v_{t-1,z,land})^2}\right) \\ &+ \omega_{3,land}\left(\frac{v_{t,x,land}}{e_{t,x,land}} + \frac{v_{t,y,land}}{e_{t,y,land}} + \frac{v_{t,z,land}}{e_{t,z,land}}\right) \end{aligned} \tag{4.60}$$

where $\omega_{2,land}$ and $\omega_{3,land}$ are positive constant parameters.

The penalty and bonus are designed to give extra values in order to reinforce the good actions or punish terrible operations. The penalty and bonus only works when the UAV altitude equals to the ground platform altitude, $e_{z,land} \approx 0$ Therefore, the $r_{t,p,land}$ and $r_{t,b,land}$ are defined as:

$$r_{t,p,land} = \begin{cases} -10 \times dis_t, & \text{When } e_{z,land} \approx 0. \\ 0, & \text{Otherwise.} \end{cases} \tag{4.61}$$

$$r_{t,b,land} = \begin{cases} 100, & \text{When } e_{z,land} \approx 0 \ \ and \ \ dis_t \approx 0. \\ 0, & \text{Otherwise.} \end{cases} \tag{4.62}$$

Along with the algorithm training process, the value of penalty and bonus will not only restrict to the land action, but also transmit to previous actions. As a result, good actions will be enhanced and receive larger values in reward function.

The total reward $R_{land}(t)$ is the sum of the above four reward functions:

$$R_{land}(t) = r_{t,d,land} + r_{t,v,land} + r_{t,p,land} + r_{t,b,land} \tag{4.63}$$

4.3.4 Algorithm Training

Compared to other problems using DRL, the control problem is more sensitive and requires more accurate control signals due to the nature of control systems. Specifically, improper control signal, which is equivalent to exploration action in DRL algorithm, can easily result in system failure. A small change between two actions can lead to a big difference in results. However, the most common used exploration method is stochastic exploring policy. The strategy of random exploration may perform well in other problems, but facing control problems, the algorithm will be difficult to find optimal actions and require considerable training time to converge. That is because good control signal always lies in a narrow range. As a result, most of the exploring actions will receive a considerable low reward value and these actions can not be used to train a better performance. Consequently, the training efficiency of stochastic exploring policy is unacceptable low in control problems and it is of great importance to develop a new exploring strategy with higher efficiency and faster converge speed.

Aiming at the requirements above, a transfer-imitation learning training approach (TILT) is proposed. The idea behind TILT is inspired from the transfer learning and imitation learning. Imitation learning can improve the algorithm performance quickly while the algorithm has minimal knowledge of a complex task. For the control problem in this section, imitation learning will help the algorithm skip through the initial exploration phase and avoid a large amount of useless and ineffective explorations. An acceptable performance will be achieved after imitation learning.

Transfer learning can store the gained knowledge of one problem and apply it to another one, which is an outstanding solution for training real-world control problem solvers. For the auto-landing problem studied in this section, it is impractical to be directly trained on the drone-vehicle system due to the following reasons. First, the training process needs a huge amount of flight data, and it takes almost 10 minutes for one complete trial. The time

cost of training is too high to afford. Second, it is quite normal to cause system failure while training. Once the drone crashes, the training process has to be suspended. In addition, the repair cost is inevitable increasing.

The TILT includes three main sequential steps, the first two steps are trained in simulation environment and the last one is trained on real-world drone-vehicle platform.

(1) *Imitation learning* Behaviour cloning is used in the proposed method to achieve imitation learning. Prior to the training phase, a system model of the drone-vehicle landing platform is established in the simulation environment. After that, a PID controller is designed to accomplish the landing mission. Then, the algorithm can focus on learning the PID controller's policy with the help of supervised learning.

(2) *Polishing learning* Although the algorithm obtained a good performance from imitation learning, it is still necessary to seek better performance. Polish learning will use the training method based on exploration and exploitation to improve the overall performance in simulation environment.

(3) *Transfer learning* The transfer learning method used in the proposed method is the simplest one, inductive transfer learning because the source and target domains remains the same. The algorithm applies the knowledge from the source model to optimize the target task. The pre-trained model is starting from a better position than if we were to train it from scratch because it already has knowledge of the features.

The pseudo-code and detailed steps of the proposed training method is shown in Algorithm 5.

Algorithm 5: Proposed transfer-imitation learning training (TILT) method

STEP 1. Imitation learning
Collect the demonstrations from PID controller
Treat the collected demonstrations as action-state pairs $(s_{pid,t}, a_{pid,t})$
Learn π_{pid} policy with supervised learning by minimizing the loss function $L(\phi_{pid}) = \mathbb{E}[(Q(s_{pid,t}, a_{pid,t}) - y_t)^2]$
STEP 2. Polishing learning
Copy policy π_{pid} to π_{simu}
Randomly initialize Q-functions $\phi_{simu,1}$ and $\phi_{simu,2}$
Empty replay buffer $\mathbf{D_{simu}}$, use the TD3 training method described in Algorithm 4
STEP 3. Transfer learning
Transfer the policy and weights from simulation training to the real-world training
π_{simu} to π_{real}, $\phi_{simu,1}$ to $\phi_{real,1}$ and $\phi_{simu,2}$ to $\phi_{real,2}$
Empty replay buffer $\mathbf{D_{real}}$, and use the same training method in Algorithm 4

4.3.5 Results and Analysis

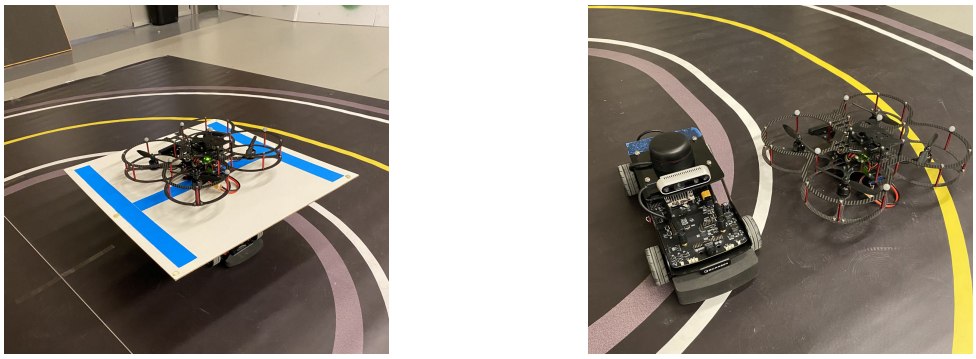
Experimental Setup

The investigated auto-landing system consists of two vehicles, a flying UAV and a ground vehicle with landing platform (see Fig. 4.24(a)). The selected UAV platform is a QDrone from Quanser because it has small size, robust system and good manoeuvrability. The selected ground vehicle platform is a QCar which is shown in Fig. 4.24(b). The QDrone UAV platform has an on-board autopilot and can be communicated within a wireless WiFi channel. Both the UAV and ground vehicle can be located with the help of an OptiTrack system.

The starting position of the UAV in $x - y$ plane is randomly located within 2 meters of the ground vehicle. Due to the limitation of OptiTrack system, the starting altitude of UAV is constantly 1.5 meters.

Training Results

The algorithm training consists of two phases, simulation and real-world. The simulation training phase include imitation learning and polishing learning. While transfer



(a) Auto-landing platform (b) QDrone and QCar

Figure 4.24: Test platform with a UAV and a ground vehicle.

learning belongs to the real-world training phase.

In the training phase, the entire landing maneuver has been carried out 10,000 episodes (which contains over 1.5 million training iterations) in simulation environment. A training trial finishes if the UAV lands on the moving platform or the altitude is lower than the moving platform. The experiment will be repeated under a large variety of situations and the training data will provide a wide range of experiences for the algorithm to learn. Fig. 4.25 illustrates the changes of average accumulated reward during simulation training phase. The imitation learning is executed at the first place, the average reward increases rapidly while learning from the PID controller. When the reward approaches a steady value, the algorithm training will switch to the next step, polishing learning. With the stochastic exploration strategy, the algorithm can achieve a better performance in spite of fluctuations.

For the training in real-world experiment, the total training episodes cannot be as many as it in simulation training. Because it will take 3-5 minutes to complete one training trial. And a fully charged battery can only last for 30 minutes. Therefore, it is necessary to choose good experiment and abandon the ones which fails to land on moving platform. As a result, 173 episodes has been selected as the training data, and the total training experiment has been carried out over 250 times within 16 hours. Based on the selected training data, the

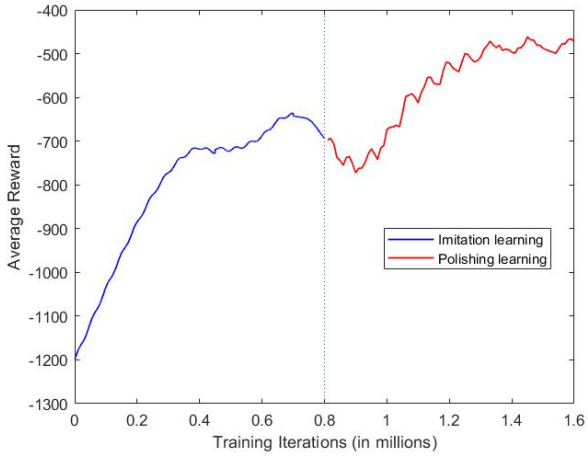


Figure 4.25: Average reward for the simulation training phase.

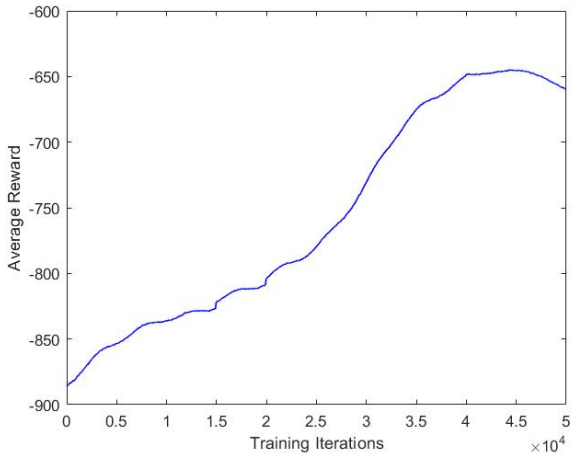
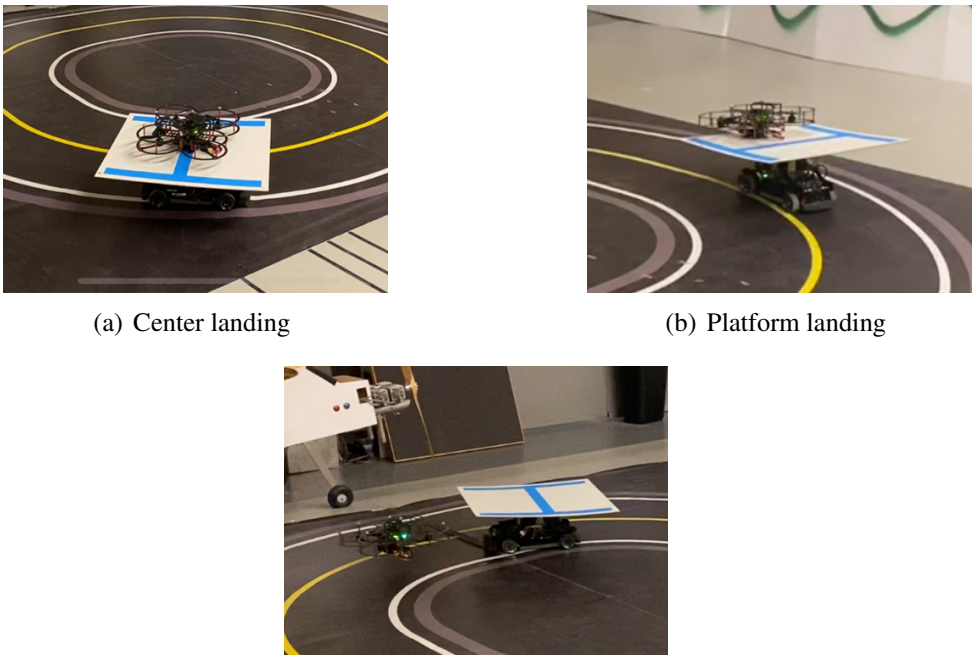


Figure 4.26: Average reward for the real flight training phase.

algorithm has been trained over 50,000 training iterations to achieve a better performance. With increasing training iterations, the trend of average reward is shown in Fig. 4.26.

To validate the effectiveness of the auto-landing controller proposed in this section, landing success rate is introduced as an evaluation indicator of the performance. Two kinds of success landing are considered, one is platform landing and the other is center landing. A center landing refers to the landing trial when the horizontal distance between the UAV and the center of the moving platform is less than the pre-defined threshold. If the horizontal distance is beyond this threshold but less than the platform range, this trial is

called a platform landing. While the UAV fails to land on the platform, it is regarded as an unsuccessful trial.



(a) Center landing

(b) Platform landing

(c) Unsuccessful landing

Figure 4.27: Definitions of different landing trials

Fig. 4.28 shows the landing success rate comparison between different training stages in simulation environment. Results are based on 100 times landing test episodes. The PID controller, which is the imitating target, has a good performance in the simulated landing mission. After the imitation learning, the trained algorithm can perform very close the PID controller. Finally, with the help of polishing learning, the trained algorithm has the highest landing success rate in simulation environment.

In real flight test, the comparison of different training stages is illustrated in Fig. 4.29. Due to the time-consuming issue of real-world test, results are based on 10 times landing test trials. Although the trained algorithm performs very well in simulation, it has poor performance in the real flight tests due to some inevitable factors, such as model uncertainties and sensor inaccuracy. After the optimization of transfer learning, the overall performance

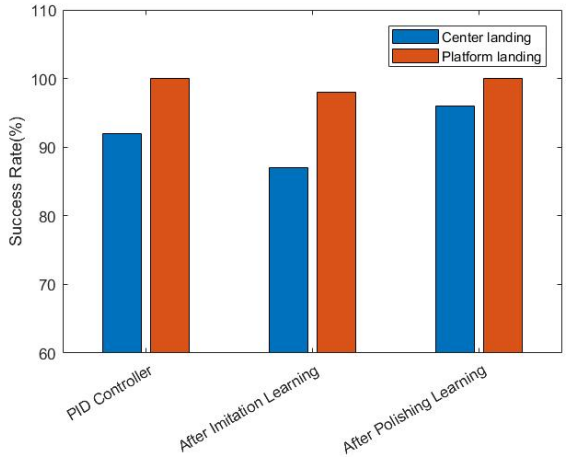


Figure 4.28: Landing success rates of different training stages based on simulation tests.

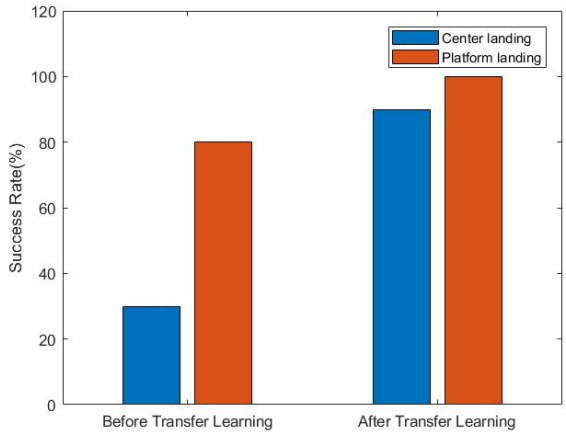


Figure 4.29: Landing success rates of different training stages based on real-world trials.

has improved greatly.

Real Flight Test Results

After algorithm has been well trained by the proposed ITLT method, two cases of flight tests will be carried out to validate the effectiveness and performance of the proposed algorithm.

Case 1: The moving platform performs a shuttle movement

Fig. 4.30 shows the trajectories of the UAV and moving platform in a shuttle movement

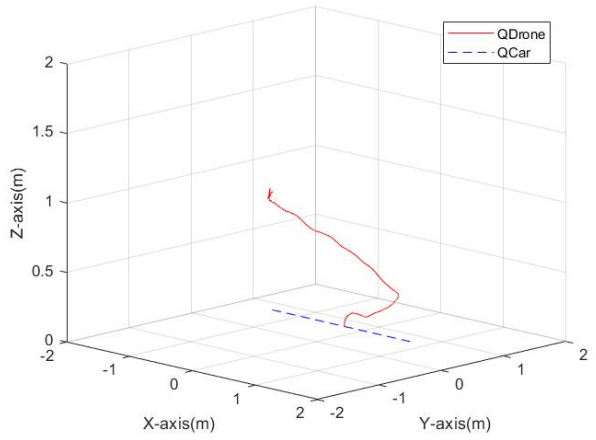


Figure 4.30: Trajectories of QDrone and QCar in a landing mission with shuttle movement.

test. The red line illustrates the trajectory of UAV (QDrone) while the blue dotted represents the movement of the ground platform (QCar).

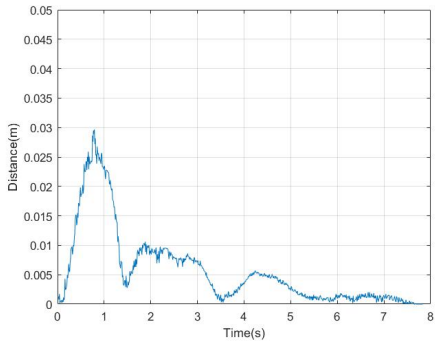
The objective of the auto-landing algorithm is to drive the UAV landing at the center of the platform. Therefore, the distance error between the UAV and the center point is an important evaluation factor of the algorithm. The distance error in the xy plane is shown in Fig. 4.31(a). From the graph, it is obviously that distance error remains at a low level and finally converges too zero before landed. The altitude error along z-axis is displayed in Fig. 4.31(b). The altitude drops smoothly and eventually landed on the platform at altitude $0.2m$.

The velocities are demonstrated in Fig. 4.32.

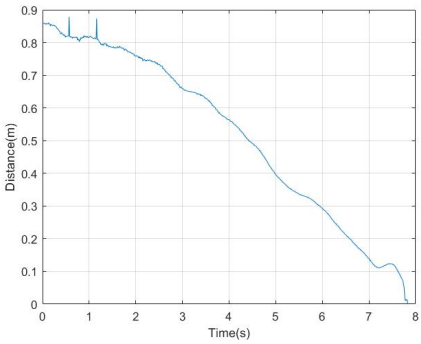
Fig. 68 illustrates the real flight test of auto-landing on a shuttle movement platform. The videos are available at *https://www.youtube.com/watch?v=6tzH4PcijeI*.

Case 2: The moving platform performs a circular movement

Compared to the above shuttle movement test, the following test with a circular movement is more complicated. The trajectories of the UAV and moving platform are shown in Fig. 4.34. Although there is a little shaking at the beginning, the UAV eventually landed on the moving platform smoothly and successfully.



(a) Distance errors in xy plane



(b) Distance errors along z-axis

Figure 4.31: Distance errors between UAV and the center of ground platform (shuttle movement).

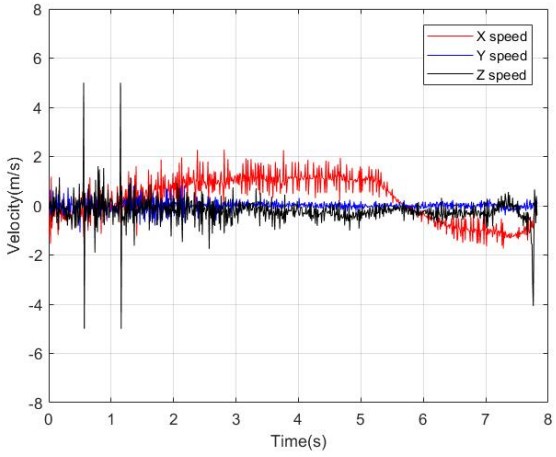


Figure 4.32: Velocities of the UAV in a landing mission with shuttle movement.

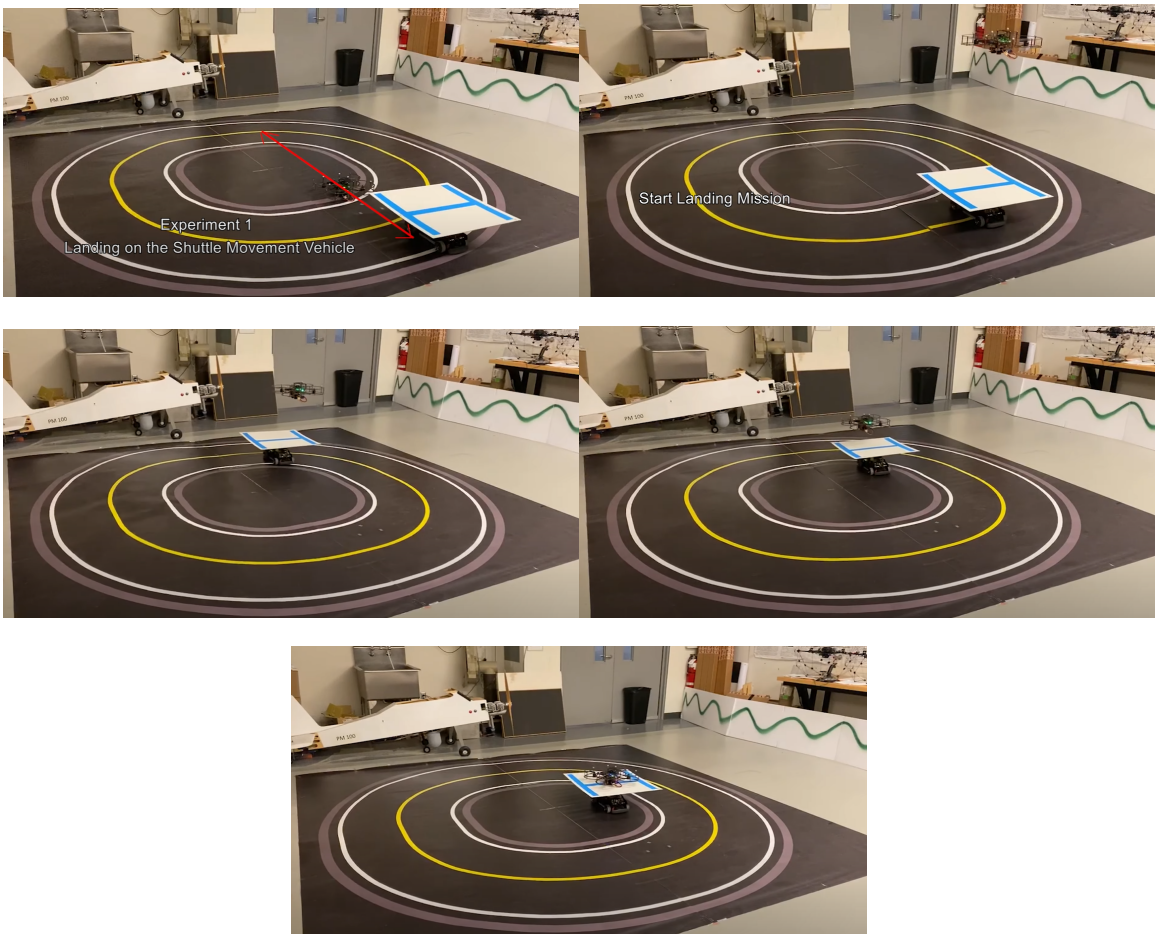


Figure 4.33: Validation in real flight test with a shuttle movement platform. (Video link: https://www.youtube.com/watch?v=6tzH4PcijeI)

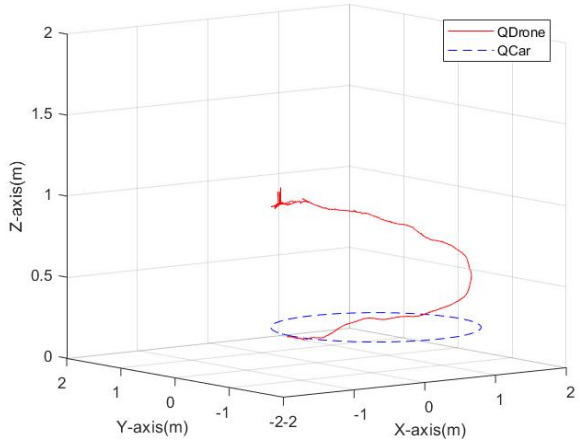
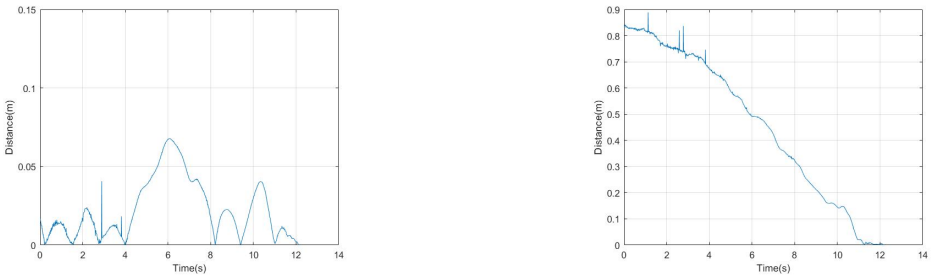


Figure 4.34: Trajectories of QDrone and QCar in a landing mission with shuttle movement.

The distance errors are shown in Fig. 61.



(a) Distance errors in xy plane (b) Distance errors along z-axis

Figure 4.35: Distance errors between UAV and the center of ground platform (circular movement).

The velocities are demonstrated in Fig. 4.36.

Fig. 72 illustrates the real flight test of auto-landing on a circular movement platform. The videos are available at *https://www.youtube.com/watch?v=6tzH4PcijeI*.

4.4 Conclusions

This section proposed three DRL algorithm aiming at different motion control problems. Firstly, a learning-based controller is constructed with the DDPG framework and trained by the DERB technique to solve the path following problem of fixed-wing UAV. A

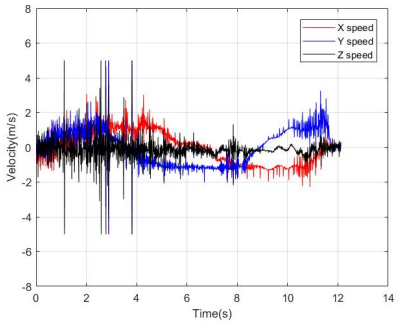


Figure 4.36: Velocities of the UAV in a landing mission with circular movement.

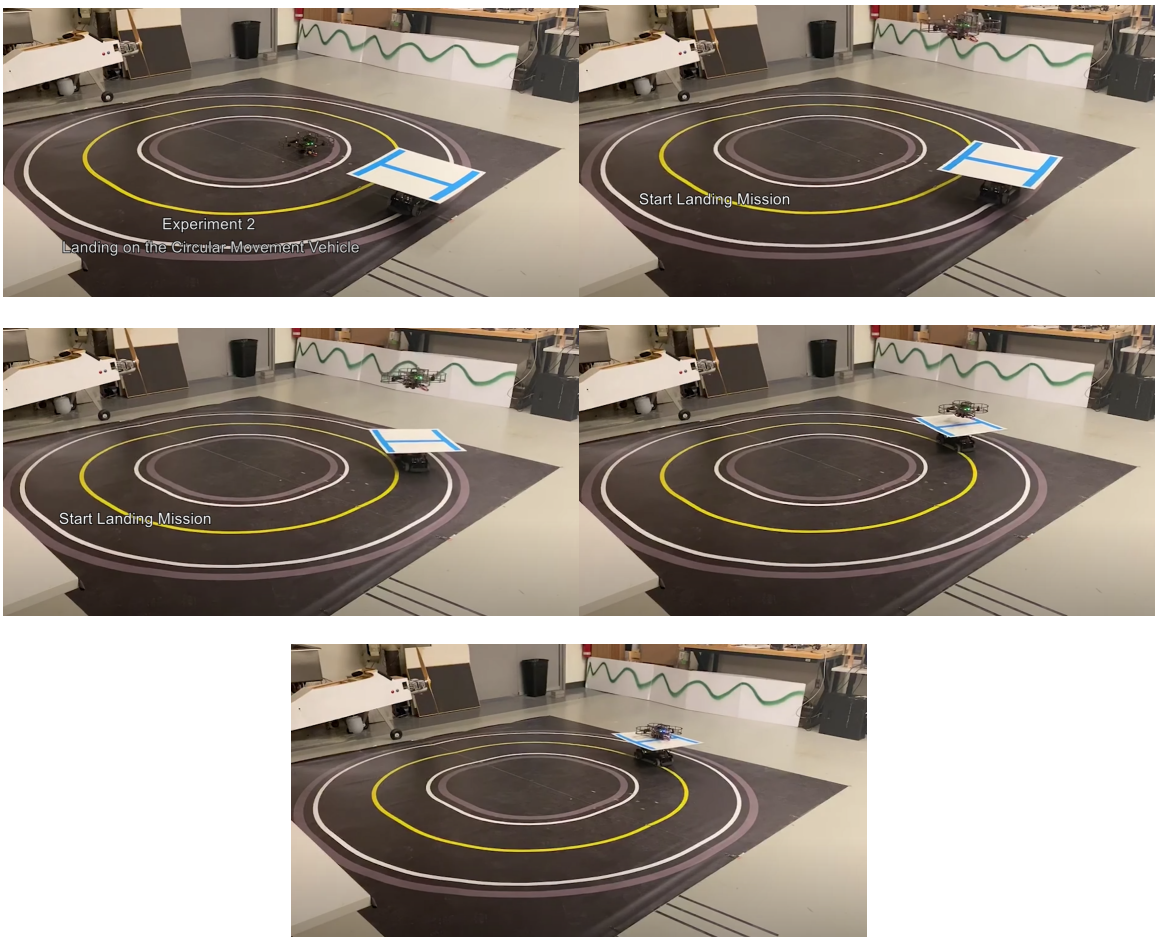


Figure 4.37: Validation in real flight test with a circular movement platform. (Video link: https://www.youtube.com/watch?v=6tzH4PcijeI)

series of simulation results showed the effectiveness, efficiency and adaptability of the proposed method. The comparison of DQN, original DDPG and DERB-DDPG proves that the DERB-DDPG can provide a better performance in the UAV path following problem under the given simulation settings. The second one investigated the problem of formation control for a leader-follower UAV team. The state-of-art deep reinforcement learning algorithm TD3 is studied in this section. First, the follower UAV kinematic model is derived. Then, the twin-delayed DDPG is proposed by introducing a double Q-learning and delayed updates. Finally, the TD3 algorithm is trained with the assistance of prioritized replay buffer with flexible capacity. Two scenarios of the simulation examples are carried out to validate the effectiveness of the proposed method. The third one studies the autonomous landing problem of multi-rotor UAV. A new training method is proposed to address the problem that traditional training approaches cannot be applied to real-world control problems. The proposed ITLT method can efficiently and effectively complete the training.

Chapter 5

Fault-Tolerant Control of Fixed-Wing UAV

This chapter investigates the fault-tolerant control (FTC) of fixed-wing uncrewed aerial vehicle (UAV) using a soft actor-critic (SAC) approach. The proposed SAC combines the algorithm of deep reinforcement learning , the concept of entropy and the mechanism of double Q-learning to construct a passive fault-tolerant controller for fixed-wing UAVs. A special robust term and a new reward function are designed to help the controller improve robustness against system fault. Simulation results in both faulty and non-faulty situations show the feasibility of using SAC in fault-tolerant control and the effectiveness of the proposed SAC algorithm.

Currently, the state-of-the-art DRL algorithms are twin-delayed deep deterministic policy gradient (TD3) and soft actor-critic (SAC). Both TD3 and SAC aim at minimizing the overestimation in DDPG by using double clipped Q-learning. However, SAC showed better learning efficiency in most control applications. Reference [136] also uses SAC as the passive fault-tolerant controller, but it does not include any additional robust mechanism, relying only on the robustness of the DRL algorithm itself. There is still room for improvement on SAC controller.

Inspired by the idea of SAC, this chapter proposes a robust SAC fault-tolerant controller for fixed-wing UAVs. Firstly, based on the actor-critic framework, the SAC algorithm is constructed by introducing two main techniques, the concept of entropy and clipped double Q-learning. Then, a robust term is defined to help the algorithm for a better awareness of the system status, which makes it easier to maintain system performance under system faults. After that, a new reward function is designed to evaluate the value of each experience even under faulty conditions. This is critical to the algorithm performance because reward function leads the direction of learning. Finally, the SAC robust controller is built by combining the robust term and the new reward function.

The main contributions of this chapter are outlined as follows:

(1) *SAC framework for fault-tolerant control*. A model-free fault-tolerant controller is developed based on the state-of-the-art SAC algorithm. The main advancements of SAC are entropy and double Q-learning. The concept of entropy helps the algorithm to have a better exploration of the data. And Double Q-learning also avoids the overestimation of NN.

(2) *Robust term*. Besides the measured state from sensors, the definition of robust term provides an extra knowledge of system status. An additional dimension of information can help the algorithm to generate better decisions, especially in faulty situations.

(3) *Reward function*. A new reward function is designed for sensitively and accurately representing the performance of an action in the flight control problem, especially under system fault conditions. The design of penalty and bonus is an accelerator for training process.

5.1 The Proposed Soft Actor-Critic Algorithm

This section introduces the detailed SAC algorithm and the actor-critic learning framework. Different from DDPG and its extension TD3, SAC uses a stochastic policy instead of deterministic policy to improve the exploration efficiency. Although SAC also has clipped double Q-learning, however, the concept of entropy still makes it a completely new idea.

5.1.1 The Concept of Entropy

The main idea of SAC is entropy regulation. This technique will motivate the policy to increase exploration efficiency and accelerate the learning speed. An entropy term $\mathbb{H}$ is added in the optimal policy π^*_{sac} to measure the randomness of the policy.

$$\begin{aligned}\pi^*_{sac} = \arg\max \mathbb{E}_\pi[\sum_{i=0}^{N} \gamma^i (R_{sac}(s_{t+i}, a_{t+i}) \\ + \alpha_{sac}\mathbb{H}(\pi_{sac}(a_{t+i}|s_{t+i})))]\end{aligned} \tag{5.1}$$

where $\alpha_{sac} > 0$ refers to the temperature parameter and the entropy term $\mathbb{H}$ can be obtained by the following equation:

$$\mathbb{H}(\pi_{sac}(a_t|s_t)) = \mathbb{E}[-\log \pi_{sac}(a_t)|s_t] \tag{5.2}$$

The value function also needs to be slightly changed to a new one with entropy term:

$$V_{\pi_{sac}}(s_t) = \mathbb{E}[\sum_{i=t}^{T} \gamma^{i-t} R_{sac}(s_i, a_i) + \alpha_{sac}\mathbb{H}(\pi_{sac}(a_t|s_t))] \tag{5.3}$$

To sum up briefly, the use of entropy will lead the algorithm favors not only the highest reward but also the most randomly distributed policy. In addition, it can also avoid the policy from converging to a local optimal one. At the same time, a broad range of actions

offers the algorithm with more selections when facing faulty conditions and improves the robustness against faults.

The actor and critic networks are deep neural network (DNN) with parameter θ_{sac} and k_{sac}, respectively. For every state-action pair, the critic Q-function can be expressed by recursive relationship, the Bellman equation also needs to be modified.

$$\begin{aligned} Q_{k_{sac}}(s_t, a_t) = \mathbb{E}[R_{sac}(s_t, a_t) + \gamma(Q_{k_{sac}}(s_{t+1}, a_{t+1}) \\ - \alpha_{sac} \log \pi_{\theta,sac}(a_{t+1}|s_{t+1}))] \end{aligned} \tag{5.4}$$

5.1.2 SAC Algorithm Updating

For the training section or parameter updating phase of the proposed SAC algorithm, there remains a problem that the training data generated by MDP are not identically distributed. These data are easily to be correlated because they are obtained sequentially from exploration. Therefore, instead of using consequent samples, a replay buffer or memory pool is established to store the experience samples. At each time step, a mini-batch of samples will be extracted from the replay buffer. Then, the actor and critic networks are updated using mini-batch samples. After that, a transition sample of the current time instant $(s_t, a_t, R_{sac}(t), s_{t+1})$ will be stored into the replay buffer.

The Q-function in equation (5.4) will be estimated by a function approximator. By minimizing the loss, the function approximator can be updated with the extracted samples $(s_t, a_t, R_{sac}(t), s_{t+1})$:

$$L_Q(k) = \mathbb{E}[(Q_{k,sac}(s_t, a_t) - y_{t,sac}))^2] \tag{5.5}$$

where $y_{t,sac}$ refers to the Q-function target which is defined as:

$$\begin{aligned} y_{t,sac} =& R_{sac}(s_t, a_t) + \gamma(Q_{k,sac}(s_{t+1}, a_{t+1}) \\ & - \alpha_{sac} \log \pi_{\theta,sac}(a_{t+1}|s_{t+1})) \end{aligned} \tag{5.6}$$

There are two Q-functions, $Q_{1,sac}$ and $Q_{2,sac}$, in the SAC. These two Q-functions share one target function which is defined with the smaller Q-value:

$$\begin{aligned} y_{\min,sac} = R_{sac}(s_t, a_t) + \gamma(&\min_{i=1,2} Q_{k_i,sac}(s_{t+1}, a_{t+1}) \\ &- \alpha_{sac} \log \pi_{\theta,sac}(a_{t+1}|s_{t+1})) \end{aligned} \tag{5.7}$$

Both two Q-functions will be updated by regressing to the target $y_{\min,sac}$:

$$\begin{aligned} L_Q(k_1, sac) =&\mathbb{E}[(Q_{k_1,sac}(s_t, a_t) \\ &- (R_{sac}(s_t, a_t) + \gamma(\min_{i=1,2} Q_{k_i,sac}(s_{t+1}, a_{t+1}) \\ &- \alpha_{sac} \log \pi_{\theta,sac}(a_{t+1}|s_{t+1})))))^2] \end{aligned} \tag{5.8}$$

$$\begin{aligned} L_Q(k_2, sac) =&\mathbb{E}[(Q_{k_2,sac}(s_t, a_t) \\ &- (R_{sac}(s_t, a_t) + \gamma(\min_{i=1,2} Q_{k_i,sac}(s_{t+1}, a_{t+1}) \\ &- \alpha_{sac} \log \pi_{\theta,sac}(a_{t+1}|s_{t+1})))))^2] \end{aligned} \tag{5.9}$$

The design of double Q-functions aims at reducing the overestimation of Q-function, which is considered to be a major drawback of DDPG.

In SAC, a dimensional multivariate Gaussian distribution with a diagonal covariance matrix is used to model the actor network. In this chapter, SAC is designed to construct a fault-tolerant controller, as a result, the generated actions of the actor network should be bounded. The actions are going to pass a $tanh$ squashing function to guarantee a finite range. The DNN with parameter θ are approximating the mean vector $\mu_{\theta,sac}$ and the covariance matrix $\sigma^2_{\theta,sac}$. Different from deterministic policy of DDPG, SAC introduces a special trick to re-parameterize the network [137], which is described in (5.10). The exploration action is obtained by applying the squashing function to a combination of the known mean

and standard deviation of stochastic policy.

$$a_{\theta,sac}(s,\xi) = \tanh(\mu_\theta(s) + \sigma_{\theta,sac}(s) \odot \xi), \xi \sim \mathcal{N}(0,1) \tag{5.10}$$

The actor network is updated by following and applying the chain rule to the expected future rewards and entropy with respect to the network parameters θ_{sac} and k_{sac}. The objective function $J_{\pi,sac}(\theta)$ is defined as:

$$\begin{aligned} J_\pi(\theta) = \mathbb{E}[\min_{i=1,2} & Q_{k_i,sac}(s_t, a_\theta(s_t,\xi)) \\ & - \alpha_{sac} \log \pi_{\theta,sac}(a_{\theta,sac}(s_t,\xi)|s_t)] \end{aligned} \tag{5.11}$$

With the increasing number of training steps, the demand for exploring is expected to diminish, hence the optimal entropy is also changing throughout training. As a result, the temperature parameter needs to be obtained as the training steps increase. The loss function $L_{sac}(\alpha)$ is designed to dynamically calculate the best temperature parameter.

$$L(\alpha) = \mathbb{E}[-\alpha_{sac} \log \pi_{\theta,sac}(a_t|s_t) - \alpha_{sac}\mathbb{H}] \tag{5.12}$$

Besides all the above mechanism, a target network is introduced at the last place. The target network is a copy of $Q_{k_{1,2},sac}$ and it is designed for soft updating. At each time instant, after $Q_{k_{1,2},sac}$ and $\pi_{\theta,sac}$ have been updated, the target network will be updated by the following equation:

$$\phi_{tar,i} \leftarrow \rho\phi_{tar,i} + (1-\rho)\phi_i, \qquad i = 1,2 \tag{5.13}$$

where $\rho \ll 1$. The mechanism of target network greatly reduces the negative effect of bad experiences and improves the stability of learning.

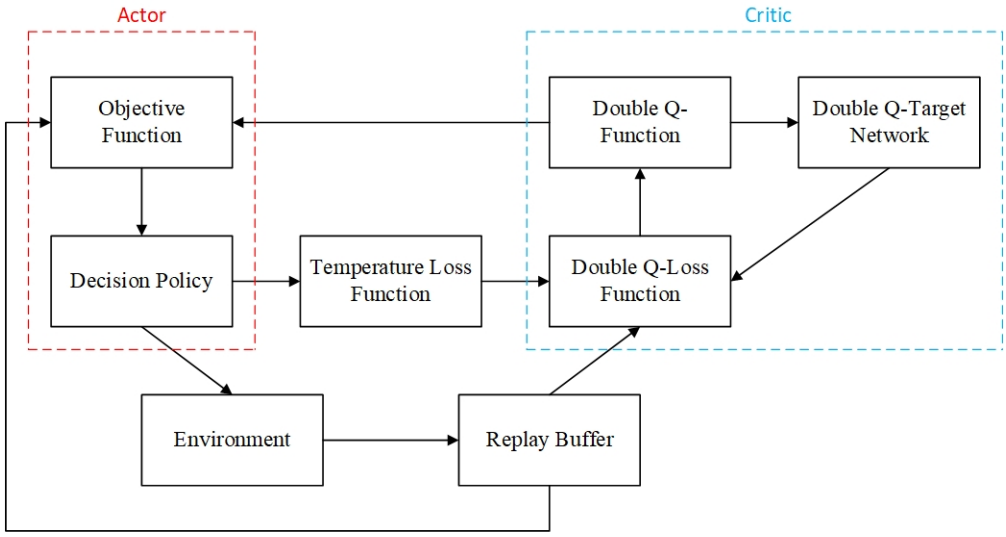


Figure 5.1: Framework of the proposed SAC algorithm.

5.1.3 The SAC Framework and Pseudo-Code

The block diagram of the proposed SAC algorithm is shown in Fig. 5.1. It consists of two cycles, an execution cycle and an update cycle. Generally, the actor network belongs to the execution cycle and makes decisions. While the critic network lies in the update cycle and updates the networks using experience samples from replay buffer. Specifically, at each time instant, the actor network generates an optimal action based on state information. Then, this action will be executed. A reward will be received from the environment and the whole system will enter the next state. The above process is a complete execution cycle. After that, the update cycle is activated. At first place, the critic network will update the double Q-function based on the samples from replay buffer and temperature loss. Then, a Q-value will be returned to the objective function to update the actor network.

Detailed steps of implementing the algorithm and its pseudo-code are introduced in the Algorithm 6.

Algorithm 6: Proposed SAC algorithm

1 h

Randomly initialize the parameters of θ_{sac}, $k_{1,sac}$ and $k_{2,sac}$ for $\pi_{\theta,sac}$ and $Q_{k_{1,2},sac}$
Copy the weights to target network $\phi_{tar,i} \leftarrow k_{i,sac},\ i = 1, 2$
Empty replay buffer and mini-batch. Initialize constant parameters.
for Episode = $[1, m]$ **do**
  for Time step = $[1, T]$ **do**
    Observe the current state s_t and generate the action $a_t \sim \pi_{\theta,sac}(a_t|s_t)$
    Execute the action a_t.
    Observe the next state s_{t+1} and obtain the reward $R_{sac}(t)$
    Store the transition $(s_t, a_t, R_{sac}(t), s_{t+1})$ into the replay buffer.
    if The replay buffer has enough transitions to update **then**
      Sample a mini-batch B_{sac} of transitions from replay buffer.
      Compute targets
      $y_{\min,sac} = R_{sac}(s_t, a_t) + \gamma(\min_{i=1,2} Q_{k_i,sac}(s_{t+1}, a_{t+1}) - \alpha_{sac} \log \pi_{\theta,sac}(a_{t+1}|s_{t+1}))$
      Update Q-functions.
      $\bigtriangledown_{k_i,sac} \frac{1}{|B_{sac}|} \sum_{((s_t,a_t,R_{sac}(t),s_{t+1}) \in B_{sac})} (Q_{k_i,sac}(s_t, a_t) - y_{min,sac})^2$, for $i = 1, 2$
      Update policy by gradient ascent
      $\bigtriangledown_{\theta,sac} \frac{1}{|B_{sac}|} \sum_{s_t \in B_{sac}} (\min_{i=1,2} Q_{k_i,sac}(s_t, a_t) - \alpha_{sac} \log \pi_{\theta,sac}(a_t|s_t))$, for $i = 1, 2$
      Update target networks
      $\phi_{tar,i} \leftarrow \rho\phi_{tar,i} + (1 - \rho)\phi_i$, for $i = 1, 2$
    end if
  end for
end for

5.2 Fault-Tolerant Controller Design

5.2.1 UAV Model

The state space model of a fixed-wing UAV can be described as follows:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{5.14}$$

where A, B, C, and D are system parameter matrices, y is the system output vector. x and u are the system state vector and control input vector, respectively.

$$x = [p, q, r, \theta, \phi, \psi, \alpha_{attack}, \beta, V_{speed}, H_{altitude}]^T \tag{5.15}$$

$$u = [\delta_e, \delta_a, \delta_r, \delta_t]^T \tag{5.16}$$

where the above parameters have normal meanings. It should be noted that the speed and throttle channel is controlled independently. The objective of the designed controller is to track the desired reference signal $D_{ref} = [\psi^d, \phi^d, h^d]^T$. The corresponding measured states are $D_{mr} = [\psi_m, \phi_m, h_m]^T$.

For the convenience of the controller design, the motion of UAV is divided into longitudinal motion and lateral motion. Specifically, the longitudinal motion and lateral motion are $[p, \theta, \alpha, a_{attack}, V_{speed}, H_{altitude}, \delta_e, \delta_t]$ and $[q, r, \phi, \psi, \beta, V_{speed}, \delta_a, \delta_r]$.

5.2.2 Fault-Tolerant SAC Controller Design

Applying SAC in a fault-tolerant controller needs to re-formulate the control problem in a MDP model. The formulation contains the definition of state-action spaces and the design of reward function, and this is critical for the algorithm's performance and learning

speed. There are two independent SAC algorithms in the SAC controller, one is designed for longitudinal motion and the other is for lateral motion. As stated above, the control input vectors are u_{lon} and u_{lat}. Therefore, the action spaces of both longitudinal SAC and lateral SAC are the same as these control inputs.

$$\begin{aligned} A_{lon} &= \{\delta_e\} \\ A_{lat} &= \{\delta_a, \delta_r\} \end{aligned} \tag{5.17}$$

The definition of state space must be unique in both time dimension and location dimension. Specifically, a state presentation such as s_t only leads to one position in the environment. And whenever the UAV reaches this state s_t, the same action a_t applied will result in an unique consequence. It should be noted that any redundancy in the state space is not acceptable because this will inevitably increase the calculation burden, decrease the learning efficiency and result in overall performance degradation. Therefore, the state definition must be accurately and concisely.

For the longitudinal motion, the state space is defined as:

$$S_{lon} = \{D_{ref}, D_{mr}, p, E_h, \eta_e\} \tag{5.18}$$

where E_h is the error between desired altitude h^d and current measured altitude h_m. η_e is a time-varying variable which is defined as follows:

$$\eta_e = e^{\Omega_h} - (\Omega_h - 1)^2 \tag{5.19}$$

where Ω_h is defined as:

$$\Omega_h = \frac{h_t^d - h_{t-1}^d}{h_{m,t} - h_{m,t-1}} \tag{5.20}$$

For the lateral motion, the state space is defined as:

$$S_{lat} = \{D_{ref}, D_{mr}, q, r, E_{\psi}, E_{\phi}, \eta_a, \eta_r\} \tag{5.21}$$

where E_{ψ} and E_{ϕ} are the error between measured and desired angles. η_a and η_r are defined as follows:

$$\begin{aligned} \eta_r &= e^{\Omega_{\psi}} - (\Omega_{\psi} - 1)^2 \\ \eta_a &= \tau_1(e^{\Omega_{\phi}} - (\Omega_{\phi} - 1)^2) + \tau_2\eta_r \end{aligned} \tag{5.22}$$

where τ_1 and τ_2 are the parameters and $\tau_1 \gg \tau_2$. Ω_{ψ} and Ω_{ϕ} are defined as:

$$\begin{aligned} \Omega_{\psi} &= \frac{\psi_t^d - \psi_{t-1}^d}{\psi_{m,t} - \psi_{m,t-1}} \\ \Omega_{\phi} &= \frac{\phi_t^d - \phi_{t-1}^d}{\phi_{m,t} - \phi_{m,t-1}} \end{aligned} \tag{5.23}$$

where η_a, η_e and η_r are the robust terms help to maintain system performance under faulty conditions.

There are several physical observations included in the above defined state and action spaces, such as heights, angles and velocities. However, the range of these observations may differ a lot, this may decrease the learning efficiency. One common approach is to normalize these physically observed variables from -1 to $+1$.

Reward function is generally considered as the most crucial part in DRL algorithms. A well-designed reward function helps to improve the overall performance greatly. The reward function is defined as follows, which consists of three parts:

$$R_{total} = r_{act} + r_{penalty} + r_{bonus} \tag{5.24}$$

where $r_{penalty}$ and r_{bonus} are the penalty and bonus. In this chapter, the actuators are assumed to have saturation limits. If the generated actions u_{lon} or u_{lat} is beyond this limit,

the reward function will receive a huge negative value.

$$r_{penalty} = \begin{cases} -100, \text{ beyond limits} \\ \\ 0, \text{ otherwise} \end{cases} \tag{5.25}$$

On the contrary, r_{bonus} will provide an extra positive reward while the error between reference and measured state is almost zero ($D_{ref} - D_{mr} \approx 0$). The bonus can help algorithm to have a better exploitation on good actions and accelerate the learning speed.

$$r_{bonus} = \begin{cases} 10, \, D_{ref} - D_{mr} \approx 0 \\ \\ 0, \text{ otherwise} \end{cases} \tag{5.26}$$

The third term in reward function (5.24) is action reward r_{act}, which is an evaluation to judge how good or how bad the generated action is. For the longitudinal motion, action reward $r_{act,lon}$ is defined as:

$$r_{act,lon} = -\mu_1(|E_h| + 1)^2 - \mu_2\eta_e + \mu_3\frac{1}{p^2} \tag{5.27}$$

For the lateral motion, action reward $r_{act,lat}$ is defined as:

$$\begin{aligned} r_{act,lat} = & -\mu_4(|E_\phi| + 1)^2 - \mu_5(|E_\psi| + 1)^2 \\ & - \mu_6\eta_a - \mu_7\eta_r + \mu_8\frac{1}{q^2} + \mu_9\frac{1}{r^2} \end{aligned} \tag{5.28}$$

where μ_i $(i = 1, ..., 9)$ are weight parameters to balance or emphasize the importance of terms in the reward function.

5.3 Simulation Results and Analysis

5.3.1 Simulation Setup and Training

The proposed SAC fault-tolerant controller is validated in a fixed-wing UAV model. The longitudinal model can be given as:

$$\begin{aligned} \dot{x}_{lon} &= A_{lon}x_{lon} + B_{lon}u_{lon} \\ y_{lon} &= C_{lon}x_{lon} + D_{lon}u_{lon} \end{aligned} \tag{5.29}$$

The system state and control input vector are defined as:

$$\begin{aligned} x_{lon} &= [\triangle\ u, w, q, \triangle\ \theta, \triangle\ H_{altitude}]^T \\ u_{lon} &= [\delta_e, \delta_t]^T \end{aligned} \tag{5.30}$$

where u and w are the velocity along x-axis and z-axis. δ_t is the throttle control input which is controlled independently. The system and control matrices are given as:

$$A_{lon} = \begin{bmatrix} -0.0069 & 0.0139 & 0 & -9.81 & 0.0706 \\ -0.0905 & -0.3149 & 235.89 & 0 & 0 \\ 0.0004 & -0.0034 & -0.4281 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -235.89 & 0 \end{bmatrix} \tag{5.31}$$

$$B_{lon} = \begin{bmatrix} -0.000057 & 2.943 \\ -5.4714 & 0 \\ -1.159 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, C_{lon} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, D_{lon} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.32}$$

The lateral model can be given as:

$$\begin{aligned} \dot{x}_{lat} &= A_{lat}x_{lat} + B_{lat}u_{lat} \\ y_{lat} &= C_{lat}x_{lat} + D_{lat}u_{lat} \end{aligned} \tag{5.33}$$

$$\begin{aligned} x_{lat} &= [v, p, r, \phi, \psi]^T \\ u_{lat} &= [\delta_a, \delta_r]^T \end{aligned} \tag{5.34}$$

The system and control matrices are given as

$$A_{lat} = \begin{bmatrix} -0.0558 & 0 & -235.9 & 9.81 & 0 \\ -0.0127 & -0.4349 & 0.4142 & 0 & 0 \\ 0.0036 & -0.0061 & -0.1458 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{5.35}$$

$$B_{lat} = \begin{bmatrix} 0 & 1.7188 \\ -0.1433 & 0.1146 \\ 0.0038 & -0.4859 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, C_{lat} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, D_{lat} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.36}$$

Choosing suitable hyper parameters for the proposed SAC controllers can significantly improve training efficiency. Specifically, the learning rate is $1 \times 10 - 3$, the discount factor is 0.98, the smooth parameter ρ is 3×10^{-3}. The deep neural network has three hidden layers and each layer has 32 units.

The SAC algorithm will be trained twice. The first time is trained on the healthy system to obtain a good control performance and provide a reasonable training base for faulty condition. In the second time, SAC is trained under actuator faults to get a strong robustness against faults.

The SAC algorithm will be trained one episode by another. Each episode is an independent training process and it contains 1,000 training steps. Since the control of aircraft is a consequent decision process, the error in previous steps will be accumulated. As a result, the aircraft may be too far away from the reference, especially in faulty conditions. Exploring in these situations is kind of useless and exploiting this experience may cause negative effect on learning performance. Therefore, the design of training episode can effectively eliminate cumulative errors and interrupt the training data from deteriorating.

Fig. 5.2 shows the curve of average reward with training episode increases. The blue line refers to the healthy system while the red line represents the faulty system. From the curves it is obviously that the average reward keeps increasing which means the performance of SAC algorithm is becoming better.

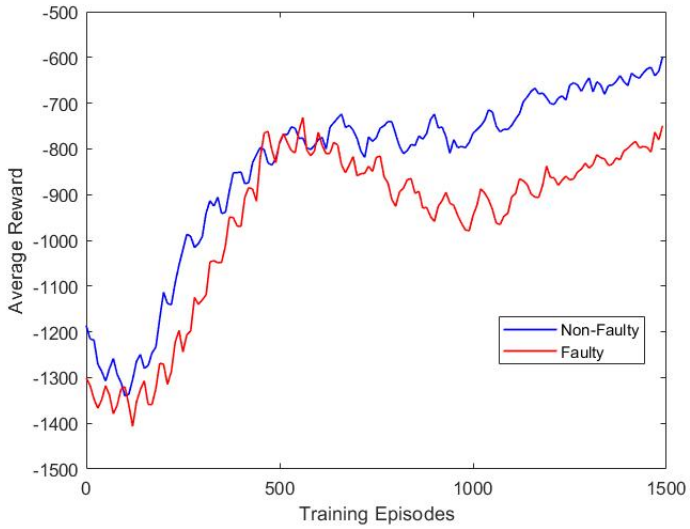


Figure 5.2: Average reward of training episodes.

5.3.2 Non-Faulty Condition

Since the proposed SAC controller is passive FTC, the performance on healthy system is also one important goal. As the Fig. 5.3 shows, the SAC controller is trying to minimize the overall errors and tracks the reference well. Blue lines are references $D_{ref} = [\psi^d, \phi^d, h^d]^T$ and red lines are measured states $D_{mr} = [\psi_m, \phi_m, h_m]^T$.

5.3.3 Loss of Effectiveness Condition

The loss of effectiveness is considered as the actuator fault in the simulation. As shown in Fig. 5.4, 75% reduction of aileron, 80% reduction of elevator and 70% reduction of rudder are applied at time $= 10s$. Initially, the aircraft's altitude drops a bit. The situation is even worse in ϕ and ψ. But soon, the aircraft keeps up with reference. Although it is not as perfect as the healthy system, the overall performance of the SAC controller is within acceptable limits.

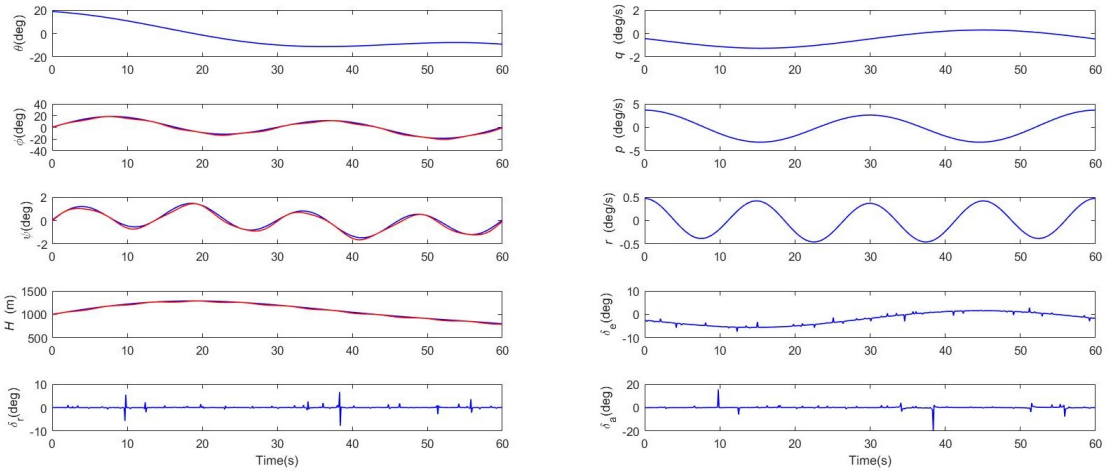


Figure 5.3: Tracking responses of the proposed SAC algorithm and system behavior under non-faulty condition.

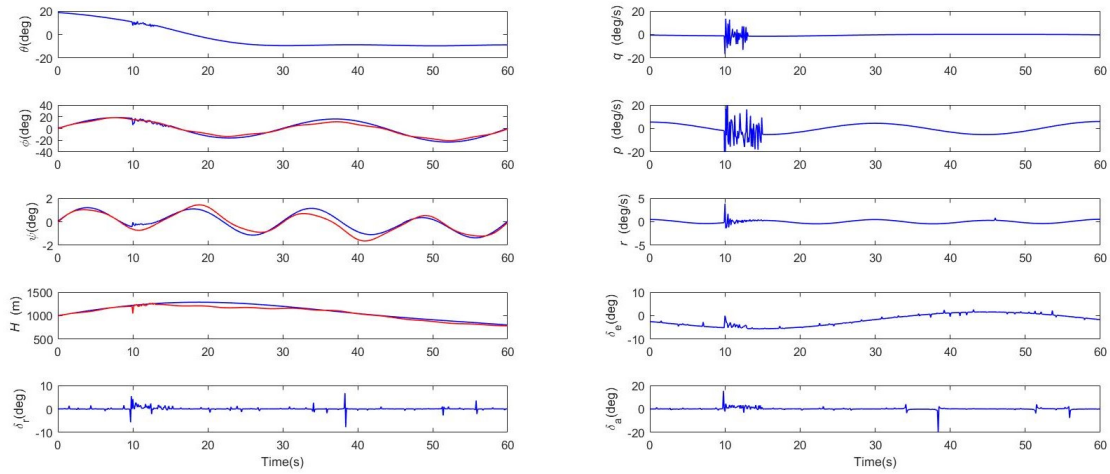


Figure 5.4: Tracking responses of the proposed SAC algorithm and system behavior under faulty condition (loss of effectiveness).

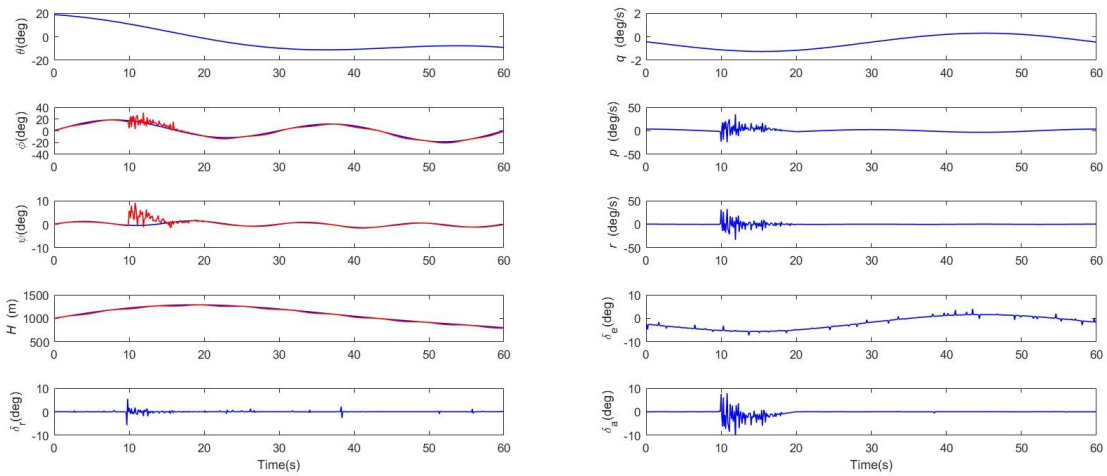


Figure 5.5: Tracking responses of the proposed SAC algorithm and system behavior under faulty condition (rudder stuck).

5.3.4 Rudder Stuck Condition

Besides the loss of effectiveness, rudder stuck is considered as another actuator fault. At time $= 10s$, the rudder is stuck at $5°$ and the system response is shown in Fig. 5.5. Despite the severe actuator fault, the system remains stable and robust. The errors in yaw and roll motion are inevitably turned to be big but they recover to normal eventually.

5.3.5 Comparison Between Traditional PID Controller and the Proposed SAC Controller

To further validate the effectiveness of the proposed SAC controller, a comparison between a PID controller with fine-tuned parameters is carried out.

Fig. 5.6 shows the system responses of two controllers under loss of effectiveness faulty conditions. The well-tuned PID controller is black line. It is obviously both two controllers have good reliability against loss of effectiveness fault. However, the performances differs. When fault occurs, the performance of PID controller degrades a lot while the performance of proposed SAC controller is less fluctuating. But after a while, the performance of PID

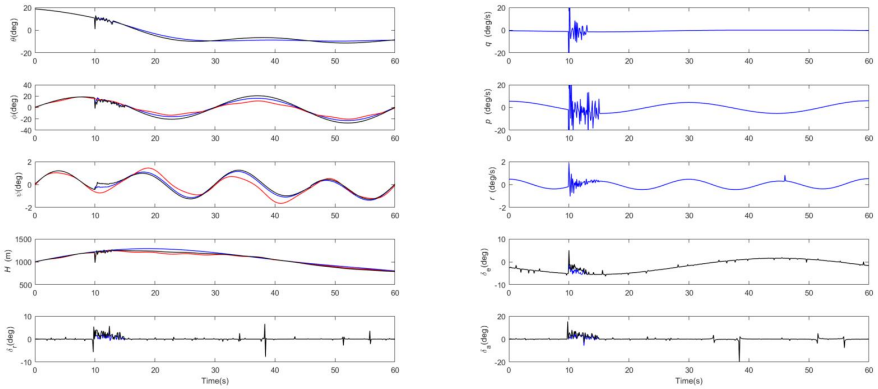


Figure 5.6: System responses of both controllers under faulty condition (loss of effectiveness).

controller is obviously better than the SAC controller. In summary, these two algorithms have their own advantages and disadvantages.

5.4 Conclusion

In this chapter, a new fault-tolerant controller based on deep reinforcement learning algorithm is proposed for a fixed-wing UAV in the presence of partial loss of effectiveness and stuck fault situations. The new SAC controller is constructed by applying the concept of entropy in an actor-critic framework. Besides, the design of reward function and the definition of robust term help the SAC controller to maintain system performance under the considered actuator faults. Numerical simulation results prove that the proposed SAC controller has a strong robustness against loss of effectiveness and rudder stuck faults. Compared to other existing literature which is also using SAC as passive fault-tolerant controller, the one proposed in this chapter has faster recovering time and less tracking errors.

Chapter 6

Conclusions and Future Works

6.1 Conclusions

In this thesis, several uncrewed aerial vehicle (UAV) issues related to real-time path planning, motion control and fault-tolerant control (FTC) are well studied, which includes:

• Comprehensive literature reviews on path planning, motion control and FTC are provided.

• Two real-time path planning algorithm are designed. The first one proposes a new state space definition method which breaks the gap between Q-learning algorithm and real-time path planning. The second one designs a new reward function inspired by the human pedestrian behavior. With the help of the proposed reward function, the trained algorithm can plan a safe and shortest path in a complex environment.

• A new learning-based controller is constructed with the deep deterministic policy gradient (DDPG) framework and trained by the double experience replay buffer (DERB) technique to solve the path following problem of fixed-wing UAV. A series of simulation results showed the effectiveness, efficiency and adaptability of the proposed method. The comparison of DQN, original DDPG and DERB-DDPG proves that the DERB-DDPG can provide a better performance in the UAV path following problem under the given simulation

settings.

• A new DRL algorithm twin-delayed DDPG (TD3) is designed for the UAV formation control problem. The TD3 algorithm is proposed by introducing a double Q-learning and delayed updates. Then, the algorithm is trained with the assistance of prioritized replay buffer with flexible capacity. Two scenarios of the simulation examples are carried out to validate the effectiveness of the proposed method.

• A new DRL training method named imitate-transfer learning training (ITLT) is designed for training DRL algorithms in real-world control applications. The ITLT is validated in the auto-landing control problem and the experimental results show the effectiveness of the proposed training method.

• A new fault-tolerant controller based on soft actor-critic (SAC) algorithm is proposed for a fixed-wing UAV in the presence of partial loss of effectiveness and stuck fault situations. The new SAC controller is constructed by applying the concept of entropy in an actor-critic framework. Besides, the design of reward function and the definition of robust term help the SAC controller to maintain system performance under the considered actuator faults. Numerical simulation results prove that the proposed SAC controller has a strong robustness against loss of effectiveness and rudder stuck faults.

6.2 Future Works

Following the current research in this thesis, the following future directions are outlined:

• Most of the developed methods in this thesis is validated in simulation, more experimental tests are needed in the future.

• Add SLAM and vision processing technologies in the path planning algorithm to make it a more intelligent and broaden its application in our daily life.

• The leader-follower formation control problem is well investigated, while more complicated and challenging issues, such as distributed control, are expected to be studied in the future.

• The faults considered in this thesis only occur in actuators, while the sensor and communication faults are not included, though they are of significance for safety critical control system design as well.

Bibliography

[1] D. W. Casbeer, R. W. Beard, T. W. McLain, S.-M. Li, and R. K. Mehra, "Forest fire monitoring with multiple small UAVs," in *American Control Conference (ACC)*, Portland, USA, 2005.

[2] C. Yuan, Z. Liu, and Y. Zhang, "UAV-based forest fire detection and tracking using image processing techniques," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, Denver, USA, 2015.

[3] C. Yuan, Y. Zhang, and Z. Liu, "A survey on technologies for automatic forest fire monitoring, detection, and fighting using unmanned aerial vehicles and remote sensing techniques," *Canadian Journal of Forest Research*, vol. 45, no. 7, pp. 783–792, 2015.

[4] Z. Li, Y. Liu, R. Walker, R. Hayward, and J. Zhang, "Towards automatic power line detection for a UAV surveillance system using pulse coupled neural filter and an improved hough transform," *Machine Vision and Applications*, vol. 21, no. 5, pp. 677–686, 2010.

[5] E. Semsch, M. Jakob, D. Pavlicek, and M. Pechoucek, "Autonomous UAV surveillance in complex urban environments," in *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, Washington, USA, 2009.

[6] D. Perez, I. Maza, F. Caballero, D. Scarlatti, E. Casado, and A. Ollero, "A ground control station for a multi-UAV surveillance system," *Journal of Intelligent and Robotic Systems*, vol. 69, no. 1, pp. 119–130, 2013.

[7] D. Popescu, L. Ichim, and T. Caramihale, "Flood areas detection based on UAV surveillance system," in *19th International Conference on System Theory, Control and Computing (ICSTCC)*, Cheile Gradistei, Romania, 2015.

[8] K. T. San, S. J. Mun, Y. H. Choe, and Y. S. Chang, "UAV delivery monitoring system," in *Asia Conference on Mechanical and Aerospace Engineering (ACMAE)*, Yokohama, Japan, 2017.

[9] S. Jung and H. Kim, "Analysis of amazon prime air UAV delivery service," *Journal of Knowledge Information Technology and Systems*, vol. 12, no. 2, pp. 253–266, 2017.

[10] H. Huang, A. V. Savkin, and C. Huang, "Optimal control of a hybrid UAV/train parcel delivery system," in *Chinese Control Conference (CCC)*, Guangzhou, China, 2019.

[11] X. Liu and N. Ansari, "Resource allocation in UAV-assisted M2M communications for disaster rescue," *IEEE Wireless Communications Letters*, vol. 8, no. 2, pp. 580–583, 2018.

[12] A. Wang, X. Ji, D. Wu, X. Bai, N. Ding, J. Pang, S. Chen, X. Chen, and D. Fang, "Guideloc: UAV-assisted multitarget localization system for disaster rescue," *Mobile Information Systems*, 2017.

[13] T. Ahn, J. Seok, I. Lee, and J. Han, "Reliable flying IoT networks for UAV disaster rescue operations," *Mobile Information Systems*, 2018.

[14] S. J. Kim, Y. Jeong, S. Park, K. Ryu, and G. Oh, "A survey of drone use for entertainment and AVR (augmented and virtual reality)," in *Augmented Reality and Virtual Reality*, 2018, pp. 339–352.

[15] M. De Marsico and A. Spagnoli, "Using hands as an easy UAV joystick for entertainment applications," in *13th Biannual Conference of the Italian SIGCHI Chapter: Designing the Next Interaction*, Padova, Italy, 2019.

[16] K. P. Valavanis and G. J. Vachtsevanos, *Handbook of unmanned aerial vehicles*. Springer, 2015, vol. 1.

[17] S. Y. Choi and D. Cha, "Unmanned aerial vehicles using machine learning for autonomous flight; state-of-the-art," *Advanced Robotics*, vol. 33, no. 6, pp. 265–277, 2019.

[18] A. I. Khan and Y. Al-Mulla, "Unmanned aerial vehicle in the machine learning environment," *Procedia Computer Science*, vol. 160, pp. 46–53, 2019.

[19] B. Zhang, Z. Mao, W. Liu, and J. Liu, "Geometric reinforcement learning for path planning of UAVs," *Journal of Intelligent and Robotic Systems*, vol. 77, no. 2, pp. 391–409, 2015.

[20] B. Kim and J. Pineau, "Socially adaptive path planning in human environments using inverse reinforcement learning," *International Journal of Social Robotics*, vol. 8, no. 1, pp. 51–66, 2016.

[21] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, P. De La Puente, and P. Campoy, "A deep reinforcement learning strategy for UAV autonomous landing on a moving platform," *Journal of Intelligent and Robotic Systems*, vol. 93, no. 1, pp. 351–366, 2019.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[23] J. Wu, H. He, J. Peng, Y. Li, and Z. Li, "Continuous reinforcement learning of energy management with deep q network for a power split hybrid electric bus," *Applied Energy*, vol. 222, pp. 799–811, 2018.

[24] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[25] J. Fan, Z. Wang, Y. Xie, and Z. Yang, "A theoretical analysis of deep Q-learning," in *Learning for Dynamics and Control*, Zurich, Switzerland, 2020.

[26] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, Beijing, China, 2014.

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *ArXiv Preprint arXiv:1509.02971*, 2015.

[28] S. Guo, X. Zhang, Y. Zheng, and Y. Du, "An autonomous path planning model for unmanned ships based on deep reinforcement learning," *Sensors*, vol. 20, no. 2, p. 426, 2020.

[29] K. F. Wan, X. G. Gao, Z. J. Hu, and G. F. Wu, "Robust motion control for UAV in dynamic uncertain environments using deep reinforcement learning," *Remote Sensing*, vol. 12, no. 4, p. 640, 2020.

[30] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, Stockholm, Sweden, 2018.

[31] W. Liu, S. Y. Chen, and H. X. Huang, "Actor-critic learning hierarchical sliding mode control for a class of underactuated systems," in *Chinese Automation Congress (CAC)*, Hangzhou, China, 2019.

[32] D. Gao, H. Zhang, C. Li, and X. Gao, "Satellite attitude control with deep reinforcement learning," in *Chinese Automation Congress (CAC)*, Shanghai, China, 2020.

[33] J.-S. Chou, M.-Y. Cheng, Y.-M. Hsieh, I.-T. Yang, and H.-T. Hsu, "Optimal path planning in real time for dynamic building fire rescue operations using wireless sensors and visual guidance," *Automation in Construction*, vol. 99, pp. 1–17, 2019.

[34] M. F. Ozkan, L. R. G. Carrillo, and S. A. King, "Rescue boat path planning in flooded urban environments," in *IEEE International Symposium on Measurement and Control in Robotics (ISMCR)*, Houston, USA, 2019.

[35] H. Liu, J. Ge, Y. Wang, J. Li, K. Ding, Z. Zhang, Z. Guo, W. Li, and J. Lan, "Multi-UAV optimal mission assignment and path planning for disaster rescue using adaptive genetic algorithm and improved artificial bee colony method," *Actuators*, vol. 11, no. 1, p. 4, 2021.

[36] R. Almadhoun, T. Taha, L. Seneviratne, and Y. Zweiri, "A survey on multi-robot coverage path planning for model reconstruction and mapping," *SN Applied Sciences*, vol. 1, no. 8, pp. 1–24, 2019.

[37] J. hua Yu, S. nyeong Heo, J.-s. Shin, and H.-h. Lee, "Fire area detection based on convolutional neural network and improved A* path planning," in *International Conference on Information and Communication Technology Robotics (ICT-ROBOT)*, Busan, Korea, 2018.

[38] J. Chen, C. Du, Y. Zhang, P. Han, and W. Wei, "A clustering-based coverage path planning method for autonomous heterogeneous UAVs," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[39] Y. Lin and S. Saripalli, "Sampling-based path planning for UAV collision avoidance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 11, pp. 3179–3192, 2017.

[40] A. Al-Hilo, M. Samir, C. Assi, S. Sharafeddine, and D. Ebrahimi, "UAV-assisted content delivery in intelligent transportation systems-joint trajectory planning and cache management," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 5155–5167, 2020.

[41] H. S. Munawar, H. Inam, F. Ullah, S. Qayyum, A. Z. Kouzani, and M. Mahmud, "Towards smart healthcare: UAV-based optimized path planning for delivering COVID-19 self-testing kits using cutting edge technologies," *Sustainability*, vol. 13, no. 18, p. 10426, 2021.

[42] J. Zhang and H. Huang, "Occlusion-aware UAV path planning for reconnaissance and surveillance," *Drones*, vol. 5, no. 3, p. 98, 2021.

[43] J. Li, J. Chen, P. Wang, and C. Li, "Sensor-oriented path planning for multiregion surveillance with a single lightweight UAV SAR," *Sensors*, vol. 18, no. 2, p. 548, 2018.

[44] P. Vadakkepat, K. C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," in *Congress on Evolutionary Computation*, Busan, Korea, 2000.

[45] Y.-b. Chen, G.-c. Luo, Y.-s. Mei, J.-q. Yu, and X.-l. Su, "UAV path planning using artificial potential field method updated by optimal control theory," *International Journal of Systems Science*, vol. 47, no. 6, pp. 1407–1420, 2016.

[46] P. Bhattacharya and M. L. Gavrilova, "Voronoi diagram in optimal path planning," in *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*, Calgary, Canada, 2007.

[47] E. S. Low, P. Ong, and K. C. Cheah, "Solving the optimal path planning of a mobile robot using improved Q-learning," *Robotics and Autonomous Systems*, vol. 115, pp. 143–161, 2019.

[48] R. A. Saeed, D. R. Recupero, and P. Remagnino, "A boundary node method for path planning of mobile robots," *Robotics and Autonomous Systems*, vol. 123, p. 103320, 2020.

[49] G. Han, H. Xu, J. Jiang, L. Shu, T. Hara, and S. Nishio, "Path planning using a mobile anchor node based on trilateration in wireless sensor networks," *Wireless Communications and Mobile Computing*, vol. 13, no. 14, pp. 1324–1336, 2013.

[50] Y. Shi, Q. Li, S. Bu, J. Yang, and L. Zhu, "Research on intelligent vehicle path planning based on rapidly-exploring random tree," *Mathematical Problems in Engineering*, 2020.

[51] S. Rodriguez, X. Tang, J.-M. Lien, and N. M. Amato, "An obstacle-based rapidly-exploring random tree," in *IEEE International Conference on Robotics and Automation (ICRA)*, Orlando, USA, 2006.

[52] Z. Wang and J. Cai, "Probabilistic roadmap method for path-planning in radioactive environment of nuclear facilities," *Progress in Nuclear Energy*, vol. 109, pp. 113–120, 2018.

[53] L. E. Kavraki, M. N. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 166–171, 1998.

[54] C. Lamini, S. Benhlima, and A. Elbekri, "Genetic algorithm based approach for autonomous mobile robot path planning," *Procedia Computer Science*, vol. 127, pp. 180–189, 2018.

[55] U. Aybars, "Path planning on a cuboid using genetic algorithms," *Information Sciences*, vol. 178, no. 16, pp. 3275–3287, 2008.

[56] M. Brand, M. Masuda, N. Wehner, and X.-H. Yu, "Ant colony optimization algorithm for robot path planning," in *International Conference on Computer Design and Applications*, Qinhuangdao, China, 2010.

[57] J. Liu, J. Yang, H. Liu, X. Tian, and M. Gao, "An improved ant colony algorithm for robot path planning," *Soft Computing*, vol. 21, no. 19, pp. 5829–5839, 2017.

[58] C. J. C. H. Watkins, *Learning from delayed rewards*. King's College, Cambridge United Kingdom, 1989.

[59] W. D. Smart and L. P. Kaelbling, "Practical reinforcement learning in continuous spaces," in *17th International Conference on Machine Learning (ICML)*, Stanford, 2000.

[60] ——, "Effective reinforcement learning for mobile robots," in *IEEE International Conference on Robotics and Automation*, Washington. USA, 2002.

[61] P. K. Das, A. Konar, R. Janarthanan *et al.*, "Extended Q-learning algorithm for path-planning of a mobile robot," in *Asia-Pacific Conference on Simulated Evolution and Learning*, Kanpur, India, 2010.

[62] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved Q-learning for path planning of a mobile robot," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 5, pp. 1141–1153, 2013.

[63] D. B. Aranibar and P. J. Alsina, "Reinforcement learning-based path planning for autonomous robots," in *Congress of the Brazilian Computer Society*, Foz do Iguacu, Brazil, 2004.

[64] H. R. Beom and H. S. Cho, "A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 3, pp. 464–477, 1995.

[65] N. Yung and C. Ye, "Self-learning fuzzy navigation of mobile vehicle," in *Third International Conference on Signal Processing (ICSP'96)*, Beijing, China, 1996.

[66] G.-S. Yang, E.-K. Chen, and C.-W. An, "Mobile robot navigation using neural Q-learning," in *International Conference on Machine Learning and Cybernetics*, Shanghai, China, 2004.

[67] B. Rubí, R. Pérez, and B. Morcego, "A survey of path following control strategies for UAVs focused on quadrotors," *Journal of Intelligent and Robotic Systems*, vol. 98, no. 2, pp. 241–265, 2020.

[68] L. Qian and H. H. Liu, "Path-following control of a quadrotor UAV with a cable-suspended payload under wind disturbances," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 3, pp. 2021–2029, 2019.

[69] B. Anderson, B. Fidan, C. Yu, and D. Walle, *UAV formation control: Theory and Application*. Springer, 2008.

[70] Y. Kuriki and T. Namerikawa, "Consensus-based cooperative formation control with collision avoidance for a multi-UAV system," in *American Control Conference*, Portland, USA, 2014.

[71] Y. Guo, J. Guo, C. Liu, H. Xiong, L. Chai, and D. He, "Precision landing test and simulation of the agricultural UAV on apron," *Sensors*, vol. 20, no. 12, p. 3369, 2020.

[72] K. E. Wenzel, A. Masselli, and A. Zell, "Automatic take off, tracking and landing of a miniature UAV on a moving carrier vehicle," *Journal of intelligent and robotic systems*, vol. 61, no. 1, pp. 221–238, 2011.

[73] T. Yamasaki, H. Takano, and Y. Baba, "Robust path-following for UAV using pure pursuit guidance," in *Aerial Vehicles*, 2009.

[74] P. Sujit, S. Saripalli, and J. B. Sousa, "An evaluation of UAV path following algorithms," in *European Control Conference (ECC)*, Zurich, Switzerland, 2013.

[75] T. I. Fossen, M. Breivik, and R. Skjetne, "Line-of-sight path following of underactuated marine craft," *IFAC Proceedings Volumes*, vol. 36, no. 21, pp. 211–216, 2003.

[76] R. Rysdyk, "UAV path following for constant line-of-sight," in *2nd AIAA" Unmanned Unlimited" Conf. and Workshop and Exhibit*, San Diego, USA, 2003.

[77] M. Breivik, *Topics in guided motion control of marine vehicles*. Tapir Uttrykk, 2010.

[78] J. Woo, C. Yu, and N. Kim, "Deep reinforcement learning-based controller for path following of an unmanned surface vehicle," *Ocean Engineering*, vol. 183, pp. 155–166, 2019.

[79] C. Nie, Z. Zheng, and M. Zhu, "Three-dimensional path-following control of a robotic airship with reinforcement learning," *International Journal of Aerospace Engineering*, 2019.

[80] Z. Yu, Y. Zhang, B. Jiang, J. Fu, Y. Jin, and T. Chai, "Composite adaptive disturbance observer-based decentralized fractional-order fault-tolerant control of networked UAVs," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2020, published online, doi: 10.1109/TSMC.2020.3010678.

[81] I. Maza, F. Caballero, J. Capitán, J. R. Martínez-de Dios, and A. Ollero, "Experimental results in multi-UAV coordination for disaster management and civil security applications," *Journal of Intelligent and Robotic Systems*, vol. 61, no. 1, pp. 563–585, 2011.

[82] Z. Yu, Y. Zhang, B. Jiang, C.-Y. Su, J. Fu, Y. Jin, and T. Chai, "Distributed fractional-order intelligent adaptive fault-tolerant formation-containment control of two-layer networked unmanned airships for safe observation of a smart city," *IEEE Transactions on Cybernetics*, 2021, published online, doi: 10.1109/TCYB.2021.3052875.

[83] G. Pajares, "Overview and current status of remote sensing applications based on unmanned aerial vehicles (UAVs)," *Photogrammetric Engineering and Remote Sensing*, vol. 81, no. 4, pp. 281–330, 2015.

[84] Z. Yu, Y. Zhang, B. Jiang, C.-Y. Su, J. Fu, Y. Jin, and T. Chai, "Fractional-order adaptive fault-tolerant synchronization tracking control of networked fixed-wing UAVs

against actuator-sensor faults via intelligent learning mechanism," *IEEE Transactions on Neural Networks and Learning Systems*, 2021, published online, doi: 10.1109/TNNLS.2021.3059933.

[85] X. W. Dong, Y. Zhou, Z. Ren, and Y. S. Zhong, "Time-varying formation tracking for second-order multi-agent systems subjected to switching topologies with application to quadrotor formation flying," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 6, pp. 5014–5024, 2016.

[86] X. W. Dong, B. C. Yu, Z. Y. Shi, and Y. S. Zhong, "Time-varying formation control for unmanned aerial vehicles: Theories and applications," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 1, pp. 340–348, 2014.

[87] C. B. Low, "A dynamic virtual structure formation control for fixed-wing UAVs," in *9th IEEE International Conference on Control and Automation (ICCA)*, Santiago, Chile, 2011.

[88] D. Cai, J. Sun, and S. T. Wu, "UAVs formation flight control based on behavior and virtual structure," in *Asian Simulation Conference*, Shanghai, China, 2012.

[89] G. R. Lee and D. K. Chwa, "Decentralized behavior-based formation control of multiple robots considering obstacle avoidance," *Intelligent Service Robotics*, vol. 11, no. 1, pp. 127–138, 2018.

[90] D. Xu, X. Zhang, Z. Zhu, C. Chen, and P. Yang, "Behavior-based formation control of swarm robots," *Mathematical Problems in Engineering*, 2014.

[91] Z. Sun, G. Zhang, Y. Lu, and W. Zhang, "Leader-follower formation control of underactuated surface vehicles based on sliding mode control and parameter estimation," *ISA Transactions*, vol. 72, pp. 15–24, 2018.

[92] Z. X. Peng, G. G. Wen, A. Rahmani, and Y. G. Yu, "Leader–follower formation control of nonholonomic mobile robots based on a bioinspired neurodynamic based approach," *Robotics and Autonomous Systems*, vol. 61, no. 9, pp. 988–996, 2013.

[93] K. Hengster-Movrić, S. Bogdan, and I. Draganjac, "Multi-agent formation control based on bell-shaped potential functions," *Journal of Intelligent and Robotic Systems*, vol. 58, no. 2, pp. 165–189, 2010.

[94] Q. L. Jia and G. W. Li, "Formation control and obstacle avoidance algorithm of multiple autonomous underwater vehicles (AUVs) based on potential function and behavior rules," in *IEEE International Conference on Automation and Logistics*, Jinan, China, 2007.

[95] Y. Zhang, Y. Zhang, and Z. Yu, "Path following control for UAV using deep reinforcement learning approach," *Guidance, Navigation and Control*, vol. 1, no. 01, p. 2150005, 2021.

[96] H. Liu, Q. Meng, F. Peng, and F. L. Lewis, "Heterogeneous formation control of multiple UAVs with limited-input leader via reinforcement learning," *Neurocomputing*, vol. 412, pp. 63–71, 2020.

[97] T. Baca, P. Stepan, V. Spurny, D. Hert, R. Penicka, M. Saska, J. Thomas, G. Loianno, and V. Kumar, "Autonomous landing on a moving vehicle with an unmanned aerial vehicle," *Journal of Field Robotics*, vol. 36, no. 5, pp. 874–891, 2019.

[98] O. Araar, N. Aouf, and I. Vitanov, "Vision based autonomous landing of multirotor UAV on moving platform," *Journal of Intelligent and Robotic Systems*, vol. 85, no. 2, pp. 369–384, 2017.

[99] A. Borowczyk, D.-T. Nguyen, A. Phu-Van Nguyen, D. Q. Nguyen, D. Saussié, and J. Le Ny, "Autonomous landing of a multirotor micro air vehicle on a high velocity ground vehicle," *IFAC-Papersonline*, vol. 50, no. 1, pp. 10 488–10 494, 2017.

[100] Y. Xu, Z. Liu, and X. Wang, "Monocular vision based autonomous landing of quadrotor through deep reinforcement learning," in *37th Chinese Control Conference (CCC)*, Wuhan, China, 2018.

[101] A. Rezelj and D. Skocaj, "Autonomous charging of a quadcopter on a mobile platform," in *Austrian Robotics Workshop*, Klagenfurt, Austria, 2015.

[102] S. Arora, S. Jain, S. Scherer, S. Nuske, L. Chamberlain, and S. Singh, "Infrastructure-free shipdeck tracking for autonomous landing," in *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013.

[103] A. Gautam, P. Sujit, and S. Saripalli, "A survey of autonomous landing techniques for UAVs," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, Orlando, USA, 2014.

[104] ——, "Application of guidance laws to quadrotor landing," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, Denver, USA, 2015.

[105] W. Kai, S. Chunzhen, and J. Yi, "Research on adaptive guidance technology of UAV ship landing system based on net recovery," *Procedia Engineering*, vol. 99, pp. 1027–1034, 2015.

[106] H. Mo and G. Farid, "Nonlinear and adaptive intelligent control techniques for quadrotor UAV–a survey," *Asian Journal of Control*, vol. 21, no. 2, pp. 989–1008, 2019.

[107] M. Skoczylas, "Vision analysis system for autonomous landing of micro drone," *Acta Mechanica et Automatica*, vol. 8, no. 4, 2014.

[108] A. Rucco, P. Sujit, A. P. Aguiar, J. B. De Sousa, and F. L. Pereira, "Optimal rendezvous trajectory for unmanned aerial-ground vehicles," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 2, pp. 834–847, 2017.

[109] K. Ling, D. Chow, A. Das, and S. L. Waslander, "Autonomous maritime landings for low-cost vtol aerial vehicles," in *Canadian Conference on Computer and Robot Vision*, Montreal, Canada, 2014.

[110] X. Guo, S. Denman, C. Fookes, L. Mejias, and S. Sridharan, "Automatic UAV forced landing site detection using machine learning," in *International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, Wollongong, Australia, 2014.

[111] V. V. R. M. K. Muvva, G. Li, and M. Wolf, "Autonomous UAV landing on a moving UAV using machine learning," in *AIAA SCITECH 2022 Forum*, San Diego, USA, 2022.

[112] J. Xie, X. Peng, H. Wang, W. Niu, and X. Zheng, "UAV autonomous tracking and landing based on deep reinforcement learning strategy," *Sensors*, vol. 20, no. 19, p. 5630, 2020.

[113] J. Ma and C. Peng, "Adaptive model-free fault-tolerant control based on integral reinforcement learning for a highly flexible aircraft with actuator faults," *Aerospace Science and Technology*, vol. 119, p. 107204, 2021.

[114] G. K. Fourlas and G. C. Karras, "A survey on fault diagnosis and fault-tolerant control methods for unmanned aerial vehicles," *Machines*, vol. 9, no. 9, p. 197, 2021.

[115] B. Wang and Y. Zhang, "An adaptive fault-tolerant sliding mode control allocation scheme for multirotor helicopter subject to simultaneous actuator faults," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 5, pp. 4227–4236, 2017.

[116] I. A. T. Association *et al.*, "Loss of control in-flight accident analysis report 2010–2014," *Montreal-Geneva: International Air Transport Association*, 2015.

[117] H. Badihi, Y. Zhang, P. Pillay, and S. Rakheja, "Application of fmrac to fault-tolerant cooperative control of a wind farm with decreased power generation due to blade erosion/debris buildup," *International Journal of Adaptive Control and Signal Processing*, vol. 32, no. 4, pp. 628–645, 2018.

[118] Z. Liu, C. Yuan, Y. Zhang, and J. Luo, "A learning-based fault tolerant tracking control of an unmanned quadrotor helicopter," *Journal of Intelligent and Robotic Systems*, vol. 84, no. 1, pp. 145–162, 2016.

[119] F. Sharifi, M. Mirzaei, B. W. Gordon, and Y. Zhang, "Fault tolerant control of a quadrotor UAV using sliding mode control," in *Conference on Control and Fault-Tolerant Systems (SysTol)*, Nice, France, 2010.

[120] I. Sadeghzadeh and Y. Zhang, "A review on fault-tolerant control for unmanned aerial vehicles (UAVs)," in *Infotech at Aerospace*, St. Louis, USA, 2011.

[121] Y. Zhang and J. Jiang, "Bibliographical review on reconfigurable fault-tolerant control systems," *Annual Reviews in Control*, vol. 32, no. 2, pp. 229–252, 2008.

[122] H. Yang, B. Jiang, H. H. Liu, H. Yang, and Q. Zhang, "Attitude synchronization for multiple 3-dof helicopters with actuator faults," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 2, pp. 597–608, 2019.

[123] K. Zhou and Z. Ren, "A new controller architecture for high performance, robust, and fault-tolerant control," *IEEE Transactions on Automatic Control*, vol. 46, no. 10, pp. 1613–1618, 2001.

[124] S. Yin, H. Gao, J. Qiu, and O. Kaynak, "Descriptor reduced-order sliding mode observers design for switched systems with sensor and actuator faults," *Automatica*, vol. 76, pp. 282–292, 2017.

[125] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[126] Y. Hou, L. Liu, Q. Wei, X. Xu, and C. Chen, "A novel ddpg method with prioritized experience replay," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Banff, Canada, 2017.

[127] J. Wu, R. Wang, R. Li, H. Zhang, and X. Hu, "Multi-critic ddpg method and double experience replay," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Miyazaki, Japan, 2018.

[128] R. Rysdyk, "Unmanned aerial vehicle path following for target observation in wind," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 5, pp. 1092–1100, 2006.

[129] H. Qie, D. Shi, T. Shen, X. Xu, Y. Li, and L. Wang, "Joint optimization of multi-UAV target assignment and path planning based on multi-agent reinforcement learning," *IEEE Access*, vol. 7, pp. 146 264–146 272, 2019.

[130] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE Conference on Computer Communications*, Honolulu, USA, 2018.

[131] X. Wu, S. Liu, T. Zhang, L. Yang, Y. Li, and T. Wang, "Motion control for biped robot via DDPG-based deep reinforcement learning," in *WRC Symposium on Advanced Robotics and Automation*, Beijing, China, 2018.

[132] R. Yu, Z. Shi, C. Huang, T. Li, and Q. Ma, "Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle," in *36th Chinese control conference (CCC)*, Dalian, China, 2017.

[133] Y. D. Liu, W. Z. Zhang, F. M. Chen, and J. L. Li, "Path planning based on improved deep deterministic policy gradient algorithm," in *3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, Chengdu, China, 2019.

[134] R. Way, F. Lafond, F. Lillo, V. Panchenko, and J. D. Farmer, "Wright meets markowitz: How standard portfolio theory changes when assets are technologies following experience curves," *Journal of Economic Dynamics and Control*, vol. 101, pp. 211–238, 2019.

[135] M. Y. Jaber and A. L. Guiffrida, "Learning curves for processes generating defects requiring reworks," *European Journal of Operational Research*, vol. 159, no. 3, pp. 663–672, 2004.

[136] K. Dally and E.-J. Van Kampen, "Soft actor-critic deep reinforcement learning for fault tolerant flight control," in *AIAA SCITECH 2022 Forum*, San Diego, USA, 2022.

[137] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*, Stockholm, Sweden, 2018.