Reinforcement learning based motion planning of quadrotors using motion primitives

Efe Camci

2020

Efe Camci. (2020). Reinforcement learning based motion planning of quadrotors using motion primitives. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/137183>

<https://doi.org/10.32657/10356/137183>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 11:25:23 SGT

REINFORCEMENT LEARNING BASED MOTION PLANNING OF QUADROTORS USING MOTION PRIMITIVES

EFE CAMCI

School of Mechanical and Aerospace Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2020

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

02-JUL-2019

.....
Date

EFE CAMC)

.....

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

02-JUL-2019

.....
Date

Prof. Chen I-Ming

.....

Authorship Attribution Statement

This thesis contains material from 3 papers published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 3 is partially published in Efe Camci and Erdal Kayacan. "Planning Swift Maneuvers of Quadcopter Using Motion Primitives Explored by Reinforcement Learning." 2019 Annual American Control Conference (ACC). IEEE, 2019 (Accepted).

The contributions of the co-authors are as follows:

- Since I am the only author besides my supervisor, I have conducted the whole research. I have developed and tested the proposed motion planning framework both in simulations and in real flights.
- Dr. Erdal Kayacan has provided general supervision and editorial comments while writing the paper.

Chapter 3 is partially published in Efe Camci and Erdal Kayacan. "Learning Motion Primitives for Planning Swift Maneuvers of Quadrotor." Autonomous Robots (2019): 1-13 (In Press).

The contributions of the co-authors are as follows:

- Since I am the only author besides my supervisor, I have conducted the whole research. I have developed and tested the proposed motion planning framework both in simulations and in real flights.
- Dr. Erdal Kayacan has provided general supervision and editorial comments while writing the paper.

Chapter 5 is partially published in Mirko Gschwindt, Efe Camci, Rogerio Bonatti, Wenshan Wang, Erdal Kayacan, Sebastian Scherer. "Can a Robot Become a Movie Director? Learning Artistic Principles for Aerial Cinematography." 2019 IEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2019 (Accepted).

The contributions of the co-authors are as follows:

- I have co-developed the proposed motion supervision framework together with Mirko Gschwindt. We have developed and tested the proposed method in simulations together by distributing the workload equally.
- Mirko Gschwindt has also helped with real flight tests which have been conducted in Pittsburgh, PA, USA.
- Rogerio Bonatti has helped with system integration of different parts in the planning pipeline in order to fit the proposed method in the existing pipeline.
- Dr. Wenshan Wang has provided technical and theoretical supervision while developing the proposed method. She has also helped with real flight tests.
- Dr. Erdal Kayacan has provided general supervision and editorial comments while writing the paper.
- Dr. Sebastian Scherer has provided general supervision while developing the proposed method and editorial comments while writing the paper.

02-JUL-2019

.....
Date

EFE CAMCI

.....

Acknowledgments

I hereby acknowledge the great support by the people below with a certain hierarchy. This PhD thesis would not have been possible without them. Besides, I would like to extend my cordial thanks to my dear friends, family, and all the women that came into my life and changed me. Thank you all!

Sanjiban Choudhary
Juliana Binte Jaapar
Ting Li Yong
Ang Hanlin
Nicholas Sadjoli
Wenshan Wang
Ng Tian Seng
Dogan Kircali
Long Tien Siew
Mojtaba Ahmadi Khansar
Aayush Ahuja
Mohammadreza Mousaei
Yunus Govdeli
Siddharth Patel
Wang Hui
Elson Ng Moo Kheo

Nursultan Imanberdiyev
Arjun Sasikumar
Ali Turker Kutay
Linlu Ma
Serkan Ozgen
Shavin Goswami
Baris Burak Kanbur
Lawrence Ang Koon Teck
Erdal Kayacan
Yiqun Dong
Daniel Maturana
Daniel Tan Khin Hwa
Devesh Raju Kripalani
Delong Zhu
Ozan Tekinalp

Ercan Gurses
Wong Zhuo Wei
Soh Meow Chng
Ran Duan
Kevin Pluckter
Azarakhsh Keipour
Sebastian Scherer
Rogerio Bonatti
Domenico Campolo
Koh Wing Leong
Nicholas Lee Han Chun
Anna Prach
Basaran Bahadir Kocer
Seet Thian Beng
I-Ming Chen
Cherie Ho
Juan Jose Saameño Perez

Ratnesh Madaan
Mirko Gschwindt
Ilkay Yavrucuk
Changhong Fu
Wilson Lee Ying Jun
Suraj Ravindrababu
Roger Lee Keng Yuen
Ow Yong See Meng
Yeo Lay Foon
Omer Burak Iskender
Rohit Garg
Andriy Sarabakha
Volkan Kumteveli
Edward Yeo Boon Chuan
Mohit Mehndiratta

Contents

Acknowledgments	vii
Contents	ix
Summary	1
1 Introduction	3
1.1 Motivational Examples	4
1.1.1 Performing Swift Maneuvers	4
1.1.2 Navigating in Partially Known Places	5
1.1.3 Filming a Moving Target	5
1.2 Specific Problems Addressed	6
1.2.1 Partially Known Situations	6
1.2.2 Onboard Computation and Sensing Limitations	6
1.2.3 Inherently Subtle Criteria for Task Success	7
1.3 Outline and Contributions	7
2 Background	11
2.1 Motion Planning of Aerial Robots	11
2.1.1 Orientation Planning	13
2.1.2 Position Planning	14
2.1.3 High-Level Planning	17
2.2 Reinforcement Learning	18
2.2.1 Associative Search	19
2.2.2 Deep Q-learning	20
2.3 Motion Primitives	20

3	Learning Primitives in Orientation Domain	23
3.1	Introduction	23
3.2	Approach	25
3.2.1	Problem Formulation	25
3.2.2	Quadrotor Model	25
3.2.3	Motion Planning	27
3.3	Results	33
3.3.1	Virtual Flights	33
3.3.2	Real Flights	38
3.4	Discussion	43
4	Learning Primitives in Position Domain	47
4.1	Introduction	47
4.2	Approach	49
4.2.1	State	49
4.2.2	Action	49
4.2.3	Reward	50
4.2.4	Algorithmic Details	51
4.3	Results	52
4.3.1	Virtual Flights	52
4.3.2	Real Flights	58
4.4	Discussion	60
5	Learning Primitives in High-Level User Input Domain	61
5.1	Introduction	61
5.2	Approach	64
5.2.1	State	64
5.2.2	Action	65
5.2.3	Handcrafted Reward	65
5.2.4	Human Reward	66
5.2.5	Algorithmic Details	67
5.3	Results	68
5.3.1	Virtual Flights	69
5.3.2	Real Flights	75
5.4	Discussion	75

6 Conclusion	79
6.1 Planning in Orientation Domain	79
6.1.1 Summary	79
6.1.2 Lessons Learned	80
6.2 Planning in Position Domain	80
6.2.1 Summary	80
6.2.2 Lessons Learned	80
6.3 Planning in High-Level User Input Domain	81
6.3.1 Summary	81
6.3.2 Lessons Learned	82
6.4 Future Work	83
Video Links	85
Bibliography	87

Summary

Motion planning of robots in real world is challenging due to the uncertainty in environments and robot models, the computation and sensing limitations on hardware, and the complexity of the tasks to be performed during operations. Motivated by these problems, the main contribution of this thesis is a Reinforcement Learning (RL) based approach for motion planning of quadrotors that can deal with uncertainties, work with rudimentary hardware, and minimize expert user intervention during complex operations. Aligned with the hierarchical motion planning pipeline of quadrotors, i.e., high-level, position, and orientation planning, we propose a novel planning framework at each domain based upon RL.

In Chapter 3, we propose a planning framework in orientation domain which allows a quadrotor to plan swift maneuvers and decrease navigation time in dense environments. Our RL agent learns the complex relations between abrupt control inputs and the motion of a quadrotor incorporating probable uncertainties such as the unmodeled dynamics during agile maneuvers. It then utilizes this knowledge during deployment to plan relatively swift and reasonably precise maneuvers.

In Chapter 4, we propose a planning framework in position domain which caters for navigation in unstructured environments using raw data obtained from onboard sensors. Our RL agent maps instantaneous raw sensor data to local motion plans in the position domain which cater for obstacle avoidance and navigation towards goal simultaneously. It learns essential navigation policies which are transferable from virtual environments to previously unseen real environments.

In Chapter 5, we propose a planning framework in high-level user input domain which creates aesthetically aware motions of a filming quadrotor that follows an actor and shoots its video. Trained using both a handcrafted and a human reward, our RL agent extracts the intricate relations between video scene context and aerial camera viewpoints. Accordingly, it creates high-level viewpoint sequences which govern low-level position planning of a filming quadrotor for shooting better video clips.

We evaluate and validate our RL-based planning frameworks by conducting virtual and real flight tests extensively. We discuss our methods in comparison with the current state-of-the-art methods. In the end, we share the lessons learned during this study and indicate a few promising directions for future work.

Introduction

Quadrotors have been at the spotlight by the robotics research community in the past few decades due to their versatility, low cost, and relatively simple dynamics for control. Their applications are of a broad spectrum: leisure activities such as drone-racing [1] or aerial filming [2]; daily life tasks such as cargo delivery [3] or drink serving [4]; emergency missions such as search and rescue [5] or surveillance [6]. Nevertheless, the number of autonomous applications of quadrotors is still relatively limited, especially outside controlled lab spaces, in daily life environments. The tasks to be accomplished in this regard are sensing, motion planning, and control. We touch each of these items briefly while mainly focusing on motion planning in this thesis.

Motion planning of quadrotors has a rich literature. From optimal trajectory planning in a virtual flight [7] to competing in a challenging drone-racing track [8], the state-of-the-art motion planning algorithms have demonstrated decent results so far. On the other hand, we do not see these algorithms deployed frequently on quadrotors in real flights. The challenges in real flights, though have been addressed by several research groups up until now as discussed in Chapter 2, are yet to be overcome completely. To name a few, planning algorithms have to be robust to faulty sensor measurements, reliable in previously unseen situations, adaptive to changing conditions, and intelligent enough to replace the expert user input in the planning pipeline completely. Reinforcement Learning (RL) [9] is a favorable choice to cater in these contexts.

Studied since 1980s, RL is substantially helpful for the tasks which are difficult to plan beforehand, which include some level of uncertainty, and which require adaptation to changing conditions. Despite its relatively long history, most of the RL-based algorithms have been proven to work in simulations not in real life with real robots. Yet, considering the above-mentioned real challenges, RL is a perfect cut for the current problems in the motion planning research. All of these motivate us to investigate whether RL can play a role in real world applications of quadrotors where conditions are far from ideal unlike virtual flights.

1.1 Motivational Examples

Quadrotors are yet to achieve their full potential in daily life, despite the substantial amount of research conducted for these vehicles. A great portion of quadrotor research falls into the scope of control [10]. In this line of research, quadrotors generally attempt to follow a predefined trajectory with high accuracy. Given the certain breakthroughs in this research direction, including the controllers which exploit differential flatness of quadrotors [11, 12], this line of research can be considered as relatively mature. Another line of research focuses on motion planning of these vehicles with the availability of full information about the environment. That is, a quadrotor with pre-modeled dynamics is deployed in a pre-mapped environment to plan its motion among obstacles in order to navigate to a goal position without any collision [13]. Along with the global path planning research conducted mainly for ground robots [14], this particular direction also has a rich literature.

Although the above-mentioned two main directions contribute vastly to the overall quadrotor research, they do not address the challenges encountered in real world scenarios completely. Real world deployment poses several specific challenges, e.g., uncertainty in dynamics or sensor measurements, computational limitations of onboard computers or range limitations of onboard sensors, changes in the environment or changes in the tasks to be accomplished. Hence, quadrotors ought to have reasoning ability, be adaptive to changing conditions, and react quickly in certain cases. We explain the significance of these points with a few motivational examples in the following subsections.

1.1.1 Performing Swift Maneuvers

Quadrotors can perform a variety of maneuvers including instantaneous turnaround motions over sharp turns, high acceleration motions on straights, or even flips in the air. These kinds of agile maneuvers are of great interest in certain cases, e.g., rapid navigation during search and rescue, quick lap completion in drone-racing (Fig. 1.1). While all these maneuvers are trivial for expert human pilots after a short practice with a real vehicle, autonomous execution of these in a controlled manner is a little more difficult. Precise and swift motion planning using a priori known mathematical model of a quadrotor may not be possible because even that model may not be valid anymore in the extreme conditions, i.e., nonlinear dynamics of quadrotor at extreme attitude angles, uncertain rotor aerodynamics during a high speed flight. Besides, faulty or delayed sensor measurements for state estimation would deteriorate the process even further. Moreover, the performance of the low-level controller of a quadrotor may decrease drastically at extreme attitude angles and rates since they are not necessarily tuned for these conditions. Thus, an intelligent motion planner which can account for changes in the mathematical model, uncertainties in the hardware, and performance decrease in the low-level controller would be needed, which we investigate in Chapter 3.

Figure 1.1: **Left:** Quadrotor performing a swift maneuver during drone-racing [15]. **Middle:** Quadrotor navigating in an unstructured forest [16]. **Right:** Filming quadrotor in action during the World Rally Championship [17].

1.1.2 Navigating in Partially Known Places

Quadrotor deployment in real world generally implies operation in unknown environments. Consider a search and rescue task in a forest (Fig. 1.1) where trees of different sizes and shapes are densely located. It is fairly difficult to have a map of this kind of environment. Another example is search and rescue in mines or within remainders of wrecked buildings after a natural disaster. Previously obtained environment maps before the disaster would not be valid anymore because buildings, structures, and roads are all wrecked after the disaster. Obstacle locations are uncertain in this kind of environments. A quadrotor can only rely on its onboard sensors to avoid obstacles and navigate safely. On the other hand, given its size and power capacity, a quadrotor can have only limited computation power onboard. It may not be able to use advanced sensing and mapping techniques which can precisely inform about obstacle locations as in self-driving cars. Thus, a motion planner ought to be contented with limited information by rudimentary sensing modules onboard. In this vein, an intelligent motion planner which can implicitly account for obstacles during navigation using limited raw sensor data would be desirable, which we investigate in Chapter 4.

1.1.3 Filming a Moving Target

Quadrotors in aerial cinematography (Fig. 1.1) are becoming more and more popular day by day due to their substantially reduced cost as compared to a full size helicopter minimally operated with a pilot and a professional cameraman onboard. On the other hand, most of the current aerial cinematography applications with quadrotors are through manual flights by expert pilots. In this context, the job of expert pilots is intractable. They are expected to deal with a complex optimization on-the-fly considering dynamic feasibility, obstacle and occlusion avoidance, motion forecast of the actor, and visual aesthetics of the video clip simultaneously. This expectation is a bit beyond the human beings' capabilities. Hence, these manual operations generally result in suboptimal performance. Therefore, the need for an intelligent motion planner that can handle these issues automatically is apparent, which we investigate in Chapter 5.

1.2 Specific Problems Addressed

Each of the motivational examples mentioned in the previous section poses unique challenges for motion planning. We investigate a few of these challenges deeply in this thesis. These challenges are mostly related to inherent difficulty of motion planning in real world. We can categorize them in three major items.

1.2.1 Partially Known Situations

The word ‘situation’ here refers to everything included in a motion planning scenario, e.g., quadrotor, static surroundings, dynamic obstacles, ambient conditions. In this context, there could be some prior information about the situation such as a rough mathematical model of quadrotor that is valid for near-hover conditions or an imprecise map of static surroundings. Nevertheless, obtaining full information about the situation in advance is not possible in real world deployments, especially in certain cases, e.g., stochastically moving obstacles or randomly changing ambient conditions. A motion planner has to account for these uncertainties in real world deployments.

The tasks of performing swift maneuvers with a quadrotor, navigating in an unknown environment, or filming a moving target which we present as the motivational examples in the previous section bring the partially known situation problem to the fore:

While *planning swift maneuvers autonomously*, a preliminary mathematical model of quadrotor might be available to give an idea about quadrotor’s motion in near-hover conditions. However, while flying at extreme attitude or speed, this model may not be able to yield precise results about the overall motion.

While *planning motions in an unknown place*, a rough map might be available informing about corridors or open areas. Nevertheless, precise obstacle location information would not be known in advance. Instead, it has to be acquired on-the-fly using onboard sensors with limited range.

While *filming a moving target*, a map including obstacle locations in the movie scene might or might not be available in advance. Yet, the paths which the target is likely to take would probably remain uncertain, which can impel the quadrotor towards unknown regions, especially in the case of personal-filming drones. Moreover, unexpected motions of the target such as sudden stops or sharp turnarounds may not be known in advance.

1.2.2 Onboard Computation and Sensing Limitations

The quadrotors of interest in this thesis include three different types as we introduce in the next chapters, but all of them can be considered as micro-scale aerial vehicles. In this vein, their payload and power capacity is strictly constrained. Accordingly, computers or sensors which would be put onboard have to be lightweight and less power consuming. Therefore, we cannot employ very advanced sensing and

mapping systems which yield substantially precise information with the use of multiple light detection and ranging (LiDAR) modules, optical flows, or high resolution cameras as in self-driving cars. Similarly, we cannot utilize advanced computers at the weight scale of kilograms. Instead, we have to make our sensor and computer choices minimally by assuring that we would still have decent flight time capacity with limited battery power onboard. Accordingly, we have to develop algorithms minimally both from constrained computational load and from limited sensor range perspectives. All three of the above-mentioned motivational examples yield this limitation, regardless of the task being performed.

1.2.3 Inherently Subtle Criteria for Task Success

For a quadrotor following a predefined trajectory, the task success criterion is fairly obvious: to follow the path with the least error possible. On the other hand, the examples which motivate our study in this thesis present intrinsically vague success criteria.

Consider *navigating in an unknown place* example. A quadrotor has to sense its surroundings and navigate safely. On the other hand, it is non-trivial or subtle to have a balance between moving slow in dense areas or moving fast in relatively free areas using only limited sensor information. The quadrotor has to reason about its current situation while deciding on moving aggressively or cautiously in order to stay collision-free and in order to navigate to goal in the quickest way possible.

Consider *filming a moving target* example. A quadrotor has some obvious subtasks to be accomplished such as following the target or flying in a collision-free area safely. However, it also has a relatively subtle criterion to satisfy, e.g., producing visually pleasing video clips. While the former subtasks can be incorporated via conventional planning methods in a way, the latter criterion is intractable to comply with. The quadrotor ought to have a reasoning ability on the video clips produced concurrently and plan its motion accordingly.

1.3 Outline and Contributions

We address all the aforementioned problems in the following chapters by developing RL-based motion planning methods. We develop our methods around the fundamental idea that:

‘In cases where uncertainty is present, be it in environment modeling or in task success criteria, learning from experience can assist in extracting essential policies which generalize to different conditions than what have been known previously.’

In Chapter 2, we present a comprehensive literature review on motion planning of aerial vehicles. We go through the current state-of-the-art methods with their pros and cons. Although, each of the subsequent chapters also includes a mini literature review, we first present a thorough one in Chapter 2 for completeness. We also give a basic background on RL and explain how it would be useful for motion

planning in real world. Moreover, we touch the topic of motion primitives in the same chapter as it is the essential bridge between motion planning and RL throughout the thesis.

In Chapter 3, we focus on performing swift maneuvers with a quadrotor by learning primitives in orientation domain. The main contribution of this chapter is a novel method for swift maneuver planning in order to decrease navigation time in dense environments. Quick navigation in cluttered environments is of great interest for autonomous aerial vehicles as it was also investigated recently in Defense Advanced Research Projects Agency’s (DARPA) Fast Lightweight Autonomy (FLA) program [18]. A remarkable method in this program [19] uses features of sensor data (camera) to select control laws or motion primitive sequences for creating safe and quick maneuvers. In the same vein, we develop a method for crafting best-fit motion primitives to cater for swift maneuvers but using preplanned trajectory segments instead of camera data. Particularly, we develop an RL-based method which enables offline learning of swift maneuver primitives and online utilization of them when a quadrotor is deployed in dense obstacle courses. Our method caters for capturing the dynamics of a real quadrotor even at extreme motions and planning swift maneuvers confidently. We show performance improvement in navigation time up to 20% by incorporating swift maneuvers into the overall planning in real flights, as compared to the default planning and control strategy of our quadrotor. We also show the computational feasibility of our method for real-time operations by a comparative computation time analysis.

In Chapter 4, we focus on planning desirable motions of a quadrotor for navigation in unstructured environments by learning primitives in position domain. The main contribution of this chapter is a novel method for navigation in partially unknown environments using onboard sensors. Aligned with the main objectives of DARPA’s FLA program [18], we keep the number of sensors minimal (a single depth camera) for navigation. Different from the conventional navigation methods utilized in DARPA’s FLA on the other hand, we develop an alternative navigation approach which incorporates end-to-end learning. Particularly, we develop a deep RL based method which uses raw image data from a front-facing stereo camera on a quadrotor in order to plan motions which enable obstacle avoidance as well as navigation to goal. We train a Deep Q Network (DQN) in a diversified set of environments in high-fidelity, virtual flights by utilizing a virtual quadrotor with a simulated stereo camera. Purely trained in these simulations, our DQN demonstrates proficiency in generalizing to unseen conditions in both virtual and real flights.

In Chapter 5, we focus on planning aesthetically aware motions of a quadrotor for aerial cinematography by learning motion primitives in high-level user input domain. The main contribution of this chapter is a novel method for autonomous viewpoint selection during aerial cinematography based upon visual aesthetics. Aligned with one of the objectives of DARPA’s FLA program [18], we replace the expert user’s role (director) with an intelligent agent that makes decisions and provides high-level inputs autonomously during aerial filming operations. In this vein, the work in Chapter 5 is one of a kind among the other aerial

cinematographers [20, 21]. Particularly, we develop a deep RL based method which supervises the overall motion planning of a filming quadrotor by creating desirable sequences of camera viewpoint primitives. We train a DQN which captures the visual aesthetics in video clips both via a set of commonly recognized cinematography rules and via a human assessment on video clips. We show improved performance in creating visually pleasing videos as compared to baseline policies through extensive simulations. In order to verify the success of our method in a cogent manner, we conduct a user study in which participants rate the video clips taken using our method and write comments about them. In the end, we also deploy our method for real world filming to demonstrate generalization from simulations to real flights.

In Chapter 6, we conclude the whole work presented in this thesis discussing about the developed methods and the lessons learned on the way. We then complete the thesis by giving insight for possible future work.

Background

Robot motion planning research is vast including motion planning of manipulators or ground vehicles. There have been numerous methods and algorithms proposed in this field so far. While these methods are generally applicable across different types of robots, e.g., robotic arm, nonholonomic car, or quadrotor, each of them is distinguishable by their very specific details which are essential for successful applications in real world. Since our main aim in this thesis is to address the problems faced during real world deployments, for the sake of brevity, we focus our literature review on motion planning of aerial robots with particular attention on the works with experimental results in real world.

2.1 Motion Planning of Aerial Robots

The research on motion planning of aerial robots, especially multirotors, has been accelerating in the past decade substantially. Recent advances in multirotor technology as well as remarkable progress in onboard sensors and computers have created a great impetus in this line of research. They have inspired the possibility of previously unthought applications such as autonomous aerial cargo delivery or autonomous aerial cinematography. On the other hand, realization of these applications requires development of motion planning methods suitable for real world deployment. Accordingly, many research groups have contributed to the literature in order to develop a variety of motion planning methods for aerial robots in the past few decades. These methods can be classified in several ways to review the literature systematically.

A common classification in a broader scope of motion planning consists of two rough categories: global and local motion planning. Global planning [22] here refers to finding a path from start to goal in an environment for which full information about the environment (or about the ‘situation’ as defined in Chapter 1) is available in advance. These planners can guarantee optimality or completeness [23] in certain cases but they are mostly computationally expensive. Besides, the full information assumption is not valid for real world deployments of aerial robots; real world situations are at most partially known or totally unknown beforehand. Local planning [22], on the other hand, refers to incremental planning based

on sensor range. These methods can account for unavailability of full information but they do not have any optimality or completeness guarantees accordingly. They seem to suit for real world deployments better in this regard. However, the current state-of-the-art methods are generally the combination of these two categories; they try to exploit the strengths of both global and local planning for successful missions.

Another classification for motion planning methods could be based upon the type of underlying algorithms. It is relatively hard to distinguish the available algorithms under different categories though, considering the great diversity of the current motion planning algorithms. However, to mention a few main categories in this context, four items can be given: road maps, potential fields, graph-based search methods, and optimization-based methods.

Road maps, including probabilistic road map [24], Rapidly-exploring Randomized Tree (RRT) [25], and Voronoi diagram [26], represent the work space as some sort of a graph. The nodes in these graphs refer to placements of the robot while the edges of the graphs represent collision-free paths. Accordingly, these methods try to find the best path on these graphs based upon a predefined criterion.

Potential fields method [27] represents a robot as a point mass in an environment. Its principle is based upon a force field in which obstacles generate repulsive forces while goals generate attractive forces. The point mass moves within this field attempting to avoid obstacles and reach goals. This approach is widely known for its low computational complexity, but also for being prone to the local minima problem.

Graph-based search methods, including A* [28] and D* [29], use a heuristic to solve a motion planning problem within a grid or node based environment representation. The heuristic is a kind of rule that assists in predicting which path segments are more desirable for moving towards the goal. These algorithms are widely used due to their low computational complexity along with suboptimal but generally adequate performance.

Optimization-based methods, including Mixed Integer Linear Programming (MILP) [30] and Model Predictive Control (MPC) [31], formulate the motion planning problem as a numerical optimization problem. Vehicle constraints, environment constraints, goals, etc., are all included in this optimization problem via an arithmetical representation. In theory, these methods can provide optimal solutions. However, in practice, they require high computational power to find close-to-optimal solutions due to the common complexity of real world environments which makes the optimization problem intricate.

Different from the above-mentioned two classifications for motion planning methods, we review the literature by a proprietary categorization which is based on the hierarchical planning pipeline of quadrotors. As depicted in Fig. 2.1, there are three main modules in the pipeline. They can be named differently in different contexts, but in the context of this thesis, we name them as orientation planning, position planning, and high-level planning. The works presented in the following chapters also follow the same convention for consistency.

Figure 2.1: Hierarchical planning pipeline of quadrotors.

2.1.1 Orientation Planning

Being one before the low-level controller module in the pipeline in Fig. 2.1, orientation planning module is widely known as high-level controller of a quadrotor. In most of the trajectory tracking applications, it caters for creating necessary orientation (or attitude) plans to be executed by the low-level controller in order to follow a predefined trajectory precisely in three dimensional (3D) Cartesian coordinate system. Hence, renowned controllers such as proportional-integral-derivative or linear quadratic regulator may fit well in this module for these common applications in which the only aim is to decrease the trajectory tracking error in 3D. On the other hand, there is another set of operations in which the desired orientation has to be planned explicitly rather than being regulated by a simple controller, e.g., passing through a slanted, narrow gap with a quadrotor. We review the literature by focusing on these works in this subsection.

In [32], the researchers conduct planning in extreme orientation regions of a quadrotor, which enables performing flip maneuvers in the air. They focus on a two dimensional vertical model of a quadrotor since they are only concerned with flips over a single axis. They formulate a five-step flip primitive with five open parameters, and they adjust these parameters by a simple learning strategy over retrospective flip maneuvers. They demonstrate improvement in flip maneuvers by repetitive trials. They suggest that the method can be extended for different acrobatic aerial maneuvers, but they only consider specific case studies (flips over a single axis) using a simplified mathematical model in this work. They present extended results in [33] where they also consider translation motions together with flips.

In [34], the researchers break down a backflip maneuver into three steps: recovery, drift, and impulse. While they also demonstrate successful flip maneuvers with a quadrotor, their focus is more on the design of the guaranteed safe switching regions between the three different steps or modes. They construct the regions using reachable sets calculated by Hamilton-Jacobi differential game formulation. Nevertheless, both the previously mentioned two works and this one consider attractive maneuvers in obstacle-free environments which may not be generally the case for real world applications.

Addressing possible real world challenges a little more appropriately in [35], the researchers develop a family of trajectories defined as a sequence of segments in order to perform challenging maneuvers for which obstacles are present, e.g., passing through a narrow, slanted gap. They develop separate controllers

for each segment using a dynamic quadrotor model and feedback control laws. By this way, they easily align the orientation of a quadrotor while moving through a slanted passage or perching to inclined surfaces. However, their method is not plug-and-play and needs iterative learning with a real vehicle similar to [32], especially for refining the launch position and velocity towards surfaces or passages.

In [36], the researchers propose a unified approach for trajectory optimization and tracking by a nonlinear MPC algorithm. Position planning is also included in the overall framework, but the researchers focus more on agile maneuvers which require orientation planning, e.g., moving through slanted passages. Their algorithm differs from the previous approach for moving through slanted passages because they propose to use sequential linear quadratic in MPC setting to solve the optimal control problem at run time, hence enable a plug-and-play method.

All the works discussed in orientation planning context so far use external motion capture system for state estimation. In [37], the researchers attempt to utilize only onboard sensors and still plan aggressive maneuvers including moving through a 90° slanted passage. They present successful results for this purpose and for several other agile maneuvers. Yet, a drawback of their method is that the gap position and orientation information has to be obtained precisely in advance because they rely on this precision as they only conduct one shot planning before executing the maneuver.

In [38] on the other hand, the researchers utilize only onboard sensors both for gap detection and state estimation. They propose to couple perception and planning in order to facilitate the visual perception task by planning desirable trajectories in this context. They calculate the relative orientation of a quadrotor with respect to the detected gap repeatedly and plan the motion repetitively on-the-fly instead of one shot planning. They present successful results in the context of moving through relatively milder slanted gaps (up to 45°) by using only onboard sensing and computation.

Inspired by some of the methods above, we propose an RL-based framework for swift maneuver planning of quadrotors [39] in Chapter 3. Similar to the methods above, we focus on challenging, instantaneous changes in the orientation of quadrotors. However, as being different from the methods above, our aim is to decrease navigation time from start to goal by planning swift maneuvers rather than performing attractive maneuvers or moving through slanted passages.

2.1.2 Position Planning

Position planning module is responsible for creating desirable position commands in the pipeline. In the common trajectory tracking applications, this module generally does not exist. Instead, the trajectory to be followed is specified by an expert user explicitly. On the other hand, in the context of motion planning of aerial robots in real world, this module is the most widely studied one. The aforementioned planning algorithms such as road maps or potential fields generally take place in this module. Over the years, many

algorithms are proposed to be employed within this module as variants of the basic planning algorithms or as standalone algorithms. We examine them briefly in this subsection with an emphasis on real world applications.

One of the most well known planning algorithms for quadrotors is presented in [11]. The researchers exploit the differential flatness property of quadrotors in order to generate minimum snap trajectories in this work. The significance of minimizing the snap is that the snap of a trajectory maps directly to the squares of the motor revolutions per minute through differential flatness. Therefore, when the snap is minimized by optimizing polynomial coefficients, a trajectory which theoretically requires less control effort (easier to track) can be obtained. On the other hand, minimizing snap using a quadratic program with equality constraints at the boundaries is non-trivial in certain cases.

In [40], the researchers approach the same problem above from a different perspective by solving derivatives at the waypoints instead of optimizing polynomial coefficients of a trajectory. This approach enforces the boundary conditions inherently. They present successful results both on a fixed-wing plane and a quadrotor, proving applicability across different platforms. Nevertheless, the trajectory optimization is initiated by the waypoints obtained beforehand using a low-dimensional search algorithm that might yield suboptimal solutions at the first place.

In [41], the researchers develop a computationally light method which incorporates the dynamic constraints of a quadrotor during planning. They first derive feasibility constraints on vehicle dynamics, and then generate trajectories which guarantee this feasibility by limiting jerk and acceleration. Being similar to the last two methods, this one also requires intermediate target points to be specified beforehand.

In [42], the researchers utilize iterative regional inflation by semidefinite programming which segments space into obstacle-free regions instead of working over previously specified waypoints. The method then conducts mixed-integer convex optimization through this segmentation in order to generate desirable trajectories. Aggressive flights in cluttered environments including poles and strings are demonstrated taking this approach. Nevertheless, the precise obstacle location information is again available in advance. In this regard, all the four methods examined so far can be named as trajectory optimization methods within pre-mapped environments.

The researchers in [43] address the problems faced in real world planning such as limited sensor range and onboard computation. They propose an algorithm called SPARTAN (acronym for Sparse Tangential Network) which is shown to create motion plans 172 times faster than RRT* algorithm [44]. By creating vertices during the obstacle cost update and adopting favorable graph construction rules, the proposed method exploits possible intermittent sparsity of environments during planning. The researchers present decent flight test results in relatively free environments. They utilize incremental occupancy grid and distance transform representation for the obstacles sensed on-the-fly.

In [45], the researchers adapt a closed-loop RRT with image space collision checking using a stereo disparity map. They create some sort of image-based configuration space obstacle map and conduct collision check by projecting 3D trajectories generated by RRT into the image space. They validate the approach by successful flight tests in a groove of trees by onboard state estimation for low flight speeds.

In [46], the researchers propose a dual range planning horizon method: A short range planner generates promising paths towards the final goal and a long range planner gets activated if the short range planner fails. They create an occupancy grid map using the onboard sensor data and conduct the planning process within this map. The method demonstrates sufficiency in real flights with a fairly free obstacle setting.

Another approach addressing the real world planning challenges is presented in [47]. The researchers adopt RRT* as a waypoint generator and conduct minimum snap trajectory generation with corridor constraints within limited sensor range. They create an octree-based representation of the environment and conduct the planning process within this representation. They show decent results in fairly dense environments but with low navigation speeds.

In [48], the researchers propose a continuous-time trajectory optimization method which caters for real-time avoidance as the obstacles appear within the sensor range. They draw inspiration from optimization-based planning methods such as Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [49], and propose an objective function of two folds: minimizing a derivative of position and a cost of collision. The method demonstrates sufficiency for avoiding the obstacles which are previously out of sensor range by quickly replanning the trajectory in real flights with a fairly free obstacle setting. They extend the method in [50] where they compare different parameters for the underlying local optimization method and deploy the method for navigation in a forest.

In [51], the researchers address the limited sensor range problem together with the uncertainty problem which might be caused by noisy sensor data or unexpected disturbances. They pre-compute a funnel library over possible maneuvers of a fixed-wing airplane which guarantees to keep the state of the vehicle within certain boundaries even in the case of uncertainty. They then use this library in real-time to sequentially generate motion plans while ensuring the safety of the vehicle. They present preliminary results in a motion capture system with a relatively free environment setting.

In [52], the researchers propose an optimization-based approach using the safe flight corridor concept. They formulate the trajectory generation as a quadratic program by exploiting a convex overlapping polyhedra which results from a safe flight corridor that is based on a piecewise linear skeleton obtained through a graph-based search algorithm. They demonstrate real flights with a quadrotor equipped with LiDAR in fairly free outdoor environments. They mainly emphasize on the low computational demand of their method, but the performance of the proposed method is also highly dependent on the underlying graph-based search algorithm's performance.

In [53], the researchers adopt a fast marching based search method to find a path on a velocity field generated by Euclidean signed distance field. They then generate a safe flight corridor within which a trajectory consisting of Bézier curve segments is optimized through a convex quadratic program considering waypoints, continuity, safety, and dynamic feasibility constraints. They demonstrate successful flights in previously unknown environments with powerful onboard computation hardware which caters for map reconstruction as well as quadratic program solving.

In [54], the researchers address the difficulty and importance of precise state estimation using only onboard sensors for autonomous quadrotor flight. They propose a motion planner which generates a library of trajectories repetitively and evaluates them on certain criteria including perception quality. In other words, the proposed motion planner attempts to find paths which would provide better image features in onboard visual sensor data that is used for visual-inertial state estimation. They demonstrate preliminary flight test results with a quadrotor equipped with a monocular camera and an inertial measurement unit (IMU) in a relatively free environment.

Most of the methods above use some sort of a map representation of the environment which might not be available in advance. The process of sensing and reconstructing a map, on the other hand, might be computationally demanding. Moreover, imprecision in sensor data may result in downgraded map representations which would degrade the navigation performance even further. Considering all these uncertainties and computational constraints, we develop an end-to-end, RL-based method in Chapter 4 which sidesteps the explicit sensing-and-reconstructing process. It instead uses raw sensor data in order to generate local motion plans directly.

2.1.3 High-Level Planning

High-level planning refers more to task planning in the overall pipeline. It creates necessary inputs for the position planning module to consider while finding desirable paths in 3D Cartesian coordinate system. In the overall autonomy concept, it is generally a replacement for abstract commands possibly given by an expert user, e.g., shot type in aerial filming. The amount of research conducted in this specific direction is relatively less as compared to the former two categories, but there are a few works worth to mention in this subsection.

In [55], the researchers propose an optimization-based approach for high-level mission planning of multiple unmanned aerial vehicles (UAVs). They exploit specifying mission objectives and respective timing constraints using Linear Temporal Logic (LTL). They develop an algorithm which formulates multiple UAV mission planning problems with complex constraints as an MILP by converting an LTL formula into mixed-integer linear constraints. They demonstrate the method over a few examples which can be faced in real life that include several UAVs and targets with fuel and communication constraints.

In [56], the above-mentioned researchers propose a novel version of the vehicle routing problem by adding metric temporal logic specifications. They again utilize an MILP-based algorithm that solves the underlying optimization problem. They demonstrate the competency of their method with several examples in which multiple UAVs, launch and landing sites, as well as target locations are included.

In [57], the researchers introduce the decentralized partially observable semi-Markov decision process concept which allows for asynchronous decision-making in multi-robot domains. They demonstrate the planning approach on a ‘multi-UAV package delivery to multiple targets’ problem under uncertainty. They extend their approach in [58] by integrating a macro-observation scheme (hierarchical Bayesian noise inference) into their planner and present results obtained with the hardware.

In [59], the researchers introduce a novel decentralized control protocol for motion planning of UAV teams under high-level goals with LTL specifications. They abstract the motion of each UAV as a finite transition system and utilize formal verification methods to comply with each UAV’s LTL specification. They demonstrate successful navigation with multiple UAVs among predefined targets while avoiding collision of UAVs with each other.

The above-mentioned methods formulate the high-level planning problem of quadrotors into different kinds of formal representations using renowned techniques such as LTL. Their task success criteria are relatively obvious though, i.e., reaching as many as possible targets with the least fuel consumption. In Chapter 5 where we focus on shooting aesthetical video clips in aerial cinematography, the criterion is rather subtle or subjective. The meaning of aesthetics can vary for different people in different contexts. Moreover, even if a universal aesthetical rule exists, it would be non-trivial to formulate it within an optimization problem. It would be even more difficult to solve this kind of an optimization while conducting high-level planning on-the-fly. Therefore, we take a deep RL based approach in which a reward by human preferences are used together with a handcrafted reward to learn desirable viewpoint primitives for aerial filming in Chapter 5.

2.2 Reinforcement Learning

In machine learning, current algorithms are generally classified as members of three types of learning: *supervised*, *unsupervised*, and *reinforcement*. Supervised learning refers to learning from a labeled data set whose labels are provided by an external supervisor. Each data in the set describes a particular situation together with a specification of an action to take during that situation. The aim is to generalize to different conditions, which are not present in the labeled data set, using a large amount of diversified data. Unsupervised learning, on the other hand, refers to learning patterns or structures in an unlabeled data set. In this case, there is no external supervisor to label different situations. The aim is now not to generalize to unseen conditions but to classify different situations as distinguished categories.

Different from supervised and unsupervised learning paradigms, RL refers to learning from experience using only a simple reward signal instead of labeled or unlabeled data. The keyword here is *experience*: an RL agent learns by interaction with its environment by evaluating its retrospective actions just as in the learning process of infants. In this vein, RL is based upon evaluative feedback that assesses the action taken, instead of instructive feedback that specifies which action to take. Instructive feedback is totally independent of the action taken as it only instructs the action to be taken explicitly. On the other hand, evaluative feedback is dependent on the action taken since it is an indication on the desirability of that action. This is why RL requires learning from interaction or experience.

The formal framework of RL defines the agent's interaction with its environment through three main elements: state, action, and reward. State refers to the situation which the agent is currently in. Action refers to the agent's choice observing its state and following a certain policy. Reward refers to the reinforcement it gets for taking that particular action at that particular state. The agent takes an action during a situation, observes its reward and the changes in the situation, and takes another action subsequently exploiting the knowledge obtained during retrospective actions. The aim of the agent is to maximize the rewards over the long run by learning desirable actions. Earlier researchers in RL field design this formal framework as a simple way of incorporating essential features of an Artificial Intelligence (AI) problem such as sense of cause and effect, uncertainty, and goals [9].

While the overall framework of RL appears to be simple, RL problems addressed are rather intricate to solve. There are a variety of real world problems which can be represented using the formal framework of RL from playing chess [60] to legged robot locomotion [61]. Hence, the diversity of the algorithms proposed proprietarily for these tasks is vast. A decent overview on these algorithms is given in [9]. We do not include all of them in this thesis for the sake of brevity. Instead, we only mention the ones which are adapted in the following chapters of this thesis: associative search and deep Q-learning.

2.2.1 Associative Search

Common RL problems can be classified into three categories: bandit problems, associative search problems, and full RL problems [9]. The bandit problems are the simplest form of RL problems which include only a single state. An RL agent is supposed to learn desirable actions only for one situation in this case. The full RL problems, on the other hand, often incorporate more than one state (in fact, numerous states, especially in real robot applications) and transitions in between. This setting is rather more complex and it is discussed in Chapters 4 and 5 in this thesis.

Associative search problems (also known as contextual bandits) are the intermediate step between the bandit problems and the full RL problems. In this case, an RL agent is supposed to associate different actions with different situations. It learns the best actions which yield the highest rewards for particular

states. It does not consider state transitions though; thus, actions affect not the next state but only the immediate reward. Formulating real world problems in this setting is beneficial as one can still account for different states without requiring to handle the complexity of the full RL problems. In Chapter 3, we introduce a novel planning framework for swift maneuver planning of quadrotors by exploiting the associative search type of RL problem definition.

2.2.2 Deep Q-learning

The full RL problem is concerned with learning actions which maximize the rewards over the long run by reckoning state transition probabilities for different actions taken. This is a more complex setting as compared to the associative search problems since the immediate reward obtained for an action is not the real value of that action taken at that particular state. The real value, referred as $Q(s, a)$ (action-value function), generally stays hidden and can only be approximated by certain action-value function estimation methods. Dynamic programming [62], Monte Carlo methods [9], or temporal-difference methods [9] are famous examples for this purpose.

In Chapters 4 and 5, we focus on a special case of temporal-difference methods, namely Q-learning [63]. We employ deep neural networks (DNNs) to approximate the complex action-value function $Q(s, a)$. The choice of DNN for this purpose results from the fact that the number of state-action pairs is huge in the formulated RL problems in Chapters 4 and 5. Accordingly, the complexity of the action-value function $Q(s, a)$ is fairly high. Inspired by the impressive capability of DNNs for approximating highly complex action-value functions [64], we follow a similar configuration with novel DNN architectures.

2.3 Motion Primitives

Both of the RL methods of interest in this thesis (associative search and deep Q-learning) work in discrete state-action spaces. However, representing a real robot's motion in a discretized space can be fairly difficult. Conventional discrete-time model definitions in which actions are assumed to be constant over fixed time intervals may limit the fidelity of this surrogate representation. Motion primitives fill this gap by catering for a more flexible representation. They serve as a connecting bridge between the motion planning of quadrotors and the discrete RL methods throughout this thesis.

In a broader concept, a motion primitive is defined as a representation of predefined motions of a quadrotor for a finite period of time [23]. These representations can be in the form of human-interpretable expressions with underlying mathematical definitions. For example, a set of motion primitives as different flight modes are defined in [65] such as hover, forward flight, left turn, and right turn. As being different from the conventional discrete-time models, these primitives are not constrained to be constant throughout a fixed duration. In fact, the underlying mathematical functions of these kinds of primitives can be

well parameterized with respect to more finely partitioned time intervals within the total duration of the primitive of interest. Alteration and concatenation of these parameterized motion primitives can specify the whole quadrotor motion during an operation fairly well as compared to constant actions for fixed time intervals. As a result, the necessary conditions (discretized state-action space) for the above-mentioned discrete RL algorithms can be satisfied without making concessions on the fidelity. We exploit the motion primitives for this purpose throughout the thesis and adapt the discrete RL algorithms to explore the desirable shapes (Chapter 3) and sequences (Chapters 4 and 5) of these parameterized motion primitives at different hierarchical levels in the overall motion planning pipeline of quadrotors (Fig. 2.1).

In Chapter 3, we focus on motion primitives at the orientation planning level. These orientation primitives (action) alter the attitude of a quadrotor for a finite, short duration of time aggressively. Once its attitude is changed, the quadrotor has also a resulting translation motion, based upon which we calculate a reward. An RL agent here explores desirable orientation primitives that yield high rewards for different approaching velocity and spatial trajectory segment pairs (state), which allow for relatively swift and reasonably precise maneuvers. Consequently, it generates rotational motions that result in desirable translational motions for decreased navigation time in dense environments.

In Chapter 4, we focus on motion primitives at the position planning level. These position primitives (action) move a quadrotor from an initial position to an intermediate goal position within a finite time duration. A deep RL agent explores the relations between raw sensor data (state) and desirable position primitive sequences during training. Desirability of a primitive is governed by a reward function which is based on collision and relative motion towards the goal point (details of which are in Chapter 4). By concatenating the desirable position primitives later during deployment, the agent achieves collision-free navigation in unstructured environments.

In Chapter 5, we focus on motion primitives at the high-level planning. These high-level primitives supervise the low-level position planning for aerial cinematography in unstructured environments. Specifically, a deep RL agent explores the relations between video scene context (state) and camera viewpoint primitives (action) during training. It then selects favorable viewpoints on-the-fly which guide the low-level position planner that generates receding time trajectories for a filming quadrotor. Favorability of a viewpoint is determined by two distinctive reward functions: one that is based on generic cinematography principles, the other one that is based on human evaluation.

Learning Primitives in Orientation Domain

In this chapter, we focus on planning swift maneuvers of quadrotors by learning motion primitives in orientation domain. We propose a novel, RL-based method to craft desirable orientation primitives which enable swift maneuvers for fast navigation in dense environments. Our method is composed of two main stages: learning a set of orientation primitives during offline training first, and utilization of them for online planning thereafter. We propose a compact disposition of orientation primitives which consists of roll, pitch, and yaw motions to build up a swift motion library. The library is then referred in real-time and the primitives are utilized by an intelligent control authority switch mechanism when swift maneuvers are needed for particular portions of a trajectory. We compare our approach with a current state-of-the-art method for planning and control, and demonstrate improved navigation time performance up to 20% in challenging obstacle courses. We validate our approach through both software-in-the-loop simulations with Gazebo and real flights with Diatone FPV250 Quadrotor equipped with PX4 Flight Management Unit (FMU). The role of our method in the overall planning pipeline of quadrotors is as in Fig. 3.1.

3.1 Introduction

Rapid task completion of quadrotors is vital in real world applications such as inspection, surveillance, search and rescue in mines, within remainders of wrecked buildings after natural disasters, or in nuclear power plants after large scale accidents. Rapid motion of quadrotors is desired also for recently emerging leisure activities such as drone-racing. In all of these applications, quadrotors may need to react to sudden changes in their dynamic surroundings and have agile motions within dense environments. Nevertheless, planning this type of maneuvers might be compelling for the conventional motion planning methods. Since they mostly require the consideration of full state equations of motion of quadrotors for planning, the planning process is computationally demanding. Previous research addresses this issue by enabling simplifications in equations of motion [41, 66]. However, the assumptions made for enabling simplifications, i.e., near-hover flight conditions, are not valid for agile maneuvers of quadrotors.

Figure 3.1: The proposed method in the hierarchical planning pipeline of quadrotors. It takes a role in the orientation planning module.

Another line of research approaches to the problem from the aspect of finding a computationally inexpensive motion primitive [67]. Promising results are presented taking this approach by demonstrating the ability of evaluating around a million motion primitives per second on a standard laptop computer. On the other hand, a decoupled approach is proposed in [68] to reformulate the path planning problem considering separate trajectories in each translational degree of freedom. This approach enables planning trajectories of quadrotors from a large class of initial states to a target state at rest. These two approaches are yet to be integrated into a complete path planning framework as steering functions.

In [37], the researchers exploit differential flatness of quadrotors and solve the aggressive maneuver planning problem using quadratic programming. In [69], the researchers again benefit from differential flatness but they use iterative regional inflation by semidefinite programming for higher level motion planning of quadrotors during agile flights. They validate the approach by experiments with increasing obstacle density levels in [42].

A different approach is developed in [35] by sequential trajectory generation for aggressive maneuvers of quadrotors. Each sequential segment is related to a particular controller for distinctive flight modes and transitions in between. Multi-modal flight planning is also exploited in [70] for agile maneuvers of a combat aircraft. Instead of searching the whole state space for solving the planning problem, the search is focused into distinctive flight modes. Similarly in [34], planning of complex aerobatic flights are decomposed into sequences of maneuvers with safe transitions in between.

The learning-based methods also yield up-and-coming results for planning aggressive maneuvers. In [71, 72], apprenticeship learning is employed for a wide range of exceptional maneuvers with a helicopter, including stall turns, tic-tocs, and in-place flips. Both approaches try to benefit from expert pilot demonstrations and extract the characteristics of the aforementioned maneuvers. In [73], RL is used to extract policies which cater for a quadrotor to stabilize from harsh initializations, i.e., upside-down throwing, by swift recovery maneuvers. Furthermore, in [32, 33], the parameters of a priori formulated flip maneuver are learned. Based on a five-step flip primitive, five free parameters which describe the maneuver are optimized. Nevertheless, all the aforementioned learning-based methods omit the obstacles in the environment.

In this chapter, we develop a novel, RL-based approach for swift maneuver planning of quadrotors. In particular, we propose learning a set of orientation primitives in an automated fashion to build up a motion library to be referred when swift maneuvers are needed. Unlike the previous motion primitive based [67] or learning-based [32, 33, 71] approaches, our method is not only developed for attractive, high performance flights in an obstacle-free environment. It is also integrated into the overall path planning by a novel control authority switch mechanism in order to decrease total navigation time while satisfying reasonable precision in dense environments. Moreover, as being different from the previous learning-based approaches [74, 75], we conduct learning not only in simulations but also in real flights in order to upgrade the quality of the motion library by mitigating probable modeling uncertainties which can degrade the planning performance.

3.2 Approach

3.2.1 Problem Formulation

Kinodynamic motion planning problem is defined as to find the required control inputs of a robot in order to drive it from a start configuration to a goal configuration while satisfying its dynamical constraints and avoiding obstacles in the environment [14]. In this context, the specific problem that we consider herein is to find (to learn) the desired values of the carefully selected control inputs which enable a quadrotor to fly from an initial configuration to a goal configuration in minimum time by obeying intermediate configuration constraints within collision-free space. In addition, since real-time applications of quadrotors are of interest, it is a requirement to develop an approach which is computationally inexpensive and suitable for usual operating frequencies (50-100Hz) of these vehicles [76].

3.2.2 Quadrotor Model

In this chapter, our platform of interest is Diatone FPV250 Quadrotor (Fig. 3.2) which has six degrees of freedom composed of kinematics and dynamics in three axes. The kinematic equations represent the translational freedom in three inertial axes (x_I, y_I, z_I). The dynamic equations denote the rotational freedom along three body axes (x_B, y_B, z_B).

We use two individual coordinate systems, namely body and inertial frames, to describe the kinematics of the quadrotor in 3D. The body frame is the coordinate frame whose origin is located at the centre of gravity of the quadrotor while its x axis (x_B) is towards the front of the vehicle and z axis (z_B) is pointing in the direction of the thrust vector as in Fig. 3.2. Consequently, the y axis (y_B) follows the orientation defined by the right hand rule. The inertial frame is the coordinate frame whose origin is located on the ground while its z axis (z_I) is pointing towards the opposite direction of the center of Earth. Using the

Figure 3.2: Diatone FPV250 Quadrotor schematic diagram.

aforementioned two coordinate frames, the location of the quadrotor is represented in 3D with respect to the inertial frame. The orthogonal transformation matrix R_{IB} between the two coordinate frames is given by (c:cos, s:sin):

$$R_{IB} = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \quad (3.1)$$

where Euler angles roll (ϕ), pitch (θ), and yaw (ψ) are the tilting angles of the body frame with respect to the inertial frame in x, y, z directions, respectively.

There are two main forces acting on the vehicle: the gravitational force in the $-z_I$ direction and the thrust force in the z_B direction. Abiding by the rigid body assumption, Newton's equation of motion is written as [11]:

$$m\ddot{r} = -mgz_I + F_T z_B \quad (3.2)$$

where r is the position vector of the center of gravity of the quadrotor denoted in the inertial frame and g is the gravitational acceleration. The symbol F_T is the total thrust created by the propellers and it is given as:

$$F_T = \sum_{i=1}^4 F_i \quad (3.3)$$

where F_i describes the thrust created by each propeller.

Besides the aforementioned two forces, there are three main moments acting on the quadrotor as described in Fig. 3.2. The moments along the x_B and y_B axes are due to the moment arms of the propellers from the center of the quadrotor while the moment along the z_B axis results from the anti-torque effect of

each propeller. Euler's equation of motion herewith is written as [11]:

$$\dot{\omega} = I^{-1} \left(-\omega \times I \omega + \begin{bmatrix} M_\phi \\ M_\theta \\ M_\psi \end{bmatrix} \right) \quad (3.4)$$

where ω is the angular velocity vector which consists of three components and I is the inertia matrix. The symbols M_ϕ , M_θ , M_ψ represent the total moments around each axis and F_T is the total thrust force. They are related to the propeller speeds Ω_i via the input allocation matrix which is given as:

$$\begin{bmatrix} F_T \\ M_\phi \\ M_\theta \\ M_\psi \end{bmatrix} = \begin{bmatrix} k_t & k_t & k_t & k_t \\ -k_t l & k_t l & k_t l & -k_t l \\ -k_t l & k_t l & -k_t l & k_t l \\ -k_m & -k_m & k_m & k_m \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} \quad (3.5)$$

where k_t and k_m are the thrust and moment coefficients while l is the vertical distance of the propellers to the x_B and y_B axes.

In the equations of both rotational and translational accelerations, we neglect aerodynamic and gyroscopic terms because they require substantial effort to be identified precisely by a sequence of experiments and simulations. We determine the other parameters of the quadrotor by creating a detailed SolidWorks design of our platform based upon the design by [77], and matching the real-time experiments with the data obtained in simulations. They are given as in Table 3.1.

3.2.3 Motion Planning

Our motion planning framework consists of two folds: offline learning of orientation primitives within obstacle-free environments and online utilization of them for planning swift maneuvers in dense obstacle courses. The aim during offline learning is to build up an easily retainable, referable, and reproducible motion library which can be rapidly utilized in real-time.

Orientation Primitives

The effectiveness and simplicity of the motion library strongly depend on the disposition of the orientation primitives. We propose this disposition to consist of carefully selected three inputs: roll, pitch, and yaw rate. Inspired by the nature of the transmitter inputs given by expert pilots during manual flights, we propose to have Bézier curve based profiles for these three inputs. During a maneuver, control inputs given by an expert pilot via transmitter resemble high order curve shapes. These shapes can be represented by best-fit curve techniques and mapped to particular states. As a result, an intelligent agent which can mimic the expert pilot can be acquired [71]. In this chapter, we explore these curves not via a prior demonstration but with a random start to the exploration thanks to the effective representation by Bézier curves.

Table 3.1: Diatone FPV250 Quadrotor specifications [77].

Symbol	Description	Value
m	Mass (kg)	0.414
I_{xx}	Moment of inertia in x (kgm ²)	0.0015
I_{yy}	Moment of inertia in y (kgm ²)	0.0015
I_{zz}	Moment of inertia in z (kgm ²)	0.0028
k_t	Thrust coefficient (Ns ²)	7.47×10^{-7}
k_m	Moment coefficient (Nms ²)	4.48×10^{-9}
l	Moment arm length (m)	0.0884

Bézier curves are parametric curves which have been used for modeling and evaluating smooth curves, especially in computer graphics. The mathematical basis of them [78] is constructed upon Bernstein polynomials. A Bézier curve is represented via its control points and the related Bernstein polynomial expansion. The curve lies within the convex space formed by its control points. The degree of the Bernstein polynomials determines the smoothness of the curve. The mathematical formulation of a Bézier curve is stated as:

$$C : [0, 1] \longrightarrow \mathbb{R}, \quad C(t) = \sum_{i=0}^n \binom{n}{i} P_i B_{n,i}(t), \quad (3.6)$$

where P_i are control points and $B_{n,i}(t)$ are Bernstein polynomials of n^{th} degree which are given as:

$$B_{n,i}(t) = (1-t)^{(n-i)} t^i. \quad (3.7)$$

Bézier curves can approximate any curve in a parameterized manner with the appropriate choice of n and P_i . They are exploited to craft input profiles for each element of the proposed orientation primitives. During offline learning, a Bézier curve profile is explored for each distinguished maneuver while satisfying a certain criterion which is explained in the next subsection.

Reinforcement Learning

The orientation primitives in the form of Bézier curve profiles can move a quadrotor from one configuration to another. Nevertheless, it is arduous to calculate the desirable combinations of these curve profiles considering the complex dynamics of quadrotors. Aside from the difficulty of the algebraic calculations resulting from the 12-state equations of motion, calculated solutions may be far from being applicable for a real vehicle if a modeling error exists. In this vein, we propose RL to be employed for exploring the desirable curve profiles of the proposed orientation primitive elements. RL offers a favorable sub-optimality (if not the optimality) which caters for swift maneuver planning without the need of complex algebraic calculations. In addition, RL can also minimize possible performance degradations resulting from probable modeling errors since the learning is based on the realistic flights in Gazebo simulations together with the flights in real world.

RL is a nature-inspired, experience-based, simple learning method which enables the paradigm of *AI in robotics*, broadly. In its principles, there are three main elements such as state, action, and reward as mentioned in Chapter 2. Agent makes a selection among possible actions. It observes state and reward as a consequence of its action. It makes a new decision using them. The aim of the agent is to find the optimal actions which maximize the sum of the rewards [9]. We introduce each of these three elements proprietarily in this chapter to cater for relatively swift (as compared to the default planning and control strategy) and reasonably precise (without making concessions on safety in terms of collision in cluttered environments) maneuvers.

State: *State* is defined as a combination of two characteristics such as the approaching velocity of the quadrotor (u) right before the maneuver starts and the spatial path segment of the maneuver. These two characteristics distinguish different maneuvers as different states since they reflect the difficulty level of each maneuver. While we include the former directly as a number within the state representation, we incorporate the latter by using the control points (P^*) of its 4th degree Bézier curve approximation (5 control points) in three axes (15 control points). Overall, the state space is denoted as:

$$s \in S_u \times \mathbb{R}^{15}, \quad S_u = \{x : x \in \mathbb{R}, \quad u_{min} \leq x \leq u_{max}\} \quad (3.8)$$

where $u \in S_u$ and $P^* \in \mathbb{R}^{15}$. In order to create an extensive library, a number of different approaching velocities and a variety of spatial path segments are considered to diversify the states of interest¹.

Action: *Action* is the low-level, attitude plans which are composed of roll (ϕ), pitch (θ), and yaw rate (ψ). Each of them has a Bézier curve based profile (as visualized in Fig. 3.7). In particular, we use linear and quadratic Bézier curves to account for simplicity without loss of generality. We represent each action by the control points of these curves which are randomly sampled within the search space of interest. We bound the search space for the agent to avoid redundancy. In order to have a reasonably extensive motion set containing full bank and instantaneous turnaround motions, we determine the overall action cluster as follows:

$$\begin{aligned} \phi &\in S_\phi, \quad \theta \in S_\theta, \quad \psi \in S_\psi \\ S_\phi &= \{x : x \in \mathbb{R}, \quad 0 \leq |x| \leq \phi_{max}\} \\ S_\theta &= \{x : x \in \mathbb{R}, \quad 0 \leq |x| \leq \theta_{max}\} \\ S_\psi &= \{x : x \in \mathbb{R}, \quad 0 \leq |x| \leq \psi_{max}\} \end{aligned} \quad (3.9)$$

where the parameters ϕ_{max} and θ_{max} are selected as 1.57rad ($\approx 90^\circ$) while the parameter ψ_{max} is chosen to be 6.28rad/s ($\approx 360^\circ/s$).

¹During training, the parameters u_{min} and u_{max} are selected as 0.2m/s and 2.0m/s while spatial path segments are generated within rough limits of turning radius ($\pm 2.0m$) and altitude change ($\pm 2.0m$) considering our lab space limitations and Diatone FPV250 Quadrotor's capabilities for experimental demonstration. However, the agent is flexible to have different limits based on the hardware.

Reward: *Reward* is the only guidance for the agent to assess its actions while crafting the desired input profiles by altering the control points of each primitive. We define it to consist of three characteristics: maneuver execution time (T), average deviation from the path² (σ), and final deviation from the local goal point (δ). Individual rewards for these characteristics form the overall reward function as follows:

$$\begin{aligned} R_T, R_\sigma, R_\delta &\in S_R, \\ S_R &= \{x : x \in \mathbb{R}, 0 < x \leq 10 \vee x = -100\} \end{aligned} \quad (3.10)$$

where R_T , R_σ , and R_δ represent the aforementioned, individual rewards which constitute the overall reward (R) by weighted arithmetic mean. The rationale behind the bounds of each reward in (3.10) is to award the desirable actions in a decent manner while punishing the undesirable ones harshly. By this way, we intend to keep the agent away from probable collisions in dense environments. The details of each individual reward are:

$$R_T = \begin{cases} 10, & \text{for } T < T_{min} \\ \frac{k_T}{T}, & \text{for } T_{min} \leq T \leq T_{max} \\ -100, & \text{for } T_{max} < T \end{cases} \quad (3.11)$$

$$R_\sigma = \begin{cases} 10, & \text{for } \sigma < \sigma_{min} \\ \frac{k_\sigma}{\sigma}, & \text{for } \sigma_{min} \leq \sigma \leq \sigma_{max} \\ -100, & \text{for } \sigma_{max} < \sigma \end{cases} \quad (3.12)$$

$$R_\delta = \begin{cases} 10, & \text{for } \delta < \delta_{min} \\ \frac{k_\delta}{\delta}, & \text{for } \delta_{min} \leq \delta \leq \delta_{max} \\ -100, & \text{for } \delta_{max} < \delta \end{cases} \quad (3.13)$$

where the upper bounds denote the allowable values for each characteristic for which a positive reward is given and the lower bounds define the values beyond which the rewards are saturated. In other words, as the value of a particular characteristic gets lower than its saturation value, the reward increases dramatically while the performance is fairly the same. Therefore, the rewards given beyond the lower saturation bounds are fixed as 10. The parameters k_T , k_σ , and k_δ , on the other hand, equate ‘the maximum reward given within the reasonable region’ and ‘the globally maximum reward’. The rationale behind these parameters lies in our experimental scenarios. In Table 3.2, the lower limits on σ and δ are selected by considering the maximum tracking precision level that our quadrotor can achieve. The upper limits on the other hand are selected considering the maximum allowable deviation level beyond which the quadrotor would collide with the obstacles in our environment settings with narrow passages as in Figs. 3.13, 3.14, 3.15. Similarly,

²Average deviation from the path is calculated by averaging the spatial distances between ‘each data point of the 4th degree Bézier curve approximation of the desired path’ and ‘the closest point on the 10th degree Bézier curve approximation of the resultant path to that point’ over the total number of data points.

Table 3.2: Design parameters for the reward function.

Parameter	min	max
T	0.05s	5.0s
σ	0.001m	0.1m
δ	0.001m	0.1m

the lower limit on T is determined by taking into account the quickest maneuver that our quadrotor can perform, while the upper limit is selected considering the slowest maneuver that our quadrotor performs using its default planning and control strategy.

We adopt incremental RL for swift orientation primitive exploration. It is a suitable choice because the RL problem addressed herein is formulated in an associative setting in which the agent learns how to behave for different states separately. It uses the evaluative feedback for active exploration as in common RL solution methods while avoiding the complexity of the full RL problem. In this setting, each complete maneuver corresponds to one episode. Each reward obtained at each episode is used directly to update Q values without requiring an excessive memory by the following formula [9]:

$$Q_{n+1}(s, a) = Q_n(s, a) + \frac{1}{n} (R_n - Q_n(s, a)) \quad (3.14)$$

where $Q_{n+1}(s, a)$ is the updated Q value for the state-action pair (s, a) and $Q_n(s, a)$ is the current Q value before the update. The symbol n denotes the number of times that the agent has taken the action a at the state s , while the resultant reward for the action is R_n .

During the learning process, action selections are governed by ϵ -greedy method. The parameter ϵ herein describes the probability of selecting random actions while $1-\epsilon$ denotes the probability of selecting the desirable actions learned throughout the episodes. The analysis for the suitable selection of ϵ -greedy policy is given in Section 3.3.

Switching Between Modes

The extensity of the learning process determines the number of situations for which swift maneuvers can be planned. Learning a larger set of motion plans for a broader set of states lets the agent to utilize the learning-based orientation primitives on a wider range of situations, or vice versa. However, it is fairly demanding to learn all possible orientation primitives which cater for every single maneuver that a quadrotor can perform. Hence, it is a necessity to differentiate the maneuvers which have been learned and the ones which are unlearned. Therefore, we introduce a function ξ , called ‘confidence level’. It is a binary function which indicates whether or not the agent is confident on creating learning-based motion plans for a particular maneuver. It governs a binary switching mechanism as described in Fig. 3.3. In detail, the function ξ indicates whether the ϵ -close states to the current state s^* are visited during the

Figure 3.3: Overall system architecture.

learning process and the Q value is positive for the action a^* taken during at least one of these ϵ -close states to the state s^* . The binary switching is provided accordingly, as in the following:

$$\xi : \mathbb{R} \times \mathbb{R}^n \longrightarrow \{0, 1\}, \quad u \in \mathbb{R}, \quad \lambda \in \mathbb{R}^n, \quad (3.15)$$

$$\xi(u, \lambda) = \begin{cases} 1, & \text{if } \exists(u, \lambda) \in S_{past} \\ 0, & \text{otherwise.} \end{cases}$$

where λ are the data points of the trajectory in the next keyframe and the set S_{past} contains the states which are ϵ -close to the states visited during learning and for which there is at least one action explored with a positive Q value. The definition of ϵ -closure is as follows:

$$\Delta u = |u^* - u|, \quad (3.16)$$

$$\Delta \lambda = \frac{\sum_{i=0}^n d(\lambda^*(i), \lambda(i))}{n}, \quad (3.17)$$

where the $d(\lambda^*, \lambda)$ states the Euclidean distance between the data points of the curves while n represents the number of data points. Accordingly,

$$\begin{aligned} \text{IF } \exists(u, \lambda) : \Delta u \leq \epsilon_u \wedge \Delta \lambda \leq \epsilon_\lambda, \\ \text{THEN } s^* \text{ is } \epsilon\text{-close.} \end{aligned} \tag{3.18}$$

Overall, if the agent is confident on creating learning-based motion plans in a particular situation, it utilizes them for the maneuver. If it is unconfident, then the attitude plans enforced by the default geometric tracking controller are utilized to account for safety. The overall system architecture of the proposed planning framework as applied to the quadrotor is shown in Fig. 3.3.

3.3 Results

In order to account for safe learning, the *modus operandi* during learning consists of two main stages: rough learning in virtual flights first and fine learning in real flights thereafter. Since quadrotors are inherently unstable systems, direct learning in real flights without any initial, stable knowledge base may lead to undesirable motions, even a crash. Therefore, we first create a virtual representation of our quadrotor in Gazebo by exploiting the identified parameters in Table 3.1. We conduct the initial training and testing processes in this environment (rough learning). After a relatively reliable knowledge base is obtained, we proceed with training and testing in real flights (fine learning).

3.3.1 Virtual Flights

Training

A set of maneuvers within a virtual, obstacle-free environment in Gazebo are performed (explored) during training (Fig. 3.4). At each iteration, the agent selects a distinguished maneuver based upon its currently available knowledge, tries to perform it with the action decisions guided by ϵ -greedy method, and learns iteratively throughout the episodes.

For the learning in simulations, we compare different ϵ -greedy policies in terms of the quality of the primitives together with the convergence pace. The comparison is based upon 10 epochs. Each epoch is composed of 200 episodes as depicted in Fig. 3.5. In terms of the average reward value, the ϵ -greedy policy of 0.01 yields the most desirable results within the limited episode number of 200. The ϵ -greedy policy of 0.1 follows it with a little degraded average reward. The linear relaxations of ϵ from 1.0 to 0.1 and to 0.01 yield similar results to each other; they both have an increasing trend of average reward throughout the episodes. However, since both the number of episodes is constrained to bound the learning duration and the relaxation policies give an emphasis on exploration in the early episodes, they seem to have less average reward values at the end of an epoch. On the other hand, they yield superior performance

Figure 3.4: Diatone FPV250 Quadrotor learning swift maneuvers in Gazebo.

Figure 3.5: Average reward over episodes with different ϵ -greedy policies.

regarding the averaged number of desirable orientation primitives acquired within 10 epochs. The reason behind this result is that the agents with the relaxation policies are more flexible to try and explore a variety of actions at the beginning of the epochs. The constant ϵ -greedy policies restrict their exploration capabilities relatively and give more importance to exploitation. As shown in Fig. 3.6, they could only obtain desirable orientation primitives of an averaged number below 10. This number goes beyond 30 in the case of the relaxation policies while the relaxation policy from 1.0 to 0.1 yielding the highest number. Overall, it is a trade-off between ‘achieving finely polished primitives in less amount’ and ‘obtaining stable but not completely crafted primitives in more amount’. Considering the fact that the initial Q look-up table acquired in simulations will be iterated and the primitives will be refined in real flights thereafter, we favor the policy which can provide not fully refined but a larger number of desirable primitives. As a result, we select ϵ -greedy relaxation policy of 1.0 to 0.1 for batched simulations as it is able to provide vast and stable enough primitives at the beginning.

After finding the most suitable ϵ -greedy policy, we proceed with learning in batched simulations. During this process, the agent spends minimum of 50 episodes for a maneuver. If a desirable primitive is not acquired within 50 episodes, the agent visits the same maneuver in the forthcoming epochs. On the other hand, if at least one initial stable primitive is acquired in 50 episodes, the agent avoids the selection of the same maneuver in the future epochs in order to avoid redundancy.

As shown in Fig. 3.7, the agent creates attitude plans randomly within the specified bounds. The strengthened curves represent the primitives which are visited many more times since they are explored to

Figure 3.6: Averaged number of desirable orientation primitives learned through 10 epochs with different ϵ -greedy policies.

Figure 3.7: Bézier curve based orientation primitives learned over 200 episodes.

yield high rewards. The other curves are explored to be undesirable for that particular maneuver once, and they are punished with a negative reward.

Overall, the agent converges to a set of motion plans which is stored in a Q look-up table by mapping the desired actions to the corresponding maneuvers. As regards to the total training time, an epoch composed of 50 episodes lasts for approximately 100-120s on a standard Dell Precision M4800 laptop computer. The total training time is highly subject to the number and type of the maneuvers but if the simulation is kept running for approximately 8 hours, roughly 60-70 different maneuvers can be explored.

Testing

Following the batched learning stage, we deploy our agent for the exploitation of the retrospective knowledge. We consider two case studies: constant altitude operation and 3D motion. We create complicated environments including challenging turns as shown in Figs. 3.8 and 3.9. In order to have a comparison with a current state-of-the-art method, we also include the navigation results obtained via finding the required controls by a geometric tracking controller [79]. In all cases, we use A*-search algorithm for finding a collision-free path, and parameterize this path by the polynomials produced using quadratic programming to have a minimum snap trajectory.

In the first case study, we deploy the agent in a maze-like environment to navigate at a constant altitude. The nominal path is the same (planned via A*-search and minimum snap parameterization) both for

Figure 3.8: **Top row:** Left: Virtual, maze-like environment. Right: Maneuver execution time. **Second row:** Left: Default tracking controller's response. Right: Agent's response. **Third row:** Left: Trajectory error for default tracking controller. Right: Trajectory error for the agent. The agent has the control authority within green-shaded areas. **Bottom row:** Navigation time comparison.

the agent and the default tracking controller. The only difference is that the agent has the flexibility to plan fast motions using its retrospective knowledge if a previously learned state is encountered during the sharp turns. The results in the third and fourth rows of Fig. 3.8 show that the agent increases the overall navigation time performance by 25%. It executes the maneuvers during the turns up to 55% faster as depicted in the first row of Fig. 3.8. This improvement comes with a cost though, as shown in the third row of Fig. 3.8. Agile maneuvers may yield a little bit larger deviation. However, ultimately, they cater for decreasing the overall navigation time safely. It is also worth to mention that the agent plans swift maneuvers only for four turns despite the total number of turns being higher than four in this environment. In other words, it refers to the swift orientation primitive library during only four of the turns whose ϵ -close neighborhood has been explored during offline learning. When the agent encounters with a maneuver which remains unexplored after offline learning, it does not get the control authority thanks to the intelligent switch mechanism.

Figure 3.9: **Top row:** Left: Virtual, challenging, drone-racing kind of track. Right: Maneuver execution time. **Second row:** Left: Default tracking controller's response. Right: Agent's response. **Third row:** Left: Trajectory error for default tracking controller. Right: Trajectory error for the agent. The agent has the control authority within green-shaded areas. **Bottom row:** Navigation time comparison.

In the second case study, we deploy the agent in an obstacle course which resembles a challenging drone-racing track. The agent is again able to decrease the overall navigation time by utilizing the swift orientation primitives. As depicted in the first row of Fig. 3.9, the agent generates agile motion plans which allow up to 62% of improvement in maneuver completion time. The aggregated enhancement in the navigation time performance eventually reaches up to 25%. In this case, however, the deviation from the path is comparably larger in z_I due to the difficulty of the maneuvers resulting from the rapid altitude changes. Nevertheless, the agent completes the task in the obstacle course safely. An important remark herein is that the nominal path is already parameterized to push the default tracking controller to its limits. The vehicle reaches maximum speed of 6m/s in this configuration. If we parameterize the path by further decreasing the time required for completing the keyframes as described in [11], the vehicle fails to complete the track due to excessively large deviations which result in a crash.

3.3.2 Real Flights

Once the initial knowledge base is obtained in simulations, we continue learning in real flights. We take the preliminary Q look-up table acquired in simulations as a baseline and improve it throughout the repetitive maneuvers by the real quadrotor. We conduct real flight learning offline within the obstacle-free environment in order to avoid possible crashes which may occur while the agent is exploring undesirable primitives that can yield large deviations from the nominal path. In the sequel, we deploy the quadrotor for testing in challenging scenarios including sharp turns, narrow passages, and sudden altitude changes.

The overall system architecture for the real setup is explained in Fig. 3.10. The whole communication is achieved over Robot Operating System (ROS). Two ROS nodes run simultaneously on the Dell Precision M4800 laptop computer. The position and orientation information of the quadrotor is provided by the OptiTrack motion capture system at 240 Hz. Together with this information, the commands generated by the learning-based planner are conveyed to PX4 FMU through WiFi at 50 Hz.

Training

The learning in real flights leads up for the possible improvements in exploring the best-fit orientation primitives for swift maneuver planning. Although the initial orientation primitives explored in simulations are stable enough to be used for performing a variety of maneuvers in an obstacle-free environment, the vehicle occasionally exceeds the allowable deviation limits defined in Section 3.2 for a positive reward. These results stem from the fact that there are a number of hardware features which can hinder the perfect representation of the real vehicle in Gazebo simulations such as battery power, electronic speed controller performance, motor and propeller efficiency, nonlinear rotor dynamics in extreme attitude configurations, or delay in communication with the vehicle. Therefore, further exploration with the real vehicle can allow for capturing the real vehicle dynamics and planning more desirable motions.

The learning stage in real flights (Fig. 3.11) is a more conservative version of the learning phase in simulations to account for safety. We focus the search space around the stable orientation primitives by restricting the actions within 0.1rad ($\approx 6^\circ$) of random walks for roll and pitch, and within 0.1rad/s for yaw rate. We keep the ϵ -greedy relaxation of 1.0 to 0.1 as the same. During an epoch, the relaxation is continued for the first 10 episodes, while it is fixed as 0.1 for the next 10 episodes until the following battery change of the vehicle. The aim of the learning stage in real flights is to evaluate and validate the retrospective primitives for the real vehicle, and improve them if possible.

As shown in Fig. 3.12, learning in real flights results in an increased average reward at the end of 10 epochs. The agent captures finely tuned primitives with upgraded reward values by exploring within the aforementioned random walks generated around the desirable primitives learned in simulations. Although the agent yields negative rewards in the beginning due to a relatively higher emphasis on exploration

Figure 3.10: Overall architecture of the real setup.

Figure 3.11: Diatone FPV250 Quadrotor learning swift maneuvers within an obstacle-free environment.

Figure 3.12: Average reward in real flights over 10 epochs with the relaxed (1.0 to 0.1) ϵ -greedy policy.

governed by adaptive ϵ -greedy policy from 1.0 to 0.1, it extracts the useful information from the repetitive flights which ultimately cater for exploring the best-fit primitives with upgraded rewards.

It is also worth to mention that even relatively conservative random walks (0.1rad) around the desirable orientation primitives can change the flight performance significantly, as can be inferred from negative rewards. This result indicates the sensitivity of the overall motion to the inputs. It also reveals that possible, slight deviations in the motion plans generated by the conventional planning methods due to probable modeling errors can result in hazardous failures. Therefore, the proposed learning-based method presents

itself as an attractive choice for exploring the real dynamics and planning swift maneuvers confidently in an obstacle course.

Testing

Following the learning phases in simulations and real flights, we deploy the vehicle to perform rapid navigation from start to goal in three different obstacle courses. The first one includes two turns with altitude changes and a narrow corridor in between, as shown in Fig. 3.13. The second one evaluates the repetitive performance of the agent for swift 180° turns for four laps as in Fig. 3.14. The third one challenges the agent by four turns with sudden altitude changes which vary from 0.6m to 1.2m, and evaluates the repetitive performance within four laps, as depicted in Fig. 3.15. As in the testing in simulations, we compare the agent's performance with the default tracking controller in real flights.

In the first scenario, the agent is able to decrease the total navigation time by around 15% via planning two swift maneuvers through the path. The agent takes the first turn around 55% faster while this ratio is 35% for the second turn. This agility comes with a cost though. The agent yields a higher deviation from the nominal path, especially in the z_I axis. This is mainly due to the fact that while the explored elements of roll, pitch, and yaw rate are commanded to the vehicle, an additional thrust input has to be commanded too. It is regulated with a proportional-derivative controller in the current version of the algorithm whose performance is highly dependent on the power available in the battery of the vehicle. This problem occasionally occurs for real vehicles since it is difficult to maintain the same level of voltage for every flight.

In the second scenario, the focus is on a particular type of turn (180°) and the repetitive performance of the agent. In this case, the agent is able to utilize its retrospective knowledge to plan its swift motions within four laps and decrease the total navigation time by around 20%. It takes the turns up to 45% quicker as compared to the default tracking controller. The overall navigation time improvement is more in this case as compared to the former one because the agent has a higher number of opportunities to utilize its swift motion plans within four laps. As regards to the quantification of the trajectory error, if the swift maneuvers are not conducted, the overall path can be considered as a mild one which enables the default tracking controller to yield precise trajectory results. The agent, on the other hand, results in a relatively erroneous but moderately accurate trajectory. The agent completes the navigation without a crash.

In the third scenario, we deploy the agent in an obstacle course which resembles a compelling drone-racing track. The agent is again able to improve the overall navigation time performance by utilizing the orientation primitives learned during offline training. The improvement in the total navigation time is around 15% within four laps by taking the turns up to 65% faster. The deviation from the path stays moderate for the x_I and y_I axes, but the aforementioned problem is present for the z_I axis. Overall, the

Figure 3.13: **Top row:** Left: Obstacle course including two turns and a narrow corridor. Right: Maneuver execution time. **Second row:** Left: Default tracking controller's response. Right: Agent's response. **Third row:** Left: Trajectory error for default tracking controller. Right: Trajectory error for the agent. The agent has the control authority within green-shaded areas. **Bottom row:** Navigation time comparison.

proposed method demonstrates adequacy to stay fairly on the nominal x_I and y_I while performing swift maneuvers for decreased total navigation time. Another important remark herein is that although the environment includes sixteen turns in four laps, the agent plans swift maneuvers only for thirteen of them. This is due to the intelligent control authority switch mechanism. When the agent encounters with a maneuver which remains unexplored after the offline learning, control authority stays on the default tracking controller. This is also a part of the reason for relatively less enhancement on the navigation time performance in this scenario as compared to the second scenario.

As regards to the computational complexity, since the agent only needs to retain the retrospective knowledge instead of conducting complex algebraic calculations as the other nonholonomic planners, the size of the previously obtained data is the most influential parameter on the overall computation time. We conduct ten experiments with exponentially growing number of randomly generated primitives in order to have generalized results. The whole computation is conducted on the aforementioned standard laptop. We

Figure 3.14: **Top row:** Left: Obstacle course including two 180° turns to be navigated at a constant altitude for four laps. Right: Maneuver execution time. **Second row:** Left: Default tracking controller's response. Right: Agent's response. **Third row:** Left: Trajectory error for default tracking controller. Right: Trajectory error for the agent. The agent has the control authority within green-shaded areas. **Bottom row:** Navigation time comparison.

create the C++ executables for the corresponding ROS nodes and directly implement them without any code optimization. The results averaged over ten experiments show that the proposed method requires less computational power than the default method up to the presence of 1,000 primitives in the library as shown in Table 3.3. The results also indicate that the proposed method shows real-time applicability easily up to the presence of 1,000,000 orientation primitives in the library as shown in Fig. 3.16. The computation time reaches and occasionally exceeds the limit of 0.02s (operational frequency of 50 Hz) through ten experiments when 10,000,000 primitives are present in the library, which is much above the number of primitives that we use for our experiments (less than 1000) in Figs. 3.13, 3.14, 3.15. Therefore, these results demonstrate the computational feasibility of the proposed method for real-time applications.

Figure 3.15: **Top row:** Left: Obstacle course including four turns and sudden altitude changes to be navigated at a constant altitude for four laps. Right: Maneuver execution time. **Second row:** Left: Default tracking controller’s response. Right: Agent’s response. **Third row:** Left: Trajectory error for default tracking controller. Right: Trajectory error for the agent. The agent has the control authority within green-shaded areas. **Bottom row:** Navigation time comparison.

3.4 Discussion

In this chapter, we propose a novel, RL-based framework for planning swift maneuvers of quadrotors. In contrast to the conventional planning methods in which full state equations of motion are considered for each sequential states algebraically, our approach is based upon creating a motion library which consists of desirable orientation primitives explored offline in simulations and real flights. These orientation primitives are later used when the vehicle is deployed in an obstacle course if similar maneuvers to the ones which are learned during training are needed to be planned and executed.

Our approach shows competency in planning swift maneuvers without loss of safety in dense environments. The overall navigation time from start to goal in different environments is decreased up to 20% by these swift maneuver plans. We validate the proposed modus operandi both in simulations and in real flights. The agent is able to learn the preliminary orientation primitives in Gazebo simulations without

Table 3.3: Averaged computation time comparison between the default method and the agent.

Method	Computation time (s)
Default tracking controller	7.09×10^{-5}
Agent with 10 primitives	4.35×10^{-5}
Agent with 100 primitives	4.99×10^{-5}
Agent with 1,000 primitives	6.15×10^{-5}
Agent with 10,000 primitives	8.77×10^{-5}
Agent with 100,000 primitives	3.73×10^{-4}
Agent with 1,000,000 primitives	0.0022
Agent with 10,000,000 primitives	0.0199

Figure 3.16: Averaged computation time for different number of motion primitives in the library.

any prior knowledge first, and improve upon them by continuing to learn in real flights thereafter. With the aid of real flight learning, the agent is able to alleviate the probable errors which may result from the uncertainties in the hardware and extract the best-fit motion plans.

We also discuss different learning strategies for the aim of exploring the most suitable learning type for quadrotors which are inherently unstable flying machines and which may end up with a crash unless the learning is conducted with a rigorous attention. We find that more relaxed policies together with relaxed boundaries can be used in the first stage of learning since the aim is to explore a higher number of preliminary orientation primitives and since there is no risk of crash in simulations. In the second stage of learning, a more conservative learning ought to be conducted without giving a chance of unstable maneuver trials which may end up with loss of the vehicle.

There are a few limitations of our approach, some of which will be investigated as a future work. Firstly, the switching between the agent's motion plans and the controls enforced by the default tracking controller is relatively coarse. Currently, it is just a binary switch mechanism which directly gives the authority either to the agent or to the default tracking controller. This might yield abrupt motions in some cases. Especially while recovering after a swift maneuver, directly switching back to the default controller might

not be fast enough to stabilize the vehicle. In order to cater for smooth transitions between modes, a more advanced switching mechanism which incorporates intermediate planning between modes can be developed. Secondly, the effect of the battery power available on the overall motion of the quadrotor is substantial. Currently, it restricts us to use only fully charged batteries and still yields degraded performance in the z_I axis. As the commands sent to the quadrotor consist of thrust and attitude plans for the execution of the orientation primitives, we observe that the execution performance of the thrust command changes drastically for different battery power levels. The last drawback of our approach is that the agent has to explore the desirable orientation primitives for each type of maneuver unless the maneuvers are ϵ -close as defined in Section 3.2. It would be beneficial to enable generalization while reasoning for the maneuvers which are not explicitly learned and are not ϵ -close to the previously learned maneuvers. This is relatively important because the initial learning effort can be decreased substantially in this way. In this regard, we investigate the approximation of the Q function by DNNs instead of using a Q look-up table in the next chapter.

Learning Primitives in Position Domain

In this chapter, we focus on planning desirable motions of quadrotors for navigation in unstructured environments by learning primitives in position domain. We propose a novel, deep RL based method which uses raw depth images from a front-facing camera on a quadrotor in order to generate local motion plans. Since the state space is substantially larger due to the use of images as compared to the state space defined in the previous chapter, we employ DNNs to approximate the Q function. Our method demonstrates adequate end-to-end learning to generate position primitive sequences on-the-fly without conducting explicit sensing-reconstructing-planning. Omitting this cumbersome procedure, our method is computationally efficient and real-time applicable. We validate our approach through both virtual flights in Microsoft’s AirSim [80] and real flights with DJI F330 Quadrotor equipped with Intel RealSense D435. The role of our method in the overall planning pipeline of quadrotors is as in Fig. 4.1.

4.1 Introduction

Autonomous navigation of quadrotors requires sensing [81, 82, 83, 84], planning [39, 43, 50, 85], and control [11, 12, 86, 87]. Majority of state-of-the-art navigation methods perform these tasks separately to attain modularity. In this way, each module can easily be designed, developed, analyzed within their own scope, and combined with the existing modules in literature. However, if a module in the pipeline fails, other modules are most likely to fail as well. If individual modules are not developed considering possible fails of other modules, overall navigation can be deteriorated. To circumvent this issue, we develop a single module which provides end-to-end reasoning from raw sensor data to motion plans using deep RL [88, 89, 90].

According to our navigation scenario, the environment is partially known without exact obstacle location information, an initial rough path to the goal is given, and the concatenation of desirable local motion plans for safe and quick navigation is to be found. To this end, we propose an RL agent which solely uses a series of raw depth images and relative position information of a moving setpoint on the initial rough

Figure 4.1: The proposed method in the hierarchical planning pipeline of quadrotors. It takes a role in the position planning module.

path in order to generate local motion plans. We particularly train a DQN with around 400,000 parameters which exploits correlations between these two coarse inputs in order to yield desirable position primitive sequences on-the-fly.

The specific problem considered herein has been a topic of recent research in robotics community. In [50], researchers present a conservative trajectory-optimization-based local planner together with a local exploration strategy for quadrotor navigation. They build upon their previous work [91] in which a reactive local planning method is proposed to avoid obstacles using disparity maps. Both methods require an explicit reconstruction of a local map first, and perform planning within this map thereafter.

In [92], researchers introduce a collision avoidance method which exploits point cloud data and samples minimum-time motion primitives. They suggest that the computational burden of building a map can be alleviated using k -d tree but the method still needs some sort of world model representation. Another computationally lightweight method [93] introduces neural network control policies which compute control commands directly from sensor inputs. However, the method requires supervision from time-free model-predictive path-following controller in order to avoid obstacles. In [94], researchers develop a vision-based reactive planning algorithm inspired by MPC. This method requires exact representation of obstacles in order to include them in a nonlinear program formulation.

From end-to-end learning perspective, researchers in [95] train a deep RL agent using virtual raw monocular images in a variety of environments within low fidelity simulations to extract collision avoidance policies. While the method circumvents the challenges of 3D reconstruction and demonstrates decent results in terms of collision avoidance, it is yet to be incorporated into a complete path planning framework in which quadrotor navigates to the goal as quick as possible. Another learning-based algorithm is proposed in [96] which follows high-level navigation instructions by direct mapping between images, instructions, pose estimates and control. In [97], researchers train a network for detecting gates in drone racing kind of tracks followed by a desired heading and velocity generation. They demonstrate successful passes through gates in real flights but sensing and planning modules are separate in this method.

Inspired by some of the methods above, our approach sidesteps separate sensing-reconstructing-planning. It provides direct reasoning from sensor data to local motion plans. It generates (mostly) smooth

motion plans by exploiting Bézier curves as position primitives. Besides decent collision avoidance, our method caters for quick navigation to goal in dense environments.

4.2 Approach

The main challenge in developing RL-based algorithms successfully is to formulate the RL problem in a fashion that the agent would be able to derive useful representations of an environment from sensor data, and it could exploit them for generalization to new situations. In this regard, success of many RL-based algorithms lies in proper design of main elements: state, action, and reward. We explain each of these elements in the next subsections while depicting them in Fig. 4.2 as the main portions of our RL system.

4.2.1 State

State includes information on the current respective situation of the agent in the environment. It consists of three consecutive 32×32 depth images taken from the front-facing camera and the relative position of the moving setpoint with respect to the quadrotor in its body frame, x_B, y_B, z_B . The former informs the agent about obstacle locations implicitly while the latter gives insight on the proximity of the agent to the moving setpoint. Since the agent's aim is to avoid a priori unknown obstacles while moving towards the goal, these two items constitute necessary information for the agent to reason about its current situation. Depth images are taken directly from the front-facing camera without any image processing and fed to the DQN. Relative position is calculated by simply subtracting the quadrotor's position vector from the moving setpoint's and transforming the resultant vector into the quadrotor's body frame.

4.2.2 Action

Action defines the agent's move at each time step following some policy to increase rewards. It is designed to be in the form of position primitives which are based on Bézier curves. They are parametric curves based on Bernstein polynomials as given in (3.6) and (3.7). We use cubic Bézier curves ($n=3$) to represent smooth position primitives in x_B, y_B, z_B for each finite time step.

The agent selects an action at each time step among the action set which consists of 18 different primitives. This set is constructed on the account that the agent has a forward-biased motion since it has only a front-facing camera as its exteroceptive sensor. The action set is depicted as in Fig. 4.3. The length of these curves (and lines) can be adjusted for different case studies based upon environment density and quadrotor agility. In this chapter, since we consider the approximate quadrotor speed as 1m/s, we select primitives which result in at most 1m of navigation in each axis and which are executed at 1Hz of replanning.

Figure 4.2: Overview of our RL system.

Figure 4.3: Position primitives as the action set.

4.2.3 Reward

Reward serves as an evaluative feedback for the agent to assess its actions. It is designed based upon the relative motion of the quadrotor with respect to the moving setpoint within a time step. The Euclidean distance between the quadrotor position and the moving setpoint is calculated at the beginning (d_{t-}) and at the end of each time step (d_t). A raw reward is obtained using the change in this quantity through the current time step ($\Delta d = d_t - d_{t-}$) and it is discounted using the same quantity obtained at the end of the current time step. Besides, a primary logic is added to the reward function to inform the agent about collisions and excessive deviations from the initial rough path. In this sense, if the agent moves further

away from the initial path, it gets a lower reward. If it stays closer to the initial path, it gets a higher reward. If it crashes into obstacles, it gets a drastic punishment. If it deviates from the initial path excessively, it gets another punishment. We design this reward logic for the agent to learn how to avoid obstacles safely while following an initial rough path fairly in unknown environments. We formally define it as:

$$R = \begin{cases} (R_l) f(d_t), & \text{for } \Delta d_u < \Delta d \\ \left(R_l + (R_u - R_l) \frac{\Delta d_u - \Delta d}{\Delta d_u - \Delta d_l} \right) f(d_t), & \text{for } \Delta d_l \leq \Delta d \leq \Delta d_u \\ (R_u) f(d_t), & \text{for } \Delta d < \Delta d_l \\ R_{dp}, & \text{for } d_{max} < |d_t| \\ R_{cp}, & \text{for collision.} \end{cases} \quad (4.1)$$

The variables R_l and R_u are lower and upper limits on the reward, and we select them as 0 and 1 in this chapter. The terms Δd_l and Δd_u are the limits on Δd beyond which the reward is saturated to its bounds, and we determine them to be 0 and 1m, respectively. The term d_{max} is the limit which determines the excessive deviation and it is equal to 5m. The term R_{dp} is the mild punishment for excessive deviation which is equal to -0.5, and R_{cp} is the drastic punishment for collision which is equal to -1. Lastly, f is a function to regulate the discount rate based upon the maximum allowable deviation (d_{max}). It is introduced both to avoid unreasonably high discount rates when the quadrotor is fairly keeping up with the moving setpoint and to yield very high discount rates if the quadrotor fails to navigate towards the goal. It is formally defined as:

$$f : \mathbb{R} \longrightarrow (0, 1), \quad f(d_t) = \frac{1}{2} \left(\tanh \left(\frac{2d_{max} - d_t}{d_{max}} \right) + 1 \right). \quad (4.2)$$

While R_l , R_u , R_{dp} , R_{cp} , and f are crafted by trial-and-error, the terms Δd_l , Δd_u , and d_{max} are determined considering the usual operational speeds of our quadrotor in the environments of interest. All of these parameters are flexible to be adjusted for different case studies.

4.2.4 Algorithmic Details

The aim of an RL agent is to find the optimal policy which maximizes sum of returns over the long run. The essential idea behind many RL algorithms is to estimate the action-value function $Q(s, a)$ which can then govern the agent's policy by yielding the expected returns for state-action pairs. In theory, if an agent would have infinite number of action trials and would use the Bellman equation [98] as an iterative update, it would converge to the optimal action-value function $Q^*(s, a)$ eventually. However, it is impractical in real world scenarios to have infinite number of trials and estimate the action-value function for each state-action pair separately. Therefore, function approximators have emerged as a common choice to estimate the action-value function by adding a certain level of generalization [64]. In this chapter, we use

a DQN (Fig. 4.4) for this purpose, specifically to approximate the following function:

$$Q(s, a) = \mathbb{E} \left(R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') \mid s_t = s, a_t = a \right) \quad (4.3)$$

where s_t is the state that the agent is in and a_t is the action taken at the beginning of the time step t . The terms s_{t+1} and R_{t+1} are the next state which the agent reaches and the reward it observes at the end of the time step t , respectively. The term γ is the discount factor which determines the present value of the future rewards [9].

The proposed RL algorithm in this chapter is an off-policy, episodic training algorithm (Algorithm 1). Instead of updating the network parameters θ_i at each time step, we update them (using Adam [99] in PyTorch with default settings) after each certain number of episodes within which minimum number of interactions (64 in this chapter) is reached. During the update, all data in experience replay (ER) are used through random minibatches of 64. While this kind of episodic update aids on decreasing computational demand since it eliminates the need for frequent network updates at each time step, randomly drawn samples for minibatches help on breaking strong correlations between consecutive data samples [64]. We use Huber loss while creating gradient for network parameters update [100]:

$$L_\delta(y_j - Q(s_j, a_j; \theta_i)) = \begin{cases} \frac{1}{2} (y_j - Q(s_j, a_j; \theta_i))^2, & \text{for } |y_j - Q(s_j, a_j; \theta_i)| < \delta \\ \delta |y_j - Q(s_j, a_j; \theta_i)| - \frac{1}{2} \delta^2, & \text{otherwise.} \end{cases} \quad (4.4)$$

The variable j refers to data sample indice in a minibatch, δ is a constant for the steepness of the piecewise loss function and it is equal to 1. The terms y_j and $Q(s_j, a_j; \theta_i)$ are the target and estimated values for the network with parameters θ_i where i stands for the episode number.

4.3 Results

The experiments for validating the method involve training and testing stages. First, we train an agent in seven virtual environments to have diversified retrospective knowledge. Then, we test the agent in ten virtual environments which are unseen during the training stage to demonstrate sufficient generalization. Lastly, we test the same agent in real flights to demonstrate successful sim-to-real knowledge transfer.

4.3.1 Virtual Flights

Training

In this section, we present the training results using the aforementioned settings of our RL system. The simulation software we use is the Microsoft's AirSim. It is an open source software for simulating drones, cars, and people. It is based on Unreal Engine game software. It offers high fidelity simulations especially for drones. We create seven training environments in AirSim as depicted in Fig. 4.5. The environments

Figure 4.4: The proposed DQN architecture. Each element of the state is processed separately in a few layers first and combined in a few other layers thereafter in order to resemble the conceivable feature extraction and decision making paradigms.

are diversified regarding their complexity, from obstacle-free environment (Env. 1) and obstacle-free corridors with different widths (Envs. 2 and 3) to left-right (Envs. 4 and 5) and up-down (Envs. 6 and 7) slalom environments. The start points in these environments are on the side where the quadrotor is

Algorithm 1 Episodic deep Q-learning

```

for i in episodes do
  observe state  $s$ 
  for t in time steps do
    take action  $a$  governed by  $\epsilon$ -greedy policy
    observe state  $s'$  and reward  $R$ 
    save data sample  $(s, a, s', R)$ 
    if  $R < 0$  then
      break
    end if
     $s = s'$ 
  end for
  update ER with new data
  if min interaction limit is reached since the last update then
    divide ER into random minibatches of size  $n$ 
    for j in minibatches do
      pick the minibatch with indice j
      apply  $y_{1:n} = R_{1:n} + \gamma \max_{a'} Q(s'_{1:n}, a'; \theta_i)$ 
      update DQN weights by  $L_{\delta}(y_{1:n} - Q(s_{1:n}, a_{1:n}; \theta_i))$ 
    end for
  end if
end for

```

Figure 4.5: Training environments: obstacle-free, corridor, and slalom tracks.

located in Fig. 4.5. Accordingly, the goal points are on the other side, and the quadrotor tries to navigate there safely by avoiding obstacles. The distance between start and goal positions is 12m in all the training environments. We merge these environments in a single AirSim session. The agent visits them randomly during the training stage in order to randomize the learning process sufficiently. We find merging them particularly helpful for improved learning performance as compared to training in different environments sequentially. Diversification of data samples during learning is enhanced by this way, which possibly improves the data sample efficiency.

We train the agent for 2000 episodes using an ϵ -greedy policy. We utilize linear decay on ϵ from 1.0 to 0.1 for the first half of the training and keep it constant at 0.1 for the second half. We utilize the discount factor γ as 0.5 to have a balance between stable but short-sighted and far-sighted but unstable agents. The average reward throughout the episodes is shown in Fig. 4.6. The average reward first increases at the earlier episodes, then stabilizes towards the end when the agent gains sufficient knowledge for near-optimal behaviour.

Figure 4.6: Average reward through the training episodes. The white line represents the mean value of each 100 episodes over a sliding window.

Testing

In this section, we deploy the agent in ten different test environments (Fig. 4.7) which are unseen during training. They consist of obstacles with different shape, size, and orientation as well as corridors with different width and length to assess the generalization capability of the proposed agent. In each environment, the quadrotor starts on the end where it is located in Fig. 4.7 and tries to navigate towards the other end. The distance between these two ends is 30m for all the test environments¹.

We test the proposed DQN in each environment for five times to have generalized evaluative results. We also compare it with three competitor algorithms. Two of these are the preliminary versions of the proposed DQN: DQN- $\beta.1$ and DQN- $\beta.2$. DQN- $\beta.1$ is a version in which the relative position information of the moving setpoint is omitted in the state definition. DQN- $\beta.2$ is another version in which the relative position information is kept but paired with only a single depth image instead of three images. By the comparison with these two versions, we aim at an ablation study which justifies our design choices for the proposed DQN. We also include a comparison with a conventional method, namely, potential fields (PF) based planning [27], in order to fit the proposed method in the existing literature with its pros and cons. In the case of PF, the explicit sensing-reconstructing-planning process is conducted using the depth sensor. We keep the resolution (32×32) and update frequency (1Hz) of depth images as the same for a fair comparison. We create an online point cloud representation of the environment by these depth images and convert it into a probabilistic voxel map with a voxel size of 0.5m, incrementally. Each voxel creates a repulsive force proportional to its occupancy probability and inversely proportional to the square of its distance to the quadrotor. Accordingly, the moving setpoint creates an attractive force proportional to the

¹We consider tracks with the same straight length in order to have consistent benchmarking in this work. However, the proposed method is flexible to be employed in different environments with a variety of lengths of straight lines, a concatenation of straight lines, or turns. Once the intermediate goal points are put to create straight path segments as initial rough paths, it is trivial to utilize the method in these environments since the RL system is designed with respect to the quadrotor's body frame.

Figure 4.7: Test environments with obstacles of different shapes and sizes.

square of its distance to the quadrotor. We calculate the overall force vector acting on the quadrotor and map it into linear velocities by keeping the maximum velocity as 1m/s.

Table 4.1 states average navigation distance, navigation time, and crash rate for the proposed DQN along with its competitors. In terms of averaged navigation distance over five trials, the proposed DQN either outperforms its competitors or yields a similar performance to them in 8 of 10 environments. PF yields a similar but little downgraded navigation distance performance especially in Envs. 3, 7, and 9. DQN- $\beta.1$ and DQN- $\beta.2$ yield worse results. The former does not have enough guidance to move towards goal, confuses its way towards dense regions, and ends up with a crash. The latter avoids the obstacles at the first glance but does not have a sense of passing them due to the lack of previous depth images. It crashes into the obstacles mostly from the side when they are out of sight.

In terms of averaged navigation time over five trials, the proposed DQN either outperforms the others or yields a similar performance to them in all the environments. Especially when compared with its closest competitor PF, the proposed DQN's performance is superior on navigation time. It can navigate to the goal around 20% faster when the obstacles are densely located (Envs. 3, 4, 5, 6, 7), which naturally decreases the velocities generated by PF. DQN- $\beta.1$ yields slightly slower behaviour than the proposed DQN since it is deprived of the moving setpoint's relative position information as its incentive to move forward. DQN- $\beta.2$ has the incentive, but it probably does not have as much grasp as the proposed DQN over the environments due to the lack of sequential depth images.

In terms of averaged crash rate over five trials, the proposed DQN either outperforms the others or yields a similar performance to them in 8 of 10 environments. Over the 50 flights (5 trials in 10 environments), the number of safe flights is 48 for the proposed DQN, 46 for PF, 36 for DQN- $\beta.2$, and 35 for DQN- $\beta.1$. Even these results alone reveal the superiority of the proposed DQN to its preliminary versions DQN- $\beta.1$ and DQN- $\beta.2$. Thus, they justify the proposed state definition consisting of depth images and relative position information. Moreover, a crash rate similar to the renowned PF reveals the reliability of the proposed method.

Table 4.1: Averaged test results in AirSim.

Env.#	Algorithm	Navigation distance (m)	Navigation time (s)	Crash rate
Env. 1	DQN- $\beta.1$	30.00	33.44	0%
	DQN- $\beta.2$	30.00	33.18	0%
	PF	30.00	33.24	0%
	DQN	30.00	33.70	0%
Env. 2	DQN- $\beta.1$	30.00	33.35	0%
	DQN- $\beta.2$	30.00	33.45	0%
	PF	30.00	34.04	0%
	DQN	30.00	33.43	0%
Env. 3	DQN- $\beta.1$	15.07	-	100%
	DQN- $\beta.2$	23.72	-	100%
	PF	28.59	47.29	20%
	DQN	30.00	35.21	0%
Env. 4	DQN- $\beta.1$	10.36	38.56	80%
	DQN- $\beta.2$	30.00	36.37	0%
	PF	30.00	41.64	0%
	DQN	30.00	35.80	0%
Env. 5	DQN- $\beta.1$	25.97	42.04	60%
	DQN- $\beta.2$	25.36	41.75	40%
	PF	30.00	41.24	0%
	DQN	30.00	34.49	0%
Env. 6	DQN- $\beta.1$	19.43	38.18	60%
	DQN- $\beta.2$	30.00	36.35	0%
	PF	30.00	39.24	0%
	DQN	26.48	35.63	20%
Env. 7	DQN- $\beta.1$	30.00	36.14	0%
	DQN- $\beta.2$	28.29	38.50	40%
	PF	25.38	47.04	40%
	DQN	29.02	35.88	20%
Env. 8	DQN- $\beta.1$	30.00	33.40	0%
	DQN- $\beta.2$	30.00	33.17	0%
	PF	30.00	32.84	0%
	DQN	30.00	33.46	0%
Env. 9	DQN- $\beta.1$	30.00	36.66	0%
	DQN- $\beta.2$	8.59	-	100%
	PF	27.44	36.54	20%
	DQN	30.00	36.49	0%
Env. 10	DQN- $\beta.1$	30.00	37.75	0%
	DQN- $\beta.2$	30.00	36.19	0%
	PF	30.00	35.64	0%
	DQN	30.00	36.01	0%

4.3.2 Real Flights

In this section, we present the real flight test results. We deploy the proposed DQN directly on our DJI F330 Quadrotor (Fig. 4.8). Our vehicle is equipped with the flight controller PX4 FMU, the companion computer Nvidia Jetson TX2, and the exteroceptive sensor Intel RealSense D435. All the planning codes are running on TX2 utilizing its GPU for DQN in PyTorch with Cuda option. D435 provides live depth images as an input to DQN while PX4 is responsible for the low-level control of the vehicle. Odometry information is obtained using OptiTrack motion capture system. All modules communicate over ROS.

We consider three different environment configurations (Fig. 4.9): one as a obstacle-free, one with a short but wide obstacle to test up-down slalom performance, and the other one with a narrow but long obstacle to test left-right slalom performance. Since our lab space is limited to roughly 3m by 3m area, we set the initial rough path of 4m diagonally. We consider the approximate speed of the vehicle as 0.5m/s in order not to cause dangerous movements of quadrotor. Accordingly, we decrease the length of position primitives to a maximum movement of 0.5m in each axis. The rest of our RL system parameters is exactly the same as in AirSim.

We conduct five trials in each environment for real flights as well to demonstrate reliability. The proposed DQN completes 15 of 15 flights without a crash. It navigates to the goal point in all the three environments while avoiding the obstacles successfully. A remark in real flights is that the average navigation time is higher as compared to AirSim trials. This is mostly due to the control ability of our quadrotor. We simply use PX4’s position controller with default parameters which are fairly sensitive to power available in the battery. As compared to the precise execution of high-level commands without considering any motor, electronic speed controller, propeller, or battery efficiency issues in AirSim, real flight control ability is obviously less. Still, the proposed DQN yields adequate end-to-end reasoning through relatively noisy depth images from D435 and completes the task slower but with 100% accuracy (Table 4.2).

From the real-time applicability aspect, the proposed DQN requires the highest computation time on TX2 as shown in Table 4.3. This computation demand mostly results from the presence of three depth images because it increases the number of connections in the DQN drastically. This can be seen when the computation time required is compared with the one for DQN- $\beta.2$ which uses a single depth image. However, this result is obtained without any code or network optimization and it is still close to the PF’s computation time. Further development can bring this number down easily and allow for deployment over low-cost hardware.

Table 4.2: Averaged test results in real flights.

Env.#	Navigation distance (m)	Navigation time (s)	Crash rate
Env.1	4.00	22.60	0%
Env.2	4.00	19.49	0%
Env.3	4.00	21.14	0%

Table 4.3: Computation time analysis on TX2.

Algorithm	Computation time (s)
DQN- $\beta.1$	0.0340
DQN- $\beta.2$	0.0161
PF	0.0304
DQN	0.0368

Figure 4.8: DJI F330 Quadrotor with Pixhawk as a flight controller, Jetson TX2 as a companion computer, and RealSense D435 as an exteroceptive sensor.

Figure 4.9: Real environments with corresponding trajectory results for the trials with the shortest navigation duration.

4.4 Discussion

In this chapter, we develop a novel, end-to-end motion planner based upon deep RL. The proposed method demonstrates decent generalization capability by performing in a variety of environments which are unseen during training. It achieves successful flights in 63 of 65 trials in these previously unseen environments. While 50 of these flights are in simulations, 15 of them are in real flights, which yields sufficient sim-to-real knowledge transfer as well. We also conduct an extensive comparison with three competitors. The proposed method yields substantially better performance as compared to its two preliminary versions in all the three metrics: navigation time, navigation distance, and crash rate. It yields similar performance to PF-based planner regarding the last two metrics but superior performance regarding navigation time.

There are a few aspects that can be studied as an extension of this work. To begin with, a totally different set of primitives (possibly larger in number) can be utilized. In this way, the action set can be enhanced to have more freedom during planning. This would obviously expand the state-action space, which would make the Q function even more complex. It might result in a difficulty for the network to approximate the Q function. Accordingly, different (possibly larger) network architectures can be trained and tested. It is also probable that the required episode numbers and data would increase with the expansion of the state-action space. Despite all these challenges, if a network can be trained with a larger set of primitives which allow for navigation in more directions, a quadrotor with a higher navigation capability in more cluttered environments can be achieved.

Through our informal training trials, we observe that network size and structure as well as hyperparameters for learning can affect the navigation performance substantially. Considering the high dependency of RL training success on network initialization, network update frequency, learning rate, etc., the selection of these variables require rigorous attention. Using the same state, action, reward definitions, we are interested in conducting a formal search for all these variables in future.

As an extension on the hardware side, it would be interesting to create a larger network which takes more depth images possibly from multiple depth sensors around the vehicle to increase the situational awareness. In this way, the quadrotor's motion would not have to be limited to a forward-biased motion. This is particularly important while operating in a cluttered environment in which local planning requires the vehicle to move backwards sometimes.

Overall, our novel planner in this chapter demonstrates relatively safe and quick navigation in partially known environments. Our ultimate aim is to enhance our method's performance on quick navigation to goal in more cluttered environments, and present a useful tool for the robotics community.

Learning Primitives in High-Level User Input Domain

In this chapter, we focus on planning aesthetically aware motions of quadrotors for aerial cinematography by learning motion primitives in high-level user input domain. We propose a novel, deep RL based method which supervises overall motion planning of a filming quadrotor by autonomously selecting desirable viewpoint primitives which are traditionally provided by an expert user as high-level inputs. Our method exploits aesthetical values of retrospective video clips evaluated by a set of handcrafted rules as well as by human preferences in order to generate high-level user input (viewpoint) primitives which govern low-level motion planning of the quadrotor. We first conduct training and testing extensively in simulations with Microsoft’s AirSim [80] exploiting its visually realistic environments. Then, we also deploy our method for testing in real flights with DJI M210 Quadrotor equipped with Nvidia Jetson TX2. To evaluate the success of our approach, we conduct a user study in which participants rate the videos taken using our method and write comments about them. The role of our method in the overall planning pipeline of quadrotors is as in Fig. 5.1.

5.1 Introduction

Aerial filming has started revolutionizing the industries that require live and dynamic camera viewpoints such as entertainment, sports, and security. It has invoked considerable attention within both large scale drone companies and well established research groups [101, 102, 103] accordingly. Today, the accessibility of filming drones is vast. There are still a number of research problems in this area though: to create safer, more compact, aesthetically aware, user-friendly, and autonomous filming drones. In this chapter, we address aesthetical awareness of autonomous filming drones at the junction of well known scientific problems such as motion planning of UAVs in unknown environments, and motion forecasting of moving targets, e.g., humans, cars, bicycles.

Figure 5.1: The proposed method in the hierarchical planning pipeline of quadrotors. It takes a role in the high-level planning module.

Current state-of-the-art methods for aerial filming generally require partial or complete trajectory inputs from an expert user. In the cases where complete trajectory inputs are required, the job of the expert pilot is intractable. He or she is required to solve a complex optimization problem intuitively on-the-fly by considering dynamic feasibility, collision avoidance, motion forecast of the actor, and visual aesthetics simultaneously. In the other cases where only high-level supervision or partial trajectory inputs are required by an expert user, the first two considerations may be circumvented, but the last two will possibly remain to be optimized by the expert user via choosing the best viewpoints at the right time. In this vein, we attempt to automate even this process by training a deep RL agent with aesthetical awareness which aids on selecting favorable viewpoints.

Intelligent viewpoint (or shot mode) selection for aerial camera control borrows concepts from the disparate fields of virtual cinematography, aerial filming, and learning aesthetics. Camera control in virtual cinematography has been extensively examined by the computer graphics community [104]. These methods usually employ through-the-lens control where a virtual camera is manipulated while maintaining focus on certain image features [105, 106, 107, 108]. Artistic features for camera control follow different empirical principles known as composition rules [109, 110]. Several works analyze the choice of which viewpoint to employ for a particular situation. For example, in [106], the researchers use an A* planner to move a virtual camera in pre-computed indoor simulation scenarios to avoid collisions with obstacles in 2D. More recently, the method in [111] post-processes videos of a scene taken from different angles by automatically labeling features of different views. The approach uses high-level, user-specified rules which exploit the formerly labeled features in order to automatically select the optimal sequence of viewpoints for the final movie.

Specifically for aerial filming, the methods in [103, 112, 113, 114] focus on keyframe navigation, filming static landscapes or structures. In [115], the researchers touches the idea of image keyframes, which are defined by a user using through-the-lens control instead of positions in the world frame. Another line of work in aerial filming focuses on tracking moving targets in dynamic contexts. In this regard, the researchers in [101, 102, 116, 117] present different approaches with varying levels of complexity in terms of obstacle and occlusion avoidance as well as real life applicability.

As regards to learning aesthetics, several works explore the idea of learning a beauty or artistic quality metric directly from data. In [118], a measure for the quality of selfies is learned. In [119], how to generate professional landscape photographs is learned. In [120], how to transfer image styles from paintings to photographs is learned. While all the aforementioned disparate fields are mature within their own context, there is no single work which investigates the intersection of these fields. Combining these disparate fields together in a deep RL based framework, we attempt to enhance autonomous aerial cinematography further in this chapter.

Deep RL has started revolutionizing many research areas in recent years. It has shown great success in accomplishing tasks which are substantially difficult to be hard-coded or planned in advance such as playing Atari games [64] or Go [121] and learning dynamic motor skills for legged robots [61] or recovery maneuvers for multirotors [73]. Besides these tasks where a well specified reward function is available, it has yielded promising results for the tasks for which designing a suitable reward function is non-trivial such as socially compliant robot navigation [122, 123]. While both classes of problems are already challenging enough to be solved, task success subjectivity makes the learning process even harder in some cases. In this regard, deep RL has also successfully incorporated human preferences (human reward functions) [124, 125] while achieving a higher level goal. Inspired by the competitive success of RL algorithms with hand-engineered and human reward functions in literature, we propose a novel, deep RL system for autonomous aerial filming by incorporating both handcrafted and human reward functions in this chapter. We discuss their performance in capturing aesthetical values of video shots and yielding viewpoint primitives for motion planning of a filming quadrotor.

We build upon our collaborators' previous work [116] in which smooth and safe trajectories are generated in real-time using CHOMP [49]. In that motion planning pipeline, CHOMP is utilized to optimize the cinematographic trajectories considering obstacle and occlusion avoidance as well as smoothness. However, it needs a higher level input from the user, i.e., viewpoint, which defines the raw cinematographic path. CHOMP uses this path as its primitive while conducting trajectory optimization. In order to automate the overall filming process even further in this chapter, we attempt to train a deep RL agent which provides this primitive intelligently, replacing the user input. Starting from no prior knowledge, the agent selects different shot types (action) for different situations (state) and observes their corresponding values (reward) based on certain aesthetical criteria. While these criteria are coded into a complex mathematical function considering widely recognized cinematographic rules for the handcrafted reward, they are solely humans' aesthetical assessment in the case of human reward function. Using this retrospective knowledge enhanced over a number of trials, the agent explores a policy which provides desirable viewpoint primitive sequences during aerial filming. Accordingly, it replaces external user inputs and supervise the whole filming process autonomously. The overall flow of our method is given in Fig 5.2.

Figure 5.2: The proposed system flow merged into a generic RL diagram.

5.2 Approach

The main aim of the deep RL agent is to create viewpoint primitive sequences which will then be utilized during low-level trajectory optimization by CHOMP. The agent is expected to create desired viewpoint sequences while satisfying the following conditions:

- The actor is in view and within desired shot angle limits.
- The overall video sequence switches filming directions to not only show one side of the actor and keep the shot interesting.
- The quadrotor does not crash into obstacles.
- Other possible, implicit human sense of aesthetics is complied.

The main challenge in applying RL successfully in real world scenarios is to formulate the problem in such a way that the agent is able to derive useful representations of the environment, and it is able to exploit these to generalize to new situations using an informative feedback, i.e., reward. In this vein, the proposed approach can be examined in three folds, each referring to the main elements of RL, i.e., state, action, and reward.

5.2.1 State

We formulate the state (s) to have three elements: a 2.5D local height map around the actor, current viewpoint, and repetition count for the current viewpoint. The 2.5D map around the actor gives insight about obstacles close to the actor and paths the actor is likely to take based on obstacle locations. It is an informative representation of the local environment which is a grayscale image in which each pixel has a value governing the highest obstacle occupying that grid. Alongside its informativeness, it is compact enough to be fed into a DQN easily. It is represented as a 24×24 matrix with values between 0 and 255.

The second element of the state (current viewpoint) provides a grasp of favorable switching between different viewpoints. That is, although a particular viewpoint works the best for a given situation, direct switching from the agent’s current viewpoint to that viewpoint may be undesirable due to a difficult transition in between, considering the control effort for the quadrotor. While the shot modes available in today’s filming drones that provide numerous viewpoints have a variety such as left/right, front/back, orbit, high pan, fly-through, etc., we focus on the four basic types in this chapter: left, right, front, and back viewpoint. We encode current viewpoint information as a one-hot vector to be fed to the DQN.

Repetition count for the current viewpoint is a number which informs how many times the current viewpoint is repeated without switching to the other viewpoints in the preceding time steps. Since keeping the same viewpoint for a long duration would be undesirable from the cinematographic perspective, this number provides insight on whether the overall video stagnates so long where it might become boring.

5.2.2 Action

The action (a) is the viewpoint primitive that the agent selects at the beginning of each time step. Similar to the current viewpoint element of state, our action space is composed of four different viewpoint primitives, i.e., left, right, front, and back. Each viewpoint primitive selection results in a new cinematographic trajectory consisting of position and orientation. Our low-level planning algorithm (CHOMP) incorporates this trajectory into its optimization considering the other costs which are described in our collaborators’ previous work [116]. The essential aim of the agent is to explore the desirable viewpoint primitive sequences in a given situation to maximize the sum of the rewards.

5.2.3 Handcrafted Reward

Handcrafted reward is designed based on aesthetical cinematography rules. While some of these rules might be subjective and may depend on the context, some of them such as actor’s presence ratio in the frame, are fairly objective and well applicable to a majority of filming scenarios. The beauty of the second group of qualitative rules is that they can also be mathematically formulated and included in a reward function. We focus on three of them, namely, actor’s presence ratio in the shots, shot angle, and current viewpoint duration.

Actor’s presence ratio yields an informative feedback on both whether the actor is occluded in the video and whether the actor is too far from or too close to the camera. It is an essential aesthetical measure together with shot angle, especially in the context of personal filming drones whose aim is to track the owner and shoot a video. Current viewpoint duration, on the other hand, is another aesthetical measure which helps in assessing whether the overall shot is too calm with long durations of the same viewpoints or it is too dynamic with frequent switching between different viewpoints.

While calculating the reward, we first consider the actor's presence ratio and shot angle for each image taken in the current time step. We set the 'best' presence ratio and shot angle which would yield the highest reward, if achieved. Then, given an image, we calculate the reward by a linear decay from the highest possible reward via considering the deviation of these two values from their 'best'. In addition, we set upper and lower boundaries for these two values beyond which an artistic punishment is given for the image. Mathematically, these two criteria are utilized to calculate subrewards for each image as follows:

$$\begin{aligned} R_{pr}, R_{sa} &\in S_R, \\ S_R &= \{x : x \in \mathbb{R}, 0 \leq x \leq R_{max} \vee x = R_{ap}\}, \end{aligned} \quad (5.1)$$

where R_{pr} and R_{sa} represents subrewards for presence ratio and shot angle. The term R_{max} is the possible maximum reward which is set to 1, and R_{ap} is the artistic punishment given if the boundaries of these criteria are exceeded, and it is equal to -0.5 in this chapter. If the values for both criteria are within the boundaries, then the weighted average of R_{pr} and R_{sa} forms the reward for the image of interest. Otherwise, the reward for the image is equated to R_{ap} .

Once the subrewards for each image are calculated, they form the overall reward for the current time step R_t . Firstly, the average of all the rewards obtained for all the images within a time step is calculated. Then, it is discounted by current viewpoint duration as in the following:

$$\begin{aligned} \alpha_c &= \begin{cases} \frac{c}{c_{opt}}, & \text{if } c \leq c_{opt} \\ \frac{c_{opt}}{c^2}, & \text{otherwise,} \end{cases} \\ R_{discounted} &= \begin{cases} R_t \cdot \alpha_c, & \text{if } R_t \geq 0 \\ \frac{R_t}{\alpha_c}, & \text{otherwise,} \end{cases} \end{aligned} \quad (5.2)$$

where c is the repetition count for the current viewpoint and c_{opt} is the optimal value for repetition amount. Afterwards, a collision check is incorporated into the reward function in order to avoid dangerous movements of the quadrotor in dense environments. If a collision is detected during the current time step, all aforementioned calculations are discarded and a collision punishment (R_{cp}) of -1 is given as the reward. Otherwise, the reward is assigned as the value obtained from the previous calculations.

5.2.4 Human Reward

Because it might be questionable to represent aesthetical values with a mathematical reward function precisely, we investigate another training approach where the reward is directly generated by the human perception of aesthetics (Fig. 5.3). We use the same training pipeline with the same hyperparameters for our human reward training as for other training sessions with handcrafted reward. The only difference is that the reward given during training is not delivered by a mathematical function, but by humans rating

Figure 5.3: **Left:** Training pipeline with human in the loop. **Right:** Interface for incorporating human reward.

the shot quality of the scene on-the-fly. During training, multiple people from our lab constantly rate the observed camera image. Every time step (6s), the rating person is asked to give an evaluation from 1 to 5 stars for shot quality, or 0 for a collision using the interface in Fig 5.3. This rating is mapped linearly to a reward ranging from -0.5 to 1, and -1 for a collision, respectively. By this way, we make the range of human rewards resemble the possible rewards from our handcrafted reward configuration.

5.2.5 Algorithmic Details

The essential idea behind the use of DQN is to estimate the highly complex action-value function $Q(s, a)$ which yields the value of an action taken on a particular state considering its long run gains for the agent as explained in Section 4.2. For the sake of completeness, it is formally reproduced here as:

$$Q(s, a) = \mathbb{E} \left(R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') \mid s_t = s, a_t = a \right) \quad (5.3)$$

where s_t is the current state and a_t is the action taken. The terms s_{t+1} and R_{t+1} are the next state which agent reaches and the reward it observes, respectively. The term γ is the discount factor which determines the present value of future rewards.

The DQN considered in this chapter is composed of fully connected neural networks which fuse three different inputs in three main lanes. The architecture is depicted in detail in Fig. 5.4. Three inputs (a 2.5D local map, current viewpoint, repetition count for the current viewpoint) are first fed through three different lanes separately. Then, they are combined in a single lane which eventually yields the Q values for each action. All consecutive layers are fully connected to each other, and all of them applies ReLU except the very last layer. We train this network using Adam optimizer [99] with default settings in PyTorch. We use Huber loss [100] for creating the gradient for the parameter update as also explained in Section 4.2. The pseudocode for overall training using ER with minibatches is given in Algorithm 2.

Figure 5.4: DQN architecture.

Algorithm 2 Episodic deep Q-learning using ER with minibatches.

```

for  $i$  in episodes do
  observe state  $s$ 
  for  $t$  in time steps do
    take action  $a$  governed by  $\epsilon$ -greedy policy
    observe state  $s'$  and reward  $R$ 
    save data sample  $(s, a, s', R)$ 
    if collision occurs then
      break
    end if
     $s = s'$ 
  end for
  update ER with new data
  if  $i \% \text{min episodes for update} == 0$  then
    divide ER into random minibatches of size  $n$ 
    for  $j$  in minibatches do
      pick the minibatch with indice  $j$ 
      apply  $y_{1:n} = R_{1:n} + \gamma \max_{a'} Q(s'_{1:n}, a'; \theta_i)$ 
      update DQN weights by  $L_\delta(y_{1:n} - Q(s_{1:n}, a_{1:n}; \theta_i))$ 
    end for
  end if
end for

```

5.3 Results

The experiments for validating the proposed framework are composed of training and testing stages. We first train an agent in visually realistic AirSim simulations extensively. We create three different environments which feature a variety of surroundings, e.g., block-shaped obstacles, mountain-shaped structures, photo-realistic urban areas. We train different agents separately in each of these environments.

We then deploy them both in their training environment and in the other environments. By this way, we present the performance of our method both for previously seen and unseen conditions. Besides, we conduct a user study in which 10 participants rate video clips by our method as well as competitor policies, i.e., fixed shot mode, random shot mode. We present these results for several selected video clips. In the end, in order to show the applicability of our method in real world conditions, we deploy the agent in real flights in the Gascola region near Pittsburgh, Pennsylvania (PA), United States of America (USA).

5.3.1 Virtual Flights

Training

We start the training phase in a relatively simple environment called ‘Block World’. The actor is walking on a straight path (red path in Fig. 5.5) with alternating blocks on left and right with varying lengths and heights in this environment. We keep the maximum episode number relatively low as 300 for this environment to account for its simplicity. The average rewards obtained during training for two different agents guided by the handcrafted reward and the human reward are given in Fig. 5.6. The increase in the rewards is prominent for both agents, which indicates successful learning.

We continue training with another agent in Big Map environment. This is a much larger map which is separated into three distinguishable areas, i.e., slightly more complex blocks, two parallel rows of pillars, and mountain-like structures. The actor is walking on a longer path this time (yellow path in Fig. 5.5), and we set the maximum episode number as 2000 accordingly. Since it is difficult for participants in the human reward function study to rate the videos for 2000 episodes (approximately 15 hours), we only train an agent guided by the handcrafted reward in this case. The increasing average reward trend is again visible as shown in Fig. 5.7. This demonstrates the learning ability of the agent even in relatively complex environments featuring a variety of obstacles, e.g., blocks, pillars, piles of rubble.

Lastly, we train two different agents in photo-realistic Neighborhood environment. This environment enables a high fidelity representation of real world conditions. It features a suburb including streets, houses, bushes, trees, cars, etc., which could be a possible candidate for a movie scene in real life. The actor follows the orange route in Fig. 5.5 during training. The maximum number of episodes is 500 for this case. The average rewards for both the agent guided by the handcrafted reward and the agent guided by the human reward yield increasing trends (Fig. 5.8). The handcrafted reward yields more steady increase as compared to the human reward having a drastic up and down through the episodes. This behaviour is also visible for the training in Block World (Fig. 5.6). It is possibly caused by subjective evaluations of different human raters. Nevertheless, the differences between the final and initial average rewards are quite prominent for both handcrafted and human reward. These results are significant to demonstrate adequate learning almost in real world conditions which promises possible deployment in real flights.

Figure 5.5: Screenshots and height maps of training environments in AirSim. **Left:** Block World. **Middle:** Big Map. **Right:** Neighborhood.

Testing

Following the extensive trainings, we deploy the agents in the same set of environments for testing. We start with testing the agent trained in Block World for 300 episodes using the handcrafted reward. We deploy the agent in the same environment but on a different path (blue path in Fig 5.5). Despite its simplicity, this environment offers a decent opportunity to evaluate the performance of the agent since it has a clear pattern of blocks that the agent can avoid. We compare the results with the result of the random policy on the same path in Table 5.1. As can be seen, the trained policy performs significantly better than the random policy even on a previously unseen path. Our agent accomplishes the following goals in this environment:

- It keeps the actor in view by preferring 90° viewpoint switches (i.e., from left to back or front) over 180° switches (i.e., from left to right) which often cause losing sight of the actor.
- It switches viewpoints regularly to keep the shot interesting.
- It avoids flying above obstacles to keep the shot angle in desirable limits.
- It avoids extremely high obstacles which might pose collision risk.

These observations are also complied by the results in Neighborhood over the purple route in Fig 5.5. We deploy the agent trained for 500 episodes using handcrafted reward together with the random policy. Table 5.2 summarizes the rewards obtained for this setting while Fig. 5.9 visualizes how primitives guide the quadrotor motion in this photo-realistic environment. The trained policy yields much higher reward as

Figure 5.6: Average rewards (over each 30 episodes) totally for 300 episodes training in Block World. **Left:** Handcrafted reward. **Right:** Human reward.

Figure 5.7: Average rewards (over each 30 episodes) totally for 2000 episodes training in Big Map for handcrafted reward.

Figure 5.8: Average rewards (over each 30 episodes) totally for 500 episodes training in Neighborhood. **Left:** Handcrafted reward. **Right:** Human reward.

compared to the random policy over a previously unseen route. Therefore, it is evident that the agent is able to learn a policy on selecting favorable viewpoint primitives even for previously unseen conditions.

Table 5.1: Average rewards obtained during testing on a previously unseen path in Block World by the agent trained in Block World for 300 episodes using the handcrafted reward.

Trained policy	0.3581
Random policy	0.0616

Table 5.2: Average rewards obtained during testing over a previously unseen route in Neighborhood by the agent trained in Neighborhood for 500 episodes using the handcrafted reward.

Trained policy	0.5882
Random policy	0.2047

Figure 5.9: Time lapse of quadrotor motion during filming in the photo-realistic environment. Since the left hand side is occupied, the primitives guide the quadrotor to go from the left to the front, then to the right viewpoint.

We further test the capabilities of our method for generalization to previously unseen classes of environments. Table 5.3 states the rewards obtained during testing in all the three environments using the three different policies explored in these three environments. As shown, all the trained policies perform better than the random policy in all environments. This implies successful learning which can be utilized both for previously seen and unseen conditions. Diagonally placed highest rewards indicate that each trained policy is superior for the environment which they are trained in. This is also an expected result as the agent would be more familiar with the environment which it has seen previously since it captures the particular features of that environment during training.

User Study

In the former subsection, our evaluations on the testing performance are based upon the average rewards obtained during testing. Hence, these evaluations are driven by the criteria we set for calculating the handcrafted reward. On the other hand, visual aesthetics can be quite subjective and difficult to define through a mathematical function. Assessing the real aesthetical beauty of shots might require more

Table 5.3: Average rewards obtained during the testing phase.

		Testing Environment		
		Block World	Big Map	Neighborhood
Trained Policies	Block World	0.3444	0.3635	0.3622
	Big Map	0.2178	0.5052	0.5142
	Neighborhood	0.1822	0.4311	0.5198
Random Policy		-0.0061	0.1944	0.2417

complicated criteria than those. In this vein, we conduct a user study for a more cogent evaluation of the video clips produced using our method. We ask 10 participants to watch the video clips taken by four different policies, and order them from ‘the most visually pleasing’ to ‘the least visually pleasing’. We also ask them to write short comments about each video clip in order to infer what the features of a desired video clip are. The four policies we deploy are:

- Policy guided by the handcrafted reward,
- Policy guided by the human reward,
- Random viewpoint policy,
- Back viewpoint policy.

We present participants totally five scenes (Fig. 5.10): two from Block World, three from Neighborhood. Each video clip has the length of 30s. The agents trained in these environments are deployed in their respective environment but over previously unseen paths. We randomize the presentation order of the video clips to the participants in order to minimize possible biases.

The linearly transformed version of the user ratings are given in Table 5.4. The agents using the handcrafted or the human reward outperform the back viewpoint and random viewpoint policies significantly. The video clips by these trained agents are consistently rated as the two most visually pleasing ones in different scenes by the participants. The policy guided by the handcrafted reward mostly ranked higher as compared to the policy guided by the human reward. While these results indicate the success in our formulation of the handcrafted reward function for capturing aerial filming aesthetics, open-ended comments provided by the participants offer further insight on human evaluation criteria on the same aesthetics. We summarize them in the following items:

- All participants criticize the back viewpoint policy as ‘boring’ or ‘unexciting’.
- All participants mention that the random viewpoint policy loses the view of the actor too often.
- The most common complaint is that the actor gets out of view.
- There is a small window of how often the view angle is supposed to change. Many participants complain about too few changes, when only one viewpoint change is performed in an episode (30s).

Table 5.4: Average score of video clips in the user study (from 0: worst to 10: best) via linear transformation from average rated position.

	Average	Scene 1	Scene 2	Scene 3	Scene 4	Scene 5
handcrafted reward	8.2	10.0	5.3	9.3	7.7	8.7
human reward	7.1	5.0	9.0	6.0	7.7	8.0
back viewpoint	3.8	4.0	4.7	4.3	4.0	2.0
random viewpoint	0.9	1.0	1.0	0.3	0.7	1.3

Figure 5.10: Five scenes selected from Block World and Neighborhood. The actor follows the same trajectory for different policy runs. The quadrotor trajectories via following the highest rated policy are shown.

On the other hand, many participants criticize too much camera movement in a scene where the quadrotor switches every time step (6s).

- Participants frequently mention that they would like to get an overview of the surrounding instead of just seeing the actor in front of a building or a wall. When the agent gives multiple fields of view around the actor by switching between viewpoints, this is positively remarked.
- The handcrafted reward policy is often described as ‘the most exciting’ while the human reward policy is described as ‘smooth’.

The main difference between the handcrafted reward policy and the human reward policy is the consistency of switching viewpoints. Our handcrafted reward function leads to a policy that tries to switch exactly every two time steps (12s), if not disturbed by an obstacle. Based on the user study results, this seems to be a relatively favorable time frame that achieves not being too boring while still giving enough time to get a decent impression on the scene context. The human reward policy is less consistent and switches viewpoints more irregularly. Another interesting result of the study is that losing the view of the actor, even for a very short moment, is always rated as ‘very poor’ by the participants. No policy that loses sight of the actor at any point is rated the highest in any scene. These results justify the actor’s presence ratio definition in the handcrafted reward well.

5.3.2 Real Flights

After presenting extensive training and testing results in virtual flights together with a user study, we deploy our method for aerial filming in real world. We use a DJI M210 Quadrotor equipped with a companion computer Nvidia Jetson TX2 and an independently controlled gimbal DJI Zenmuse X4S as seen in Fig. 5.11. We deploy this vehicle in the Gascola region near Pittsburgh, PA, USA.

We employ the agent trained in Big Map for 2000 episodes for real flight testing. We compare the video clips produced by two other policies, i.e., back viewpoint and random viewpoint. In the videos, it is fairly visible that the agent outperforms the other two. It switches viewpoints based on the relative location of the heap of rubble to the actor as well as to itself. These viewpoint selections comply with the learned principles to form a smooth, occlusion-free, and interesting video clip. A scene from the experiments in which the agent is switching shot types near the heap of rubble is given in Fig. 5.12.

It is worth to mention that the overall performance in real flights is little downgraded as compared to the virtual flights. At some point, the agent selects not the most desirable but the second best primitive near the heap of rubble. There might be two reasons for this result. Our experimental facility is a relatively open area with less obstacles around as compared to the environments in AirSim. Even while flying quite near the heap of rubble, the agent might interpret the situation as fairly relaxed and select the shot modes which would fly the quadrotor even closer to the obstacle. Another reason might be a possible misalignment in the local map of Gascola due to probable odometry errors. This might cause the agent to interpret near obstacles as slightly far and may affect the decision-making accordingly. Nevertheless, despite all these possible hindrances in real world conditions, our agent performs better than the baseline competitors. While random policy results in a turbulent movement which cause losing the sight of the actor, back policy produces a relatively stable but also unexciting video clip due to the presence of the same viewpoint all along. Therefore, our agent is able to transfer what it has learned in virtual flights to the real world conditions fairly.

5.4 Discussion

In this chapter, we propose a novel, deep RL based method for planning aesthetically aware motions of quadrotors in the scope of aerial filming. As shown in the results in the previous section, our method generates desirable primitive sequences which guide the quadrotor from one viewpoint to another at the right time to produce smooth and interesting video clips. The results obtained in both virtual and real flights show that a deep RL agent can learn the complex correlations between selecting viewpoints and producing visually pleasing video clips. Accordingly, the agent can replace the high-level inputs provided by an expert user and autonomously supervise the overall motion planning of a filming quadrotor.

Figure 5.11: **Left:** DJI M210 used for the field experiments. **Right:** Testing facility with a heap of rubble in the Gascola region near Pittsburgh, PA, USA.

Figure 5.12: The agent follows the actor while switching viewpoints autonomously in real world. **Top row:** It starts at the back of the actor. **Middle row:** The heap of rubble on the right side of the actor poses occlusion and collision risk. Thus, the agent switches to the left viewpoint. **Bottom row:** The agent switches to the front viewpoint in the open area.

In all the case studies included in this chapter, our method outperforms the baseline policies, i.e., back viewpoint and random viewpoint. Although the success margins can change for different environments, our method always produces visually pleasing video clips in contrast to the back viewpoint and the random viewpoint policies. It is evident both from the average reward values and the rankings by the participants in the user study that the aesthetical quality of the video clips taken using our method is significantly better. While these results serve as a proof of concept for aesthetically aware motion planning in aerial filming, there is still room for improvement in a few aspects.

Currently, a time step lasts for 6s in order for a viewpoint primitive to be utilized by CHOMP and executed by the quadrotor. This is the shortest possible duration for a time step: a limitation imposed by CHOMP’s replanning rate and the controller’s ability to follow the path generated by CHOMP in between the viewpoints. It takes time to move between viewpoints, especially when the actor is on the move. On the other hand, shorter the time step length is, more chances the agent would have for switching viewpoints. This might be particularly important in excessively dense environments unlike our testing facility in the Gascola region. In order not to lose the view of the actor, the agent may require more frequent switches between viewpoints. To this end, the action space can be expanded to include more intermediate viewpoints between the four main viewpoints used in this chapter. This might ease the job of CHOMP while utilizing consecutive primitives and enable for decreasing the time step length. Once the time step length is shortened, the capacity of the agent can be enlarged by giving the opportunity of more frequent switching. The only probable con of this approach might be the need for a longer training duration as the action space would be significantly enlarged.

Another aspect to study is to increase the number of participants as well as to diversify participants’ background for providing grades on the video clips as rewards. Currently, they are a relatively small number of volunteers from our lab. Expanding the participant cluster both in quantity and diversity may lead to interesting results. There can be more conflicts between different people’s choices. This might cause hardships for the agent to converge to a desirable policy, but at the same time once the agent is converged, the policy obtained would yield more widely accepted aesthetical features for video clips.

Another aspect to investigate can be shaping the reward function. Currently, the handcrafted reward function seems to yield the best results. This is due to the rigorous crafting of the reward function by designing each element with a great attention. Nevertheless, it may require substantial effort to engineer these and find suitable hyperparameter selections for each element. Moreover, the crafted reward might still not perfectly represent the aesthetical values of a video clip since the real aesthetics remain as fairly subjective and difficult to be included in a mathematical function. To this end, we also employ human reward function but it requires even more effort for people to sit in front of a computer and rate video clips for hours. Therefore, a more elegant and less effortful technique can be developed for shaping the reward function. Inverse RL is one of the promising options for this purpose. Given a huge amount of previously taken professional video clips (from movies, series, or online platforms), an inverse RL agent might have a good chance to learn a reward function which can capture the human sense of aesthetics. Subsequently, it can guide the motion of a filming quadrotor by trying to comply with the learned aesthetics.

Conclusion

In this thesis, we have developed several motion planning methods for quadrotors based upon RL. We have focused on real world applications of these vehicles and addressed respective challenges such as partially known situations, onboard computation and sensing limitations, and inherently subtle criteria for task success. We have shown that our methods can be applied proprietarily at each hierarchical level in the overall motion planning pipeline of quadrotors.

6.1 Planning in Orientation Domain

6.1.1 Summary

In Chapter 3, we showed that the orientation primitives learned by experience cater for relatively swift and reasonably precise maneuvers. We first started with learning orientation primitives in Gazebo simulations since learning in a virtual environment is both cheap and safe. However, it is not necessarily the most precise one in terms of capturing the real vehicle dynamics due to the limited fidelity. Hence, we continued learning in real flights to capture what could not have been simulated by Gazebo, e.g., delay in executing commands. Our RL agent explored a set of primitives over the real vehicle based upon the primitives learned during simulations. We then deployed our agent for navigation in dense obstacle courses. We observed faster maneuver execution up to 65% which catered for improved navigation time up to 20% as compared to a conventional trajectory tracking algorithm. We also conducted a computational demand analysis via employing exponentially increasing number of primitives. Our approach required less computational power up to the presence of 1,000 primitives as compared to the baseline method. All these results validated our RL-based approach for swift maneuver planning of quadrotors when a precise mathematical model is not available and onboard computation is limited.

6.1.2 Lessons Learned

Certain lessons have been learned from our work in Chapter 3. Since we adopted associative search with a Q look-up table for exploring desirable orientation primitives, the generalization capability to unseen situations was limited. In other words, each primitive was explored for a particular state (and for its ϵ -close neighbours as defined in Chapter 3) and was utilized when those states are encountered during deployment. We opted for DNNs in Chapters 4 and 5 for better generalization capability but it is worth to mention that this limiting feature in Chapter 3 had also a positive side effect. Since the agent only utilized the orientation primitives within the neighborhood of previously seen situations, it has never made concessions on safety which is a major concern in robotics. It allowed for the utilization of the learning-based motion plans only during the situations where it was confident.

Another important lesson in Chapter 3 was the effect of the battery voltage on the overall motion of the quadrotor. We observed that the same orientation primitive utilized during two different battery voltage levels resulted in slightly different motions. This was probably due to the low-level controller of our customized quadrotor and may not be valid for branded quadrotors. However, it forced us to use only fully charged batteries during real flight experiments.

6.2 Planning in Position Domain

6.2.1 Summary

In Chapter 4, we showed that the position primitives can be correlated with raw data from onboard sensors for learning navigation policies which can be applied in unstructured environments. In the light of the lessons learned in Chapter 3, we approximated the Q function using a DNN for better generalization in relatively high dimensional state-action space in Chapter 4. We first trained a DNN in high fidelity AirSim simulations. We created a diverse ensemble of environments including combination of different shapes, lengths, and heights of obstacles. Our RL agent has learned certain policies which cater for obstacle avoidance and navigation to goal simultaneously. It used only a pair of depth images and relative position information with respect to a virtual moving setpoint for decision-making at each time step. We demonstrated successful navigation by deploying our agent in previously unseen environments in AirSim. Furthermore, we deployed our agent in real flights without any modification on the DNN which was trained in simulations and showed sufficient knowledge transfer from virtual to real environments.

6.2.2 Lessons Learned

The lessons learned in Chapter 4 include the importance of network architecture and hyperparameter selections on the overall performance. While attempting to obtain a decent set of results with our method,

we have tried a number of different network architectures. We have observed that different architectures may result in a similar performance if they are trained properly. On the other hand, there are certain architectures which perform much better than the others. In DNN research, it is quite difficult to set the ultimate guidelines in this regard, but our observations suggest that simple architectures work better unless they are too simple. Reduced number of layers or decreased layer sizes always have a potential to yield better results according to our observations. This obviously does not mean that the simplest architecture would work the best, which implies a single neuron would do the job perfectly. What we suggest is that it is always worth to try a simpler version of a well working DNN for the same task. It might have a greater potential than the original DNN.

Hyperparameter selection is equally important as network architecture for DNN training. Discount factor (γ), balance between exploration and exploitation (ϵ) are the most significant hyperparameters in RL according to our observations. Discount factor determines the current value of the future rewards in the current Q value. If it is too low, then the agent is said to have myopia and it tries to maximize mostly the immediate rewards. If it is too high, then the agent is said to be farsighted and it tries to maximize the rewards over the long run. While the second configuration seems more desirable for sequential decision-making problems, it might result in divergence of Q values during DNN training easily. The main reason is that the parameters of the DNN are updated at each episode using the Q values which are estimated by the previous version of the same DNN. If the previous version is not mature enough to provide precise Q value estimations, a high discount factor results in taking those faulty estimations into account more during the current update. According to our observations, this creates a chain reaction at each update and yields totally divergent Q estimations in the end. Therefore, we selected a discount factor of 0.5 which achieves a balance in this trade-off. For the balance between exploration and exploitation, we incorporated an annealing logic for the ϵ parameter. As the DNN became more mature, our agent started to rely on it more and emphasized on exploitation while it was mostly opting for exploration at the beginning of each training.

6.3 Planning in High-Level User Input Domain

6.3.1 Summary

In Chapter 5, we showed that a deep RL agent can map a video scene context to camera viewpoint primitives during aerial filming for creating aesthetically aware motions of a filming quadrotor. Similar to Chapter 4, we employed a DNN for the approximation of Q function. Different from Chapter 4, we trained the DNN using both handcrafted reward and human reward. We conducted the whole training in visually realistic AirSim simulations which are substantially helpful when people in our lab take part in

human reward function and evaluate the video clips. They could easily do it by watching realistic videos provided by AirSim. We created a diversified set of training environments which feature a variety of video scene contexts. After training different agents in each of these environments separately, we deployed the agents in all the environments. In order to quantify the success of our agents, we compared their rewards with those of random and fixed viewpoint policies. Moreover, we conducted a user study on the video clips obtained during testing. Our trained agents outperformed the two baseline policies in terms of creating aesthetically pleasing video clips. In the end, we deployed an agent trained purely in simulations for real aerial cinematography in real world. We demonstrated sufficient generalizability to real world conditions in these flight tests.

6.3.2 Lessons Learned

The lessons learned in Chapter 5 share similarities with the ones in Chapter 4, e.g., importance of hyperparameters and network architecture. Moreover, there are a few application-specific lessons such as the significance of the simulation fidelity for generalization from virtual to real environments. The first element of video scene context was a 2.5D map in our RL framework in Chapter 5. The exact map was provided to the agent during the training in AirSim. However, when we deployed the agent in an unstructured environment in real world, there was only a map obtained using the LiDAR onboard. Since it was reconstructed online with relatively noisy data, it was difficult for the agent to reason about it. Although the agent outperformed the baseline policies, its performance was not as good as in the simulations during testing in real flights. Similarly, odometry information and actor heading estimation was close to ideal in simulations while they were little downgraded in real flights. We anticipate that if the agent would be trained in a simulation environment which could incorporate all these details, its performance could have been better in real flights. Training phase would include these uncertainties and the DQN could have been fed with more realistic data. All of these suggest that higher fidelity levels in simulations could help in improving real world deployment performance.

Another application-specific lesson was the effect of sparse actions. We have incorporated only four main viewpoint primitives, i.e., left, right, front, back due to the limitations enforced by our low-level planning algorithm, CHOMP. This selection constrained the frequency of decision-making to once in every 6s. CHOMP could only generate the paths between different viewpoints and the tracking controller of our quadrotor could follow these in a recognizable manner only in around 6s. This duration could be decreased by introducing intermediate viewpoints among the main four viewpoints utilized in Chapter 5. In this way, the RL agent would have more chances for switching viewpoints. This might be particularly useful to deal with high occlusion risks possibly caused by numerous obstacles in dense environments.

6.4 Future Work

In this thesis, we have taken important steps on employing RL to enhance motion planning of quadrotors at different hierarchical planning levels. We have demonstrated the benefits of RL for motion planning in real world by developing and testing novel, RL-based planning frameworks. Looking forward, all the planning frameworks proposed in this thesis could be improved by aiming to overcome their current drawbacks and limitations discussed in the previous subsections.

In a broader view for future work, we have observed that the great potential of RL for motion planning in real world lies in the end-to-end learning setting. Learning from experience could help in dealing with uncertainties in different contexts as we demonstrated throughout this thesis, but specifically the end-to-end learning configuration in Chapter 4 seems much promising to us for future investigation. Currently, end-to-end learning is fairly rudimentary in robotics research, especially when it comes to real hardware deployment. The main reason behind this is safety and reliability. Presently, it is not possible to predict how worse a DNN can behave if it is trained with unhealthy data or if its training is not complete to comprehend all possible situations that a robot can encounter. If further research can enable a metric for assessing the ‘wisdom level’ of a DNN and utilizing this information during deployment (similar to our ‘confidence level’ in Chapter 3), it would be an important step regarding safety and reliability. Another gap in end-to-end learning is being not able to access the information in intermediate layers of a DNN. At present, a DNN acts as a powerful black box which has all the authority from observing data to taking action. Obviously, this is precisely the idea behind end-to-end learning: extracting complex data-action relations directly and taking action given raw data by discarding intermediate modules in between. However, what we point out here is the ability of seeing into those complex relations in intermediate layers. This can accelerate the end-to-end learning research regarding safety and performance. In the terms of safety, a quantifiable relation in between intermediate layers can help while assessing whether the knowledge captured by a DNN is desirable or not. In terms of performance on the other hand, it can be substantially helpful during the training phase by focusing the learning towards what has not been learned yet, instead of conducting it randomly. If further research can enable this ability, we believe that it will create a great impetus for deep RL agents to be deployed on real robots. Currently, the closest research branch in this regard is the explainable AI [126], but it is mostly considered over supervised learning not RL.

Overall, we believe that unified, intelligent modules which can reason about raw and noisy data will successfully pave the way for a different era in motion planning once the aforementioned safety issues are solved. Accordingly, we also foresee that the future with the aerial robots which can think, reason, and react in our daily life using only the data available by their onboard sensors is nearer than what most people think today.

Video Links

- Planning in Orientation Domain: <https://youtu.be/TX0RYuDpvfw>
- Planning in Position Domain: <https://youtu.be/hw0sxNwliqs>
- Planning in High-Level User Input Domain: <https://youtu.be/67qbXfSwb54>

Bibliography

- [1] <https://thedroneracingleague.com/>, (last accessed in April 2019), April 2019.
- [2] <https://thedrone.co/>, (last accessed in April 2019), April 2019.
- [3] <https://www.logistics.dhl/global-en/home/insights-and-innovation/thought-leadership/trend-reports/unmanned-aerial-vehicles.html>, (last accessed in April 2019), April 2019.
- [4] Steven Millward. Singapore restaurant shows off autonomous drone waiters. <https://www.techinasia.com/singapore-restaurant-autonomous-drone-waiters>, (last accessed in April 2019), February 2015.
- [5] Zacc Dukowitz. Drones in search and rescue: 5 stories showcasing ways search and rescue uses drones to save lives. <https://uavcoach.com/search-and-rescue-drones/>, (last accessed in April 2019), January 2019.
- [6] <https://www.microdrones.com/en/industry-experts/security/monitoring/>, (last accessed in April 2019), April 2019.
- [7] Y Bouktir, M Haddad, and T Chettibi. Trajectory planning for a quadrotor helicopter. In *2008 16th mediterranean conference on control and automation*, pages 1258–1263. Ieee, 2008.
- [8] Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. *arXiv preprint arXiv:1810.06224*, 2018.
- [9] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. <http://incompleteideas.net/book/bookdraft2017nov5.pdf>, (last accessed in December 2019), November 2017.
- [10] Katsuhiko Ogata and Yanjuan Yang. *Modern control engineering*, volume 4. Prentice-Hall, 2002.

-
- [11] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2520–2525. IEEE, 2011.
- [12] Ezra Tal and Sertac Karaman. Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 4282–4288. IEEE, 2018.
- [13] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. Springer, 2016.
- [14] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.
- [15] Scott Graham. Drone racing league was our idea, tech enthusiasts say. <https://www.law.com/sites/almstaff/2017/10/04/drone-racing-league-was-our-idea-tech-enthusiasts-say/?slreturn=20190413053420>, (last accessed in May 2019), October 2017.
- [16] <https://phys.org/news/2016-02-drones-forest-trails-lost-people.html>, (last accessed in May 2019), February 2016.
- [17] https://www.youtube.com/watch?v=uq7ey7r_sk4, (last accessed in May 2019), November 2013.
- [18] <https://www.darpa.mil/news-events/2018-07-18>, (last accessed in December 2019), July 2018.
- [19] Steve Paschall and Julius Rose. Fast, lightweight autonomy through an unknown cluttered environment: Distribution statement: A—approved for public release; distribution unlimited. In *2017 IEEE Aerospace Conference*, pages 1–8. IEEE, 2017.
- [20] <https://www.skydio.com/technology/>, (last accessed in December 2019), December 2019.
- [21] <https://www.dji.com/mavic>, (last accessed in December 2019), December 2019.
- [22] Farid Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 29(2):315–378, 2012.
- [23] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

- [24] LE Kavraki, P Svestka, J-C Latombe, and MH Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.
- [25] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.
- [26] Osamu Takahashi and Robert J Schilling. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on robotics and automation*, 5(2):143–150, 1989.
- [27] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.
- [28] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- [29] A Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 3310–3317. IEEE, 1994.
- [30] Kieran Culligan, Mario Valenti, Yoshiaki Kuwata, and Jonathan P How. Three-dimensional flight experiments using on-line mixed-integer linear programming trajectory optimization. In *2007 American Control Conference*, pages 5322–5327. IEEE, 2007.
- [31] David H Shim, Hoam Chung, and S Shankar Sastry. Conflict-free navigation in unknown urban environments. *IEEE robotics & automation magazine*, 13(3):27–33, 2006.
- [32] Sergei Lupashin, Angela Schöllig, Michael Sherback, and Raffaello D’Andrea. A simple learning strategy for high-speed quadcopter multi-flips. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1642–1648. IEEE, 2010.
- [33] Sergei Lupashin and Raffaello D’Andrea. Adaptive fast open-loop maneuvers for quadcopters. *Autonomous Robots*, 33(1-2):89–102, 2012.
- [34] Jeremy H Gillula, Haomiao Huang, Michael P Vitus, and Claire J Tomlin. Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1649–1654. IEEE, 2010.
- [35] Daniel Mellinger, Nathan Michael, and Vijay Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, 31(5):664–674, 2012.

- [36] Michael Neunert, Cedric De Crousaz, Fadri Furrer, Mina Kamel, Farbod Farshidian, Roland Siegwart, and Jonas Buchli. Fast nonlinear model predictive control for unified trajectory optimization and tracking. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1398–1404. IEEE, 2016.
- [37] Giuseppe Loianno, Chris Brunner, Gary McGrath, and Vijay Kumar. Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu. *IEEE Robotics and Automation Letters*, 2(2):404–411, 2017.
- [38] Davide Falanga, Elias Mueggler, Matthias Faessler, and Davide Scaramuzza. Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5774–5781. IEEE, 2017.
- [39] Efe Camci and Erdal Kayacan. Learning motion primitives for planning swift maneuvers of quadrotor. *Autonomous Robots*, pages 1–13, 2019.
- [40] Adam Bry, Charles Richter, Abraham Bachrach, and Nicholas Roy. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *The International Journal of Robotics Research*, 34(7):969–1002, 2015.
- [41] Markus Hehn and Raffaello D’Andrea. Quadrocopter trajectory generation and control. *IFAC Proceedings Volumes*, 44(1):1485–1491, 2011.
- [42] Benoit Landry, Robin Deits, Peter R Florence, and Russ Tedrake. Aggressive quadrotor flight through cluttered environments using mixed integer programming. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1469–1475. IEEE, 2016.
- [43] Hugh Cover, Sanjiban Choudhury, Sebastian Scherer, and Sanjiv Singh. Sparse tangential network (spartan): Motion planning for micro aerial vehicles. In *2013 IEEE International Conference on Robotics and Automation*, pages 2820–2825. IEEE, 2013.
- [44] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *arXiv preprint arXiv:1005.0416*, 2010.
- [45] Larry Matthies, Roland Brockers, Yoshiaki Kuwata, and Stephan Weiss. Stereo vision-based obstacle avoidance for micro air vehicles using disparity space. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 3242–3249. IEEE, 2014.
- [46] Sikang Liu, Michael Watterson, Sarah Tang, and Vijay Kumar. High speed navigation for quadrotors with limited onboard sensing. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1484–1491. IEEE, 2016.

- [47] Jing Chen, Tianbo Liu, and Shaojie Shen. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1476–1483. IEEE, 2016.
- [48] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. Continuous-time trajectory optimization for online uav replanning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5332–5339. IEEE, 2016.
- [49] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. 2009.
- [50] Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto. Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles. *IEEE Robotics and Automation Letters*, 3(3):1474–1481, 2018.
- [51] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.
- [52] Sikang Liu, Michael Watterson, Kartik Mohta, Ke Sun, Subhrajit Bhattacharya, Camillo J Taylor, and Vijay Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robotics and Automation Letters*, 2(3):1688–1695, 2017.
- [53] Fei Gao, William Wu, Yi Lin, and Shaojie Shen. Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 344–351. IEEE, 2018.
- [54] Zichao Zhang and Davide Scaramuzza. Perception-aware receding horizon navigation for mavs. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2534–2541. IEEE, 2018.
- [55] Sertac Karaman and Emilio Frazzoli. Complex mission optimization for multiple-uavs using linear temporal logic. In *2008 american control conference*, pages 2003–2009. IEEE, 2008.
- [56] Sertac Karaman and Emilio Frazzoli. Vehicle routing problem with metric temporal logic specifications. In *2008 47th IEEE Conference on Decision and Control*, pages 3953–3958. IEEE, 2008.
- [57] Shayegan Omidshafiei, Ali-Akbar Agha-Mohammadi, Christopher Amato, and Jonathan P How. Decentralized control of partially observable markov decision processes using belief space macro-

- actions. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5962–5969. IEEE, 2015.
- [58] Shayegan Omidshafiei, Shih-Yuan Liu, Michael Everett, Brett T Lopez, Christopher Amato, Miao Liu, Jonathan P How, and John Vian. Semantic-level decentralized multi-robot decision-making using probabilistic macro-observations. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 871–878. IEEE, 2017.
- [59] Christos K Verginis, Ziwei Xu, and Dimos V Dimarogonas. Decentralized motion planning with collision avoidance for a team of uavs under high level goals. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 781–787. IEEE, 2017.
- [60] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharmashan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.
- [61] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.
- [62] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.
- [63] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.
- [64] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.
- [65] Emilio Frazzoli. *Robust hybrid control for autonomous vehicle motion planning*. PhD thesis, Massachusetts Institute of Technology, 2001.
- [66] Mark Müller, Sergei Lupashin, and Raffaello D’Andrea. Quadrocopter ball juggling. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 5113–5120. IEEE, 2011.
- [67] Mark W Mueller, Markus Hehn, and Raffaello D’Andrea. A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE Transactions on Robotics*, 31(6):1294–1310, 2015.

- [68] Markus Hehn and Raffaello D'Andrea. Real-time trajectory generation for quadcopters. *IEEE Transactions on Robotics*, 31(4):877–892, 2015.
- [69] Robin Deits and Russ Tedrake. Efficient mixed-integer planning for uavs in cluttered environments. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 42–49. IEEE, 2015.
- [70] N Kemal Ure and Gokhan Inalhan. Autonomous control of unmanned combat air vehicles: Design of a multimodal control and flight planning framework for agile maneuvering. *IEEE Control Systems*, 32(5):74–95, 2012.
- [71] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.
- [72] Jie Tang, Arjun Singh, Nimbus Goehausen, and Pieter Abbeel. Parameterized maneuver learning for autonomous helicopter flight. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1142–1148. IEEE, 2010.
- [73] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.
- [74] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *2017 IEEE international conference on robotics and automation (icra)*, pages 1527–1533. IEEE, 2017.
- [75] Efe Camci and Erdal Kayacan. Waitress quadcopter explores how to serve drinks by reinforcement learning. In *Region 10 Conference (TENCON), 2016 IEEE*, pages 28–32. IEEE, 2016.
- [76] Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 33(1-2):21–39, 2012.
- [77] Lionel Degiovanni and Philippe Bernard. Tp drone diatone 250 fpv. <http://eduscol.education.fr/>, (last accessed in June 2017), June 2017.
- [78] Paul De Casteljau. Outillages méthodes calcul. *Andr e Citro en Automobiles SA, Paris*, 1959.
- [79] Taeyoung Lee, Melvin Leoky, and N Harris McClamroch. Geometric tracking control of a quadrotor uav on se (3). In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 5420–5425. IEEE, 2010.

-
- [80] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.
- [81] Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard. A fully autonomous indoor quadrotor. *IEEE Transactions on Robotics*, 28(1):90–100, 2012.
- [82] Raúl Mur-Artal and Juan D Tardós. Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters*, 2(2):796–803, 2017.
- [83] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Robotics Research*, pages 235–252. Springer, 2017.
- [84] Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018.
- [85] Benjamin Morrell, Rohan Thakker, Gene Merewether, Robert Reid, Marc Rigter, Theodore Tzaneetos, and Gregory Chamitoff. Comparison of trajectory optimization algorithms for high-speed quadrotor flight near obstacles. *IEEE Robotics and Automation Letters*, 3(4):4399–4406, 2018.
- [86] Mark W Mueller and Raffaello D’Andrea. Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 45–52. IEEE, 2014.
- [87] Mohit Mehndiratta, Efe Camci, and Erdal Kayacan. Automated tuning of nonlinear model predictive controller by reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3016–3021. IEEE, 2018.
- [88] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.
- [89] Siddharth Reddy, Anca Dragan, and Sergey Levine. Shared autonomy via deep reinforcement learning. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.
- [90] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

- [91] Helen Oleynikova, Dominik Honegger, and Marc Pollefeys. Reactive avoidance using embedded stereo vision for mav flight. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 50–56. IEEE, 2015.
- [92] Brett T Lopez and Jonathan P How. Aggressive 3-d collision avoidance for high-speed navigation. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 5759–5765. IEEE, 2017.
- [93] Stefan Stevšić, Tobias Nägeli, Javier Alonso-Mora, and Otmar Hilliges. Sample efficient learning of path following and obstacle avoidance behavior for quadrotors. *IEEE Robotics and Automation Letters*, 3(4):3852–3859, 2018.
- [94] Bryan Penin, Paolo Robuffo Giordano, and François Chaumette. Vision-based reactive planning for aggressive target tracking while avoiding collisions and occlusions. *IEEE Robotics and Automation Letters*, 3(4):3725–3732, 2018.
- [95] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.
- [96] Valts Blukis, Nataly Brukhim, Andrew Bennett, Ross Knepper, and Yoav Artzi. Following high-level navigation instructions on a simulated quadcopter with imitation learning. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.
- [97] Sunggoo Jung, Sunyou Hwang, Heemin Shin, and David Hyunchul Shim. Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. *IEEE Robotics and Automation Letters*, 3(3):2539–2544, 2018.
- [98] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.
- [99] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [100] Peter J Huber et al. Robust estimation of a location parameter. *The annals of mathematical statistics*, 35(1):73–101, 1964.
- [101] Tobias Nägeli, Lukas Meier, Alexander Domahidi, Javier Alonso-Mora, and Otmar Hilliges. Real-time planning for automated multi-view drone cinematography. *ACM Transactions on Graphics (TOG)*, 36(4):132, 2017.
- [102] Quentin Galvane, Christophe Lino, Marc Christie, Julien Fleureau, Fabien Servant, Fran Tariolle, Philippe Guillotel, et al. Directing cinematographic drones. *ACM Transactions on Graphics (TOG)*, 37(3):34, 2018.

-
- [103] Christoph Gebhardt, Stefan Stevšić, and Otmar Hilliges. Optimizing for aesthetically pleasing quadrotor camera motion. *ACM Transactions on Graphics (TOG)*, 37(4):90, 2018.
- [104] Marc Christie, Patrick Olivier, and Jean-Marie Normand. Camera control in computer graphics. In *Computer Graphics Forum*, volume 27, pages 2197–2218. Wiley Online Library, 2008.
- [105] Michael Gleicher and Andrew Witkin. Through-the-lens camera control. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 331–340. ACM, 1992.
- [106] Steven M Drucker and David Zeltzer. Intelligent camera control in a virtual environment. In *Graphics Interface*, pages 190–190. Citeseer, 1994.
- [107] Christophe Lino, Marc Christie, Roberto Ranon, and William Bares. The director’s lens: an intelligent assistant for virtual cinematography. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 323–332. ACM, 2011.
- [108] Christophe Lino and Marc Christie. Intuitive and efficient camera control with the toric space. *ACM Transactions on Graphics (TOG)*, 34(4):82, 2015.
- [109] Daniel Arijon. *Grammar of the film language*. Silman-James Press, 1976.
- [110] Christopher J Bowen and Roy Thompson. *Grammar of the Shot*. Taylor & Francis, 2013.
- [111] Mackenzie Leake, Abe Davis, Anh Truong, and Maneesh Agrawala. Computational video editing for dialogue-driven scenes. 2017.
- [112] Mike Roberts and Pat Hanrahan. Generating dynamically feasible trajectories for quadrotor cameras. *ACM Transactions on Graphics (TOG)*, 35(4):61, 2016.
- [113] Niels Joubert, Mike Roberts, Anh Truong, Floraine Berthouzoz, and Pat Hanrahan. An interactive tool for designing quadrotor camera shots. *ACM Transactions on Graphics (TOG)*, 34(6):238, 2015.
- [114] Ke Xie, Hao Yang, Shengqiu Huang, Dani Lischinski, Marc Christie, Kai Xu, Minglun Gong, Daniel Cohen-Or, and Hui Huang. Creating and chaining camera moves for quadrotor videography. *ACM Transactions on Graphics*, 37:14, 2018.
- [115] Ziquan Lan, Mohit Shridhar, David Hsu, and Shengdong Zhao. Xpose: Reinventing user interaction with flying cameras. In *Robotics: Science and Systems*, 2017.
- [116] Rogerio Bonatti, Yanfu Zhang, Sanjiban Choudhury, Wenshan Wang, and Sebastian Scherer. Autonomous drone cinematographer: Using artistic principles to create smooth, safe, occlusion-free trajectories for aerial filming. *International Symposium on Experimental Robotics*, 2018.

- [117] Niels Joubert, Dan B Goldman, Floraine Berthouzoz, Mike Roberts, James A Landay, Pat Hanrahan, et al. Towards a drone cinematographer: Guiding quadrotor cameras using visual composition principles. *arXiv preprint arXiv:1610.01691*, 2016.
- [118] Andrej Karpathy. What a deep neural network thinks about your# selfie. <http://karpathy.github.io/2015/10/25/selfie>, 2015.
- [119] Hui Fang and Meng Zhang. Creatism: A deep-learning photographer capable of creating professional work. *arXiv preprint arXiv:1707.03491*, 2017.
- [120] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.
- [121] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.
- [122] Henrik Kretzschmar, Markus Spies, Christoph Sprunk, and Wolfram Burgard. Socially compliant mobile robot navigation via inverse reinforcement learning. *The International Journal of Robotics Research*, 35(11):1289–1307, 2016.
- [123] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.
- [124] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. Cooperative inverse reinforcement learning. In *Advances in neural information processing systems*, pages 3909–3917, 2016.
- [125] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.
- [126] David Gunning. Explainable artificial intelligence (xai). <https://www.darpa.mil/program/explainable-artificial-intelligence>, (last accessed in July 2019), July 2019.