ISBN 978-82-326-6446-7 (printed ver.)
ISBN 978-82-326-6581-5 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (electronic ver.)

NTNU
Norwegian University of
Science and Technology

Eivind Bøhn

Doctoral thesis

Doctoral theses at NTNU, 2022:45

NTNU
Norwegian University of
Science and Technology
Thesis for the degree of
Philosophiae Doctor
Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

NTNU

Doctoral theses at NTNU, 2022:45

Eivind Bøhn

Reinforcement Learning for Optimization of Nonlinear and Predictive Control

NTNU
Norwegian University of
Science and Technology

Eivind Bøhn

Reinforcement Learning for Optimization of Nonlinear and Predictive Control

Thesis for the degree of Philosophiae Doctor

Trondheim, March 2022

Norwegian University of Science and Technology
Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

© Eivind Bøhn

ISBN 978-82-326-6446-7 (printed ver.)
ISBN 978-82-326-6581-5 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (electronic ver.)

ITK-report: 2022-4-W

PRINTED IN
NORWAY
NO - 1598

NORDIC SWAN ECOLABEL
Printed matter
2041 0731

Printed by Skipnes Kommunikasjon AS

Summary

Autonomous systems extend upon human capabilities and can be equipped with superhuman attributes in terms of durability, strength, and perception to name a few, and can provide numerous benefits such as superior efficiency, accuracy and endurance, and the ability to explore dangerous environments. Delivering on this potential requires a control system that can skillfully operate the autonomous system to complete its objectives. A static control system must be carefully designed to handle any situation that might arise. This motivates the introduction of learning in the control system since a learning system can learn from its experiences to manage novel unexpected events and changes in its operating environment.

Traditional formal control techniques are typically designed offline assuming exact knowledge of the dynamics of the system to be controlled. These knowledge-based approaches have the important benefit that the stability properties of the control algorithm can be analyzed and certified, such that one can have confidence in the control system's ability to safely operate the controlled system. However, linear control techniques applied to nonlinear systems (which all real systems are to some extent) lead to increasingly conservative and therefore suboptimal control performance the more nonlinear the controlled system is. Nonlinear control techniques often have considerable online computational complexity, which makes them infeasible for systems with fast dynamics and for embedded control applications where computational power and energy are limited resources.

Reinforcement learning is a framework for developing self-optimizing controllers, that learn to improve its operation through trial-and-error and adjusting its behaviour based on the observed outcomes of its actions. In general, reinforcement learning requires no knowledge about the dynamics of the controlled system, can learn to operate arbitrarily nonlinear systems, and its online operation can be designed to be highly computationally efficient. It is therefore a valuable tool for control systems where the dynamics are fast, nonlinear, or uncertain, and difficult to model. A central challenge of reinforcement learning control on the other hand is that its behaviour is complex and difficult to analyze, and it has no inherent support for specification of operating constraints.

An approach to remedy these challenges for reinforcement learning control is to combine its learning capabilities with an existing trusted control technique. In Part I of this thesis, we employ reinforcement learning for optimization of the model predictive control (MPC) scheme, a powerful yet complex control technique. We propose the novel idea of optimizing its meta-parameters, that is, parameters affecting the structure of the control problem the MPC solves as opposed to internal parameters affecting the solution to a given problem. In particular, we optimize the meta-parameters of when to compute the MPC and with what prediction horizon, and show that by intelligently selecting the conditions under which it is computed, the control performance and computational complexity can be simultaneously improved. We subsequently present a framework in which these meta-parameters as well as any other internal parameter of the MPC can be jointly optimized with a configurable objective. Finally, Part I of the thesis also considers how an existing controller can be used to accelerate the learning process of a learning controller.

Control of unmanned aerial vehicles (UAVs) is precisely such an embedded application with limited computational- and energy-resources, and moreover where the dynamics are highly nonlinear and affected by significant disturbances such as turbulence. In Part II of this thesis, we propose the novel idea of employing deep reinforcement learning (DRL) for low-level control of fixed-wing UAVs, a UAV-design that exhibit superior range and payload capacity compared to the popular multirotor drone design. We present a method capable of learning flightworthy DRL controllers with as little as 3 minutes of interaction with the controlled system, and demonstrate through field experiments with the real UAV that the DRL controller is competitive with the state-of-the-art existing autopilot, generating smooth responses in the controlled states and in the control signals to the actuators.

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of Philosophiae Doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The work that constitutes the basis of this thesis has been carried out at the Department of Engineering Cybernetics at NTNU during the first year, and at the Analytics and Artificial Intelligence group at the Department of Mathematics and Cybernetics at SINTEF Digital for the remaining two years.

Professor Tor Arne Johansen has been the main supervisor of the PhD project, and Signe Moe and Professor Sebastien Gros have been co-supervisors. The Research Council of Norway has co-funded this project through the funding scheme PhD Scholarships at SINTEF, grant number 272402.

Acknowledgements

My PhD project was a great and unique opportunity in that it was essentially a blank slate with a pouch of funding attached, and the initial part of the PhD consisted of designing my own project in terms of formulating a project description and finding a suitable supervisor. I would first of all like to thank SINTEF for giving me this opportunity. Having just finished a master's of science in cybernetics and having a newfound interest in machine learning, reinforcement learning for control was a natural choice as the topic of the PhD, and with the renowned and experienced Professor Tor Arne Johansen graciously accepting the role of main supervisor, things were looking up. However, with great opportunity comes great responsibility, and I quickly got to feel the flip side of too much freedom as a fresh and inexperienced researcher. Having chosen a topic that neither of my initial supervisors had their core expertise within, I had to rely on my own intuition in deciding what ideas were worth pursuing and which ideas work in theory but whose implementation was flawed (which is especially pertinent to deep reinforcement learning, which I am sure all researchers and practitioners in this field can attest to). This issue was rectified in the third year of the PhD with the addition of co-supervisor Professor Sebastien Gros, whose reinforcement learning expertise helped tremendously. Still, I am happy that I chose the topic of reinforcement learning, some of the principles of which I have incorporated into my daily life as well: while exploitation is comfortable, remember to do some exploration as well!

I am very grateful for the help and encouragement I have received from my supervisors throughout the PhD. Tor Arne, your thorough feedback and insightful comments have been instrumental in raising the quality of every work undertaken, not to mention your world-class email-based customer support, seemingly open 24/7 and with instant response time. I would like to thank my supervisor at SINTEF, Signe, for your tutelage on being a PhD student, and for our brainstorming sessions resulting in many great ideas. Last but not least, thank you Sebastien for our numerous discussions which always resulted in a lighter yet wiser mind, there was no problem that could not be solved with a simple conversation.

Big thanks also goes out to Erlend, my personal aviation consultant and co-author of first and last works during the PhD. The first article we published together set a great precedent and bar for the later works I did during the PhD, with your nose for details constantly finding ways we could improve the work we did and ensure it is accurately communicated. I would like to thank my colleagues in the Analytics and AI group at SINTEF where I resided in the second and third year of my PhD. The people at the NTNU UAV-Lab also deserve recognition, without whom the flight experiments undertaken would not have been possible.

A negative shout-out goes to COVID-19 for cancelling my research stay abroad in the Netherlands, which I am sure would have been a great and enriching experience, and for delaying experiments such that the tail end of the PhD project became busier than it had to be. Further, the global pandemic made building a professional network and spreading the research that was done exponentially more difficult.

Finally, I would like to thank my sisters, Mette and Hege, and my parents, Per and Anne Kathrine, for continued support throughout my academic endeavours. A further thanks is in order for housing me as I transitioned from NTNU in Trondheim to SINTEF in Oslo, and tolerating my presence in the house again, 6 years after I first moved out.

Contents

Summary iii

Preface v

Contents vi

1 Introduction 1

1.1 Background and Motivation 1

1.2 Objectives 4

1.3 Contributions 5

1.4 Publications 6

1.5 Outline 7

2 Preliminaries 9

2.1 Model Predictive Control 9

2.1.1 Adaptive Horizon Model Predictive Control 10

2.1.2 Event-Triggered Model Predictive Control 11

2.1.3 Note on Feasibility 11

2.2 Value Function Estimation for Model Predictive Control 12

2.3 Linear Quadratic Regulator 13

2.4 Reinforcement Learning 15

2.4.1 Policy Gradient Methods 17

2.5 Proximal Policy Optimization 18

2.5.1 Trust Region Policy Optimization 18

2.5.2 The Proximal Policy Optimization Objective 19
2.5.3 Advantage Function Estimation 20
2.6 Value-Based Deep Reinforcement Learning Algorithms 20
2.7 Twin Delayed DDPG 22
2.8 Soft Actor Critic 23
2.9 Hindsight Experience Replay 25

I Combining Knowledge-Based Control Techniques with Reinforcement Learning 27

3 Introduction 29
3.1 Motivation and Related Work 30

4 Reinforcement Learning of the Model Predictive Control Computation Timing 33
4.1 Introduction 33
4.2 Learning-based Event-Triggered MPC 35
4.2.1 Proposed Control Algorithm 35
4.2.2 Reward Function 37
4.2.3 Initialization 37
4.2.4 Policy Representation 37
4.3 Experiment 38
4.3.1 Baseline Policies 39
4.3.2 Training and Evaluation 40
4.4 Results and Discussion 40
4.5 Conclusion 42

5 Reinforcement Learning of the Prediction Horizon in Model Predictive Control 43

5.1 Introduction 43

5.2 Learning the Prediction Horizon using SAC 45

5.2.1 Horizon Policy 45

5.2.2 MPC Value Function 46

5.2.3 Evaluation 46

5.3 Experiments 47

5.3.1 Inverted Pendulum 47

5.3.2 Collision Avoidance 48

5.4 Results 49

5.5 Conclusion 53

6 A Complete Framework for Optimization of Model Predictive Control with Reinforcement Learning 55

6.1 Introduction 55

6.2 Control Algorithm 56

6.3 A Reinforcement Learning Parameter and Meta-Parameter Optimization Framework for Model Predictive Control 58

6.3.1 A State Space with the Markov Property 58

6.3.2 Policies 59

6.3.3 Optimizing LQR with Reinforcement Learning 66

6.3.4 Summary of Control Algorithm with Learning 68

6.3.5 Reward Function 70

6.3.6 Initialization of the Learning Procedure 71

6.4 Numerical Results 71

6.4.1 The Inverted Pendulum System 72

6.4.2 Training and Evaluation 73

6.4.3 Inverted Pendulum Results and Discussion 75

6.5 Conclusion 81

7 Accelerating Reinforcement Learning with Suboptimal Guidance: Improving the Adaptivity of the Q-filter Approach 83

7.1 Introduction 84
7.2 Related Work 86
7.3 Improving the Adaptivity of the Q-filter 88
7.3.1 Problem Description 88
7.3.2 Proposed Method 88
7.4 Experiments 90
7.4.1 Guiding Controllers 91
7.4.2 Configuration 91
7.4.3 Ablation Studies 91
7.5 Results and Discussion 92
7.5.1 Results of Experiments 92
7.5.2 Evaluation of Results 95
7.6 Conclusion 95

II Reinforcement Learning for Attitude Control of Fixed-Wing UAVs: An Application of Reinforcement Learning Control in the Physical World 97

8 Introduction 99

9 Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs: A Proof of Concept 101

9.1 Introduction 101
9.2 Related Work 104
9.3 UAV Model 106
9.3.1 Aerodynamic Forces and Moments 106
9.3.2 Propulsion Forces and Moments 108

9.3.3 Actuator Dynamics and Constraints 109

9.4 Learning an On-Policy Attitude Controller using PPO 109

9.4.1 Action Space 110

9.4.2 Training of Controller 110

9.4.3 Evaluation 112

9.5 Results and Discussion 114

9.5.1 Key Factors Impacting Training 115

9.5.2 Evaluation of Controller 117

9.6 Conclusion 119

10 Data Efficient Deep Reinforcement Learning Control of Fixed-Wing UAVs in the Field 123

10.1 Introduction 124

10.2 Learning an Off-Policy Attitude Controller for the Real UAV using SAC 126

10.2.1 Controller Architecture and State Design 127

10.2.2 Reward and Objective Design 129

10.2.3 UAV Model 130

10.2.4 Sim-to-Real 131

10.2.5 Simulator Episode Design 133

10.2.6 RL Algorithm 134

10.2.7 Experimental Platform 135

10.3 Experimental Results 136

10.3.1 FBWA Mode 136

10.3.2 Auto mode 137

10.3.3 Step sequences and Comparison with ArduPlane PIDs . . 139

10.4 Discussion 141

10.4.1 Non-symmetric roll response 141

10.4.2 Steady-State Errors 142
10.4.3 Oscillations: Illustration of the Iterative Development Process 143
10.4.4 Data Requirement of the RL controller 145
10.4.5 Linear Analysis 145
10.4.6 Learning Phase 148
10.5 Conclusion 150

III Concluding Remarks 153

11 Conclusions and Future Possibilities 155

Appendix A The ArduPlane Attitude Controller 161

Bibliography 163

1

Introduction

1.1 Background and Motivation

Human history has been marked by continual development and introduction of new technologies, which allow people to accomplish their tasks more efficiently, freeing them up to focus on more productive objectives. The emergence of agriculture enabled the creation of permanent settlements and domestication of livestock. When combined with technological innovations such as the animal-driven plough, this eventually led to a significant food surplus such that only a fraction of the community needed to dedicate their time to food acquisition, and people were now free to specialize into other tasks. The ultimate technology in this sense is automation, as it has the potential to entirely eliminate the need for human participation in the process loop. Today, automation is ubiquitous and rapidly expanding; factories are increasingly automatized with robotic assembly lines, nearly every household contains appliances such as washing machines and dishwashers, and companies are starting to experiment with automatic last-mile delivery using unmanned aerial vehicles (UAVs). Tomorrow, the automotive industry's push towards self-driving cars has revolutionized transportation. However, as pointed out by Lisanne Bainbridge in her article ”Ironies of Automation” [14], the more efficient the automated system is, the more crucial the role of the human operator. As the complexity and scope of automated systems grows, the consequences of their errors and the difficulty in correcting the errors is ever increasing as well. Therefore, I would argue, that a truly autonomous system needs to be antifragile [177].

An antifragile system is one that is not merely resilient to stressors and volatility but rather improves in the face of unexpected events and stressors. The evolutionary nature of biological life exhibits exactly these properties. A process starting from humble single-cell origins with seemingly no intelligent driving force behind it has developed an unimaginable richness and complexity, all while surviving countless catastrophic events such as asteroid impacts, volcanic eruptions and ice ages. Random mutations of the genome and a ruthless survival-of-the-fittest natural selection process together constitute an antifragile system that can continually adapt to and increasingly master its environment, as opposed to a perfectly engineered robust system that would need to envision and account for every possible event in order to ensure its continued operation. One way to enable antifragility for autonomous systems is to equip them with the capability of learning. For a learning system, unexpected events and uncertainty constitute information and an opportunity to improve itself for the future. One might object that the evolutionary process has taken billions of years to develop intelligent behaviour, however, consider then the faster replicating viruses that mutate and develop novel strains that are continuously able to circumvent the defense systems of their targets. Thus, for a machine learning system, the rate at which it can be improved even through such a simple learning paradigm as evolution is simply a question of compute power (i.e. replication speed), which has been increasing exponentially the last decades.

Machine learning is the study of how computer algorithms can incorporate and learn from new information in order to improve its operation. It is broadly categorized into three types: 1) supervised learning, for which the objective is to learn a mapping between known input-output pairs, 2) reinforcement learning (RL), which is concerned with learning optimal sequential decision making for dynamical systems, and 3) unsupervised learning, which involves identifying patterns and structure in data. Of these, the two former are the most applicable forms of machine learning for control of autonomous systems, and they can enter into the control-loop of autonomous systems shown in Figure 1.1 in several ways [138]. In this depiction of the traditional control-loop, the plant block is the controlled system whose state is observed (or estimated) by the internal observation block. This observation in conjunction with an observation of the external environment of the plant is used by the kinematic and dynamic controllers to regulate the system. The kinematic control block decides reference signals for the dynamic control block by comparing the state of the plant to the desired state. The dynamic control block corresponds to the lowest-level controller and decides how the actuators of the plant are used to regulate the controlled states of the plant to the reference signals.

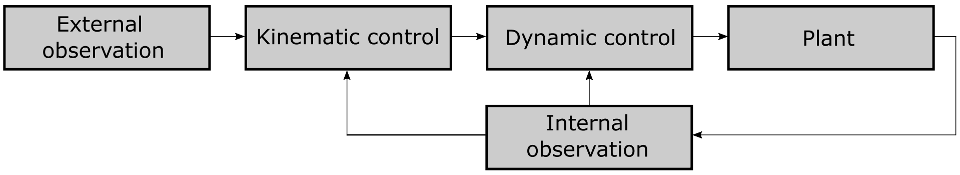


Figure 1.1: The control-loop consists of several blocks relating to estimation and control, in which learning can enter in numerous ways.

Supervised learning can be used as a tool for system identification, that is, to learn a model of the plant's dynamics. The learned model can then be employed in a number of ways, e.g. to simulate the plant block in order to facilitate learning or tuning of other blocks in the control loop, it can be used for estimation of system states [181, 50], or it can be used in the control blocks as is done in [107] where the learned dynamics model is employed in a model predictive control (MPC) framework. It should be noted that combining data-driven system identification and control is a well established field within control theory known as adaptive control [120]. However, while adaptive control typically considers parametric uncertainty in the model, learning-based control models typically employ machine learning for estimation of unknown parameters or unknown functions of the system model [181, 162]. Supervised learning can also be used for kinematic control where online analysis of the environment is complex, e.g. because the observation of the external environment consists of camera images [68]. Finally, it can also be used directly in the dynamic control block by learning to mimic an existing controller [203], the learned controller can then be superior in terms of lower online computational complexity or not requiring access to hidden state information.

When it comes to developing novel control laws and control strategies the most relevant machine learning technique is RL, which is applicable when one can formulate a feedback signal describing the utility of the state of the plant, called the reward function. RL is a framework for self-optimization of a policy (analogous to a controller in control engineering terminology) that maps states to actions, using a trial-and-error process and the evaluative feedback received in terms of rewards and state transitions. This basic RL problem has due to its generality been studied in many different contexts and under many different names. Its roots can be traced back to the 1950s and ’60s in the fields of artificial intelligence, optimal control, and dynamic programming. Following the success of deep learning, RL has seen a resurgence in interest with the advent of deep reinforcement learning (DRL) which combines RL algorithms with neural networks (NNs) as function approximators. In 2013, Mnih et al. [137] demonstrated that DRL could learn to play a wide range of Atari video games with minimal prior knowledge, exceeding the performance of contemporary methods and even human players. This method operated in discrete action spaces, and as such was not immediately applicable to continuous control problems. This was rectified in 2015 when Lillicrap et al. [117] presented the deep deterministic policy gradients (DDPG) algorithm allowing the potential of DRL to be harnessed for continuous control.

As is the case for supervised learning, RL can be applied for several tasks in the control-loop. Historically, RL has been applied for kinematic control [196, 90, 114, 94] as this block often involves complex-decision making processes on conflicting objectives as well as path planning in uncertain environments where crafting a fixed rule-based system can be difficult. RL is also applied for dynamic control [84, 148, 40, 8, 191], and even for incorporating both the kinematic and dynamic control blocks into a single control scheme, as is done in [123] for a non-holonomic wheeled mobile robot. In general, RL requires no knowledge about the system to be optimized and it is therefore applicable to a large class of control problems where no model suitable for real-time control exists or is readily obtainable. However, the learning nature of RL makes its behaviour and stability properties difficult to analyze, especially when combined with function approximation.

Traditional formal control approaches on the other hand are knowledge-based and designed offline assuming knowledge of the system dynamics, which enables their closed-loop behaviour to be studied and therefore trust to be put in their safe operation of the plant. However, accurate modeling of systems is a difficult process, often necessitating idealizations and assumptions of negligibility of certain effects under certain conditions [184, 43], and further, simplifications of the system model is often necessary for the model to be suitable for real-time control [99, 51]. For nonlinear and uncertain systems, providing stability guarantees using linear control techniques yields increasingly conservative and therefore suboptimal control performance the more nonlinear or uncertain the system is. Nonlinear control techniques such as nonlinear model predictive control (NMPC) often have intractably high online computational complexity for many embedded control applications or systems with fast dynamics. Because RL controllers can be learned in a model-free manner directly on the nonlinear and unknown dynamics through feedback gained by operating the plant, RL is a valuable tool for dynamic control of nonlinear systems where the dynamics are unknown or affected by significant disturbances, or where the hardware platform is computationally limited and can benefit from RL's efficient online operation.

1.2 Objectives

Motivated by the described challenges of traditional control approaches for control of nonlinear systems, this thesis considers RL-based control. The applications and experiments presented focuses on dynamic low-level control, but the methods developed in this thesis are applicable also to kinematic control.

Part I of the thesis considers the combination of traditional approaches and reinforcement learning approaches to control, and how these can be combined in order to leverage their unique strengths and shore up each of their weaknesses. In particular, the thesis focuses on the NMPC method and attempt to address some of its challenges wrt. embedded control applications, namely its high online computational complexity and the need to tune (and re-tune with time and changes in the controlled system) its parameters to the task at hand. Building upon previous research considering the MPC's dynamic model and cost objective as parameters to be learned with RL, Part I of the thesis investigates how RL can be used to learn what we call the meta-parameters of the MPC scheme, which majorly impact its computational complexity and control performance. Finally, a second research question considered in Part I is when basing RL-control on tabula rasa control laws (e.g. neural networks) instead of existing control methods (e.g. MPC), how can an existing suboptimal control approach to the task be used to accelerate the learning of a tabula-rasa control law.

Part II of this thesis considers the specific control application of attitude control of fixed-wing UAVs, i.e. the lowest-level dynamic control of its orientation. This is a highly complex control application due to several reasons: highly nonlinear dynamics, significant disturbances from weather phenomena, and limited computational power available to the control system onboard the UAV. As a result, the current state-of-the-art approaches are based on linear control approaches, yielding a limited set of feasible operating conditions and maneuvers that can be performed autonomously, which is together referred to as the flight envelope of the control system. Extending the flight envelope of fixed-wing UAVs is therefore of great interest especially for hazardous search and rescue missions in extreme weather conditions and in general for more efficient operation of the control system. To achieve an extended flight envelope, the control system must be nonlinear with low online computational complexity, and Part II investigates how RL-based control approaches can fulfill this role.

1.3 Contributions

In light of the above, this thesis has contributed to the research community in the following ways:

- Proposed the novel idea of optimizing meta-parameters of the MPC scheme using RL, and showed that this approach can jointly optimize control performance and computational efficiency, extending MPC's viability for computationally- or energy-limited applications.
- Developed an RL-based optimization (tuning) framework in which the above-mentioned meta-parameters, as well as any other parameter (e.g. objective function, constraints etc.) of the MPC scheme, can be jointly optimized.
- Improved the adaptivity of the Q-filter approach to guiding the exploration

phase of RL controllers.

- Proposed and experimentally demonstrated the viability of RL for dynamic control of fixed-wing UAVs.
- Developed an open-source Python flight simulator that has been widely adopted by the research community [34, 35].
- Demonstrated a successful application of RL-based dynamic control, validated through field experiments.

1.4 Publications

Seven articles in total were produced and published in peer-reviewed international conferences and journals during the PhD. This thesis is based on the following lists of works:

Conference publications

- [26] Eivind Bøhn, Erlend M Coates, Signe Moe, and Tor Ame Johansen. Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 523–533. IEEE, 2019
- [36] Eivind Bøhn, Signe Moe, and Tor Arne Johansen. Accelerating reinforcement learning with suboptimal guidance. *IFAC-PapersOnLine*, 53(2): 8090–8096, 2020. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2020.12.2278. 21st IFAC World Congress
- [37] Eivind Bøhn, Sebastien Gros, Signe Moe, and Tor Arne Johansen. Optimization of the model predictive control update interval using reinforcement learning. *IFAC-PapersOnLine*, 54(14):257–262, 2021. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2021.10.362. 3rd IFAC Conference on Modelling, Identification and Control of Nonlinear Systems MICNON 2021
- [38] Eivind Bøhn, Sebastien Gros, Signe Moe, and Tor Arne Johansen. Reinforcement learning of the prediction horizon in model predictive control. *IFAC-PapersOnLine*, 54(6):314–320, 2021. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2021.08.563. 7th IFAC Conference on Nonlinear Model Predictive Control NMPC 2021

Journal publications

- [28] Eivind Bøhn, Sebastien Gros, Signe Moe, and Tor Arne Johansen. Optimization of the model predictive control meta-parameters through reinforcement learning. *IEEE Transactions on Cybernetics*, 2021. Submitted
- [27] Eivind Bøhn, Erlend M. Coates, Dirk Reinhardt, and Tor Arne Johansen. Data-efficient deep reinforcement learning for attitude control of fixed-wing uavs validated through field experiments. *IEEE Transactions on Neural Networks and Learning Systems*, 2021. Submitted

Additionally, the following article was published during the PhD, but is not part of the thesis:

Publications not included in this thesis

- [29] Eivind Bøhn, Signe Moe, and Tor Arne Johansen. On the effects of properties of the minibatch in reinforcement learning. *Accepted into the 4th International Conference on Intelligent Technologies and Applications*, 2021

1.5 Outline

This thesis is organized into two main parts: Part I covers the contributions made in combining existing control techniques based on system knowledge with model-free RL for optimization and guiding of RL controllers, while Part II covers RL-based dynamic control of fixed-wing UAVs experimentally verified in the field. However, before this, Chapter 2 presents the fundamental methods that this thesis is based upon.

The first chapter of Part I introduces and motivates the work that was performed on optimization of the meta-parameters of the MPC scheme using RL as well as a framework for adaptive guiding of RL controller's exploration phase using an existing controller. Chapter 4 considers the timing of when to compute the MPC as a meta-parameter, and how this can be optimized with RL. Then Chapter 5 considers the prediction horizon of the MPC scheme as a meta-parameter to be optimized, and we show how the optimal prediction horizon can be learned as a function of the state. Chapter 6 asserts that the meta-parameters considered in the two preceding chapters are interconnected, and presents a unified framework that jointly optimizes these meta-parameters. Finally, Chapter 7 presents an alternative use of the framework from Chapter 6 for guiding, something that would have been investigated with more time for the PhD, before presenting the adaptive Q-filter guiding method.

In Part II of the thesis, Chapter 8 motivates the importance of fixed-wing UAVs. Chapter 9 presents a proof of concept of dynamic control of fixed-wing UAVs using RL in a simulated environment. The next chapter then builds on this encouraging result, where we target control of the real UAV in the field. To this end, Chapter 10 presents a new method for developing RL controllers for the UAV control problem with a focus on data efficiency, as well as presenting the experimental results obtained during flight experiments, demonstrating that RL-based control has comparable performance to the existing state-of-the-art autopilot.

Finally, the third and last part in Chapter 11 summarizes the work that has been performed in this PhD and offers some perspectives on the work and possible future directions.

2

Preliminaries

We consider in this thesis control problems on the form:

$$x_{t+1} = f(x_t, u_t), \;\; \min_{x,u} \sum_{t=0}^{T} C_t(x_t, u_t) \tag{2.1}$$

where x is the state vector, u is the control input vector, the function f defines the (discrete-time) system dynamics and C describes some cost objective to be minimized. The system runs in an episodic fashion, beginning in some initial state x_0 and terminating after some predetermined time T has passed. For any statement that holds regardless of the time, we omit the time subscript. To denote a contiguous sequence of points we use the subscript $x_{t:t+n}$, i.e. the sequence of states from time t to time $t+n$. We denote the time dimension of variables internal to any controller scheme with a subscript k, e.g. the future states in the optimal control problem (OCP) solved by the MPC. Finally, matrices are denoted with bold uppercase letters, e.g. $\mathbf{A}$.

2.1 Model Predictive Control

MPC is a control method in which a model of the plant-to-be-controlled is used to predict the response to the controlled variables, in order to identify the optimal control inputs that yield the most desirable behaviour. To this end, the MPC solves a numerical OCP at every time step with the current state of the plant as the initial conditions, yielding the sequence of inputs over the prediction horizon that minimizes the objective function of the OCP, as well as yielding the predicted state trajectory. The first input of this sequence is then applied to the plant, and the OCP is solved again at the subsequent time step. The recomputation of the OCP is motivated by the assumption that the model of the plant is not an exact representation of the true plant dynamics, and thus the predicted state trajectory will increasingly drift from the true system trajectory as the time since the OCP was solved increases. One can therefore limit the effects of the model mismatch by recomputing the OCP as new information is available at the current time. Note that in this thesis we use MPC to refer to both linear and nonlinear MPC, but state explicitly which one we refer to (e.g NMPC) when it is appropriate.

In this thesis, we consider nonlinear state-feedback discrete-time MPC [4, 155]. The MPC receives as arguments the current state of the plant, $\bar{x}_t$, current (possibly time-varying) exogenous input variables (e.g. reference signals), $\hat{p}_t$, as well as the prediction horizon, N_t, for the OCP. We label the MPC control law (for a given horizon, this will be elaborated on in Chapters 5 and 6) as:

$$u^{\mathrm{M}}_{t:t+N_t-1},\ \hat{x}_{t:t+N_t} = \pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(\bar{x}_t, \hat{p}_t, N_t) \tag{2.2}$$

where the first return value is the optimal input sequence, the second return value is the predicted optimal state trajectory, and θ^{M} are the tunable parameters of the MPC scheme. The OCP is formulated as:

$$\begin{aligned}
\min_{x,u} \quad & \sum_{k=0}^{N_t-1} \rho^k \ell_{\theta^{\mathrm{M}}}(x_k, u_k, \hat{p}_k) + \rho^{N_t} m_{\theta^{\mathrm{M}}}(x_{N_t}), && (2.3)\\
\text{s.t.} \quad & x_0 = \bar{x}_t && (2.4)\\
& x_{k+1} = \hat{f}_{\theta^{\mathrm{M}}}(x_k, u_k, \hat{p}_k) \quad \forall\ k \in 0, \dots, N_t - 1 && (2.5)\\
& h_{\theta^{\mathrm{M}}}(x_k, u_k) \le 0 \qquad\qquad \forall\ k \in 0, \dots, N_t - 1 && (2.6)
\end{aligned}$$

Here, $\ell_{\theta^{\mathrm{M}}}$ is the stage cost, consisting of a task-specific objective (e.g. C) and the input change term $\Delta u_k^\top \mathbf{D}_{\theta^{\mathrm{M}}} \Delta u_k$ where $\mathbf{D}_{\theta^{\mathrm{M}}}$ is the input-change weight and $\Delta u_k = u_k - u_{k-1}$, $m_{\theta^{\mathrm{M}}}$ is the terminal cost, $\rho \in (0, 1]$ is the discount factor, $\hat{f}_{\theta^{\mathrm{M}}}$ is the MPC dynamics model (which might differ from the real system dynamics f), and $h_{\theta^{\mathrm{M}}}$ is the constraint vector. The state and control inputs must satisfy the constraints over the whole optimization horizon for the MPC solution to be considered feasible.

2.1.1 Adaptive Horizon Model Predictive Control

The stage cost evaluates the computed solution locally up to $N_t - 1$ steps, the terminal cost $m_{\theta^{\mathrm{M}}}(x_{N_t})$ should therefore provide global information about the desirability of the terminal state of the computed trajectory, which helps the MPC avoid local minima. Therefore, the more accurate the terminal cost is wrt. to the total cost of the infinite horizon solution, the shorter horizons can be used in the MPC scheme while still delivering good control performance [205, 121]. Due to this synergistic relationship between the prediction horizon and the terminal cost, we propose to learn the value function of the MPC, which measures the total cost of the controller accrued over an infinite horizon, and use it as the terminal cost. Moreover, the control performance sensitivity of the MPC to the prediction horizon varies over the state space, an observation that motivated the adaptive horizon model predictive control (AHMPC), for which the prediction horizon varies according to some criteria (see Chapter 5 for some examples). In Chapters 5 and 6 we propose to learn the optimal prediction horizon as a function of the state using RL.

2.1.2 Event-Triggered Model Predictive Control

We further consider a modification to the MPC framework called event-triggered MPC, in which the OCP is not recomputed at every time step, but rather a triggering policy decides at every time step whether the OCP should be recomputed. The control algorithm then consists of two parts, a triggering policy that decides when to compute the control law, and the control law itself. Note that we refer to the triggering policy as the recomputation policy to better highlight its role in the control algorithms presented in Chapters 4 and 6. Thus, not only the first input of the MPC input sequence u_t^{M} is applied to the plant, but rather a variable number of inputs $u_{t:t+n}^{\mathrm{M}}$, $n < N_t$ are applied sequentially at the corresponding time instance until the recomputation policy triggers the recomputation of the OCP at time step $t + n$.

2.1.3 Note on Feasibility

There is no mechanism in the methods presented in Part I that ensures the recursive feasibility of the MPC scheme or the stability of the control algorithm, a problem which is made increasingly complex with the adaptive-horizon and event-triggered formulations of the MPC. However, the control algorithm and optimization frameworks we present are agnostic wrt. to the implementation of the underlying controllers. As such, one could modify the MPC scheme in (2.3)-(2.6) by adding e.g. assumptions on the form and magnitude of the disturbances, adding terminal constraints, or entirely replacing the MPC scheme with more complex formulations, e.g. robust MPC [20] or tube MPC [132]. Moreover, if one has information about the plant to be controlled and the control system itself, such as the region of attraction, one could enable a "watchdog" component to monitor the learning procedure to ensure that it only operates within regions that are deemed safe. Such considerations are however outside the scope of this thesis.

The prediction horizon is one of the most important tunable parameters to achieve stable control with the MPC scheme [133, 77]. Learning approaches can therefore be an important tool in automatically identifying prediction horizons that yields a stable control system. RL is an optimization procedure that seeks optimal behaviour wrt. its reward function, therefore, if it produces non-stabilizing solutions this would suggest that the learning problem itself is ill-posed.

2.2 Value Function Estimation for Model Predictive Control

The ideal choice for the terminal cost $m_{\theta^{\mathrm{M}}}(x_N)$ in the MPC scheme would be the optimal value function V^{π^*} (2.8) which is the value function corresponding to the optimal control law π^* that delivers the optimal control input at every step. A value function measures the expected total sum of costs accrued from being in any given state and from there always acting according to a given control law π, until the end of the episode (or in an infinite horizon in the non-episodic case) (2.7). Equation (2.8) is written as a recursive relationship in V^{π^*} called the Bellman equation, where the value of a given state is decomposed into the one-step optimal cost and the total value from the subsequent state. Computing V^{π^*} exactly from this equation is however intractable for problems with continuous state and/or input spaces, and iterative approaches such as Q-learning requires a prohibitively large amount of data.

$$V^{\pi}(x_t, \hat{p}_t) = \mathbb{E}\left[\sum_{t'=t}^{T} \rho^{t'} \ell(x_{t'}, \pi(x_{t'}, \hat{p}_{t'}), \hat{p}_{t'})\right] \tag{2.7}$$

$$V^{\pi^*}(x_t, \hat{p}_t) = \min_{u} \ell(x_t, u_t, \hat{p}_t) + \rho V^{\pi^*}(x_{t+1}, \hat{p}_{t+1}), \ \ \forall\, t, x, \hat{p} \tag{2.8}$$

The MPC scheme delivers local approximations to π^*, and as such the MPC's value function V^{M} is a good surrogate for V^* as the terminal cost. This value function is not necessarily directly computable either, as it would require running the MPC with an infinite prediction horizon (or equal to the remaining steps of the episode in episodic problems). However, it can be approximated using fitted value iteration on data gathered by running $\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}$ on the system to be optimized.

We label the approximated MPC value function $\hat{V}^{\mathrm{M}}_{\theta^V}$, where θ^V are the parameters of the value function approximator. These parameters are iteratively updated to minimize the mean squared Bellman error (MSBE):

$$\theta^V \leftarrow \arg\min_{\theta^V} \mathbb{E}\left[\left(y(x,\hat{p}) - \hat{V}^{\mathrm{M}}_{\theta^V}(x,\hat{p})\right)^2\right] \tag{2.9}$$

$$y(x_t,\hat{p}_t) = \mathbb{E}\left[\ell(x_t, \pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(x_t,\hat{p}_t,N_t),\hat{p}_t) + \rho\hat{V}^{\mathrm{M}}_{\theta^V}(x_{t+1},\hat{p}_{t+1})\right] \tag{2.10}$$

Here, (2.10) is called the regression target and corresponds to the condition on the value function imposed by the one-step Bellman equation (2.8). Since the MPC scheme delivers N_t-step approximations to the optimal control law π^*, one can modify the update rule to instead regress onto the N-step target [121]:

$$y(x_t,\hat{p}_t) = \mathbb{E}\left[\sum_{k=0}^{N-1} \rho^k \ell(x_k,u_k,\hat{p}_k) + \rho^N \hat{V}^{\mathrm{M}}_{\theta^V}(x_N,\hat{p}_N)\right] \tag{2.11}$$

With this regression target, a larger share of the value of the future trajectory from x_t is known exactly (through the observed costs ℓ) and the contribution of the estimated bootstrapping component $\hat{V}^{\mathrm{M}}_{\theta^V}(x_N,\hat{p}_N)$ is reduced by a factor of ρ^N. This typically leads to accelerated convergence and better stability in the learning process [182, 176].

2.3 Linear Quadratic Regulator

The linear quadratic regulator (LQR) [25] is a state-feedback controller that arises as the optimal solution to unconstrained control problems where the dynamics are linear, and the objective is quadratic. In this thesis we focus on the discrete-time formulation of the LQR. In the infinite-horizon case, the control law (2.14) consists of a feedback gain matrix, $\mathbf{K}_\infty$ (2.13), that is derived from the dynamics matrices $\mathbf{A}$ and $\mathbf{B}$, the cost weighting matrices $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{N}$, and the solution $\mathbf{S}_\infty$ to the discrete-time algebraic Riccati equation (DARE) (2.12) parameterized by $\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}, \mathbf{N}$. In Chapter 6 we consider $\theta^{\mathrm{L}} = [\mathbf{Q}, \mathbf{R}, \mathbf{N}]^\top$ as parameters of the

LQR controller that can be optimized:

$$\begin{aligned} 0 =& \mathbf{Q} + \mathbf{A}^\top \mathbf{S}_\infty \mathbf{A} - \mathbf{S} \\ & - (\mathbf{A}^\top \mathbf{S}_\infty \mathbf{B} + \mathbf{N})(\mathbf{B}^\top \mathbf{S}_\infty \mathbf{B} + \mathbf{R})^{-1}(\mathbf{B}^\top \mathbf{S}_\infty \mathbf{A} + \mathbf{N}^\top) \end{aligned} \tag{2.12}$$

$$\mathbf{K}_\infty =(\mathbf{B}^\top \mathbf{S}_\infty \mathbf{B} + \mathbf{R})^{-1}(\mathbf{B}^\top \mathbf{S}_\infty \mathbf{A} + \mathbf{N}^\top) \tag{2.13}$$

$$u_k^{\mathrm{L}}(x_k) =\pi_{\theta^{\mathrm{L}}}^{\mathrm{L}}(x_k) = -\mathbf{K}_\infty x_k \tag{2.14}$$

The LQR control problem we consider is formally stated as:

$$\min_{x,u} \quad \sum_{k=0}^{\infty} \left[\frac{1}{2} x_{k+1}^\top \mathbf{Q} x_{k+1} + \frac{1}{2} u_k^\top \mathbf{R} u_k + x_k^\top \mathbf{N} u_k \right], \tag{2.15}$$

$$\text{s.t.} \quad \mathbf{Q} - \mathbf{N}\mathbf{R}^{-1}\mathbf{N}^\top \succ 0,\ \mathbf{R} > 0 \tag{2.16}$$

$$x_{k+1} = \mathbf{A} x_k + \mathbf{B} u_k \tag{2.17}$$

where the notation $\succ$ indicates positive semidefiniteness. We also consider the finite-horizon case, denoting the horizon by N_i. In this case, the $\mathbf{S}_k$ and $\mathbf{K}_k$ matrices are time-varying, solved backwards in time using (2.18, 2.19) from the initial starting point $\mathbf{S}_{N_i} = \mathbf{S}_\infty$. The other system matrices can also be time-varying:

$$\begin{aligned} \mathbf{S}_k =& \mathbf{Q}_k + \mathbf{A}_k^\top \mathbf{S}_{k+1} \mathbf{A}_k - (\mathbf{A}_k^\top \mathbf{S}_{k+1} \mathbf{B}_k + \mathbf{R}_k) \\ & (\mathbf{R}_k + \mathbf{B}_k^\top \mathbf{S}_{k+1} \mathbf{B}_k)^{-1} (\mathbf{B}_k^\top \mathbf{S}_{k+1} \mathbf{A}_k + \mathbf{R}_k^\top) \end{aligned} \tag{2.18}$$

$$\mathbf{K}_k = - (\mathbf{R}_k + \mathbf{B}_k^\top \mathbf{S}_{k+1} \mathbf{B}_k)^{-1} (\mathbf{B}_k^\top \mathbf{S}_{k+1} \mathbf{A}_k + \mathbf{R}_k^\top) \tag{2.19}$$

$$u_k^{\mathrm{L}}(x_k) = \pi_{\theta^{\mathrm{L}}}^{\mathrm{L}}(x_k) = -\mathbf{K}_k x_k \tag{2.20}$$

The finite-horizon LQR control problem can then be formally stated as:

$$\min_{x,u} \quad \sum_{k=0}^{N_i - 1} \left[\frac{1}{2} x_{k+1}^\top \mathbf{Q} x_{k+1} + \frac{1}{2} u_k^\top \mathbf{R} u_k + x_k^\top \mathbf{N} u_k \right], \tag{2.21}$$

$$\text{s.t.} \quad \mathbf{Q}_k - \mathbf{N}_k \mathbf{R}_k^{-1} \mathbf{N}_k^\top \succ 0,\ \mathbf{R}_k > 0 \tag{2.22}$$

$$x_{k+1} = \mathbf{A}_k x_k + \mathbf{B}_k u_k \tag{2.23}$$

2.4 Reinforcement Learning

The system to be optimized in the RL framework is typically formulated as a a Markov decision process (MDP). The MDP is defined by a state space $s \in \mathcal{S}$, a set of actions $a \in \mathcal{A}$, a transition probability matrix $\mathcal{T}$ that governs the evolution of states as a function of time and actions, i.e. $s_{t+1} = \mathcal{T}(s_t, a_t)$, a reward function $R(s, a)$ that describes the desirability of the states of the problem, and finally, the discount factor $\gamma \in [0, 1)$ (note the different limits from ρ) that describes the relative importance of immediate and future rewards. Note that rewards, R, are interchangeable with costs, C, through the substitution $R = -C$ (i.e. rewards are negative costs) and changing maximization of the objective to minimization. We will discuss rewards in the context of RL as this is customary. Further, since we in this thesis employ RL for control, we consider an extension to the MDP in the form of a specified goal state, from a set of possible goals $\mathcal{G}$. [164] show that value functions conditioned on this additional goal parameter can successfully be trained to generalize to unseen goals. We therefore include the goal state in the state s.

A policy π is a function that maps from states to actions, and the goal in RL is to find the optimal policy π^* that is optimal in the sense that it maximizes the expected sum of discounted rewards, either over an infinite or finite horizon. A policy can be deterministic, which we denote $\pi(s)$, or stochastic, which we denote $\pi(a|s)$. In this thesis we study the finite horizon case, and state the RL objective as:

$$\begin{aligned} J^{\mathrm{RL}}(\pi) =& \mathbb{E}_{s \sim \mathcal{T}(s, \pi(s))} \left[R(s_T) + \sum_{t=0}^{T-1} \gamma^t R(s_t, \pi(s_t)) \right] \\ =& \mathbb{E}\left[G(\tau)\right] = V^\pi(s_0), \quad s_0 \in \mathcal{S}_0 \end{aligned} \tag{2.24}$$

$$\pi^* = \arg\max_\pi J^{\mathrm{RL}}(\pi) \tag{2.25}$$

Here, $R(s_T)$ is the terminal reward, $\mathcal{S}_0$ is a distribution of initial states, $G(\tau)$ is the observed return (i.e. sum of discounted rewards) over an episode described by the trajectory $\tau = (s_0, a_1, s_1, \dots, a_{T-1}, s_T)$, $\sim$ signifies that the left hand side is distributed according to the right hand side. The expectation is therefore taken over the state-visitation distribution induced by the transition dynamics and the (possibly stochastic) policy. V^π is the value function for the current policy π, measuring the total expected reward accrued when acting according to the corresponding policy π from the state in question. A final useful primitive in the RL framework is the state-action value function:

$$Q^{\pi}(s_t, a_t) = R(s_t, a_t) + \gamma V^{\pi}(\mathcal{T}(s_t, a_t)) \tag{2.26}$$

this function, also called the Q-function, is identical to the value function V except that the first action in the trajectory generated by the policy from the starting state is left open. This function then allows one to compare the total reward obtained by using the policies' choice of action in the given state, with any other action. The value function is in the RL context often referred to as the critic, while the policy is often called an actor. Moreover, the instantiation of the RL algorithm that undergoes the learning process is often called the agent.

When it comes to the taxonomy of RL algorithms, there are several meaningful categories one can employ. One such categorization is between model-free and model-based algorithms. Model-based RL algorithms use a model of the controlled system, which can either be given or learned jointly with the policy, that is incorporated into the policy to plan ahead and evaluate the future state of the system to choose the best course of action, similar to the MPC scheme. A model-free RL algorithm on the other hand assumes no knowledge of the system dynamics and learns a policy mapping states to actions directly. These approaches both have their strengths and weaknesses, model-based methods generally being more data-efficient, while model-free methods are often more computationally efficient and often have better asymptotic control performance because the control performance of a model-based method is limited by the accuracy of the model.

Another useful categorization is on-policy vs off-policy algorithms, referring to whether the algorithm requires the optimization data to be generated by the current iteration of the policy (on-policy) or if it is able to learn from data generated by any arbitrary policy (off-policy). Off-policy methods are therefore typically significantly more data-efficient than on-policy methods, as they can reuse past data gathered by previous iterations of the policy (or even learn entirely without interaction with the environment, i.e. offline RL [113]), while on-policy methods must discard the gathered data after it is used to update the policy.

A final categorization used here is between value-based and policy-based algorithms, with the divide then being based on how the policy is developed. A policy-based algorithm attempts to directly estimate the optimal policy, while a value-based algorithm first estimates a value function and from this derives the policy as the actions that maximize the value function, such as:

$$\pi^* = \arg\max_a Q^{\pi^*}(s, a) \ \ \forall s \ \in \mathcal{S} \tag{2.27}$$

In this thesis, the focus is on model-free algorithms, and a mix of policy-based and value-based methods are employed.

2.4.1 Policy Gradient Methods

Policy gradient algorithms optimize parameterized policies, denoted π_θ, where θ is the collection of parameters, directly in the parameter space of the policy, by estimating the gradient of the objective (2.24) and applying a gradient ascent scheme. The objective (2.24) however depends on the state-visitation distribution, which would require perfect knowledge of the transition dynamics to evaluate. Thus, policy gradients algorithms rely on the policy gradient theorem (2.28), which removes this dependence on the state-visitation distribution from the gradient [176]. The parameters are then updated iteratively according to the gradient estimate and the gradient ascent scheme (2.29)

$$\nabla_\theta J^{\mathrm{PG}}(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right] \tag{2.28}$$

$$\theta = \theta_{\mathrm{old}} + \eta \nabla_\theta J^{\mathrm{PG}}(\theta) \tag{2.29}$$

where η is the update step size, also called the learning rate. The expectation in (2.28) can be replaced by sample averages, i.e. to evaluate this gradient, one simply runs the policy on the system to be optimized, and observe the outcomes in terms of returns and state trajectories. Since the update scheme described above is smooth in the parameters, policy gradient methods have good convergence guarantees in theory, but since it is a sampling-based approach it suffers from high variance in the gradient estimates in practice [71, 149]. Thus, advances in these algorithms typically revolve around reducing the variance in these estimates. One important such measure is to alter the return in (2.28) by subtracting a baseline, e.g. replacing $G(\tau)$ with the advantage function:

$$A^\pi(s_t, a_t) = R(s_t, a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) \tag{2.30}$$

The advantage function estimates the value of an action relative to the average value of the actions that the policy would take in that state. Thus, actions that are better than average have positive advantages and vice versa, such that the update procedure above will raise the probability of good actions and decrease the probability of bad actions. This yields faster convergence as opposed to all sampled actions having their probability increased (assuming all rewards are positive) and relying on the fact that better actions with higher rewards are increased in probability faster than bad actions.

2.5 Proximal Policy Optimization

Proximal policy optimization (PPO) [169] is a policy gradient RL algorithm, which is popular due to its high data-efficiency (relative to other on-policy algorithms) and simplicity, both in terms of implementation and run-time complexity. The increased data-efficiency over other policy gradient algorithms is achieved by modifying the standard policy gradient objective (2.28), to allow for multiple parameter updates over the same set of data. We include the parameters of the advantage estimator $\hat{A}^{\pi_\theta}$ (and consequently the parameters of the value function estimator $\hat{V}^{\pi_\theta}$) also in the parameter vector θ.

2.5.1 Trust Region Policy Optimization

The PPO algorithm is inspired by the predecessor trust region policy optimization (TRPO) [167], which in a principled way ensures that parameter updates improve upon the objective. This is done by first estimating a region in parameter space where updates are deemed safe in the sense that they will result in an improvement of the objective, and then identifying the optimal parameter point within this region:

$$\max_\theta \quad J^{\text{TRPO}}(\theta) = \mathbb{E}\left[r(\theta)\hat{A}^{\pi_{\theta_{\text{old}}}}(s,a)\right], \quad r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \tag{2.31}$$

$$s.t. \quad \mathbb{E}\left[\text{KL}\left(\pi_{\theta_{\text{old}}}(a|s), \pi_\theta(a|s)\right)\right] \leq \delta \tag{2.32}$$

Here, $r(\theta)$ is the probability ratio between the policy under two different sets of parameters, θ and θ_{old}, the latter of which represents the policy parameters frozen at the start of the parameter update procedure, KL is the Kullback–Leibler probability divergence measure, and δ is the maximum allowed divergence. Implementing this update procedure requires second-order information about the objective, which is computationally expensive to obtain.

2.5.2 The Proximal Policy Optimization Objective

PPO emulates trust region optimization using only first order information, and it is thus considerably more computationally efficient and scalable, while simultaneously empirically demonstrating increased data-efficiency. The PPO algorithm optimizes a surrogate objective function defined as:

$$J^{\text{PPO}}(\theta) = \mathbb{E}\left[\min\left(r(\theta)\hat{A}^{\pi_{\theta_{\text{old}}}}(s,a), \text{clip}\left(r(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}^{\pi_{\theta_{\text{old}}}}(s,a)\right)\right] \tag{2.33}$$

The probability ratio $r(\theta)$ measures for any given action the ratio of the probability that it would be drawn from the policies in the numerator and denominator. As more parameter updates are performed, the two policies will increasingly differ, and the probability ratio will increasingly deviate from unity. It will be decreasing for actions that are now less likely and increasing for actions that are more likely under the new parameters. The clip operator ensures that its first argument will be contained in the limits provided as its second and third argument, such that the probability ratio is constrained to the range decided by ϵ, which typically takes on values in the range $0.1 - 0.4$. This ensures that when the advantage of the action is positive the objective value is saturated when the action surpasses $1+\epsilon$ in increased probability, while when the advantage is negative the objective saturates when the action surpasses $1-\epsilon$ in decreased probability. Lastly, the minimum operator returns the unclipped objective when the advantage of an action is negative, yet its probability has been increased under the new parameters. The unclipped objective would then make the gradient decrease the probability of this action, correcting the previous update which made the policy worse. This is the mechanism behind PPO's first-order approximation of trust-region optimization. The gradient of the PPO objective (2.33) is the same as (2.28), and is easily obtained with auto-differentiation software.

In practice, the probability ratio $r(\theta)$ is implemented in log space as the difference between the negative action-conditional probability of the policy under the two parameterizations:

$$\log r(\theta) = \sum_{\tilde{a}\in\mathcal{B}} -\log P_\theta(\tilde{a}|s) - (-\log P_{\theta_{\text{old}}}(\tilde{a}|s)) \tag{2.34}$$

$$r(\theta) = \exp(\log r(\theta)) \tag{2.35}$$

where the notation $\tilde{a}$ signifies an action drawn from the policy's action distribution. The training procedure involves running the policy in the environment for a number of time steps, generating data on the form $(s_t, \tilde{a}_t, R(s_t, \tilde{a}_t), s_{t+1}, \hat{A}^{\pi_\theta}(s_t, \tilde{a}_t))$ which is stored in a buffer. Then, minibatches $\mathcal{B}$ are sampled from this buffer, the objective (2.33) and its gradient are evaluated, and the parameters are updated according to (2.29). This is repeated for the configured number of epochs such that each data point is reused several times, increasing data-efficiency, after which the data in the buffer is discarded and new data is gathered.

2.5.3 Advantage Function Estimation

The advantage function is estimated using the generalized advantage estimation (GAE) algorithm [168]. The advantage is defined in terms of the low bias but high variance information in the sampled rewards R, and the high bias but low variance information estimated by the value function V. Equation (2.30) shows the 1-step advantage estimate, which we now denote $\hat{A}_t^{(1)}$, but this can be unrolled further to produce an estimate that contains more information from R, and less of the information from V. In GAE this bias-variance tradeoff is summarized as in (2.36) and controlled through the factor $\kappa \in [0, 1]$, where $\kappa = 0$ recovers the 1-step advantage in (2.30) — which has the highest bias but lowest variance estimate — and $\kappa = 1$ corresponds to using all the observed rewards R as an estimate, which yields the lowest bias and the highest variance. The GAE algorithm (2.36) for $0 < \kappa < 1$ yields a compromise between bias and variance, controlled by κ, where H is the length of the trajectory collected by the policy:

$$\hat{A}^{\pi_\theta}(s_t, a_t) = \delta_t + (\gamma\kappa)\delta_{t+1} + \cdots + (\gamma\kappa)^{H-t+1}\delta_{H-1} \tag{2.36}$$

$$\delta_t = R(s_t, a_t) + \gamma\hat{V}^{\pi_\theta}(s_{t+1}) - \hat{V}^{\pi_\theta}(s_t) \tag{2.37}$$

The value function in turn is estimated by a parameterized function approximator $\hat{V}^{\pi_\theta}$, that is iteratively improved using fitted value iteration in the manner described by (2.9), where ℓ corresponds to R, x and $\hat{p}$ corresponds to s, u corresponds to a, and ρ corresponds to γ.

2.6 Value-Based Deep Reinforcement Learning Algorithms

The DDPG [117] algorithm was the first algorithm to successfully demonstrate continuous control with DRL. We will in this and the next section omit the implicit superscript π_θ indicating the policy whose value is estimated by the value functions.

DDPG and its derivations (e.g. twin delayed DDPG (TD3) [63]) are off-policy, model-free, actor-critic algorithms, that are simultaneously policy-based and value-based algorithms. That is, they simultaneously estimate the state-action value function and a parameterized policy, and uses the fact that the neural network function approximator employed for the state-action value function is differentiable wrt. to its inputs to provide gradients for the policy. Thus, the policy can probe the state-action value function in order to iteratively identify the actions that maximizes the state-action value function, Q, similar to (2.27). This provides a unique time-scale problem for these types of algorithms: Since the Q-function measures the total cost obtained by a particular policy, changes in the policy therefore leads to changes in the Q-values, and since these Q-values are again then employed to further update the policy, the Q-function should ideally be allowed to converge before it is used to update the policy. That is, the Q-function should converge on a much faster time-scale than the policy, such that the policy essentially seems fixed from the perspective of the Q-function.

As discussed earlier, off-policy RL algorithms like DDPG typically enjoy better data-efficiency than on-policy algorithms like PPO, as they can reuse the gathered data. However, one problem that arises with the reuse of past experiences is the bias introduced by time-correlation in the data. Therefore, DDPG-style algorithms save past experiences in a replay buffer, $\mathcal{D}$, from which data is sampled uniformly in time, breaking the time-correlation. Further, since the policy in these algorithms is deterministic, it is necessary to introduce some means of exploration, which is typically achieved by gathering data with a separate behaviour policy, which is typically obtained by adding noise to the deterministic policy (2.38). The Q-function is then learned in a similar fashion as described in Section 2.2 on data sampled from the replay buffer:

$$\pi_\theta^b(s) = \pi_\theta(s) + \zeta^b, \ \ \zeta^b \sim \mathcal{N}(\mu^b, \sigma^b) \tag{2.38}$$

$$y^{\mathrm{DDPG}}(s_t, a_t) = R(s_t, a_t) + \gamma Q_{\theta Q'}(s_{t+1}, \pi_\theta(s_{t+1})) \tag{2.39}$$

$$J^{\mathrm{DDPG,Q}}(\theta^Q) = \mathbb{E}_{(s_t, \tilde{a}_t, R(s_t, \tilde{a}_t), s_{t+1}) \sim \mathcal{B}} \left[(y^{\mathrm{DDPG}}(s_t, \tilde{a}_t) - Q_{\theta Q}(s_t, \tilde{a}_t))^2 \right] \tag{2.40}$$

where θ^Q is the parameterization of the Q-function, $\mathcal{B}$ is the minibatch of sampled data, $\tilde{a}_t$ is the action produced by the behaviour policy (2.38) in state s_t, μ^b and σ^b are hyperparameters, and y_t^{DDPG} is again the regression target. A final measure proposed in the DDPG algorithm is the use of a target Q-function in the regression target (2.39) to stabilize the learning process of the Q-function. The target Q-function is a separate neural network whose parameters $\theta^{Q'}$ are Polyak averaged from θ^Q:

$$\theta^{Q'}_{\text{init}} \leftarrow \theta^{Q}_{\text{init}} \tag{2.41}$$

$$\theta^{Q'}_{\text{new}} = \varrho\theta^{Q'}_{\text{old}} + (1-\varrho)\theta^Q \tag{2.42}$$

such that the (2.39) provides a slower moving target to regress towards, where ϱ is the Polyak averaging factor. Finally, the policy is iteratively improved to maximize the Q-function over the sampled data, using gradient ascent with gradients from the following objective:

$$J^{\text{DDPG},\pi}(\theta) = \arg\max_{\theta} \mathbb{E}_{s\sim\mathcal{B}}\left[Q_{\theta^Q}(s, \pi_\theta(s))\right] \tag{2.43}$$

The gradients of (2.40) and (2.43) are readily obtained with automatic differentiation software.

2.7 Twin Delayed DDPG

In [63] the authors identify several failure modes of DDPG, and proposes the twin delayed DDPG (TD3) algorithm to rectify these failure modes. First, DDPG is prone to overestimating the Q-values. High Q-values dominate the immediate reward $R(s_t, \tilde{a}_t)$ in the regression target (2.39), thus successfully leading to low MSBE as the variance introduced by the immediate reward is negligible in comparison. This subsequently leads to a poor policy, as it is trained to optimize an increasingly meaningless Q-function, perpetuating a vicious cycle. To remedy this overestimation, TD3 employs two Q-functions whose parameters $\theta^{Q,1}$ and $\theta^{Q,2}$ are initialized differently, such that they provide different Q-value estimates, and the minimum of the two Q-values are chosen for the regression target. Moreover, to further disincentivize the policy to exploit erroneous Q-values, the action used to calculate the target Q-value is sampled from a target policy (which again is a set of parameters θ' polyak averaged from θ) and further smoothed with noise ζ^a:

$$y^{\text{TD3}}(s_t, a_t) = R(s_t, a_t) + \gamma \min_{i\in\{1,2\}} Q_{\theta^{Q'},i}(s_{t+1}, a'(s_{t+1})) \tag{2.44}$$

$$a'(s) = \text{clip}(\pi_{\theta'}(s) + \text{clip}(\zeta^a, -\zeta^a_{\max}, \zeta^a_{\max}), a_{\min}, a_{\max}), \;\; \zeta^a \sim \mathcal{N}(0, \sigma^a) \tag{2.45}$$

where $\pi_{\theta'}$ is the target policy, $\zeta^a_{\max}$ is the maximum action noise, and $a_{\min}, a_{\max}$ are the minimum and maximum actions of the controlled system, respectively. Finally, to address the time-scale issue described earlier, TD3 updates the policy less often than the Q-function with the frequency of policy updates being a hyper-parameter of TD3. The objective functions of the TD3 algorithm are therefore as follows, where both Q-functions are trained independently using (2.46):

$$J^{\text{TD3,Q}}(\theta^Q) = \mathbb{E}_{(s_t,\tilde{a}_t,R(s_t,\tilde{a}_t),s_{t+1})\sim\mathcal{B}}\left[(y^{\text{TD3}}(s_t,\tilde{a}_t) - Q_{\theta^Q}(s_t,\tilde{a}_t))^2\right] \tag{2.46}$$

$$J^{\text{TD3},\pi}(\theta) = \arg\max_{\theta} \mathbb{E}_{s\sim\mathcal{B}}\left[Q_{\theta^{Q,1}}(s,\pi_\theta(s))\right] \tag{2.47}$$

and where the choice of which Q-function to optimize in (2.47) is arbitrary.

2.8 Soft Actor Critic

The last RL algorithm employed in this work is soft actor critic (SAC) [85]. Like DDPG it is an off-policy, actor-critic, value-based algorithm, but it bridges the gap towards the policy-gradient class of algorithms in that it maintains a stochastic policy, and exploration is done on-policy. The defining characteristic of SAC compared to DDPG and TD3 is that it is an entropy-regularized RL algorithm, which is to say that in addition to maximizing the expected sum of rewards as in (2.24) it jointly aims at maximizing the entropy of the policy (i.e. the randomness of the policy). This changes the formulation of the value functions as well, then referred to as soft value functions:

$$V(s_t) = \mathbb{E}_{a\sim\pi_\theta(a|s),s\sim\mathcal{T}(s,\pi_\theta(a|s))}\left[\sum_{t'=t}^{T}\gamma^{t'}\Big(R(s_{t'},a_{t'}) + \chi\mathcal{H}(\pi_\theta(\cdot|s_{t'}))\Big)\right] \tag{2.48}$$

$$Q(s_t,a_t) = \mathbb{E}_{s\sim\mathcal{T}(s,\pi_\theta(a|s))}\Bigg[R(s_t,a_t) + \gamma V\left(\mathcal{T}(s_t,a_t)\right)\Bigg] \tag{2.49}$$

where $\mathcal{H}(\pi_\theta(\cdot|s)) = \mathbb{E}_{a\sim\pi_\theta(a|s)}\left[-\log\pi_\theta(\cdot|s)\right]$ is the entropy, equal to the negative log probability of the action-distribution of the policy in the state in question, and χ is the weighting coefficient between the two objectives, balancing exploration and exploitation. The authors propose in a later work a method to automatically learn the appropriate value of the weighting factor χ [86].

Because the policy is incentivized to act as randomly as possible while maximizing the rewards, it tends to yield robust policies that are explicitly trained to handle perturbations, it is inherently exploratory, and it can give rise to effects such as multi-modal behaviour which can be beneficial in many situations. Further, a highly random policy is less likely to specialize in certain behaviours such as abusing quirks of a simulated training environment and therefore tends to transfer better to other environments than non-entropy-regularized policies.

SAC maintains a value function parameterized by θ^V, two Q-functions parameterized by $\theta^{Q,1}, \theta^{Q,2}$ as is the case for TD3, and a stochastic policy parameterized by θ. The value function is trained with the following objective, where (2.51) is an unbiased estimator of the gradient:

$$J^{\text{SAC,V}}(\theta^V) = \mathbb{E}_{s_t \sim \mathcal{B}}\left[\left(V_{\theta^V} - \mathbb{E}_{a_t \sim \pi_\theta}\left[Q_{\theta^Q}(s_t, a_t) - \log \pi_\theta(a_t, s_t)\right]\right)^2\right] \tag{2.50}$$

$$\hat{\nabla}_{\theta^V} J^{\text{SAC,V}} = \nabla_{\theta^V} V_{\theta^V}(s_t)\left(V_{\theta^V}(s_t) - \min_{i \in \{1,2\}} Q_{\theta^{Q,i}}(s_t, a_t) + \log \pi_\theta(a_t|s_t)\right) \tag{2.51}$$

and where the actions a_t importantly are sampled from the current iteration of the policy π_θ, as opposed to sampled from the dataset. The two Q-functions are then trained to minimize the soft MSBE as follows:

$$J^{\text{SAC,Q}}(\theta^Q) = \mathbb{E}_{s_t, a_t \sim \mathcal{B}}\left[\left(y^{\text{SAC}}(s_t, a_t) - Q_{\theta^Q}(s_t, a_t)\right)^2\right] \tag{2.52}$$

$$y^{\text{SAC}}(s, a) = R(s, a) + \gamma \mathbb{E}_{s \sim \mathcal{T}}\left[V_{\theta^{V'}}(s_{t+1})\right] \tag{2.53}$$

$$\hat{\nabla}_{\theta^Q} J^{\text{SAC,Q}} = \nabla_{\theta^Q} Q_{\theta^Q}(s_t, a_t)\left(Q_{\theta^Q}(s_t, a_t) - R(s_t, a_t) - \gamma V_{\theta^{V'}}(s_{t+1})\right) \tag{2.54}$$

where again $\theta^{V'}$ is the target function of the value function approximator. Finally, the policy is optimized to minimize the following objective:

$$J^{\text{SAC},\pi}(\theta) = \mathbb{E}_{s_t \sim \mathcal{B}}\left[\text{KL}\left(\pi_\theta(\cdot|s_t), \frac{\exp(Q_{\theta^Q}(s_t, \cdot))}{Z_{\theta^Q}(s_t)}\right)\right] \tag{2.55}$$

where KL is the Kullback-Leibler divergence measure and Z_{θ^Q} is a normalization function ensuring that the second argument of the Kullback-Leibler divergence is a proper probability distribution. The objective in (2.55) is however not readily optimizable because the expectation is taken over actions, which again depend on the parameters of the policy that we want to optimize. Therefore, the policy is reparameterized using a neural network transformation:

$$a_\theta(s, \xi) = \tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \xi), \quad \xi \sim \mathcal{N}(0, I) \tag{2.56}$$

here, μ_θ and σ_θ are two parameterized deterministic functions of the input, representing the mean and covariance of the output, respectively. The notation $\odot$ denotes element-wise matrix-multiplication and ξ is independently sampled Gaussian noise. The entropy can therefore be controlled in a state-dependent manner through the σ_θ function. Finally, the output is saturated with the hyperbolic tangent function, which squashes the Gaussian's infinite support to the domain $[-1, 1]$, limiting the adverse effects of extreme noise values and giving bounded outputs. Note that when referring to the policy trained with SAC in later chapters of this thesis, it is this transformed policy that is referred to and denoted by π_θ to ensure consistency with the standard RL nomenclature.

Using this reparameterization, the objective in (2.55) can be rewritten as (2.57) where the expectation is now over the noise ξ and the objective is therefore differentiable for θ. An approximate gradient for this objective is given in (2.58):

$$J^{\mathrm{SAC},\pi}(\theta) = \mathbb{E}_{s_t \sim \mathcal{B}, \xi}\left[\log \pi_\theta(a_\theta(s_t, \xi) | s_t) - \min_{i \in \{1,2\}} Q_{\theta^{Q,i}}(s_t, a_\theta(s_t, \xi))\right] \tag{2.57}$$

$$\begin{aligned}\hat{\nabla}_\theta J^{\mathrm{SAC},\pi}(\theta) =& \nabla_\theta \log \pi_\theta(a_t, s_t) \\ &+ \left(\nabla_{a_\theta} \log \pi_\theta(a_t | s_t) - \nabla_{a_\theta} \min_{i \in \{1,2\}} Q_{\theta^{Q,i}}(s_t, a_t)\right) \nabla_\theta a_\theta(s_t, \xi)\end{aligned} \tag{2.58}$$

2.9 Hindsight Experience Replay

Hindsight experience replay (HER) [6] is a technique to improve the data-efficiency of off-policy RL algorithms, particularly when solving MDPs with sparse rewards. A sparse reward only yields a feedback signal for some subset of the state space and is typically constant elsewhere. In these problems, only episodes where the agent reaches the episodes' goal state will yield a learning signal, and without any intelligent exploration or guidance measures undertaken, reaching the goal state is dependant on the typically random exploration of the initial policy. To accelerate learning in these scenarios, Andrychowicz et al. [6] proposed to retroactively exchange the episodes' goal state for some other state that was reached by the controller during the episode, and recalculate the rewards as if this new state was the intended goal state. In this way, learning signals are reached more quickly, and existing data can be augmented and reused to improve sample complexity. For each real experience the agent has, several synthetic ones are inserted into the dataset where the goal state and rewards are recalculated. How many such synthetic experiences to generate and how to select the new goal state are hyperparameters of the HER technique, but empirical observations suggest that states that were reached sometime in the not too distant future from the current state serve as the best candidates for new goal states, and that 2-16 synthetic experiences per real experience is optimal.

Part I

Combining Knowledge-Based Control Techniques with Reinforcement Learning

3 Introduction

In this part of the thesis, we consider the combination of conventional control techniques with reinforcement learning, both in the direction of enhancing existing control approaches and conversely using existing control approaches to enhance RL for control. The first three chapters concern automatic optimization of what we call the meta-parameters of the MPC scheme. By meta-parameters we mean parameters that affect the structure of the OCP (including when it is computed, i.e. its initial conditions), as opposed to what we consider internal parameters of the MPC that affect the solution to a given OCP. Chapter 4 considers optimization of the recomputation meta-parameter, that is, instead of recomputing the solution to the OCP at every step as is the default MPC paradigm, we consider the timing of when to compute the MPC a decision variable that we optimize with RL. In Chapter 5 we look at how one can learn the optimal prediction horizon meta-parameter of the MPC as a function of the state using RL. Chapter 6 then takes a more holistic approach; we argue that the questions of *when* and *how* to compute the MPC are related and optimizing these meta-parameters in isolation fails to consider the interconnections and indirect effects at play. We therefore develop a single RL policy that can incorporate any parameter of the MPC scheme (the described meta-parameters and any internal parameter), and jointly optimize them in order to improve the control algorithm. While the method presented in Chapter 6 therefore in a sense supersedes the two preceding chapters and their methods, there is an argument to be made for simplicity and the fact that tuning all meta-parameters might not make sense for every application. Moreover, a gradual introduction of these concepts constitute a more accessible presentation and corresponds to the order in which these concepts were conceived and developed.

Finally, this part of the thesis concludes with Chapter 7 which presents a method to accelerate the learning of an RL controller by guiding the initial exploration phase with an existing (suboptimal) controller. While this topic might not seem directly related to the other topics treated in this part of the thesis, in Chapter 7 we present a relationship between the optimization framework presented in Chapter 6 and the topic of guiding RL with existing solutions to the control problem, a relationship we would have looked further into given more time.

The following section serves as a shared introduction for the next three chapters, in order to limit redundancy due to their overlapping subject matter. In this thesis we implement the MPC with the CasADi framework [5], the Ipopt optimizer [192] and the do-mpc Python library [122].

3.1 Motivation and Related Work

MPC is a powerful optimizing control technique, capable of successfully controlling a wide range of systems with high control proficiency while respecting system constraints. The NMPC (henceforth referred to as just MPC) can even handle nonlinear dynamics and nonlinear constraints, and while not as simple as its linear counterpart, is increasingly being considered for applications with fast dynamics [3, 74, 200, 95]. However, one of the main drawbacks of the MPC method is its high computational complexity, which makes it ill-suited for applications with low-powered hardware platforms or battery energy restrictions, necessitating some form of compromise in its implementation, see e.g. [56] and [70].

The high computational complexity of the MPC comes as a result of its online operation consisting of solving a numerical OCP at every time step, executing the first control input of the computed optimal solution, and then solving the OCP again at the subsequent time step. A further challenge of the MPC method is the need to tune its parameters to the task at hand, greatly affecting the control proficiency, robustness and computational complexity of the controller. There are several ways to reduce the computational requirements of MPC, with the two main tunable parameters being the optimization horizon length and the step time. Reducing the optimization horizon can lead to the controller exhibiting myopic behaviour, favouring short term gains over long term performance, while large step sizes can in turn lead to worse control performance as the controller is unable to respond to high-frequency dynamics and transient disturbances. The primary technique to reduce computational complexity is to reduce the number of iterations of the optimization procedure. This early termination scheme [171] terminates optimization before the optimality conditions are met, yielding a sub-optimal solution and subsequent sub-optimal control performance. Some theoretical guarantees on stability and sub-optimality bounds have been presented [171, 157]. Other notable approaches are explicit MPC, semi-explicit MPC, and move blocking, the latter of which is a technique to reduce the number of decision variables by assuming these are constant over several contiguous steps [39]. In explicit MPC [21] the control law is precomputed offline as a piecewise linear function, and online computation consists of identifying the current operating region and evaluating the corresponding linear control law, while semi-explicit methods precompute some parameterization of the OCP such that online computation is of lower dimension [69]. Other parameters of the MPC scheme are also subject to tuning, e.g. discretization step size, objective functions, optimality tolerances, etc., including some non-obvious facets such as the constraints which can in some instances be tuned more aggressively yielding better or safer behaviour. How to tune all these parameters for the MPC scheme is a non-trivial question.

RL [176] is a field of machine learning concerned with optimal sequential decision making. While RL has proven to be the state-of-the-art approach for certain classes of problems requiring complex decision making over long time horizons, such as game-playing [137, 23, 166] and dexterous robotic in-hand manipulation [8], it has in general seen limited adoption for control. This is in large part due to its data intensive nature, combined with its inability to handle constraints and therefore lack of guarantees for safe operation of the system, both in the exploration phase and in the exploitation phase. Several works have suggested approaches to incorporate learning into the control algorithm in a safe manner by learning how to augment existing control techniques such as MPC [10, 159, 124, 185, 104, 57, 199, 116, 125]. However, concerns about the stability and recursive feasibility of the closed-loop systems as is considered in these works are outside the scope of this thesis.

Other works have suggested combining RL and MPC in a model-based manner where the dynamics are learned and then used in an MPC scheme [141, 107]. In this thesis, we employ the approach of using the MPC as a function approximator (i.e. to implement the policy) in RL following [72], which establishes that such an approach can recover the optimal policy for the controlled system, even if the underlying model in the MPC scheme is incorrect, by tuning of MPC objective costs alone. The efficacy of this approach is also demonstrated in [134] for trajectory tracking of multirotor UAVs. Later works have extended these results to more classes of RL algorithms, i.e. deterministic policy gradients [75] and stochastic policy gradients [76]. The novelty in the methods of this thesis on the combination of MPC and RL lies in considering the meta-parameters of the MPC scheme as parameters that can be optimized using RL.

4

Reinforcement Learning of the Model Predictive Control Computation Timing

This chapter represents our first foray into optimizing the meta-parameters of the MPC scheme, in which we look exclusively at the problem of deciding when to compute the MPC. While the method presented in this chapter is identical to the recomputation part of the method in Chapter 6, the experiment we present is distinct: The system dynamics are different, featuring unmeasured process disturbance rather than model mismatch between the plant and the MPC as is the case in Chapter 6, and we impose an additional state constraint. Moreover, we present a comparison to the suboptimal early-termination MPC as a baseline. This chapter is based on the following article:

- [37] Eivind Bøhn, Sebastien Gros, Signe Moe, and Tor Arne Johansen. Optimization of the model predictive control update interval using reinforcement learning. *IFAC-PapersOnLine*, 54(14):257–262, 2021. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2021.10.362. 3rd IFAC Conference on Modelling, Identification and Control of Nonlinear Systems MICNON 2021

Note that since we in this chapter limit the scope to optimization of the recomputation meta-parameter, and not of the controllers themselves, the subscript indicating the tunable parameters of the controllers described in Chapter 2 are omitted.

4.1 Introduction

As an alternative to the normal digital control system approach of measuring system state and calculating control signals at equidistant points in time, one can instead select these points by some other criteria, yielding a control system that consists of a control law and a triggering policy. This event-triggered control paradigm could be used to increase control performance or reduce the frequency of computations based on how the triggering policy is designed [207, 88]. Event-triggered MPC has been suggested in the literature to reduce resource usage particularly in networked communication systems and multi-agent systems [93, 22, 115, 42]. In these works, the triggering policy is a predetermined hand-crafted policy based on domain knowledge about the system and the controller, a learned policy on the other hand can be derived without requiring system knowledge and be adaptive to changing dynamics and uncertain disturbances. In [198] the authors use a learned empirical risk-minimization model to predict the unknown system noise in an MPC framework, and use its output to determine the triggering thresholds. For a recent review of machine learning applied to event-triggered networked control systems, see [172].

In this chapter we look at the single agent setting and suggest learning the triggering policy with RL. Several other works such as [16] and [197] have proposed RL for learning a triggering policy, however these works simultaneously learn the control law as opposed to our work which employs MPC for this purpose. The contribution of this chapter lies in introducing a novel three-part control architecture, combining a dual mode MPC and LQR control law [136] with an RL triggering policy, and in deriving how the event-triggered MPC problem can be framed as an MDP, facilitating RL. The MPC solves the OCP, providing a plan in the form of an input sequence and the predicted state trajectory from executing said input sequence, while the LQR provides an additive compensatory input based on the state trajectory prediction errors, extending the viability of the computed MPC input sequence. Finally, the RL triggering policy, henceforth referred to as the recomputation policy, selects time instants when the improved control performance from recomputing the MPC solution outweigh the computational costs. We empirically demonstrate the effectiveness of the proposed architecture through an experiment with the inverted pendulum system.

A motivation for this type of architecture can be found in the dual process theory of the human mind [53]. This theory hypothesizes that the mind in principal has two systems: the explicit system 2 which has to be actively engaged by the consciousness and therefore consumes the limited resource of focus, while the implicit system 1 is automatically engaged to decide reactions to external events whenever the world behaves according to our expectations. System 1 is fast, reactive and based on heuristics, while system 2 is slower, planning and more proactive in nature. In this analogy, the MPC acts as system 2, providing a plan that accounts for all the nonlinearities and global effects of the controlled problem, the LQR is the fast heuristics-based system 1, while the learned recomputation policy acts as the consciousness monitoring the controlled problem and activating system 2 as needed.

4.2 Learning-based Event-Triggered MPC

4.2.1 Proposed Control Algorithm

In this chapter we employ the event-triggered MPC formulation [115] (see Section 2.1.2 for a description). The recomputation policy $\pi_\theta(a \mid s)$ is a stochastic policy parameterized by θ, defining a probability distribution over a binary output action, a, where $a_t = 1$ corresponds to recomputing the MPC solution at step t and $a_t = 0$ corresponds to not recomputing it. The recomputation policy π_θ receives a state s that is not simply the real system state x, because π_θ depends on the state of the system when the MPC was last computed. For RL optimization theory to apply for π_θ the state s must be a Markov state. We will therefore define next an augmented state space $\mathcal{S}$ that does have this property.

Assume that the MPC solution delivers an input sequence $u^{\mathrm{M}}_{i:i+N-1}$ associated to a measured state $\bar{x}_i$, at time instant i and that the sequence $u^{\mathrm{M}}_{i:i+n}$ is applied to the system. The inputs received by the plant in the time interval $i : i+n$ are then a function of state $\bar{x}_i$ at time i. Since $n \in \{0, \ldots, N-1\}$ is not known a priori, one ought to generally view the control system stemming from the triggering policy and the MPC scheme as being a control law from an augmented state, $s \in \mathcal{S}$, containing 1. the current state of the system, 2. the state of the system when the last optimization took place, and 3. the number of time samples from which the last optimization took place. Labelling the current time of the system as t and the last time when the optimization occurred as i, that augmented state reads as:

$$s_i = \begin{bmatrix} \bar{x}_t \\ \bar{x}_i \\ t-i \end{bmatrix} \tag{4.1}$$

where $i < t \leq i + N - 1$, $\bar{x}_t$ has progressed from $\bar{x}_i$ according to the real system dynamics, and the deterministic state transition $\bar{x}_i \leftarrow \bar{x}_t,\ i \leftarrow t$ occurs when $a_t = 1$ is drawn from the stochastic policy $\pi_\theta(a_t \mid s_t)$. This Markov state s_t is illustrated in Figure 4.1. It is not necessary to include the MPC's predicted trajectory $\hat{x}_{i+1:i+N}$ in s to ensure the Markov property, as the prediction is fully determined by $\bar{x}_i$ and the (known) MPC control law. The MPC control law actually deployed on the system then reads as:

$$\pi^{\mathrm{M}}(s_t) = u^{\mathrm{M}}_{t-i}(\bar{x}_i) \tag{4.2}$$

where u^{M}_{t-i} is the $(t-i)^{\mathrm{th}}$ element of the MPC solution solved for the initial state $\bar{x}_i$. The dual mode control law reads as:

$$\pi^{\mathrm{DM}}(s_t) = \begin{cases} u^{\mathrm{M}}_{t-i}(\bar{x}_i) + u^{\mathrm{L}}(\hat{x}_t - \bar{x}_t) & \text{, if } \pi_\theta(a_t|s_t) = 0 \\ u^{\mathrm{M}}_0(\bar{x}_t) & \text{, if } \pi_\theta(a_t|s_t) = 1 \end{cases} \tag{4.3}$$

This control law defines the inputs actually applied to the plant in a closed-loop system. The control system and the plant dynamics together define the state transition dynamics in the augmented state space $\mathcal{S}$, where the states s does have the Markov property. The event-triggered MPC problem is therefore an MDP in the augmented state space $\mathcal{S}$, such that classic RL can be applied on s. In that context exogenous input variables (e.g. forecasts) $\hat{p}$ can be seen as being part of the states $\bar{x}_i$, $\bar{x}_t$.

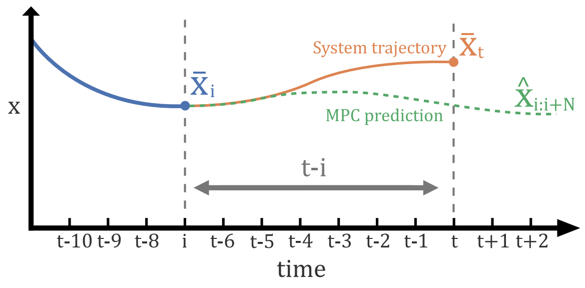


Figure 4.1: A sufficient state representation to ensure the Markov property for the recomputation policy, illustrated for a system with a one-dimensional state space.

The proposed control system is described in Algorithm 1. The MPC solution is computed at the first time step, generating an optimal predicted state trajectory $\hat{x}_{1:N}$ and a sequence of optimal inputs that produces this trajectory $u^{\mathrm{M}}_{0:N-1}$. The first input is then applied to the system as usual, but instead of discarding the rest of the MPC solution, we allow a learned policy to decide based on an observation of the current recomputation state, s, whether the last MPC solution is still of acceptable quality, or if it should be recomputed. If the recomputation policy decides not to recompute, a tracking LQR is applied to the state prediction error e to correct the state trajectory to match the plan generated by the MPC. Algorithm 1 describes the operation of the control system with a fixed recomputation policy, e.g. after the RL policy is done learning, as how and when the learning is performed would depend on the chosen learning algorithm. In Algorithm 1, the function LQR produces the gain matrix as a function of the LQR matrices as described in Section 2.3.

4.2.2 Reward Function

The objective of the recomputation policy is to reduce the frequency of the control algorithm computations while maintaining acceptable control performance. It therefore needs to be incentivized to perform well on the control objective, respect problem constraints and receive some penalty for engaging the computationally expensive controller. With this in mind we employ a cost function of the form:

$$R(s, a) = R_P(s') + \lambda_H(T - t)R_H(s') + \lambda_C R_C \tag{4.4}$$

Where $s' = \mathcal{T}(s, a)$, i.e. the resulting state from applying action a in state s, and λ_* are weighting factors. R_P is the control objective, e.g. the MPC stage cost ℓ, R_H is a binary predicate indicating if a constraint has been violated upon which the episode is ended, such that the cost is proportional to how many steps are left in the episode. Finally, $R_C = a$, indicating if a new plan is computed, and its weight should therefore reflect the relative computational complexity of the two controllers.

4.2.3 Initialization

As the objective of this architecture is to reduce the energy usage of the MPC by identifying when it is viable to *not* recompute the MPC solution (as opposed to when it is necessary to recompute), we initialize the recomputation policy to (with high probability) mimic the standard MPC approach and always elect to recompute. This is achieved by setting the bias term of the policy's recompute output to a high value. Further, since the LQR's purpose in this architecture is to execute the MPC's plan, we tune it to be a local approximation to the MPC scheme in steady-state. That is, we linearize the MPC model at the steady-state of the system to obtain the $\mathbf{A}$ and $\mathbf{B}$ matrices, and then we set the $\mathbf{Q}$ and $\mathbf{R}$ weighing matrices as the Hessian of the MPC objective also at the steady-state.

4.2.4 Policy Representation

We model the recomputation policy output as a Bernoulli random variable, where the policy itself can be any approximator capable of outputting the moments of the distribution, e.g. logistic regression models or NNs. This yields a stochastic policy such that exploration is done on-policy. See Section 4.3.2 for a specific example of such a policy realization.

Algorithm 1: Control System Algorithm

```
$\mathbf{A}, \mathbf{B} = \nabla_{x,u} \hat{f}(x, u, \hat{p})|_{x=0,u=0}$ ;
$\mathbf{Q}, \mathbf{R} = \nabla^2_{x,u} \ell(x, u, \hat{p})|_{x=0,u=0}$ ;
Initialize LQR: $\mathbf{K} = \mathrm{LQR}(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R})$ ;
Compute initial MPC solution and execute input: $i \leftarrow 0, u^{\mathrm{M}}_0$ ;
**for** $t = 1, 2, \ldots, T$ **do**
    Measure system state at next sampling instant: $\bar{x}_t$ ;
    **if** $t = i + N$ or recomputation policy $\pi_\theta(a_t \mid s_t)$ draws $a_t = 1$ **then**
        Update MPC, state, and variables: $i \leftarrow t$ ;
        Compute MPC solution: $\hat{x}_{i+1:i+N}, \ u^{\mathrm{M}}_{i:i+N-1} = \pi^{\mathrm{M}}(x_i, \hat{p}_i)$ ;
        Execute MPC control input: $u^{\mathrm{M}}_i$ ;
        Break loop ;
    **else**
        Compute prediction errors: $e_t = \hat{x}_t - \bar{x}_t$ ;
        Compute additive LQR input: $u_t = u^{\mathrm{M}}_{t-i} + u^{\mathrm{L}}(e_t)$ ;
        Apply input constraints: $u_t = \mathrm{proj}_h u_t$ ;
        Execute control input: $u_t$ ;
    **end**
    (Update $\theta$ as dictated by the RL algorithm)
**end**
```

4.3 Experiment

We illustrate the proposed control system on the classic control task of balancing an inverted pendulum mounted on a small cart. This system has hard physical constraints that must be respected, in terms of the carts' position and the angle of the pendulum, and has regions of the state space that are highly nonlinear while stable conditions with the pendulum in the up position are approximately linear. The MPC controller is therefore necessary to handle the constraints of the problem and bring the system to stable conditions, while the LQR is sufficient to maintain stability. Moreover, the cart pendulum system is an embedded system where the energy source powering the controllers is finite, limiting the computation of the control system is therefore of interest.

The state space consists of $x = [\psi, v, \beta, \omega]^\top$ which is position and velocity of cart along horizontal axis, and angle of pendulum to the vertical axis and angular velocity of the pendulum, respectively. We discretize the system equations in (4.5-4.6) with step time $\Delta_t = 0.04s$, and add unmeasured process disturbance $w_t = [0, 0, 0, \mathcal{N}(0, 1)]^\top$, which means that feedback control from the LQR is a necessary addition to the MPC plan to stabilize the pendulum in between computations. We sample initial states according to $x_0 = [0, \mathcal{U}(-1, 1), \mathcal{U}(-0.78, 0.78), \mathcal{U}(-1, 1)]^\top$ and a time-varying position reference $\psi_{r,t} \in \mathcal{U}(-0.5, 0.5)$ that is redrawn every 50 steps, where $\mathcal{U}$ is the uniform distribution. We put constraints on the input, pendulum angle and position of the cart (4.7). Finally, the physical parameters are the pendulum length and weight, $l = 0.25,\ \ m = 0.2$, the total weight of the cart and the pendulum, $M = 0.8$, and the gravitational acceleration, $g = 9.81$.

The MPC is configured with prediction horizon $N = 25$, $\mathbf{D} = 0.1$, $\ell(x_t, u_t, p_t) = m(x_N) = E_{\text{kin}} - E_{\text{potential}} + 10(\psi_{r,t} - \psi_t)^2$. This objective function promotes stabilization of the pendulum in the upright position through minimizing the kinetic energy and the negative potential energy of the system, while tracking the position reference ψ_r. We set $\lambda_h = 10$ and $\lambda_C = 0.05$. An episode is terminated after a maximum of 150 steps, or when one of the state constraints are violated.

$$\dot{\psi} = v, \ \ \dot{v} = \frac{mgsin(\beta)cos(\beta) - \frac{4}{3}(u + ml\omega^2 sin(\beta))}{mcos^2(\beta) - \frac{4}{3}M} \tag{4.5}$$

$$\dot{\beta} = \omega, \ \ \dot{\omega} = \frac{Mgsin(\beta) - cos(\beta)(u + ml\omega^2 sin(\beta))}{\frac{4}{3}Ml - mlcos^2(\beta)} \tag{4.6}$$

$$-10 \leq u_t \leq 10, \ -90^\circ \leq \beta_t \leq 90^\circ, \ -1 \leq \psi_t \leq 1 \tag{4.7}$$

4.3.1 Baseline Policies

To assess the quality of the RL policy we compare the learned policy to the fixed baseline policies: standard MPC, never recomputing until end of horizon, and a recomputation policy that statically recomputes every t time step. For the two latter policies, the LQR is applied between computations as is the case for the RL policy. We also compare with an early termination (suboptimal) MPC for which the termination tolerances are relaxed compared to the standard MPC.

4.3.2 Training and Evaluation

We train and evaluate the policies in an episodic setting with randomly generated initial conditions, process-noise, time-varying exogenous variables etc. as discussed above. To evaluate the policies and minimize the effects of the random variables on the results, we construct a test set consisting of 100 episodes where all random variables are drawn in advance such that the episode is consistent across policy evaluations. During evaluation, we take the deterministic action of the policy, i.e. the mode of the action distribution. We use the average negative undiscounted return $-G(\tau)$ (i.e. set $\gamma = 1$) of the episodes as the objective to compare models, where τ is the sequence of states and actions in the episode.

For PPO we use the hyperparameters suggested in the paper from [7], with the following exceptions: the policy is a two-layer feed-forward neural network with 16 nodes in each layer, $\gamma = 0.97$. Furthermore, since RL algorithms are known to be sensitive to the random initialization of its parameters, we train and report mean results over 5 initialization seeds.

4.4 Results and Discussion

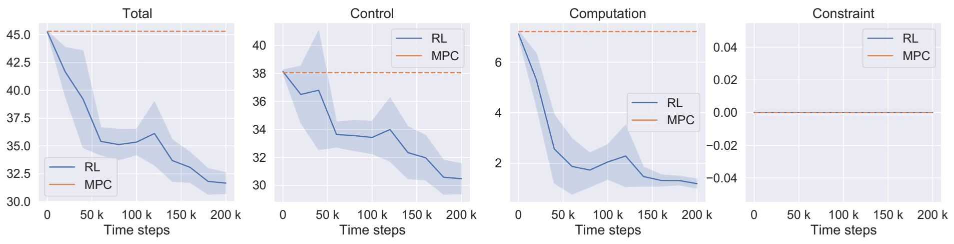


Figure 4.2: Learning curves for the training phase of the RL recomputation policy showing mean values and standard deviation (shaded region) over five different seeds. The graphs correspond to the (dimensionless) components of the cost function (4.4).

Figure 4.2 shows the evolution of the learning process of the RL recomputation policy evaluated every 20,000 time steps on the test set. It starts out emulating the MPC strategy achieving similar objective value scores, and from there identifies instances where similar control performance (or better) can be achieved without recomputing the MPC solution. It converges after around 200,000 time steps corresponding to about 2 hours and 20 minutes of real-world data collection time for the inverted pendulum system. For this system, we ensured feasibility for the dual mode control law even with the lowest recomputation frequency by tuning the parameters of the MPC, and as such the RL policy always respect the constraints even during the learning process. See Section 2.1.3 for a discussion about guaranteeing feasibility.

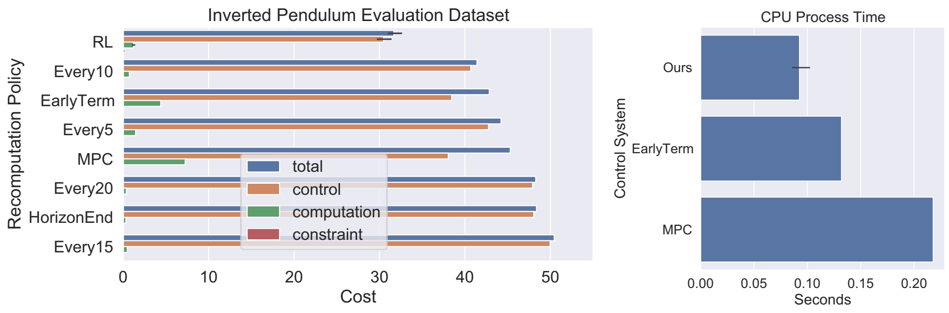


Figure 4.3: Left) Undiscounted return G over the test set for the different recomputation policies, where each bar corresponds to the components of the cost function (4.4) (note that none of the models violated any constraints). The bars for RL show mean values with the lines representing one standard deviation, over the five initialization seeds. Right) Total time spent calculating all the control signals of an episode, averaged over the episodes in the evaluation set.

The trained RL policy is compared to the baseline controllers in Figure 4.3. It outperforms the standard MPC approach by $\sim$30% on the overall objective. A noteworthy result is that the improvement stems not only from reductions in computation, but also a sizeable improvement in the control objective ($\sim$20%), being the only model to outperform the MPC standard paradigm (computed every step) in this regard. The RL policy is able to uncover some synergistic relationship between the MPC and LQR laws and realize a control algorithm that exceeds the performance of either control law alone. As discussed in Section 4.2.3, we initialize the control algorithm to a "best guess" of the optimal control law, and any improvements found by RL are therefore non-trivial to explain. We do however observe that the LQR appears slightly more successful than the MPC at rejecting the process noise added to the pendulum's angular velocity, thus applying the LQR in stable conditions would result in some control performance improvement. Figure 4.3 shows that there is no clear relationship between the control performance and the frequency of recomputation, which suggests that RL has learned to apply the recomputation strategically, rather than just some periodic function. With the early termination MPC, computational complexity is nearly halved while the control performance is only reduced by a few percent. This variant of the MPC still has considerably higher computational complexity than the control algorithm we propose, while losing rather than gaining control performance.

The right graph of Figure 4.3 shows the CPU process time to calculate the control signals of the standard MPC approach, the early termination MPC, and control algorithm we propose, summed over every step in an episode and then averaged over the episodes in the evaluation test set. For our approach, this involves the calculation of the recomputation policy as well as the calculations of the MPC and LQR control laws as selected by the policy. While this adds some overhead, it is small compared to the MPC computation time, and the reductions in processing time are therefore considerable at about $\sim 60\%$. The algorithm we propose would therefore significantly reduce the energy usage of the control system, and free up resources for other components of the embedded system. The memory requirements are also significantly reduced between MPC computations, but it is not clear how much utility this intermittently freed memory has in an embedded context.

One major effect we do not account for in this work is the time it takes to obtain the MPC solution, which is often significant compared to the sampling time. The recomputation policy could therefore instead be trained as a self-triggering policy predicting the number of time steps the current solution is viable for. The MPC could then be scheduled so that a new solution is ready when needed, while also giving the MPC more optimization time resulting in a higher quality solution.

4.5 Conclusion

In this chapter we have presented a novel three-part control algorithm architecture involving a dual mode event-triggered MPC and LQR control law, and an RL recomputation policy. The learned recomputation policy monitors the state of the system, and dynamically decides on when to compute the MPC. We show that optimization of the recomputation meta-parameter can provide significant improvements, both in terms of a decrease in computational expenditure and in terms of an increase in control performance. This could open up new applications for the MPC scheme, particularly for battery-driven systems and other applications where energy is a limiting factor. The role of RL in the proposed control system could be extended to act as the simple controller itself, or to co-optimize the two other controllers by e.g. tuning parameters, as is done in Chapter 6.

5

Reinforcement Learning of the Prediction Horizon in Model Predictive Control

In this chapter we continue the investigation into optimizing the meta-parameters of the MPC scheme, this time focusing on the prediction horizon meta-parameter. The prediction horizon is the single most impactful parameter wrt. the computational complexity of the MPC, and it therefore warrants a thorough examination. In this work, we take a different approach to modelling the prediction horizon parameter than the approach in Chapter 6, and use the valued-based off-policy RL algorithm SAC rather than the on-policy policy-gradient PPO algorithm used in Chapters 4 and 6. This provides better data efficiency and higher adaptivity of the horizon selector to the state of the system. Note that we also submit the subscript indicating the tunable parameters of the MPC controller in this chapter. This chapter is based on the following article:

- [38] Eivind Bøhn, Sebastien Gros, Signe Moe, and Tor Arne Johansen. Reinforcement learning of the prediction horizon in model predictive control. *IFAC-PapersOnLine*, 54(6):314–320, 2021. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2021.08.563. 7th IFAC Conference on Nonlinear Model Predictive Control NMPC 2021

5.1 Introduction

The prediction horizon length is a key parameter of the MPC framework. In conjunction with the step size it controls how far into the future the controller evaluates the consequences of its actions. If chosen too short, the computed trajectories are myopic in nature and might lead to instability and poor approximations of the infinite horizon solution, while the computational complexity grows at best linearly with increasing prediction horizon. Moreover, different regions of the state space might have varying requirements on the horizon length for stability and to find nearly optimal trajectories. This observation motivated the AHMPC scheme, and several implementations of AHMPC have been suggested in the literature.

The AHMPC formulation was pioneered in [136], where the horizon is adapted so that a terminal constraint is satisfied and the system enters a known region of attraction of a second terminal controller. [105] proposes a heuristics-based approach, presenting one ideal but not implementable approach, and one practical method using iterative deepening search where stability criteria are checked on each iteration to determine the lowest stabilizing horizon. A more direct approach is presented in [170] where the prediction horizon is included as a decision variable of the MPC scheme.

When it comes to learning-based approaches to AHMPC, [66] is the only work we could identify in the literature. Their approach is based on supervised learning, and operates by first generating a rich dataset of numerous combinations of states and MPC computations with varying horizons to obtain the cost objective corresponding to a certain horizon in a given state, and then train a neural network optimal horizon predictor in a supervised manner on this dataset. Of note are also the works in [15, 150, 61], who all use derivative-free optimization to tune the prediction horizon of the MPC scheme. However, these methods tune a single global horizon as opposed to an adaptive horizon as is the case for our method. In [15] and [150] the authors tackle the problem of identifying control-oriented models that maximize closed-loop performance (as opposed to system identification methods that aim at maximal prediction accuracy of the identified model), and employ Bayesian optimization on data gathered in closed-loop experiments to update the selected optimization parameters (including the prediction horizon) of the control algorithm in order to improve closed-loop performance. Finally, [61] is a follow-up work of [150] considering the effect of different hardware architectures on the optimization procedure. Note that as is the case for the methods presented in this thesis, none of the learning-based approaches described above treat the issue of guaranteeing the closed-loop stability of the system.

In this chapter, we propose to learn the optimal prediction horizon length of the MPC scheme as a function of the state using RL. To the best of our knowledge, this is the first work to employ RL for AHMPC. The contribution of this chapter lies in exploring how the RL problem of optimizing the MPC prediction horizon can be formulated and showcasing its effectiveness on two control problems. Further, we suggest jointly learning the MPC value function as described in Section 2.2 due to its synergistic relationship with the prediction horizon, enhancing the adaptive capabilities (note that we do not claim this to a be a novel proposal). While the non-learning-based AHMPC approaches described earlier can be designed with favourable properties such as theoretical stability guarantees, they often assume access to privileged information such as terminal sets and control Lyapunov functions. Learning approaches on the other hand typically assume little is known, and as such are applicable to more problems.

5.2 Learning the Prediction Horizon using SAC

5.2.1 Horizon Policy

We learn a policy π_θ^N to output the prediction horizon N of the MPC scheme using SAC. At every time step, the state of the system is measured and the policy then outputs a prediction horizon. The MPC problem (2.3)-(2.6) is solved using this prediction horizon, and the first element of its control sequence is applied to the system and the process is repeated. The prediction horizon is a positive integer, that for convenience we choose to upper bound. As such we modify the output of the SAC policy by linearly scaling the output from the $\tanh$'s limits of -1 and 1, to 1 and N_{max}, and then round the output to the closest integer:

$$a_t = \text{round}\left(\text{scale}\left(\pi_\theta^N(s_t), [-1, 1], [1, N_{\text{max}}]\right)\right) \tag{5.1}$$

The gradients of the RL problem are not affected by these transformations as the transformations are applied in the environment, while the gradients are calculated based on the unscaled and unrounded outputs from $\pi_\theta^N(s_t)$. This does however mean that the agent must "learn" that similar outputs from the policy will be rounded to the same action in (5.1), and thus lead to the same subsequent state and cost. We considered alternative ways of formulating the policy as a discrete distribution from which integer horizon lengths could be drawn directly, such as N-head NNs, Poisson models, and negative binomial models, but settled on the described rounding approach due to its simplicity and favourable results.

The cost function of the horizon policy consists of a control performance cost R_P, i.e. the MPC stage cost ℓ, a constraint violation cost R_C, and a computation cost R_N to encourage lower horizons when suitable:

$$R(s, a) = R_P(s') + \lambda_C(T - t)R_C(s') + \lambda_N R_N(a) \tag{5.2}$$

where $\lambda_C, \quad \lambda_N$ are weighting factors, and $s' = \mathcal{T}(s, a)$. $R_C(s)$ is a binary variable indicating whether a hard constraint of the problem was violated — upon which the episode is ended — and $T - t$ is the number of steps left in the episode such that the agent receives a penalty proportional to how early the episode is ended. We assume the computational complexity of the MPC scheme grows linearly in the horizon length, i.e. $R_N(a) = a$, as a lower bound for the true complexity. This generally holds true for the interior point method we use in the MPC scheme if one assumes local convergence and an initial guess that is reasonable [154].

The RL state space $\mathcal{S} = \{x, \hat{p}\}$ consists of the MPC state vector x and the time-varying exogenous input variables $\hat{p}$, as these are necessary to ensure the Markov property.

5.2.2 MPC Value Function

The MPC's value function is trained jointly with the RL horizon policy to minimize the MSBE as described in Section 2.2, using 32-step bootstrapping. We found that n-step learning provided sufficient stabilization such that other common techniques in value estimation such as target networks and multiple estimators were not needed [63]. We experimented with two types of approximators: two-layer fully connected NNs and polynomial regression models. We found that for the problems considered in this chapter, quadratic polynomial models achieved similar prediction accuracy as higher-order polynomial models as well as the NNs. We therefore employ a quadratic polynomial model as the value function estimator due to its convexity, which contributes significantly less to the computational complexity of the MPC scheme compared to the other models that were considered.

5.2.3 Evaluation

Since the environments are randomized we construct a test set consisting of 10 episodes for which all stochastic variables such as state initial conditions and references are drawn in advance and thereby fixed for all policies, ensuring a fair comparison. The learned horizon policy is compared against the standard MPC scheme with a range of fixed horizons, to assess the contribution of the learning. Each fixed horizon MPC also has its own value function estimated using a dataset of 15k time steps.

5.3 Experiments

We illustrate our approach on two systems. We set $N_{\text{max}} = 50$ in (5.1), $\lambda_N = 3 \cdot 10^{-3}, 1 \cdot 10^{-3}$ and $\lambda_C = 10, 2$ in (5.2) for the inverted pendulum and collision avoidance systems, respectively. We use the hyperparameters suggested in the SAC paper [85], with the following exceptions: π_θ^N is a 2-layer fully connected NN with 32 nodes in each layer, a reward scaling (i.e. relative weighting of entropy vs reward objective in SAC) of $\chi = 0.6$ for the inverted pendulum system and $\chi = 0.3$ for the collision avoidance system, and $\gamma = \rho = 0.97$ discounting factors for the MPC scheme and the RL algorithm, such that the controller values short term rewards comparatively high in relation to long term rewards, and an effective optimization horizon of $\sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-0.97} \approx 33$ time steps for the RL horizon policy.

5.3.1 Inverted Pendulum

The first system we experiment on is the classic control problem of stabilizing an inverted pendulum mounted on a cart that is fixed on a track, so that the cart can only move back and forth in one dimension. The cart's position is constrained to the size of the track and the pendulum angle is constrained to be above perpendicular to the surface. The controller should also track a time-varying position reference. As the position of the cart and stabilization of the pendulum are intricately linked, respecting both of the constraints while tracking the position reference requires a fairly high optimization horizon. Each episode is terminated after a maximum of 100 time steps, or when a constraint is violated.

The state space consists of the states $x = [\psi, v, \beta, \omega]$, where ψ and v is position and velocity of the cart along the horizontal axis, while β and ω is the angle to the upright position and the angular velocity of the pendulum. The system dynamics are described by equations (5.3)-(5.6), where $m = 0.2$ and $M = 0.8$ are the mass of the pendulum and total mass of cart and pendulum, and $l = 0.25$ is the length of the pendulum. For the MPC model the dynamics are discretized with a step time of $\Delta_t = 0.04s$.

The stage cost $\ell(x_t, u_t, \hat{p}_t) = E_{\text{kinetic}} - E_{\text{potential}} + 10 \cdot (\psi_t - \psi_{t,r})^2 + 0.1u_t^2$ reflects the objective of stabilizing the pendulum in the up position, formulated through minimizing the negative potential energy of the system, while tracking the position reference $\psi_{k,r}$.

$$\dot{\psi} = v \tag{5.3}$$

$$\dot{v} = \frac{mgsin(\beta)cos(\beta) - \frac{4}{3}(u + ml\omega^2 sin(\beta))}{mcos^2(\beta) - \frac{4}{3}M} \tag{5.4}$$

$$\dot{\beta} = \omega \tag{5.5}$$

$$\dot{\omega} = \frac{Mgsin(\beta) - cos(\beta)(u + ml\omega^2 sin(\beta))}{\frac{4}{3}Ml - mlcos^2(\beta)} \tag{5.6}$$

$$-5 \leq u \leq 5, \;\; -1.5 \leq \psi \leq 1.5, \;\; -90^\circ \leq \beta \leq 90^\circ \tag{5.7}$$

5.3.2 Collision Avoidance

The second system we consider is a reference tracking problem, in which a vehicle is controlled to follow a trajectory τ where obstacles are placed in the path that needs to be avoided. The MPC receives information about the reference trajectory as well as any obstacles in its vicinity (i.e. reachable at the maximum velocity within the prediction horizon), however the position of the obstacles grows more uncertain the farther away the obstacle is. This means longer horizons considers increasingly uncertain information, and a short or medium horizon might be more suited in some situations. The episode is ended when reaching the endpoint of the trajectory, when colliding with an obstacle, or after a maximum of 150 time steps.

For the vehicle we employ a unicycle model (5.8)-(5.10) where the MPC provides a forward velocity u_s as well as an angular velocity u_ω to turn the vehicle. The MPC model is discretized with a step time of $\Delta_t = 0.1s$. The positions and sizes of the obstacles are randomly generated at the beginning of every episode, and their projected positions supplied to the MPC are randomly drawn within a two-dimensional cone originating from the vehicle, such that the uncertainty grows the further away the object is from the vehicle. An episode is illustrated in Figure 5.1.

$$\dot{\psi}_x = u_s \cos(\beta) \tag{5.8}$$

$$\dot{\psi}_y = u_s \sin(\beta) \tag{5.9}$$

$$\dot{\beta} = u_\omega \tag{5.10}$$

$$0 \leq u_s \leq 5, -4 \leq u_\omega \leq 4 \tag{5.11}$$

The stage cost is defined as $\ell(x_t, u_t, \hat{p}_t) = ||\psi_t - \tau_t||_2^2$ where $\psi_t = [\psi_x, \psi_y]^\top$ and τ_t is the vehicle position and trajectory reference at time t. Further, soft constraints

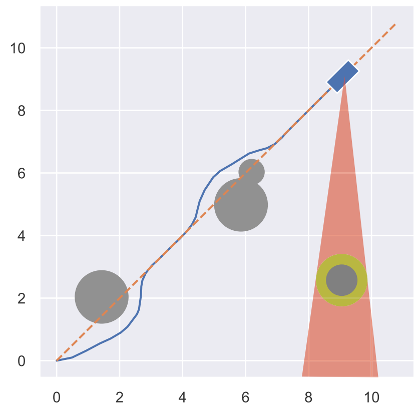


Figure 5.1: An episode in the collision avoidance environment. The blue rectangle represents the vehicle tracking the trajectory (the orange dashed line) from left to right, while avoiding the grey obstacles. The sensor beam's inaccuracy grows with distance, such that the projected position of the object in the beam is drawn from the yellow circle. The vehicles' size is enlarged for visual clarity.

with slack variables are added around each obstacle with 150% of the obstacles radius.

5.4 Results

The standard MPC scheme with various prediction horizons and the learned RL AHMPC are compared in Figure 5.3, where all models employ their corresponding value function estimator $\hat{V}^{\mathrm{M}}_{\theta\mathrm{V}}$. The RL policy outperforms the standard MPC scheme for all horizons lengths, improving on the second-best achieving policy by about 4% and 8% for the inverted pendulum and collision avoidance systems, respectively. The improvement is more significant for the latter system as the performance objective varies more with the prediction horizons, and the RL policy is able to identify when to use long and short horizons. For the inverted pendulum system, all horizons capable of respecting the constraints achieve similar performance costs, and as such, the difference lies mainly in the computation term, although the RL policy achieves the lowest performance cost here as well. RL's ability to find improvement in a problem with such a noisy cost landscape and with such little potential improvement speaks to its strength. Moreover, the gains from reducing computation would be greater when using e.g. active set methods for the MPC scheme which typically yields quadratic growth in computational complexity

[108].

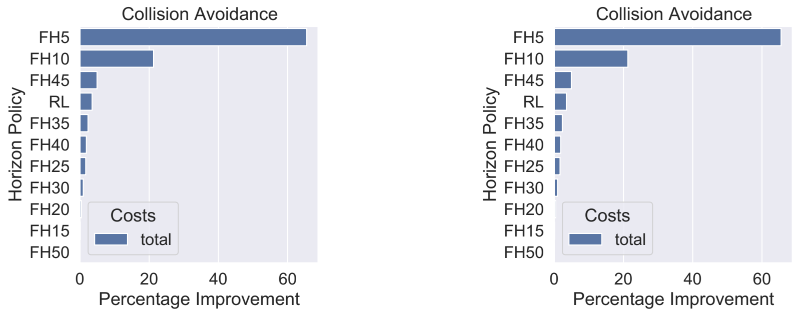
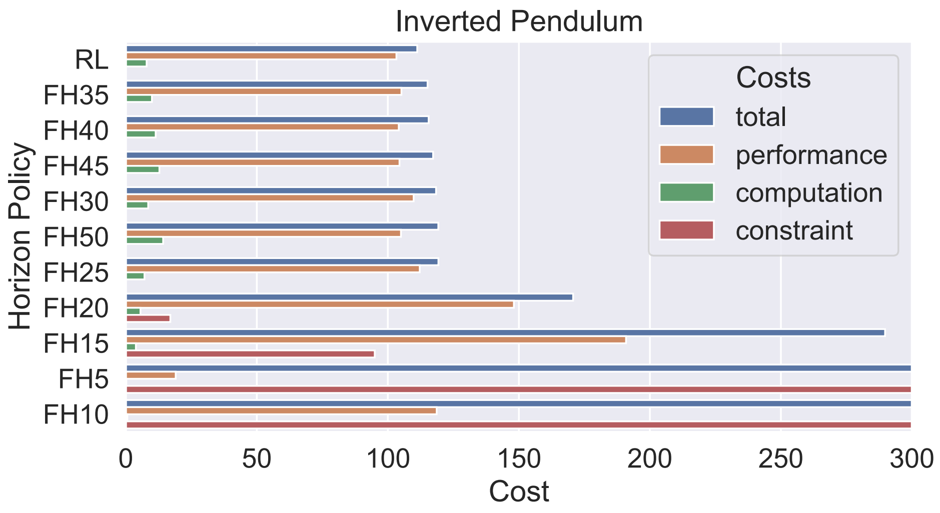


Figure 5.2: Total cost improvement over the test sets with the value function as the terminal cost in the MPC scheme.

In the collision avoidance environment, the best performing fixed horizon is the short 10 step horizon. With the shortest 5 step horizon, the MPC is unable to navigate around all the obstacles, preferring to stay still in front of large obstacles, although the addition of the value function mitigates this issue to some extent. Longer prediction horizons allow the MPC to recognize that sometimes the long way around the closest obstacle yields a shorter total path due to other obstacle locations, but its planned routes are more sensitive to the uncertainty in the projected locations. A robust MPC scheme could alleviate this deficiency, however the RL policy is also able to recognize this issue and leverage the strengths of both short and long horizons.

We found that implementing value function estimation in the MPC scheme could significantly improve the performance when using horizons in a neighbourhood of the horizon spectrum where performance changes abruptly, as illustrated in Figure 5.2, which shows the percentage improvement for each policy when including $\hat{V}^{\mathrm{M}}_{\theta^{\mathrm{V}}}$ as the terminal cost. The RL horizon policy does not benefit as much from the value function as we would expect, even being the best performing policy when removing the value function from it but not from the fixed horizon MPCs. The benefit would probably be more significant in problems that are more temporally or spatially complex. In the collision avoidance problem, the shortest horizons show the largest improvements, while for the inverted pendulum system the most improved horizons are the ones that lie close to the apparent minimum horizon required to successfully stabilize the pendulum and track the position reference. For both systems, the longer horizons benefit less from the addition of the value function. This is in part due to the fact that both these systems are heavily influenced by future information that is not available to the value function estimator, i.e. accurate information about distant obstacles and the future position reference



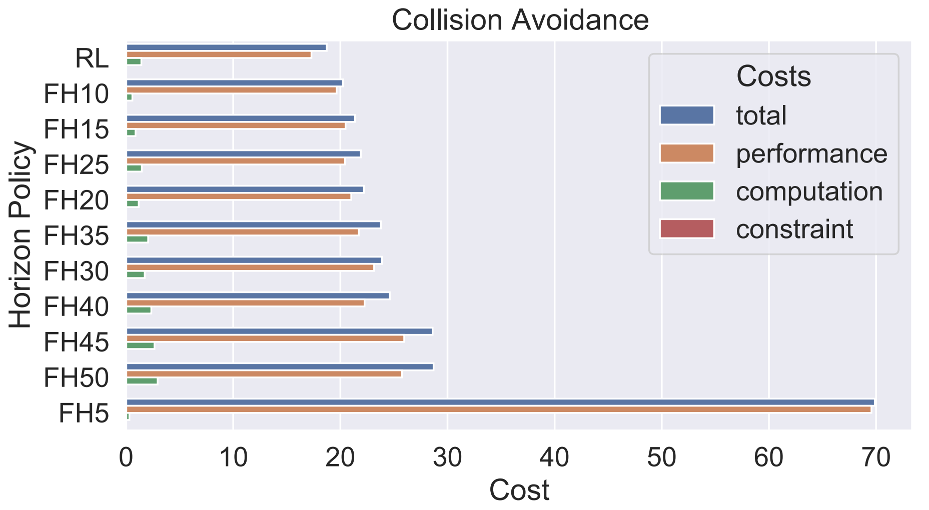


Figure 5.3: Mean episode costs for the horizon policies on the test sets for the two systems using the value function as the terminal cost, where lower is better. The objectives are connected in that the policy does not accrue performance or computation cost after the episode is terminated from a constraint violation. The top figure is cut off due to the worst-performing policies having a significantly higher cost.

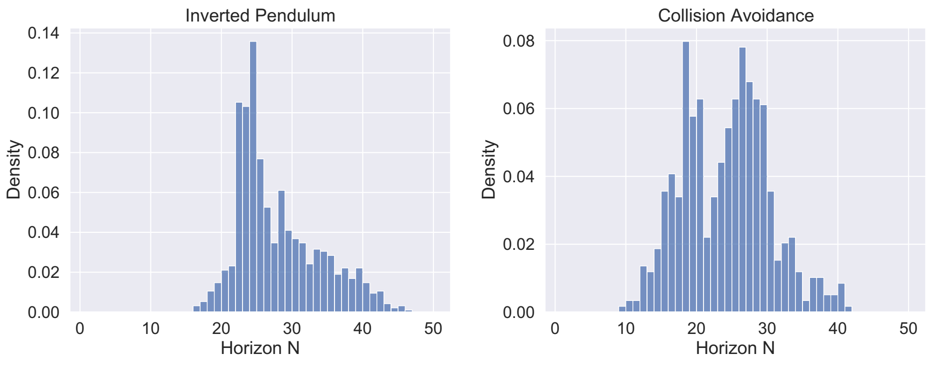


Figure 5.4: The distribution of the horizons selected by the RL policy over the two test sets.

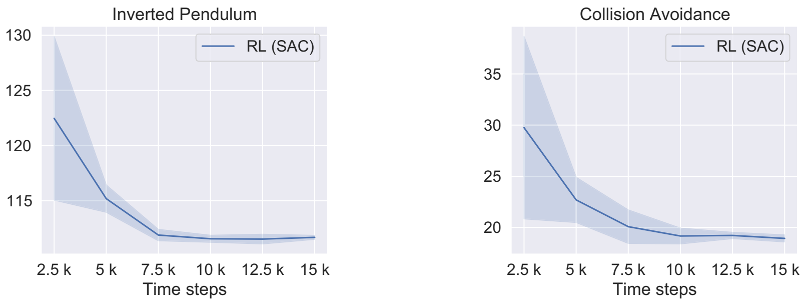


Figure 5.5: Total cost on the test set for the RL policy at different stages of the learning process. The solid line is the mean score while the shaded region is one standard deviation over three initialization seeds.

for the cart.

We note that the performance costs and the value function improvement is not monotonic wrt. the horizon length. This could partly be explained by the randomness in the data collection stage for the value function estimation.

The distribution of the horizons selected by the RL policy over the evaluation test set is shown in Figure 5.4. The policy selects a wide range of horizons and has a floor that in both cases largely agrees with the minimum horizons that achieve the highest control performance in Figure 5.3. As discussed in Section 2.1.3, this shows that RL is able to learn the range of stabilizing horizons. One interesting thing to note is that in the collision avoidance environment it displays bi-modal behaviour, presumably corresponding to the short-long horizon dichotomy discussed in the preceding paragraph. The mean, median and mode horizon are 28, 26, and 23, and 24.2, 25, and 26 for the inverted pendulum and collision avoidance environments, respectively.

Figure 5.5 shows the progression of the training process of the RL horizon policy. It learns quickly, converging after around 15 thousand time steps for both systems. Considering the sampling time of 0.04s and 0.1s for the inverted pendulum and collision avoidance systems, this corresponds to about 10 and 25 minutes of running the systems in real-world time to collect data. Moreover, we find that the RL horizon policy itself converges even faster and that the value function estimation is the slower, less data efficient component. From these results, it seems evident that RL is able to cope well with the rounding described in Section 5.2.1.

5.5 Conclusion

This chapter has shown that RL can be used to automatically tune and adapt the prediction horizon of the MPC scheme online with only minutes of data collection, at least for simple systems. An important further work is to investigate how this affects the stability properties of the MPC framework, and if any guarantees can be given.

The method presented in this chapter is quite significantly different from the method to tune the prediction horizon that is presented in Chapter 6, and was showcased on different systems. First, it uses the off-policy algorithm SAC as opposed to the on-policy algorithm PPO that is used in Chapter 6. Off-policy methods are generally more data-efficient, and we observe as expected that the data requirement of the method in this chapter is about an order of a magnitude lower than that of the method in Chapter 6. Moreover, the horizon variable is in this chapter modelled as a Gaussian random variable that is then discretized into the integer values accepted by the MPC scheme, instead of it being directly modelled as a discrete variable as is the case in Chapter 6. While we empirically demonstrate that both approaches work, in the sense that they both produce optimizing behaviour, it is not clear how the discretization of the continuous horizon variable (which is not communicated explicitly to the RL algorithm) affects the optimization procedure, and the discrete horizon variable formulation therefore constitutes a more technically correct approach. As discussed in Section 6.5, even the most fitting discrete formulation we could identify (i.e. the GP-2 variant of the generalized Poisson distribution (GPD)) had its issues in terms of high minimum variance, necessitating variance reduction techniques such as frame-skip. With the caveat that the control problems of Chapters 5 and 6 are different and it is therefore unclear how comparable the resulting behaviour of the two methods are, the entropy-maximizing nature of SAC seems to lead to the horizon policy's output being more state-dependent, even exhibiting multi-modality in the collision-avoidance experiment where this was beneficial. The PPO based method on the other hand tended towards employing a small range of horizons that worked well over a large region of the state space.

6

A Complete Framework for Optimization of Model Predictive Control with Reinforcement Learning

In this chapter, we conclude our research efforts into the optimization of meta-parameters of the MPC scheme using RL, and develop a framework in which these meta-parameters as well as any other parameters of the controllers can be jointly optimized. This chapter is based on the following article:

- [28] Eivind Bøhn, Sebastien Gros, Signe Moe, and Tor Arne Johansen. Optimization of the model predictive control meta-parameters through reinforcement learning. *IEEE Transactions on Cybernetics*, 2021. Submitted

6.1 Introduction

Learning can be an important tool in assisting the tuning process of the MPC scheme. In this chapter, we propose the novel idea of tuning the meta-parameters of the MPC scheme using RL. By meta-parameters we mean parameters that affect the structure of the OCP and under what conditions it is solved, rather than parameters that affect the solution to a given OCP, the tuning of which has previously been demonstrated in the literature [72, 52]. This work extends upon the methods presented in the two preceding chapters. Here, we propose a unified framework in which these meta-parameters, as well as any other parameters of the control algorithm, are jointly optimized to simultaneously maximize the control performance and reduce the computational complexity in a configurable manner. The control algorithm we propose consists of a recomputation policy that decides when the MPC solution should be computed, a state-feedback controller (i.e. the LQR) that is applied on the predicted state trajectory produced by the MPC in between MPC computations, and an RL algorithm that incorporates the parameters of these controllers and optimizes them according to the specified objective. Although we can enforce input constraints, there is no mechanism to ensure that the state constraints hold in our control algorithm, and the constraint satisfaction properties will be determined by the behaviour identified as optimal through RL. While we demonstrated in the preceding chapters the effectiveness of learning these meta-parameters in isolation, it is clear that the questions of *when* and *how* (that is, wrt. its tunable parameters) to compute the control algorithm are related, and treating them separately fails to consider the interactions and indirect effects at play.

Introducing learning for the LQR problem has previously been suggested in the literature [158, 179, 126, 54]. What separates our method from the aforementioned works, is that in the control algorithm we propose, the dynamics matrices stem from the MPC controller, and have a clear purpose in making the LQR compatible with the MPC scheme. Since we therefore can assume that these matrices are known and fixed, we can take advantage of the structure in the LQR problem in terms of the Riccati equation, and calculate the gradients of the weighting matrices through the Riccati equation. Introducing structure simplifies a learning problem by restricting the solution space that is searched.

The contributions of this chapter can be summarized as:

1. We propose the novel idea of optimizing the meta-parameters of the MPC scheme using RL.
2. We develop a novel MPC formulation in which the performance and computational power usage are jointly optimized in a configurable, automated manner, which could open up new applications for MPC.
3. To realize the proposed algorithm, we employ novel use of mixture distributions for RL.

6.2 Control Algorithm

The control algorithm we propose in this chapter consists of an event-triggered adaptive-horizon nonlinear discrete-time MPC, denoted $\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}$, and a discrete-time time-varying LQR, denoted $\pi^{\mathrm{L}}_{\theta^{\mathrm{L}}}$ (see Chapter 2 for more details). The MPC is computed whenever the recomputation policy $\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}$ signals to compute, with a prediction horizon as decided by the horizon policy $\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}$. Due to the adaptive prediction horizon, we propose to use a value function as the terminal cost of the MPC scheme as described in Section 2.2. Finally, note that while we assume state-feedback for the MPC for simplicity, the control algorithm could easily be extended with an estimator such as the moving horizon estimator which can be tuned in unison with the MPC [143].

The LQR [25] is a state-feedback controller that arises as the optimal solution to unconstrained control problems where the dynamics are linear, and the cost is quadratic. The role of the LQR in our control algorithm is to act as the linear feedback correction of the MPC, that can be applied to compensate for errors in the open loop predicted state trajectory between MPC recomputations. To accomplish this, we employ a time-varying LQR where the $\mathbf{A}_t$ and $\mathbf{B}_t$ matrices are obtained from the MPC scheme as the linearization of the MPC model each time it is computed:

$$\mathbf{A}_{i+1:i+N_i}, \mathbf{B}_{i+1:i+N_i} = \text{linearize}(\hat{f}_{\theta^{\mathrm{M}}})\big|_{x_{i+1:i+N_i}, u_{i:i+N_i-1}} \tag{6.1}$$

$$\mathbf{A}, \mathbf{B} = \text{linearize}(\hat{f}_{\theta^{\mathrm{M}}})\big|_{x_t^s, u_t^s} \tag{6.2}$$

After the horizon end (i.e. when $t > i + N_i$ where i is the time of last MPC computation), we set the LQR dynamics matrices as the time-invariant matrices corresponding to the steady-state (equilibrium) of the system x_t^s, u_t^s (6.2). The specific linearization procedure depends on the implementation of the dynamics model in the MPC, i.e. discrete vs continuous model and the accompanying discretization schemes. The $\mathbf{Q}$ and $\mathbf{R}$ cost-weighting matrices are initialized from the MPC objective as follows, and then tuned further as described in Section 6.3.

$$\mathbf{Q}_{\text{init}} \leftarrow \frac{\partial^2 \ell_{\theta^{\mathrm{M}}}(x, u, \hat{p})}{\partial x^2}\Big|_{x_0^s, u_0^s}, \quad \mathbf{R}_{\text{init}} \leftarrow \frac{\partial^2 \ell_{\theta^{\mathrm{M}}}(x, u, \hat{p})}{\partial u^2}\Big|_{x_0^s, u_0^s} \tag{6.3}$$

See Section 2.3 for a formal statement of the LQR control problem and its implementation.

6.3 A Reinforcement Learning Parameter and Meta-Parameter Optimization Framework for Model Predictive Control

6.3.1 A State Space with the Markov Property

For RL theory to hold for the control system we wish to optimize, the state space needs to have the Markov property, i.e. future states should not depend upon past states given the current state. This section outlines such a Markovian state.

Consider the input sequence $u^{\mathrm{M}}_{i:i+N_i-1}$ and the predicted state trajectory $\hat{x}_{i+1:i+N_i}$ computed by the MPC at time i. As discussed in Section 2.1, a variable number $n \in \{0, 1, \dots, N_i - 1\}$ of these inputs are applied to the plant. Since n is not known a priori, the state vector x is not a sufficient state representation to yield the Markov property for the control system consisting of the recomputation policy, the horizon policy, and the MPC and LQR control laws. We define an augmented state s in (6.4) that contains the current plant state and exogenous variables, labeled $\bar{x}_t$ and $\hat{p}_t$, the state of the system, exogenous variables, and prediction horizon used when the last MPC computation took place, labeled $\bar{x}_i$, $\hat{p}_i$, and N_i, as well as the number of time steps since the MPC computation, $t - i$

$$s_t = [\bar{x}_i, \hat{p}_i, N_i, \bar{x}_t, \hat{p}_t, t - i]^\top \tag{6.4}$$

where $\bar{x}_t$ has evolved from $\bar{x}_i$ according to the real system dynamics. When the MPC problem is recomputed, the deterministic transition $i \leftarrow t, \bar{x}_i \leftarrow \bar{x}_t, \hat{p}_i \leftarrow \hat{p}_t, N_i \leftarrow N_t$ takes place. The MPC and MPC plus LQR control laws deployed on the plant is then (6.5) and (6.6), respectively, while the total control system is defined as (6.7), where the $\mathrm{proj}_{h_{\theta^{\mathrm{M}}}}$ operator projects the control input onto the constraint vector $h_{\theta^{\mathrm{M}}}$ (2.6):

$$\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(s_t) = \begin{cases} u^{\mathrm{M}}_{t-i}(\bar{x}_i, \hat{p}_i, N_i) & \text{, if } t - i < N_i \\ 0 & \text{, otherwise} \end{cases} \tag{6.5}$$

$$\pi^{\mathrm{ML}}_{\theta^{\mathrm{M,L}}}(s_t) = \begin{cases} u^{\mathrm{M}}_{t-i}(\bar{x}_i, \hat{p}_i, N_i) + u^{\mathrm{L}}_t(\hat{x}_{t-i} - \bar{x}_t), & \text{if } t - i < N_i \\ u^{\mathrm{L}}_t(x^s_t - \bar{x}_t) & \text{, if } t - i \geq N_i \end{cases} \tag{6.6}$$

$$\pi^{\mathrm{CS}}_{\theta^{\mathrm{M,L}}}(s_t) = \mathrm{proj}_{h_{\theta^{\mathrm{M}}}}\left(\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(s_t) + \pi^{\mathrm{ML}}_{\theta^{\mathrm{M,L}}}(s_t)\right) \tag{6.7}$$

and where x_t^s is the steady-state equilibrium of the system at time t, which is time-variant in the case of time-varying references.

Assumption 1. *The time-varying exogenous input variables $\hat{p}_t$ are generated by a Markovian process.*

Proposition 1. *The state s has the Markov property, i.e.* $P(s_{t+1}|s_t) = P(s_{t+1}|s_{0:t})$

Proof. First, note that by definition from (2.1) x is Markovian given u

$$x_{t+1} = f(x_t, u_t) \tag{6.8}$$

$$\Rightarrow P(x_{t+1}|x_t, u_t) = P(x_{t+1}|x_{0:t}, u_{0:t}) \tag{6.9}$$

However, the control law $\pi^{\mathrm{CS}}_{\theta^{\mathrm{M,L}}}$ (6.7) that determines u_t consists of $\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}$ (6.5) — which depends on the state and exogenous variables $\bar{x}_i, \hat{p}_i$ and the prediction horizon N_i at the last MPC computation — and $\pi^{\mathrm{M,L}}_{\theta^{\mathrm{ML}}}$ (6.6), which depends on the current state x_t and the $(t-i)^{\mathrm{th}}$ element of the MPC's predicted state trajectory, and x_t^s which depends on $\hat{p}_t$. Therefore, with s as defined in (6.4) we have:

$$P(u_t|\pi^{\mathrm{CS}}_{\theta^{\mathrm{M,L}}}, s_t) = P(u_t|\pi^{\mathrm{CS}}_{\theta^{\mathrm{M,L}}}, s_{0:t}) \tag{6.10}$$

Finally, note that the MPC input sequence $u^{\mathrm{M}}_{i+1:i+N_i-1}$ and the predicted state trajectory $\hat{x}_{i:i+N_i}$ follows from $\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}$ and its arguments (which are all known), and as such does not need to be contained in s for s to be Markovian:

$$\Rightarrow P(s_{t+1}|s_t) = P(s_{t+1}|s_{0:t}) \tag{6.11}$$

□

6.3.2 Policies

We use the notation $\pi_\theta(s)$ to label a deterministic function of the state s that is parameterized by θ, and $\pi_\theta(\cdot|s)$ to label the stochastic version of the same function. The notation $\tilde{\cdot}$ signifies that the value is drawn from the policy's probability distribution. The implementation of the PPO algorithm employed in this chapter requires the log probability of the policies, and as such we will list the probability distributions, denoted P, and corresponding log probabilities, denoted $\log P$, for the policies we optimize. For brevity, we omit listing input-independent conditional variables as arguments (e.g. covariance) of the distributions.

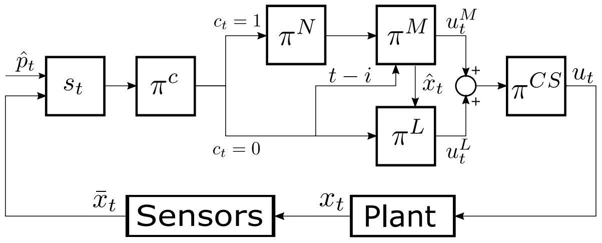


Figure 6.1: An overview of the control algorithm. Not shown here is the connection of each policy's output to the RL algorithm that updates the policy's parameters.

The Recomputation Policy

The recomputation policy $\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}$ decides on each step whether the MPC problem should be recomputed, or if the previously computed solution is still acceptable. In other words, it is a binary variable that chooses among the two different options: recompute or not. As such, we model it as a Bernoulli-distributed random variable, where the policy outputs the logit of the probability that the OCP should be recomputed (6.12), from which we can deduce the probability of not recomputing. We label the output of the recomputation policy $c \in \{0, 1\}$. The Bernoulli distribution has the probability mass function (PMF) (6.13).

$$w = \frac{1}{1 + \exp\left(-\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(s)\right)} \tag{6.12}$$

$$P^{\mathrm{c}}(c|s) = \begin{cases} 1 - w & , \text{if } c = 0 \\ w & , \text{if } c = 1 \end{cases} \tag{6.13}$$

The log probability can be expressed as follows:

$$\log P^{\mathrm{c}}(c|s) = c\log(w) + (1 - c)\log(1 - w) \tag{6.14}$$

Finally, the policy gradient of the recomputation policy is:

$$\begin{aligned} \nabla_{\theta^{\mathrm{c}}} \log \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(c|s) &= \nabla_{\theta^{\mathrm{c}}} \left(c\log(w) + (1 - c)\log(1 - w)\right) && (6.15) \\ &= -c\nabla_{\theta^{\mathrm{c}}}\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(s)w + \nabla_{\theta^{\mathrm{c}}}\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(s)(1 - c)\left(-1 - w\right) && (6.16) \\ &= \nabla_{\theta^{\mathrm{c}}}\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(s)\left(c - 1 - w\right) && (6.17) \end{aligned}$$

The Horizon Policy

The MPC prediction horizon N is a positive integer, which we model with the GP-2 variant of the generalized Poisson distribution (GPD) [187]. Other models we considered but decided against includes categorical classification models, as they do not consider the ordinal information of the horizon variable, and the standard Poisson distribution, which we found to be too inflexible due to its mean and variance being equal. The horizon model outputs the rate parameter of the GPD, μ, as a function of s, while the dispersion parameter α is learned as an input-*independent* variable.

$$P^{\mathrm{N}}(N|s) = \left(\frac{\mu}{1+\alpha\mu}\right)^{\mathrm{N}} \frac{(1+\alpha N)^{N-1}}{N!} e^{\left(-\frac{\mu(1+\alpha N)}{1+\alpha\mu}\right)} \tag{6.18}$$

$$\mathbb{E}(N) = \mu, \;\; \mathbb{V}(N) = \mu(1+\alpha\mu)^2 \tag{6.19}$$

In this model, α is restricted according to $1 + \alpha\mu > 0$ and $1 + \alpha N > 0$. To address these constraints, we introduce two hyperparameters $N_{\min}$ and $N_{\max}$ that correspond to the minimum and maximum horizons that the MPC should operate with. To constrain the rate parameter μ we apply the hyperbolic tangent function, denoted $\tanh \in [-1, 1]$, and then linearly scale it to the limits defined by $N_{\min}$ and $N_{\max}$. Finally, the learned α parameter is clipped according to restrictions outlined above

$$\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s) = \mathrm{scale}\left(\tanh(\mu), N_{\min}, N_{\max}\right) \tag{6.20}$$

$$\alpha_{\mathrm{new}} = \max\left(\alpha, -\frac{1}{N_{\max}}\right) \tag{6.21}$$

The log probability of the GPD and the policy gradient of the horizon policy is defined as follows:

$$\begin{aligned}\log P^{\mathrm{N}}(N|s) =& N\log\left(\frac{\mu}{1+\alpha\mu}\right) + (N-1)\log(1+\alpha N) \\ &- \frac{\mu(1+\alpha N)}{1+\alpha\mu} - \log(N!)\end{aligned} \tag{6.22}$$

$$\begin{aligned}\nabla_{\theta^{\mathrm{N}}} \log \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(N|s) =& \nabla_{\theta^{\mathrm{N}}}\Bigg(N(\log(\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s)) - \log(1+\alpha\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s))) \\ &+ \frac{\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(1+\alpha N)}{1+\alpha\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s)}\Bigg)\end{aligned} \tag{6.23}$$

$$\begin{aligned}=& \nabla_{\theta^{\mathrm{N}}}\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s)\Bigg(\frac{N}{\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s)} + \frac{\alpha}{1+\alpha\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s)} \\ &+ \frac{1+\alpha N + \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s)(\alpha(1+N+\alpha N)+1)}{1+2\alpha\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s) + (\alpha\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s))^2}\Bigg)\end{aligned} \tag{6.24}$$

There are several techniques to sample from this distribution [49]. We favor the normal approximation sampling technique for its low run-time complexity: With a sufficiently high rate parameter (i.e. $\mu \gtrapprox 10$), the GPD is approximately normally distributed with mean and variance as given in (6.19):

$$\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(N|s) \approx \mathrm{clip}(\lfloor \mu + \sqrt{\mu(1+\alpha\mu)^2}\zeta + 0.5\rfloor, N_{\min}, N_{\max}), \quad \zeta \sim \mathcal{N}(0,1) \tag{6.25}$$

When optimizing the horizon policy in isolation (that is, not together with the rest of the control system) we find that faster convergence and more stable solutions are obtained by learning on an augmented MDP where every action selected by the policy is repeated $d > 1$ times. Hence the policy only senses every d'th state, and the rewards it receives are the cumulative reward over every step of the control system in the d steps. This technique is known as "frame-skip" in RL and is an effective method to enhance learning for problems with discrete actions, see e.g. learning to play atari-games [137, 31], but also for continuous control [96]. While the exact mechanisms behind the improvements stemming from frame-skipping is not fully understood, it is clear that in certain problems it increases the signal-to-noise ratio of every data sample, which simplifies the credit assignment problem. With high control frequency, the consequences embedded in the rewards of an action are highly dependent on the actions that precede and come after the current action, which when sampling from the policy's action distribution are often dissimilar. Therefore, high control frequency can lead to noisier gradients. When using $d > 1$ during exploration, we use $d = 1$ during exploitation (evaluation) as this increases performance. We find that the horizon policy is not sensitive to the exact value of d, and all values in $d \in [2, T]$ generally accelerate learning. While $d = T$ (that is, only one action per episode) might seem excessive, consider that the standard MPC scheme employs one fixed horizon that is tuned by selecting the horizon that on average performs best overall the operating ranges of the control system. Finally, note that when optimizing the horizon policy and the recomputation policy together, the latter electing to not recompute the MPC will naturally enforce a form of frame-skipping, as the previous MPC solution computed with the previously selected prediction horizon is employed until the MPC is again computed.

The Controller Policies

To ensure sufficient exploration we formulate a stochastic version of the MPC, and the MPC plus LQR control laws (denoted $\pi^{\mathrm{ML}}_{\theta^{\mathrm{M,L}}}(s) = \pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(s) + \pi^{\mathrm{L}}_{\theta^{\mathrm{L}}}(s)$), modeling them as Gaussian random variables (6.26). The mean is then the output of the control laws, and the covariance of each controller is a learned input-*independent* variable of the RL algorithm. To be concise, we use $*$ in place of the superscript for the control laws as in (6.26). We will first develop these policies for the MPC scheme with a given horizon N, where the output is made stochastic as follows:

$$\pi^*_{\theta^*}(u^*|s, N) = \mathcal{N}\left(\pi^*_{\theta^*}(s, N), \Sigma^*\right), \quad * \in \{\mathrm{M}, \mathrm{ML}\} \tag{6.26}$$

$$P^{\mathrm{u}}(u^*|s, N) = \frac{1}{\Sigma^* \sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{u^* - \pi^*_{\theta^*}(s, N)}{\Sigma^*}\right)^2\right) \tag{6.27}$$

Note that the argument N to the policies is redundant (and thus these policies are equivalent to those in Section 6.3.1), as N_i in s will always reflect the latest N that is decided by the horizon policy ahead of control law calculation. We use this argument here to clarify the derivations. The log probability of the distribution and the policy gradient is:

$$\log P^{\mathrm{u}}(u^*|s,N) = -\frac{1}{2{\Sigma^*}^2}\left(u^* - \pi^*_{\theta^*}(s,N)\right)^2 - \frac{1}{2}\log({\Sigma^*}^2) - \frac{1}{2}\log(2\pi) \tag{6.28}$$

$$\nabla_{\theta^*}\log\pi^*_{\theta^*}(u^*|s,N) = {\Sigma^*}^{-1}(\pi^*_{\theta^*}(s,N) - u^*)\nabla_{\theta^*}\pi^*_{\theta^*}(s,N) \tag{6.29}$$

Then, note that the adaptive-horizon MPC control law is a distribution of the MPC schemes over the range of prediction horizons, with weights P^{N} assigned by the horizon policy. We first query the horizon policy for which prediction horizon to employ, and then the solution to the MPC problem (2.3)-(2.6) is computed with the selected horizon

$$\pi^{\mathrm{MN}}_{\theta^{\mathrm{M,N}}}(s) = \pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}\left(x_t, \hat{p}_t, \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(s)\right) \tag{6.30}$$

where the superscript N indicates that this is the adaptive-horizon MPC policy. Using the indicator function

$$\mathbb{1}_A = \begin{cases} 1, & \text{if } A \\ 0, & \text{otherwise} \end{cases} \tag{6.31}$$

we can formulate the probability distributions and log distributions for the stochastic adaptive-horizon control policies

$$P^{\mathrm{Nu}}(u^*|s) = \sum_{N=N_{\min}}^{N_{\max}} P^{\mathrm{N}}(N|s)P^{\mathrm{u}}(u^*|s,N), * \in \{\mathrm{MN}, \mathrm{MLN}\} \tag{6.32}$$

$$\log P^{\mathrm{Nu}}(u^*|s) = \log\left(\sum_{N=N_{\min}}^{N_{\max}} P^{\mathrm{N}}(N|s)P^{\mathrm{u}}(u^*|s,N)\right) \tag{6.33}$$

$$= \sum_{N=N_{\min}}^{N_{\max}} \mathbb{1}_{N=\tilde{N}}\left(\log P^{\mathrm{N}}(N|s) + \log P^{\mathrm{u}}(u^*|s,N)\right) \tag{6.34}$$

$$= \log P^{\mathrm{N}}(\tilde{N}|s) + \log P^{\mathrm{u}}(u^*|s,\tilde{N}) \tag{6.35}$$

where the $\log$ operator can be applied inside the summation over the prediction horizons in (6.34), because we know the value $\tilde{N}$ of the horizon variable N that is sampled in advance of the calculation of the control laws [62].

The Complete Policy

We collect all the parameters of the meta-parameter-deciding recomputation and horizon policies described above, and the parameters of the controllers into a single parameter vector $\theta = \left[\theta^{\mathrm{c}}, \theta^{\mathrm{N}}, \alpha, \theta^{\mathrm{M}}, \theta^{\mathrm{L}}, \Sigma^{\mathrm{M}}, \Sigma^{\mathrm{ML}}\right]^{\top}$, and define the complete policy π_θ, whose input is the state s and output is the action $a = \left[c, N, u^{\mathrm{M}}, u^{\mathrm{ML}}\right]$

Section 6.3.2 presents the recomputation problem of the MPC and the policy that decides when to compute it. We now view the recomputation policy $\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}$ as selecting the active controller between the two control laws. It is then clear that π_θ is a mixture distribution between the Gaussian control policies where the weights of the mixture are assigned by the Bernoulli recomputation policy $\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}$. We label the probability distribution of the complete policy π_θ as P^{a} and define it and the log probability as follows:

$$P^{\mathrm{a}}(\tilde{a}|s) = P^{\mathrm{c}}(0|s) P^{\mathrm{Nu}}(\tilde{u}^{\mathrm{MLN}}|s) + P^{\mathrm{c}}(1|s) P^{\mathrm{Nu}}(\tilde{u}^{\mathrm{MN}}|s) \tag{6.36}$$

$$\begin{aligned} \log P^{\mathrm{a}}(\tilde{a}|s) =& \mathbb{1}_{\tilde{c}=0}\left(\log P^{\mathrm{c}}(0|s) + \log P^{\mathrm{Nu}}(\tilde{u}^{\mathrm{MLN}}|s)\right) + \\ & \mathbb{1}_{\tilde{c}=1}\left(\log P^{\mathrm{c}}(1|s) + \log P^{\mathrm{Nu}}(\tilde{u}^{\mathrm{MN}}|s)\right) \end{aligned} \tag{6.37}$$

$$\begin{aligned} =& \mathbb{1}_{\tilde{c}=0}\left(\log P^{\mathrm{c}}(0|s) + \log P^{\mathrm{N}}(\tilde{N}_i|s) + \log P^{\mathrm{u}}(\tilde{u}^{\mathrm{ML}}|s, \tilde{N}_i)\right) + \\ & \mathbb{1}_{\tilde{c}=1}\left(\log P^{\mathrm{c}}(1|s) + \log P^{\mathrm{N}}(\tilde{N}|s) + \log P^{\mathrm{u}}(\tilde{u}^{\mathrm{M}}|s, \tilde{N})\right) \end{aligned} \tag{6.38}$$

where we again used the fact that we know the values of the sampled variables $\tilde{c}, \tilde{N}$ to take the logarithm of the sums in (6.37) and (6.38), and $\tilde{N}$ represents the output of the horizon policy at the current step t in the case the recomputation policy signals to recompute the MPC. While (6.38) is all that is strictly needed to use the PPO algorithm, we derive the policy gradient as well to highlight the connection to the LQR gradient presented in Section 6.3.3, as well as for future applications of this control framework with other RL algorithms.

$$\nabla_\theta \log \pi_\theta(\tilde{a}|s) = \nabla_\theta \log P^{\mathrm{a}}(\tilde{a}|s) \tag{6.39}$$

$$\begin{aligned} =& \mathbb{1}_{\tilde{c}=0}\Big(\nabla_{\theta^{\mathrm{c}}} \log \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(0|s) + \nabla_{\theta^{\mathrm{N}}} \log \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(\tilde{N}_i|s) \\ &\quad + \nabla_{\theta^{\mathrm{M,L}}} \log \pi^{\mathrm{ML}}_{\theta^{\mathrm{M,L}}}(\tilde{u}^{\mathrm{ML}}|s,\tilde{N})\Big) + \\ & \mathbb{1}_{\tilde{c}=1}\Big(\nabla_{\theta^{\mathrm{c}}} \log \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(1|s) + \nabla_{\theta^{\mathrm{N}}} \log \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(\tilde{N}|s) \\ &\quad + \nabla_{\theta^{\mathrm{M}}} \log \pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(\tilde{u}^{\mathrm{M}}|s,\tilde{N})\Big) \end{aligned} \tag{6.40}$$

$$\begin{aligned} =& \mathbb{1}_{\tilde{c}=0}\Big(\nabla_{\theta^{\mathrm{c}}} \log \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(0|s) + \nabla_{\theta^{\mathrm{N}}} \log \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(\tilde{N}_i|s) + (\Sigma^{\mathrm{ML}})^{-1} \\ &\quad (\pi^{\mathrm{ML}}_{\theta^{\mathrm{M,L}}}(s,\tilde{N}) - \tilde{u}^{\mathrm{ML}})(\nabla_{\theta^{\mathrm{M}}}\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(s,\tilde{N}_i) + \nabla_{\theta^{\mathrm{L}}}\pi^{\mathrm{L}}_{\theta^{\mathrm{L}}}(s))\Big) + \\ & \mathbb{1}_{\tilde{c}=1}\Big(\nabla_{\theta^{\mathrm{c}}} \log \pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(1|s) + \nabla_{\theta^{\mathrm{N}}} \log \pi^{\mathrm{N}}_{\theta^{\mathrm{N}}}(\tilde{N}|s) \\ &\quad + (\Sigma^{\mathrm{M}})^{-1}(\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(s,\tilde{N}) - \tilde{u}^{\mathrm{M}})\nabla_{\theta^{\mathrm{M}}}\pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(s,\tilde{N})\Big) \end{aligned} \tag{6.41}$$

With this policy, one can optimize any parameter of the MPC and LQR controllers jointly, by providing the gradient of these controllers wrt. the parameters. In the RL context, we show how one can obtain these gradients for the LQR controller in Section 6.3.3. For the gradient of the MPC see e.g. [72, 76].

6.3.3 Optimizing LQR with Reinforcement Learning

The Time-Invariant Case

To apply the policy-gradient theorem to tune the LQR, we need to compute the gradients of the $\mathbf{K}$-matrix wrt. to the $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{N}$ matrices. As equations (2.12) and (2.13) show, the feedback matrix $\mathbf{K}$ is only implicitly defined in terms of the these matrices, and so direct differentiation of equations (2.12) and (2.13) is not possible. Since we obtain the system matrices $\mathbf{A}$ and $\mathbf{B}$ directly from the MPC scheme and they have a clear purpose in making the MPC and LQR objectives compatible, we assume that they are fixed wrt. $\mathbf{Q}, \mathbf{R}, \mathbf{N}$ such that $\nabla_{\mathbf{Q},\mathbf{R},\mathbf{N}}\mathbf{A} = \nabla_{\mathbf{Q},\mathbf{R},\mathbf{N}}\mathbf{B} = 0$:

Assumption 2. *The weighting matrices $\{\mathbf{Q}, \mathbf{R}, \mathbf{N}\}$ values are such that the DARE has a solution $\mathbf{S}_\infty$.*

Proposition 2. *We flatten the matrices $\mathbf{S}_\infty, \mathbf{K}_\infty, \mathbf{Q}, \mathbf{R}$ and $\mathbf{N}$ into vectors, and organize them as follows: $y = \{\mathbf{S}_\infty, \mathbf{K}_\infty\}$ and $z = \{\mathbf{Q}, \mathbf{R}, \mathbf{N}\}$, such that the DARE and $\mathbf{K}$-matrix equations can be written on the vector form $F(y, z) = 0$. The gradient of $\mathbf{K}_\infty$ wrt. the weighting matrices $\mathbf{Q}, \mathbf{R}, \mathbf{N}$ can then be found as:*

$$\frac{\partial y}{\partial z} = -\frac{\partial F}{\partial y}^{-1} \frac{\partial F}{\partial z} = \nabla_{\mathbf{Q},\mathbf{R},\mathbf{N}} \mathbf{S}_\infty, \mathbf{K}_\infty \tag{6.42}$$

Proof. We rewrite (2.12)-(2.13) on the general vector form $F(y, z) = 0$ where $y = \{\mathbf{S}_\infty, \mathbf{K}_\infty\}$ and $z = \{\mathbf{Q}, \mathbf{R}, \mathbf{N}\}$.

$$\mathbf{A}^\top \mathbf{S}_\infty \mathbf{A} - \mathbf{S}_\infty - (\mathbf{A}^\top \mathbf{S}_\infty \mathbf{B} + \mathrm{N})\mathbf{K}_\infty + \mathbf{Q} = 0 \tag{6.43}$$

$$(\mathbf{B}^\top \mathbf{S}_\infty \mathbf{B} + \mathbf{R})\mathbf{K}_\infty - (\mathbf{B}^\top \mathbf{S}_\infty \mathbf{A} + \mathbf{N}^\top) = 0 \tag{6.44}$$

We then apply the implicit function theorem (IFT) which states that:

$$\frac{\partial F}{\partial y}\frac{\partial y}{\partial z} + \frac{\partial F}{\partial z} = 0 \;\Rightarrow\; \frac{\partial y}{\partial z} = -\frac{\partial F}{\partial y}^{-1} \frac{\partial F}{\partial z} \tag{6.45}$$

These gradients are easily obtained with automatic differentiation software. □

Assumption 2 holds if $\mathbf{Q} - \mathbf{N}\mathbf{R}^{-1}\mathbf{N}^\top \succ 0$ and $\mathbf{R} > 0$, and further, it is required that the symplectic pencil of the problem has eigenvalues sufficiently far from the unit circle, which is satisfied if the pair $(\mathbf{A}, \mathbf{B})$ is stabilizable and the pair $(\mathbf{A},\ \mathbf{Q})$ is detectable [109]. These conditions can be ensured by estimating the gradients as described above, and then solving a semidefinite program (SDP) using the estimated gradients subject to these constraints. However, for simplicity and to avoid constrained optimization, we set $\mathbf{N} = 0$, simplifying the constraints on the positive (semi)-definiteness to $\mathbf{Q} \succ 0$ and $\mathbf{R} > 0$. Further, we write $\mathbf{Q}$ and $\mathbf{R}$ in terms of their Cholesky decompositions: $\mathbf{Q} = \mathbf{Q}_C^\top \mathbf{Q}_C$, $\mathbf{R} = \mathbf{R}_C^\top \mathbf{R}_C$, and let the RL algorithm adjust the elements of $\mathbf{Q}_C$ and $\mathbf{R}_C$, ensuring that the original $\mathbf{Q}$ and $\mathbf{R}$ matrices are always positive definite.

Time-Varying Gradients

In what follows we will assume $\mathbf{N} = 0$, for simplicity of the derivation and expressions (and the constraint reasons outlined above). We find the gradients with respect to p, where p is an arbitrary scalar element of the $\mathbf{Q}$ and $\mathbf{R}$ matrices. The full gradients $\nabla_{\mathbf{Q},\mathbf{R}} \mathbf{K}_k$ can then be found by solving (6.48) for each parameter of

$\mathbf{Q}$ and $\mathbf{R}$ and arranging the results into the appropriate matrix structure. The derivations involve repeated application of the chain rule and the following relation for the gradient of the matrix inverse, where we define $\mathbf{E}$ for convenience:

$$\mathbf{E}_k = \mathbf{R} + \mathbf{B}_k^\top \mathbf{S}_{k+1}\mathbf{B}_k, \;\; \nabla_p \mathbf{E}^{-1} = -\mathbf{E}^{-1}\nabla_p \mathbf{E}\mathbf{E}^{-1} \tag{6.46}$$

$$\begin{aligned}\nabla_p \mathbf{S}_k =& \nabla_p \mathbf{Q} + \mathbf{A}_k^\top (\nabla_p \mathbf{S}_{k+1}\mathbf{A}_k - \nabla_p \mathbf{S}_{k+1}\mathbf{B}_k\mathbf{E}_k^{-1}\mathbf{B}_k^\top \mathbf{S}_{k+1}\mathbf{A}_k \\ &+ \mathbf{S}_{k+1}\mathbf{B}_k(\mathbf{E}_k^{-1}\nabla_p \mathbf{E}_k\mathbf{E}_k^{-1}\mathbf{B}_k^\top \mathbf{S}_{k+1}\mathbf{A}_k - \mathbf{E}_k^{-1}\mathbf{B}_k^\top \nabla_p \mathbf{S}_{k+1}\mathbf{A}_k))\end{aligned} \tag{6.47}$$

$$\nabla_p \mathbf{K}_k = -\mathbf{E}_k^{-1}\nabla_p \mathbf{E}_k \mathbf{E}_k^{-1}\mathbf{B}_k^\top \mathbf{S}_{k+1}\mathbf{A}_k + \mathbf{E}_k^{-1}\mathbf{B}_k^\top \nabla_p \mathbf{S}_{k+1}\mathbf{A}_k \tag{6.48}$$

Note that these gradients could easily be extended with time-varying $\mathbf{Q}$ and $\mathbf{R}$ matrices. The time-varying gradients of $\mathbf{S}_k$ and $\mathbf{K}_k$ are given as follows:

$$\nabla_{\mathbf{Q},\mathbf{R}} \mathbf{S}_k = \begin{cases} \nabla_p \mathbf{S}_\infty \big|_{p\in \mathbf{Q},\mathbf{R}} & \text{, if } k \geq N_i \\ \nabla_p \mathbf{S}_k \big|_{p\in \mathbf{Q},\mathbf{R}} & \text{, otherwise} \end{cases} \tag{6.49}$$

$$\nabla_{\mathbf{Q},\mathbf{R}} \mathbf{K}_k = \begin{cases} \nabla_p \mathbf{K}_\infty \big|_{p\in \mathbf{Q},\mathbf{R}} & \text{, if } k \geq N_i \\ \nabla_p \mathbf{K}_k \big|_{p\in \mathbf{Q},\mathbf{R}} & \text{, otherwise} \end{cases} \tag{6.50}$$

$$\nabla_{\theta^{\mathrm{L}}} \pi^{\mathrm{L}}_{\theta^{\mathrm{L}}}(s_t) = \nabla_{\mathbf{Q},\mathbf{R}} \mathbf{K}_{t-i}(\hat{x}_t - \bar{x}_t) \tag{6.51}$$

6.3.4 Summary of Control Algorithm with Learning

The complete control algorithm is outlined in Algorithm 2. This shows the control algorithm in the exploration phase, but the exploitation phase is identical with the two exceptions: 1) there is no data collection (and therefore no parameter updates), and 2) we use the deterministic version (i.e. the mode of the probability distribution) of all policies except for the recomputation policy. The Bernoulli distribution of the recomputation policy does not generally tend to quasi-determinism as exploration settles, unlike the other policies. As such, we find that the deterministic version of this policy has worse control performance and less consistency in the plant response than the stochastic version which we are in fact optimizing the response for.

The system to be optimized runs in an episodic fashion, and every Z steps the RL algorithm updates the parameters of the control system.

Algorithm 2: Control Algorithm with Learning

```
while Running do
    Initialize episode: $x_0, \hat{p}_0$
    Compute initial MPC solution: $u^{\mathrm{M}}_{0:N_{\max}-1},\ \hat{x}_{1:N_{\max}} = \pi^{\mathrm{M}}_{\theta^{\mathrm{M}}}(x_0, \hat{p_0}, N_{\max})$
    Execute first MPC input: $u^{\mathrm{M}}_0$
    Calculate LQR system matrices: $\mathbf{A}_{1:N_{\max}}, \mathbf{B}_{1:N_{\max}}$
    for $t = 1, 2, \ldots, T$ do
        Measure system state: $\bar{x}_t$
        if $\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}}(c_t \mid s_t)$ draws $\tilde{c}_t = 1$ then
            Compute MPC solution: $\tilde{u}^{\mathrm{MN}}_{t:t+N_t-1},\ \hat{x}_{t+1:t+N_t} = \pi^{\mathrm{MN}}_{\theta^{\mathrm{MN}}}(\tilde{u}^{\mathrm{MN}}|s_t)$
            Execute control input: $\tilde{u}^{\mathrm{MN}}_t$
            Calculate LQR system matrices: $\mathbf{A}_{t:t+N_t}, \mathbf{B}_{t:t+N_t}$
            Update last computation states: $i \leftarrow t$
        else
            Compute input: $\tilde{u}^{\mathrm{CS}}_t = \pi^{\mathrm{CS}}_{\theta^{\mathrm{M,L,N}}}(u^{\mathrm{CS}}_t|s_t)$
            Execute input: $\tilde{u}^{\mathrm{CS}}_t$
        end
        Collect data: $\mathcal{D} \leftarrow (s_t, \tilde{a}_t, R(s_t, \tilde{a}_t), s_{t+1})$
        if size$(\mathcal{D}) = Z$ then
            for $e = 1, \ldots, \mathrm{NumEpochs}$ do
                for $\mathcal{B} \in \mathcal{D}$ do
                    Evaluate PPO objective over minibatch $\mathcal{B}$ (2.33)
                    Update parameters (2.29)
                end
            end
            Empty dataset: $\mathcal{D} = \{\}$
        end
    end
end
```

6.3.5 Reward Function

The RL reward function codifies the behaviour that the control algorithm is designed to exhibit, wrt. stability, control performance, and computational complexity. As RL is a trial-and-error based optimization approach, the control algorithm will necessarily have to attempt risky maneuvers and experience the consequences in order to learn how to control the system. However, with a feasible composition of the learning problem in terms of hyperparameters and expressive power of the learned components, the end result of the converged behaviour should be stable and yield good control performance. A failure to achieve this would therefore indicate a misalignment between the choice of the reward function, and the control design requirements. With this in mind, we formulate a reward function on the following form:

$$R(s_t, a_t) = R_\ell(s_{t+1}) + \lambda_h(T - t)R_h(s_{t+1}) + \lambda_c R_c(a_t) R_N(a_t) \tag{6.52}$$

Here, λ_h and λ_c are weighting factors that represent the relative importance of the different terms. R_ℓ is the control performance term that incentivizes the RL policy to achieve good control performance. We set it to be the same as the stage cost from the MPC objective, i.e. $R_\ell = -\ell$ but this is not a strict requirement. R_h is a term that indicates whether any system constraints are violated. If using hard constraints as is the case in this chapter, this term is binary and is further weighted by $T - t$, that is, the number of time steps remaining in the episode, since the episode is prematurely terminated if any constraints are violated. If the system to be controlled has soft constraints, the R_h term could be made continuous. The R_c term indicates whether the MPC was computed at the current step ($R_c = 1$), and as such, it should reflect the relative computational complexity of the two modes of the control system. Finally, R_N represents the computational complexity of the MPC as a function of the prediction horizon N. We assume that the computational complexity grows linearly in the prediction horizon, i.e. $R_N(N_t) = N_t$, as a lower bound for the true complexity. We favour a lower bound as this would bias the RL policy towards better control performance rather than lower computation. The relationship between the prediction horizon and the computational complexity depends upon the algorithms used in the MPC implementation. We employ an interior point method, which under the assumption of local convergence and a guess of an initial solution that is reasonable, generally yields linear complexity [154], while other methods such as active set methods typically yield quadratic growth in computational complexity [108].

6.3.6 Initialization of the Learning Procedure

How to initialize the control algorithm, that is, defining its behaviour before any learning takes place is a question that has several valid answers. One could favour the most computationally expensive initialization, i.e. compute MPC every step with the maximum horizon, which without any prior knowledge about the task would be the “safest” initialization that is most likely to give the best control performance. This is however not necessarily the initialization that would facilitate the learning process most optimally and might trap the learned components in local minima. To simplify the learning problem one might therefore instead favour initializing the components in the center of their operating range and with high entropy (wrt. to its output) such that exploration is maximal.

We offered some guidance in the case of the LQR in Section 6.2, i.e. the LQR should be initialized such that it is compatible with the MPC. We view the purpose of the recomputation policy as finding instances where comparable (or better) control performance can be achieved without computing the MPC, and as such, we choose to initialize it to emulate the MPC paradigm and with high probability elect to compute the MPC. For the horizon policy, we initialize it equal to the best performing fixed horizon MPC scheme. Initializing the policies can be done by adjusting the bias terms of the policy outputs:

$$\pi^{\mathrm{c}}_{\theta^{\mathrm{c}}} \leftarrow -\log\left(\frac{1}{c_{\mathrm{init}}} - 1\right) \tag{6.53}$$

$$\pi^{\mathrm{N}}_{\theta^{\mathrm{N}}} \leftarrow \tanh^{-1}\left(\frac{(N_{\mathrm{max}} - N_{\mathrm{min}})(N_{\mathrm{init}} - (-1))}{1-(-1)} + N_{\mathrm{min}}\right) \tag{6.54}$$

6.4 Numerical Results

This section illustrates the proposed control method as outlined in Section 6.3 on the simulated inverted pendulum system. Additionally, to aid in highlighting the contributions of the different meta-parameters of the control algorithm we also present experiments for each meta-parameter in isolation.

We set $N_{\mathrm{min}} = 1$ and N_{max} = 40, and note that this horizon does not cover a full swing-up maneuver of the pendulum (requiring $N > 100$). The maximum horizon is chosen to emulate the effects of a computationally limited embedded hardware platform. The weighting terms of the reward function (6.52) are set to $\lambda_h = -10$ and $\lambda_c = 10^{-2}$. This ensures that violating constraints incur a higher cost than finishing the episode, and makes the computation account for approximately 8% of the total cost of the standard MPC scheme (i.e. MPC computed at every step) with the highest prediction horizon.

While our method supports tuning parameters internal to the MPC (i.e. objective, constraints etc.), doing so requires reevaluating the log-probabilities and recomputing the OCP for every data sample after every parameter update, which is highly relevant when using minibatches or several passes over a dataset as PPO does. This adds considerable computation, and optimization of these parameters with RL has been successfully demonstrated in previous works [52, 76]. Therefore, since this chapter focuses on optimizing the meta-parameters of the MPC scheme (that is, the prediction horizon and when to compute) we do not tune any parameters internal to the MPC in this example. We do however optimize the parameters of the LQR to illustrate the concept, as the LQR is considerably less costly to evaluate Finally, we omit learning the value function of the MPC, as it adds considerable complexity to the experiments, and was found to not add much benefit for control of similar systems in [38].

6.4.1 The Inverted Pendulum System

The inverted pendulum system is a classic control task in which a pendulum, consisting of a rigid rod with a mass at the end, is mounted on top of a cart that is fixed on a track. The control system exerts a horizontal force on the cart, which moves the cart back and forth on the track, which subsequently swings the pendulum. The control objective is to stabilize the pendulum in the up-position and position the cart at the position reference, while respecting the constraint that the cart's position is limited by the physical size of the track. The dynamics of the system are highly nonlinear, and further, the system is unstable, meaning that a controller is necessary to guide the system to stable conditions and then to maintain the stability. We perturb the parameters of the MPC dynamics model such that its dynamics differs from the plant dynamics, as shown in Table 6.1, and thus the LQR is a useful addition to correct for prediction errors in between the MPC computations. The state x consists of the cart position ψ and velocity v, pendulum angle ϕ and angular velocity ω.

Each episode lasts a maximum of 150 steps (i.e. T=150), or until the position constraint (6.57) is violated. We sample initial conditions according to (6.60), and a position reference (6.59) that is redrawn every 50 steps, where $\mathcal{U}$ is the uniform distribution. The MPC objective is defined as (6.62), i.e. minimize the kinetic energy of the system and the distance of the cart to the position reference, while maximizing the potential energy.

$$\dot{v} = \frac{mg\sin\phi\cos\phi - \frac{7}{3}\left(u + ml\omega^2\sin\phi - \mu_c v\right) - \frac{\mu_p\omega\cos\phi}{l}}{m\cos^2\phi - \frac{7}{3}M} \tag{6.55}$$

$$\dot{\omega} = \frac{3}{7l}\left(g\sin\phi - \dot{v}\cos\phi - \frac{u_p\omega}{ml}\right), \dot{\phi} = \omega,\ \ \dot{\psi} = v \tag{6.56}$$

$$-5 \leq u_t \leq 5,\ \ -2 \leq \psi_t \leq 2 \tag{6.57}$$

$$x = [\psi, v, \phi, \omega]^\top \tag{6.58}$$

$$\psi_r(t) \sim \mathcal{U}(-1,1) \tag{6.59}$$

$$x_0 \sim [0, \mathcal{U}(-1,1), \mathcal{U}(-\pi,\pi), \mathcal{U}(-1,1)]^\top \tag{6.60}$$

$$x_t^s = [\psi_r(t), 0, 0, 0]^\top \tag{6.61}$$

$$\ell(x_t, u_t, \hat{p}_t) = \mathrm{E_k} - 10\mathrm{E_p} + 10(\psi_t - \psi_r(t))^2 \tag{6.62}$$

$$\mathbf{D} = [0.1] \tag{6.63}$$

Table 6.1: Parameters of the inverted pendulum system.

Name	Plant	MPC	Description
m	0.1	0.2	Mass of pendulum
M	1.1	1.5	Mass of cart and pendulum
g	9.81	9.81	Gravitational constant
l	0.25	0.25	Half the length of the pendulum
μ_c	0.01	0.01	Friction coefficient between track and cart
μ_p	0.001	0.001	Friction coefficient between pendulum and cart
Δ_t	0.04	0.04	Discretization step size in seconds

6.4.2 Training and Evaluation

The hyperparameters of the PPO algorithm are listed in Table 6.2. The horizon and recomputation policies are fully-connected NNs with 2 hidden layers with 64 nodes in each, whereas the value function is an NN with 2 hidden layers of 128 nodes, all using the $\tanh$ activation function. We use the same hyperparameters for the isolated experiments, rather than tuning them specifically, and as such the isolation experiments serve mainly to assess how the two meta-parameters contribute. For the horizon experiment, we use a frame-skip of $d = 10$, and for the other experiments, we use a varying $d \in [1, 2, 3, 4]$ that is drawn at the start of every episode (with $d = 1$ when evaluating).

Table 6.2: Hyperparameters of the PPO algorithm

Name	Value	Description
γ	0.99	Discount factor
Z	256	Length of trajectory collected by each actor
n_envs	4	Number of actors run in parallel
ent_coef	0	Coefficient for entropy loss term
η	$3 \cdot 10^{-4}$	Learning rate
vf_coef	0.5	Coefficient for value function loss term
max_grad_norm	0.5	Maximum global norm of gradients
κ	0.9	Bias-variance tradeoff for GAE
nminibatches	1	Number of minibatch partitions for dataset
noptepochs	10	Number of passes over the dataset
ϵ	0.25	Clip parameter for the objective function

We initialize the recomputation policy using (6.53) to compute the MPC with 90% probability, thus within two time steps there is a 99% probability that the MPC is computed. It therefore initially mimics the traditional MPC scheme closely. The horizon policy is initialized at the best performing fixed-horizon (Figure 6.2), i.e. $N = 31$, using equation (6.54).

To evaluate the performance of the control system governed by the RL policy, we construct a ”test-set” consisting of 25 episodes where all stochastic elements are drawn in advance (i.e. initial conditions and position references) such that the episodes are consistent across evaluations. This gives us an objective way to compare and order policies on their performance, and to compare against the MPC baselines. We set the cost objective (2.1) $C = -R$, and evaluate models based on the total sum of costs over the episodes in the test-set.

Moreover, for every experiment we report the average over five random initial seeds (referred to as models), which impact the initializations of NNs and the randomness in exploration and episodes. Figure 6.2 shows the total cost of the control system baselines as a function of a static prediction horizon, and a static recomputation schedule. The minimum cost of the baselines is achieved with a prediction horizon of $N = 31$ and a schedule of recompute at every step. It is important to note that while this figure indicates that the optimization landscape as

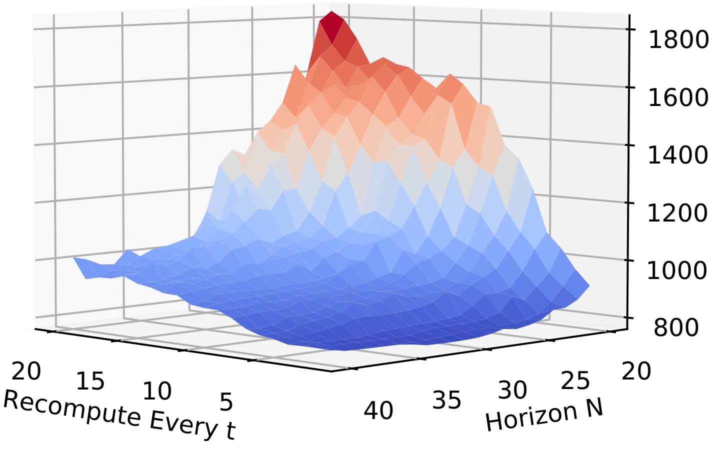


Figure 6.2: Total cost over the evaluation set for the MPC as a function of fixed horizons and fixed recomputation schedules. The minimum is found at horizon $N = 31$ and recompute every step with a cost of 781.

a function of these two variables is monotonic and amenable to optimization, the reward landscape as a function of the parameters θ (which in this example consists of the parameters of the LQR and the parameters of the recomputation and horizon NNs) is likely very different — containing many valleys and hills to overcome in order to minimize the cost objective.

6.4.3 Inverted Pendulum Results and Discussion

The results of the experiments are presented in Figures 6.5 and 6.4, and Table 6.3. When reporting the cost of the tuned system, we average over the best performing policy of each model, as well as over five seeds to account for the stochastic recomputation policy. When describing the policies' behaviour as in Figure 6.4 however, we use the best performing policy and one specific seed. We summarize our main findings as follows:

Learning improves both computational complexity and control performance. The RL framework we propose is able to improve upon the total objective by 21.6%, which interestingly is not only due to reductions in the computation term (71%) but also a sizable improvement on the control performance objective (18.5%). For this example it takes about 20 hours of data of interaction with the system to reach convergence, corresponding to 700 thousand data samples. Figure 6.5(a) shows that generally the performance monotonically improves with more data, we therefore do not view the data requirement as a major issue. Note that we did not spend significant time tuning the hyperparameters of the RL algorithm, and we favored consistency over faster rate of improvement. Thus, the algorithm's data requirement could potentially be improved with hyperparameter optimization. Moreover, DRL is a relatively new field, and more data-efficient algorithms are likely to be developed in the near future. Finally, if the RL policies are learned in a simulation environment before being deployed on the real system, the wall-clock time can be made (nearly) arbitrarily small by parallelization and more compute power.

Figure 6.5 shows that the recomputation meta-parameter is the most impactful parameter, reaching a cost around 700 when optimized by itself, which is about 13% higher than the converged value of the optimized complete policy. It also shows why we favor biasing towards higher computation, as while initializing the recomputation policy to compute the MPC with 10% probability reaches the same asymptotic performance as the 90% initialization, it is the only policy we trained that intermittently violates the constraint.

Because we initialize the learning problem at a best-effort of optimal tuning, the control performance improvements we observe are non-trivial to explain and arise due to complex interactions between multi-step adaptive-horizon MPC and the LQR control law. Figure 6.4 shows the distribution of the prediction horizon, and the steps between MPC computations chosen by the policies. The horizon policy has converged to only selecting the maximum horizon, while the MPC is mostly computed at every step (70% of steps) with some significantly longer streaks. Because of the finite-horizon nature of the OCP the MPC will produce solutions of varying optimality based on the exact initial conditions it is computed from. The RL policy has learned to recognize a set of conditions for which the computed solutions are more optimal than neighboring conditions, and therefore not recomputing and employing a longer section of the more optimal input sequence will produce better control performance.

The control algorithm we propose adds some overhead compared to MPC, i.e. evaluating the policies deciding if the MPC should be computed and with what horizon, and computing the LQR gain-matrix as necessary. The CPU processing time of our control algorithm vs the best performing MPC baseline is shown in Figure 6.3. The overhead is however small compared to the execution time of the MPC, and therefore our framework results in a 36% reduction in the total processing time of the control system compared to the best performing MPC scheme. This frees up resources for other on-board tasks the controlled system might have, or could be leveraged to increase the battery life of the system. Figure 6.3 that the calculation of the LQR gain matrices accounts for a significant portion of the total processing time of our control algorithm, and as such the processing time

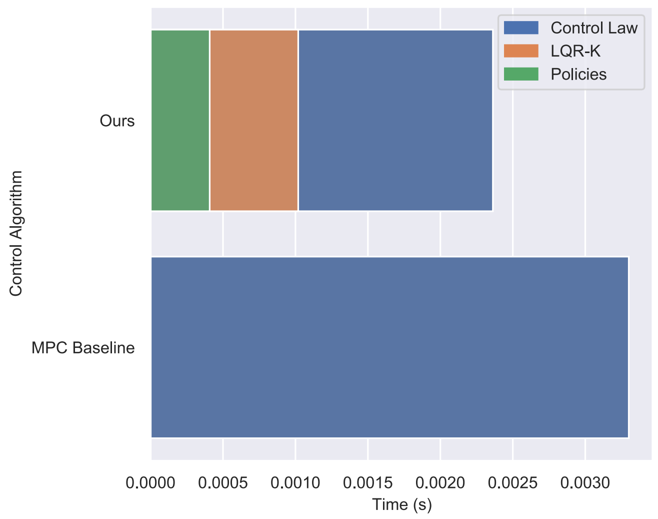


Figure 6.3: Average processing time required to evaluate the control algorithms (our proposed algorithm vs best performing MPC baseline) on the test set. For our control algorithm, in addition to evaluating the MPC and LQR controllers, this includes calculating the LQR gain-matrices K after MPC computations, and evaluating the policies deciding the meta-parameters.

improvements of our method could be considerably improved by optimizing the implementation of the LQR gain matrix calculation routine. Also note that the best performing MPC baseline is computed at every step, and as such it is not necessary to compute any LQR matrices for this control algorithm.

One of the models we trained for the complete policy, and one of the models for the isolated recomputation policy got stuck in the local minima of computing the MPC at every step, and since this didn't give any interesting results (essentially maintaining the initial behaviour and performance) we excluded them from Figure 6.5 and the discussions, replacing them with new models with new seeds. Since policy gradient algorithms are local search methods it is to be expected that it finds local minima, and random exploration can cause it to sometimes settle in sub-optimal local minima. This is also a question of how much data is used to generate the gradient.

Joint optimization delivers additional improvements. These results support the conjecture stated in the introduction; by optimizing the different parameters of the control algorithm together, we are able to enhance the control algorithm in ways that we are not able to when optimizing the same parameters in isolation. The horizon policy tends to a mean horizon of 28-31 when optimized by itself (Fig-

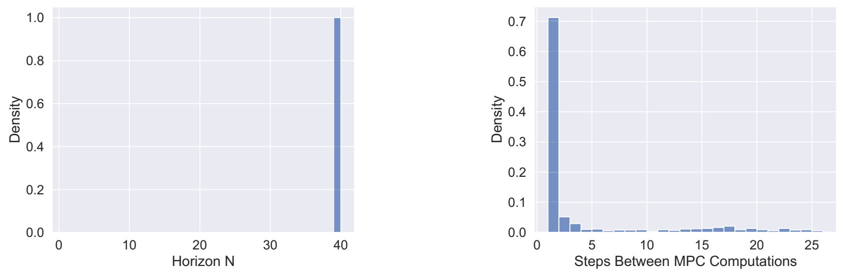
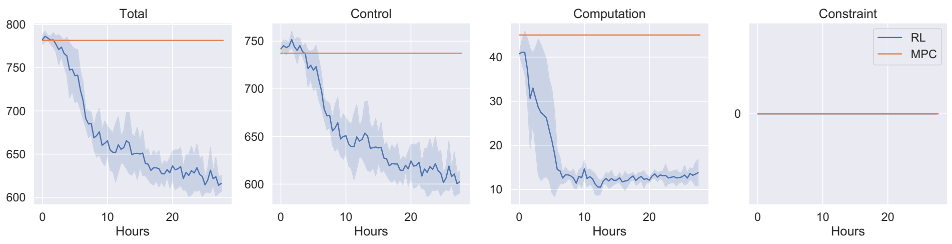


Figure 6.4: Distribution of prediction horizons and steps between MPC computations selected by the best performing policy on the evaluation test-set.

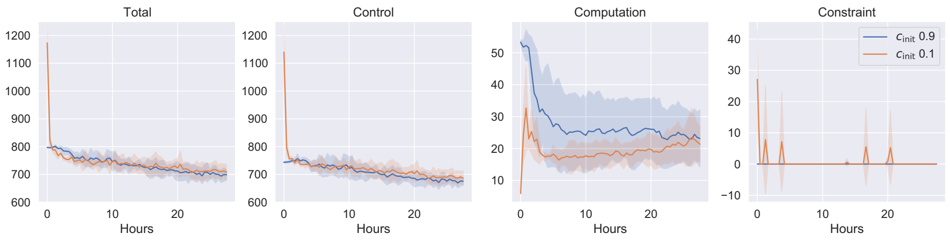
ure 6.5(c)), which is consistent with the best performing fixed horizon MPC as seen in Figure 6.2. However, as Figure 6.4 shows the complete RL policy favoured higher horizons when optimizing all meta-parameters together. As shown in Table 6.3 this improves performance significantly, where imposing a maximum horizon of 31 results in a 12% increase in cost. When solely tuning the LQR applied on an MPC on a fixed recomputation schedule, we were not able to achieve any consistent improvements. We hypothesize that this is due to the model mismatch (the MPC is overestimating the weight of the system by 36%) such that the computed LQR is not very well suited for the actual control problem, and therefore its gradients are flat and noisy. When combining LQR tuning with meta-parameter optimization we are able to tune the LQR to achieve meaningful improvements, with the tuned LQR improving by 1.95% over the initialization described in Section 2.3. This might be due to the recomputation policy generating trajectories for the LQR that are similar, thus yielding more consistent gradients. The tuned LQR weights during training are shown in Figure 6.6. The last entry in Table 6.3 shows a scenario where the same amount of computation is expended uniformly in time rather than dynamically allocated by the recomputation policy.

Table 6.3: Ablation analysis: We take the best performing policy and alter one aspect at a time, observing its effect on the cost.

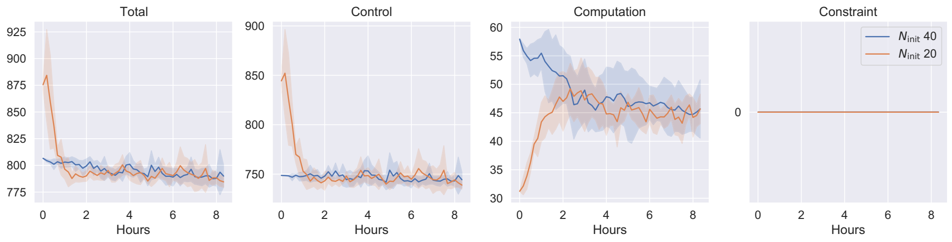
Scenario	Change
Default LQR tuning	$+1.95\%$
Max horizon 31	$+13.41\%$
Recompute at the average frequency (every $t = 4$)	$+35.45\%$



(a) Optimizing both meta-parameters and LQR jointly. The MPC baseline corresponds to the best-performing MPC scheme tuned as a function of recomputation schedule and prediction horizon.



(b) Optimizing the recomputation meta-parameter in isolation.



(c) Optimizing the prediction horizon meta-parameter in isolation.

Figure 6.5: Training process for the proposed learned control algorithm, and for each meta-parameter in isolation. In the isolated cases, we show that the tuning process is capable of recovering from sub-optimal initializations.

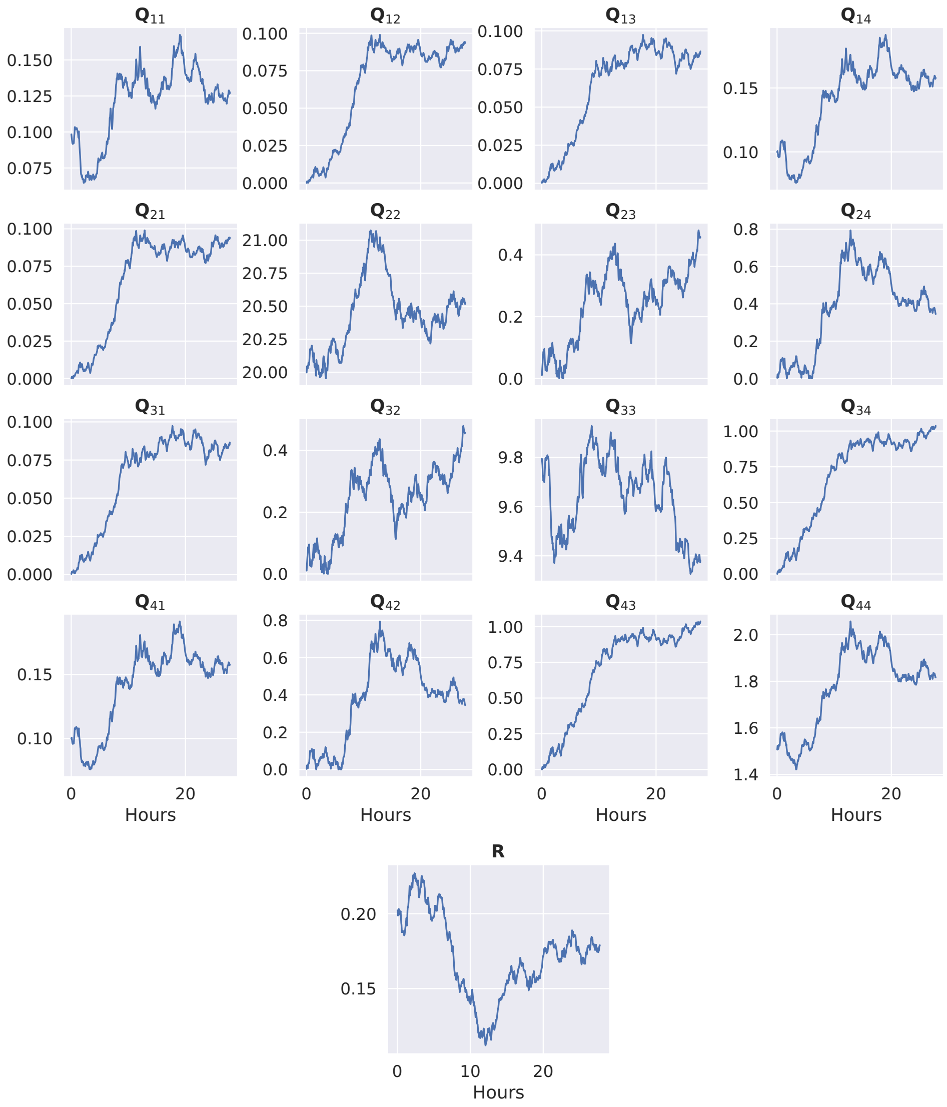


Figure 6.6: The evolution of the LQR weights as tuned by the complete RL policy.

6.5 Conclusion

This chapter has introduced a novel control algorithm that is tuned with RL to jointly improve the computational power usage and the control performance. We focus on optimizing the meta-parameters of the MPC scheme and demonstrate its efficacy on the classic control task of balancing an inverted pendulum. We show that by selecting the conditions under which the MPC is computed, control performance can be improved over the paradigm of computing at every step and that the control algorithm can be further improved by considering all the optimized parameters together. Seeing as MPC is increasingly being considered for applications with fast dynamics or limited computational power and energy resources, our framework could be an important tool in enabling such applications to harness the good control performance and constraint satisfaction abilities of the MPC. We found that with model mismatch, tuning the dual mode MPC and LQR control law was difficult. Future work could evaluate if state-dependent $\mathbf{Q}$ and $\mathbf{R}$ matrices alleviate this issue, which would also provide a gain-scheduling property to the control algorithm. Whether any stability guarantees and control satisfaction properties can be given under the learned meta-parameter-deciding policies should be investigated.

The method presented in this Chapter produced good results on the experiment and was stable in the sense that 5 of 6 models exhibits behaviour producing costs within a few percent of each other on the experiment objective (with the sixth stuck in a local minima near the initial behaviour). We did however find the variance of the meta-parameter policies to be problematically high, necessitating such measures as frame-skip and favouring the stochastic version of the recomputation policy in evaluation, in order to decrease the volatility in system response from one iteration of the parameters to the next. The chosen formulations of these meta-parameter variables, i.e. logistic regression for the recomputation variable and the GPD for the horizon variable, are convenient in that they make the log-probabilities used in the optimization procedure simple and well-conditioned. However, one could investigate other choices for this, e.g. replacing the logistic function with another saturating function which might have lower variability. In this way, the horizon formulation in Chapter 5 might be favourable, as the variance of the Gaussian random variable can be made arbitrarily small as opposed to the variance of the GPD.

7

Accelerating Reinforcement Learning with Suboptimal Guidance: Improving the Adaptivity of the Q-filter Approach

The idea of using existing control approaches to enhance RL is something we wanted to look more into given more time for the PhD. We were among other ideas envisioning using the framework presented in Chapter 6 as a framework for guiding a learning controller. Consider the recomputation policy of Chapter 6 that chooses whether to recompute the MPC solution or not: One can instead view this policy as selecting among two different controllers, where one is the guiding controller, $\pi_{\theta^{\mathrm{g}}}^{\mathrm{guide}}$, and the other is the learning controller itself, $\pi_{\theta^{\mathrm{RL}}}^{\mathrm{RL}}$. Denoting the selector policy $\pi_{\theta^{\mathrm{s}}}^{\mathrm{selector}}$, one can assemble a mixture-distribution policy that incorporates both these controllers as follows:

$$\pi(s) = \pi_{\theta^{\mathrm{s}}}^{\mathrm{selector}}(s) \cdot \pi_{\theta^{\mathrm{RL,g}}}^{\mathrm{controller}}(s) = \begin{cases} \pi_{\theta^{\mathrm{RL}}}^{\mathrm{RL}}(s), & \text{if } \pi_{\theta^{\mathrm{s}}}^{\mathrm{selector}}(s) = 0 \\ \pi_{\theta^{\mathrm{g}}}^{\mathrm{guide}}(s), & \text{if } \pi_{\theta^{\mathrm{s}}}^{\mathrm{selector}}(s) = 1 \end{cases} \tag{7.1}$$

where all parts of the framework, i.e. the selector component and the controllers can be made fully differentiable and therefore optimizable. The selector policy is initially biased towards predominantly selecting the guiding controller, and can then autonomously decide when and in what situations to apply the learning controller, initially to collect data for optimization, and later hopefully due to its superiority. If the learned controllers surpass the performance of the guiding controller, one can look into how the learned controller's strategy can be used to enhance the guiding controller as well. With a sufficiently malleable guiding controller (i.e.

its operation can be adjusted locally in the state space), this could be as simple as making the guiding controller match the strategy of the learning controller in the states that the learning controller is superior.

One potential issue with this type of guiding framework is similar to the time-scale issue described for DDPG-style algorithms in Section 2.6. There, the critic provides gradients for improvement of the actor, and should therefore converge much faster than the actor such that it accurately provides information on how to improve the actor. In this guiding approach, the selector policy should converge at a much faster pace than the learning controller changes behaviour, such that the learning controller effectively remains fixed from the perspective of the selector policy, allowing the selector policy to accurately fulfil its objective of identifying the superior policy in each state.

This chapter presents a different approach to guiding learning controllers. It presents the work we performed on using existing control approaches to enhance RL-based controllers, which takes the form of guiding the RL controller towards a viable solution to the control problem which simplifies the exploration phase. This is achieved through a combination of supervised learning and RL, where the learning controller is made to mimic the guiding controller, but only so long as the guiding controller is deemed to be superior to the learning controller. This yields adaptive supervision that allows the learning controller to eventually develop its own strategy and surpass the guiding controller, as opposed to a fully supervised approach. This chapter is based on the following article:

- [36] Eivind Bøhn, Signe Moe, and Tor Arne Johansen. Accelerating reinforcement learning with suboptimal guidance. *IFAC-PapersOnLine*, 53(2): 8090–8096, 2020. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2020.12.2278. 21st IFAC World Congress

The main contribution of this chapter is in enhancing the adaptivity of the Q-filter method for guiding RL controllers.

7.1 Introduction

RL [176] is a field of machine learning concerned with sequential decision-making problems. The theory and methods in RL have produced some impressive results in the last few years, ranging from high-level reasoning tasks such as game-playing [173, 146] to control of fast dynamics such as actuation in robotics [83].

RL has received much attention and interest due to its framework being very general, capable of discovering optimal strategies for systems with complex nonlinear dynamics without access to a model of the system, granted one can define some notion of utility of different states of the system.

This utility measure, called the reward function in the RL framework, is central to both the definition of the problem and the performance of the algorithm. Often, a sparse reward function[1] is preferable, as they are easier to formulate, easier to estimate, and importantly, there is less room for ambiguity and misinterpreting the objective of the task. The obvious downside is however that only a small region of the problem space actually gives a learning signal to the agent, and a significant portion of the training time is spent exploring the state space (often in a random manner) until a minimally viable strategy[2] is found, from which further progress can be made.

Often in robotics and other control applications, one already has a controller, which due to e.g. nonlinearities in the dynamics, uncertainties in model parameters, or strict computational requirements might be suboptimal. This controller could be used to guide the learning controller towards a minimally viable strategy, thereby reducing the long initial exploration phase. Further, guiding in this manner can offer more explainable behaviour from the learning controller, in the sense that its behaviour is close to that of the guiding controller. In turn, this approach may also lead to worse asymptotic performance, as the learning controller is searching for an optimum of the policy space in the vicinity of the guiding controller, while the global optimum might lie somewhere else. Getting to the global optimum might entail climbing several unattractive regions of less optimal policies, which might not be achievable with the usual reinforcement learning optimization instruments.

The most naive implementation of imitation learning (IL), that is, trying to directly copy the pre-existing controller is seldom successful, mainly due to a data distribution mismatch [160]. Furthermore, the potential performance of the learning controller is bounded by the performance of the existing controller. The data distribution problem arises as the data collected by the demonstrator will only consist of a subset of the state space, and will offer no guidance on how to course-correct back into this subset when the learning controller inevitably makes a small deviation from the controller it is imitating. Small errors therefore tend to accumulate into trajectories that stray far from those produced by the demonstrator. Additionally, when combining IL with RL there needs to be some mechanism in place to

[1]A sparse reward function is one that only yields signals in some subset of the state space, e.g. in the subset defined as the set of goal states, and is typically constant elsewhere.

[2]That is, a strategy that consistently is able to reach reward-yielding states

allow the learning controller to make different choices than the original controller when appropriate, in order to exceed the performance of the existing controller.

In this chapter, we tackle the problem of automatically deciding when the learning controller should imitate the pre-existing controller, and when it should develop its own behaviour. We base our method on the concept of the Q-filter [142], which makes the learning controller imitate the pre-existing controller only when the pre-existing controller is deemed to yield a higher reward than what the learning controller would achieve in that situation. The contribution of this work lies in identifying some key issues with the original formulation of the Q-filter and proposing modifications to mitigate these issues, resulting in improved adaptivity to the difficulty of the task and to the proficiency of the pre-existing controller, and thus also improved performance. Moreover, in [142], the authors only suggest using this approach on a pre-gathered demonstration dataset which is mixed with the learner's own data during training. In this chapter we propose that the Q-filter approach can be extended to the setting where one has online access to a guide, and can thus employ the Q-filter guiding on all data gathered.

7.2 Related Work

The DAGGER algorithm [160] is an approach to mitigating the data distribution mismatch problem. A learning controller is trained to mimic the behaviour of some expert controller from a demonstration dataset, and then this learning controller is used in the environment to collect more data. The newly collected data is appended to the training set, and the expert demonstrator is asked to label the new data with its action choices. Our method builds upon the DAGGER framework in the sense that we assume access to an online demonstrator, and iteratively expand the dataset with the demonstrators knowledge. However, we account for suboptimality in the demonstrator, consequently our learning controller is only made to mimic the demonstrator in some selected states and this is only one of its optimization objectives. Deeply AggreVaTeD [175] is a further extension of DAGGER to continuous state and action spaces and nonlinear function approximation with NNs. Both these approaches are pure IL methods and thus do not allow for the agent to surpass the demonstrator.

Another approach to extract knowledge from demonstration data is inverse reinforcement learning (IRL) [144], in which the aim is to recover the reward function that the agent that generated the dataset was trying to optimize, rather than directly learning how to act from the data. Armed with the reward function, the learning agent can reason on how the expert would act even in situations that are not covered in the dataset, or otherwise develop its own behaviour that is different but also in some sense optimal with respect to the reward function.

Deep deterministic policy gradients from demonstrations (DDPGfD) [183] is a method for leveraging demonstration data for RL algorithms in domains with continuous state and action spaces. In this work, human generated demonstrations are included in the replay buffer for which a prioritized sampling technique is used to ensure a suitable mix of demonstration data and self collected data in each training batch. They further use a mix of 1-step and n-step return losses for training the Q-networks. They show increased performance over the demonstrations, as well as over regular DDPG, even when the latter uses hand-crafted shaped rewards and DDPGfD uses sparse rewards.

[142] propose to address the problem of choosing when the learning controller should emulate the demonstrator with the use of the Q-filter. The Q-filter is an indicator function used to select when to apply a behaviour cloning loss on the action chosen by the demonstrator. The filter evaluates to true when the estimated value from the Q-function is higher for the demonstrator action than it is for the action chosen by the learning controller. This provides a natural way to anneal behaviour cloning loss during the training process that is adaptive to the task at hand. Their demonstrations come in the form of a fixed set of trajectories collected by a human demonstrator in a virtual reality version of the environment. In each training batch, a portion of the data is sampled from the regular replay buffer and a portion is sampled from the demonstration buffer, on which behaviour cloning (BC) is applied for the actions selected by the Q-filter. Our method borrows the concept of the Q-filter from this work, suggesting some important adjustments to the concept. Further, their source of demonstrations consists of a fixed dataset, while we assume online access to a guiding controller.

In [193] the authors propose Assisted deterministic policy gradients (AsDDPG), a novel architecture for incorporating a simple controller to guide the initial exploration phase. In their architecture, the critic module has an additional branch that estimates the Q-values of the pre-existing controller's action and the actor's action for the state in question. Based on these estimates the method greedily chooses the one with the highest Q-value, and applies this action as the exploration action during training. In this way, the guiding controller is used to show the actor some primitives it can use to aid exploration. As in our method, online access to the guiding controller is assumed, but the Q-filter is applied to select actions for exploration instead of to shape the gradients of the optimization procedure.

7.3 Improving the Adaptivity of the Q-filter

7.3.1 Problem Description

In our setup, we assume online access to a pre-existing sub-optimal controller which is capable of reaching goal states from a nonempty set of initial states. We also assume that we have access to the Q-function of this controller, denoted by Q^G. However, we do not assume prior access to demonstration trajectories from this controller.

7.3.2 Proposed Method

We modify the actor objective function of DDPG style algorithms (2.43) by adding a BC loss term (7.2) weighted by λ_{BC}, and as in [142] we selectively apply this loss with the indicator function conditioned on the Q-filter:

$$J^{\text{BC}}(\theta|s,a) = \|a - \pi_\theta(s)\|_2 \, \mathbb{1}_{Q^G(s,a) > Q^{\pi_\theta}_{\theta Q}(s,\pi_\theta(s))} \tag{7.2}$$

$$J(\theta) = \max_\theta \mathbb{E}_{s,\bar{a}\sim\mathcal{B}} \left[Q^{\pi_\theta}_{\theta Q}(s,\pi_\theta(s)) - \lambda_{\text{BC}} J^{\text{BC}}(s,\bar{a}) \right] \tag{7.3}$$

$$\text{where } \mathbb{1}_{A>B} = \begin{cases} 1 & \text{if } A > B \\ 0 & \text{otherwise} \end{cases} \tag{7.4}$$

$$Q^{\pi_{\theta_0}} \leftarrow Q^G \tag{7.5}$$

where $\bar{a}$ is the action chosen by the guiding controller in the corresponding state. Note that we employ the TD3 algorithm, a derivation of the DDPG algorithm that shares the same objective for the actor. The suggested approach is therefore equally applicable to all variants of the DDPG algorithm. We use the online availability of the pre-existing controller to apply the BC loss term to all samples in the batch. This does not add considerable overhead, as the pre-existing controller's action can be evaluated once and then saved to the replay buffer alongside the other data.[3] In this work we suggest modifying the original formulation of the Q-filter by replacing the left-hand side of the condition with the pre-existing controller's own estimated Q-function, which is kept static throughout the training process. As the TD3 algorithm has two separate Q-networks, one can look at different ways of

[3]Training with a guiding controller increases the wall clock time of the training process by about 5% compared with normal training in our experiments.

combining these to make up the Q-filter. In this work we have chosen to only use the Q-network that is used in the optimization objective (2.47) as a basis for the comparison.

In the original Q-filter formulation of [142], the same value function $Q_{\theta Q}$ is used to estimate the Q-value of both the action chosen by the agent and the action chosen by the guide. The motivation behind the proposed modifications are therefore based on two key insights:

First, the quantities that are compared, $Q^{\pi_\theta}_{\theta Q}(s_t, \pi_\theta(s_t))$ vs $Q^{\pi_\theta}_{\theta Q}(s_t, \bar{a}_t)$, are of similar magnitude especially for problems with a sparse reward function and for problems with low sampling time as is common in control applications.

$$Q_{\theta Q}(s,a) = R(s,a) + \gamma Q_{\theta Q}\left(\mathcal{T}(s,a), \pi_\theta(\mathcal{T}(s,a))\right) \tag{7.6}$$

$$\approx R(s,a) + G(\tau), \;\; \tau = (\mathcal{T}(s,a), \pi_\theta(\mathcal{T}(s,a)), \dots) \tag{7.7}$$

As shown in (7.6) and (7.7), the Q-value estimates two quantities: the immediate reward and the return from the next state, e.g. the total discounted reward over the trajectory produced by the controller from the next state. With low sampling time, each action is applied only for a short time, and each state in a sequence is similar to the previous even for very different actions, i.e.:

$$\lim_{\Delta_t \to 0} |\mathcal{T}(s_t, a_t) - s_{t+1}| = 0 \;\; \forall a_t \in \mathcal{A} \tag{7.8}$$

where Δ_t is the step size (sampling time) of the MDP. The two trajectories that are compared are generated by the same controller from similar initial states and thus have similar returns. The immediate rewards for the two controllers' actions are also similar, especially when using sparse rewards where they can only differ if the current state is in close proximity to the border of the reward-yielding subset of the state space, and the next state of the system crosses the border as a result of one action but not of the other. Great accuracy is therefore needed to correctly assess the superior action. By using the guiding controllers' own Q-function on one side of the inequality in (7.2) and the agent's Q-function on the other we compare the total value of two trajectories produced by different controllers, thus the difference in the return is greatly increased, allowing for easier discrimination.

Second, as explained the Q-function needs to be highly accurate to properly assess the values of the actions. The comparison will therefore be greatly impacted by the randomness inherent in an unconverged neural network. Since the actors' objective (2.47) in the TD3 algorithm is precisely to find the maximization of actions over the Q-function, it will quickly learn to optimize the unconverged Q-network, and therefore seemingly (most times erroneously) offer better options than the guiding controller. The action comparison performed in the Q-filter is consequently highly stochastic for much of the learning process. The Q-filter formulation in [142] will henceforth be referred to as the naive method.

In practice, the naive Q-filter therefore tends to prefer the guiding controller's actions infrequently in the beginning and converges to select the guiding controller as superior with a non-zero frequency. The aim of a guiding approach is to have the learning controller emulate the pre-existing controller to a large degree in the beginning, and then have the BC loss taper off to allow the learning controller to surpass the pre-existing controller. With the original formulation of the Q-filter however, the BC loss does not seem to be strategically applied.

We obtain Q^G by running the TD3 algorithm with the guiding controller as the actor until the objective (2.47) stabilizes. Since the actor in this case is the static guiding controller, all data is on-policy and n-step updates can be used to learn the Q-function for faster and more accurate convergence.[4] See Section 7.5 for a discussion on other ways of obtaining Q^G. In order for the actor's and the guiding controller's Q-functions to have comparable magnitudes, the actor's Q-network is initialized to that of the guiding controller (7.5). This also provides a starting point that should be closer to optimal than a random initialization.

7.4 Experiments

We train and evaluate our method on the OpenAI Gym Fetch environments [152], which are based on the MuJoCo physics simulator [178]. The FetchReach environment is easy to solve for the RL algorithms even without the use of a guiding controller, and as such has been excluded from the experiments. For each environment, a test set consisting of 100 random initial states and goal states has been constructed, and the agents are evaluated on this set every 20k time steps. We evaluate our method against the original formulation of the Q-filter, against an approach where the weight of the BC loss is linearly decayed with time steps, as well as an unguided approach using TD3 + HER. We run each experiment with five different random seeds and show mean results with one standard deviation confidence bounds.

[4]Convergence was achieved after less than 100k time steps in our experiments for all environments.

7.4.1 Guiding Controllers

The guiding controllers are handcrafted proportional controllers with some conditional logic, e.g. move hand over block, then lower hand and grip block etc. The guiding controllers exhibit varying proficiency levels, from a controller capable of achieving any goal in the test set for the pick-and-place environment to a controller achieving merely 21% of the goals in the slide task.

7.4.2 Configuration

Due to limited computational resources, we employed the TD3 algorithm instead of DDPG, as convergence of the latter is often dependent on running several actors in parallel. To increase sample efficiency we use HER with the future goal selection strategy, generating four imagined goals for every real experience sample. We use the Stable-Baselines [89] implementation of the algorithms, running on an I7-9700k 8-core CPU and an RTX 2070 GPU.

Following the original paper introducing the Fetch environments [6], we use a decay factor of 0.98 and clip the regression Q-targets (2.44) to the ranges possible in the given environment, and also bootstrap on environment termination due to time limit. We use the hyperparameters from TD3, with an actor and critic consisting of two fully-connected hidden layers of 400 and 300 nodes respectively, a learning rate of 0.001 for both actor and critic, and a replay buffer size of 10^6. Parameter updates are performed every 1000 time steps of the environment for 1000 iterations, with a batch size of 100. We apply an $\mathcal{L}_2$ regularization term with strength 0.01 to the pre-activation values of the actor's output layer, in order to mitigate vanishing gradients issues. The BC loss term weighting factor λ_{BC} (7.3) is set to 2.

The linear schedule is set to be fully decayed after 500k time steps, with an initial value equal to that of the other methods. This schedule is not optimal for all environments, but a static schedule was chosen to highlight some of the issues with this approach.

7.4.3 Ablation Studies

It is conceivable that any improvements of our method stem mainly from the knowledge encoded into Q^G and the initialization of Q_{θ^Q} to this. A pretrained Q^G already contains information about which action would maximize the rewards when further following the same action choices as the pre-existing controller at each state. The optimization procedure (2.47) implemented for the actor could take advantage of this information to achieve some of the same benefits that the BC loss offers. To test this hypothesis, we ran one version of our method without the BC loss term, and one version where the naive implementation of the Q-filter is also initialized to Q^G as in (7.5).

7.5 Results and Discussion

7.5.1 Results of Experiments

The right-most column of Figure 7.1 shows the fraction of samples in each minibatch for which the Q-filter is activated. This figure illustrates the unwanted behaviours of the naive Q-filter as described in Section 7.3: it typically starts low and converges to some value in the range of 0.3 - 0.5 for all environments even with varying proficiency of the guide. Our method on the other hand delivers on its promises of adaptivity: it copies a large portion of the actions produced by the optimal guiding controller in the pick-and-place environment, selects only some of the actions produced by the suboptimal pre-existing controllers in the slide and push environments, and decreases the frequency of imitation as the agent improves. The two Q-filter approaches have the same asymptotic performance for the push and slide environments, but our Q-filter method has considerable more success in the pick-and-place environment. The linear method matches the performance of the adaptive methods for the push environment, but falls a bit short in the pick-and-place environment, and is only able to find any success for a single seed in the most difficult slide task. The linear scheduling method seems to be in general the method with the quickest initial learning, due to indiscriminately imitating all actions from the guiding controller at the start, while the two Q-filter approaches seem to share a similar timing of when reward signals are converted into policy improvements.

Note that the unguided baseline agent using only TD3 + HER is unable to achieve any success at all in the time frame considered here, except for a single seed for the FetchSlide environment. By comparing with the guided methods in Figure 7.1, one can clearly see how much one can accelerate the learning curve and get better consistency by incorporating knowledge from existing solutions to the task in the training process of RL agents. Our results for the unguided TD3 + HER method are considerably worse than the DDPG + HER results reported in [6] and [152], in which they show that these methods are able to find working approaches without guidance in the Fetch environments, albeit at a much slower pace than the guided approaches in this chapter. This discrepancy might stem from their work leveraging extensive computational resources to run many data-collecting agents in parallel, which generates more uncorrelated and diverse exploration data, and that these methods are reliant on this trick. We also did not perform specific hyperparameter optimization for the unguided approach, but rather used hyperparameters reported in previous works as described in Section 7.4.

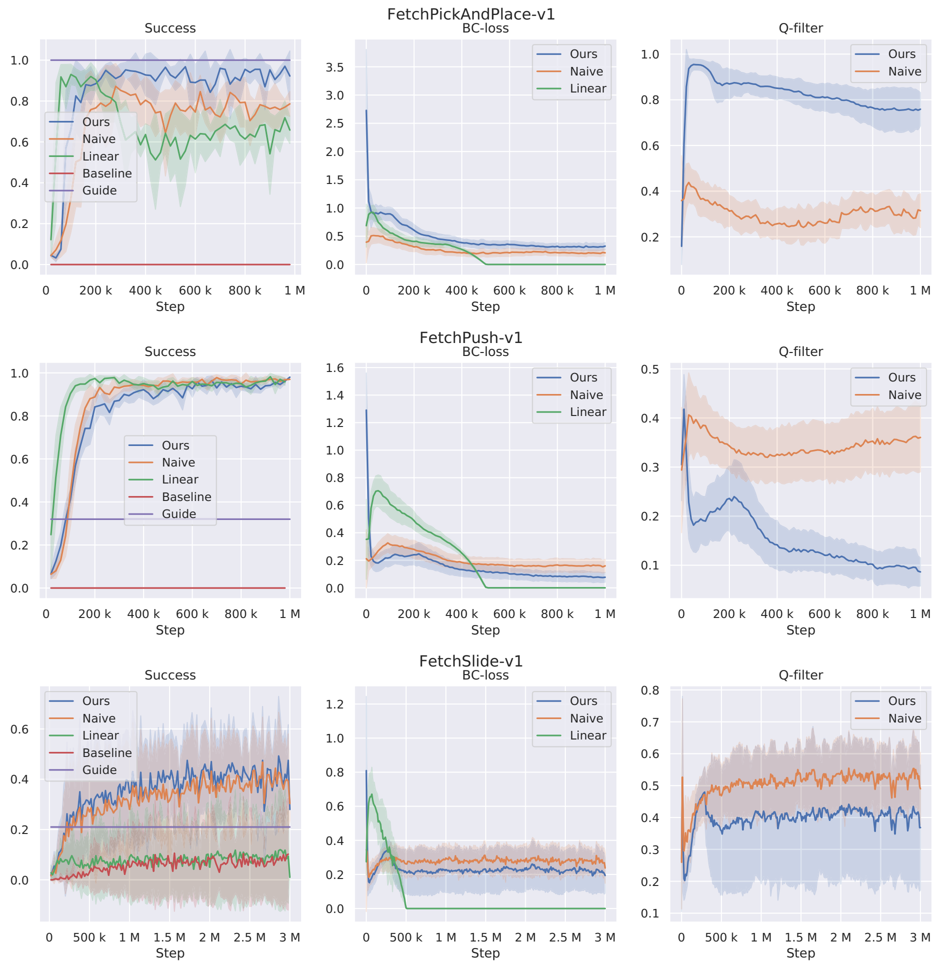


Figure 7.1: Results of experiments: The graphs show for each environment the success rate on the test set, the BC loss, and the fraction of actions for which the indicator function evaluates to true in each minibatch.

The results of the ablation experiments are shown in Figure 7.2. These experiments show that initializing to Q^G is not enough in and of itself to explain the perform-

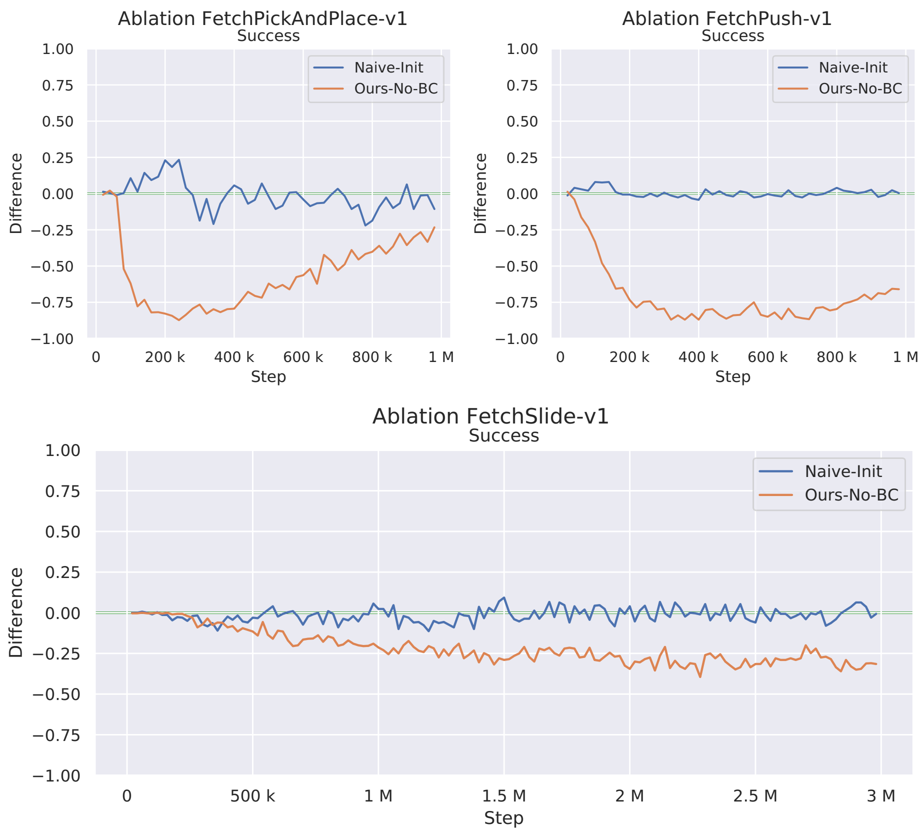


Figure 7.2: The difference in performance when including initialization to Q^G in the naive method and when removing the BC loss term from our method, compared to its counterpart in Figure 7.1 as a baseline.

ance differences shown in Figure 7.1. Our method is significantly degraded by the removal of the BC loss term, while the naive method is largely unaffected by the addition of the initialization procedure. These results suggest that the observed performance gains from our method stem in large part from an improved comparison in the Q-filter. However, since the model without the BC term in the ablation experiments and the unguided model in Figure 7.1 differ only in initialization, it is clear that the algorithm is capable of utilizing the information contained in Q^G in some way to achieve more success.

7.5.2 Evaluation of Results

While our formulation of the Q-filter does lead to more deliberate application of the BC loss, it does introduce a credit assignment problem: Since the two Q-functions that are compared correspond to two different controllers, it is unclear which actions in the future trajectory generated by the controllers that are responsible for the observed difference in value. Imitating only the initial action is therefore a potential source of error, but introducing some randomness and perturbation to the training process in RL is a known measure to avoid local minima, and therefore this might not be a significant issue.

Even though the linear scheduling method is less complex than the other methods it can sometimes achieve similar or better performance in terms of time at which the policy starts improving and asymptotic performance. The slide environment clearly illustrates that this approach requires careful tuning, as the method is in this case unable to represent a policy capable of achieving any goals for most seeds. The Q-filters on the other hand are adaptive and provide reasonable performance for any task with little tuning, and in particular, our method will imitate a good guiding controller to a larger degree than a bad guiding controller, unlike the linear scheduling method. Our version of the Q-filter does however require additional information in the form of Q^G, and the linear method might therefore be preferable in some scenarios despite its limitations.

The Q-function of the pre-existing controller need not be obtained in a complicated manner such as Q-learning. Often, one has some historical data of usage of this controller, which one could use to estimate the expected sum of rewards in an entirely offline manner. Furthermore, one might not need a function approximator as complicated as a neural network for this purpose, and some simpler methods such as linear regression might be sufficient. This would remove the possibility of directly initializing the Q-function of the actor to that of the pre-existing controller, but one could simply pretrain the Q-network in a supervised manner on outputs from the pre-existing controller's Q-function to achieve much of the same effect.

7.6 Conclusion

We have shown that our method is capable of accelerating learning using guiding controllers with a wide spread of proficiency levels. An important further work is looking into how the optimality of the guiding controller affects the rate of convergence and asymptotic performance of the learning controller.

Having a pre-existing controller available online opens up many possibilities. If one knows the stability properties of this controller, one can have the agent explore freely while still in the region of attraction (RoA) of the guiding controller, and then have the guiding controller assume control and stabilize the system when necessary. In this way, one can achieve a form of safe training, in the sense that only safe regions of the state space are visited, and the risk of damage to the system is minimized. An interesting further work is a study on what fraction of actions the agent needs to control itself for learning to be successful in such a scenario.

Part II

Reinforcement Learning for Attitude Control of Fixed-Wing UAVs: An Application of Reinforcement Learning Control in the Physical World

8

Introduction

RL, and artificial intelligence in general, has become one of the most active fields of research in the last few years, with tens of thousands of articles published every year just on the topic of RL alone. This interest is at least to some degree warranted, as the framework of RL is very general, and when combined with the empirically demonstrated expressive power of NNs, DRL promises to solve many problems previously unimaginable, such as beating humans at the board game of Go [166]. However, the vast majority of the research on RL and results presented are performed exclusively in simulated environments, and there is a clear lack of commercial application of RL systems. This part of the thesis presents the work undertaken in the PhD where we aim to contribute to the field of RL by demonstrating an RL-based, field-validated control application that is of particular importance to Norway, namely control of fixed-wing UAVs.

Norway has an extensive coastline and vast offshore resources, thus, the sea has always been an important domain for Norway. In fact, its two largest industries in terms of revenue are the oil and gas sector and the seafood sector. Protecting these interests require a great deal of monitoring and considering the size and expanse of these areas, a key enabling technology are autonomous fixed-wing UAVs. These are an excellent tool in the maritime environment as they can cover large geographical areas and carry multiple sensors and payloads, performing tasks such as search-and-rescue, surveillance and transportation. Due to this importance, NTNU has established the Unmanned Aerial Vehicle laboratory (UAV-Lab), which among other research has greatly contributed to the modelling of the Skywalker X8 fixed-wing UAV [80]. A sophisticated model enables simulation of the aircraft, which facilitates the development of learning controllers for UAVs in a safe manner.

With this starting point, we targeted low-level dynamic RL-based control of the Skywalker X8 aircraft shown in Figure 8.1. An open-source python flight simulator [34, 35] was developed based on the previous modeling efforts, and two articles on this subject was published, constituting the two chapters of this part of the thesis. Chapter 9 presents the first article we published on this topic, in which we demonstrate a proof of concept in a simulation environment of the suitability of RL for attitude control of fixed-wing UAVs. In Chapter 10 we extend the work presented in the preceding chapter, targeting control of the real UAV in the field, and develop a new data-efficient method to this end.

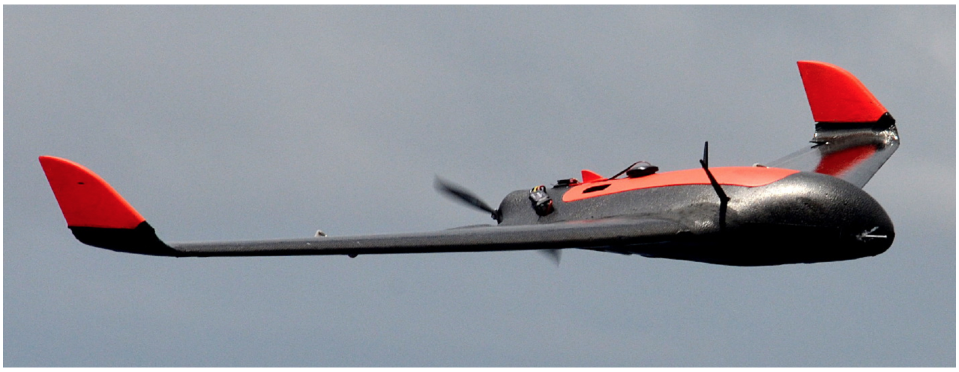


Figure 8.1: The Skywalker X8 Fixed-Wing UAV in flight.

9

Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs: A Proof of Concept

This chapter is based on the following article:

- [26] Eivind Bøhn, Erlend M Coates, Signe Moe, and Tor Ame Johansen. Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 523–533. IEEE, 2019

The aforementioned article was the first to propose DRL for the attitude control problem of fixed-wing UAVs. The main contributions of this chapter lie in proposing the use of DRL for dynamic control of fixed-wing UAVs, contextualizing the use of RL for flight control, detailing the simulated UAV model, and in presenting convincing data on the suitability of RL for dynamic control of UAV based on simulations. In particular, we show that the DRL controller is capable of successfully controlling the UAV in all the same operating conditions as the existing state-of-the-art approach, while being more robust towards unmeasured disturbances in the form of wind and turbulence.

9.1 Introduction

UAVs are extensively employed to increase safety and efficiency in a plethora of tasks such as infrastructure inspection, forest monitoring, and search and rescue missions. Many tasks can however not be accomplished fully autonomously, due to several limitations of autopilot systems. Low-level stabilization of the UAV's attitude provided by the inner control loops is increasingly difficult as the attitude and airspeed deviate from stable, level conditions, due to various nonlinearities.

The outer control layers providing path planning and guidance has to account for this and settle for non-agile and safe plans. Equipping the autopilot with the stabilization skills of an experienced pilot would allow fully autonomous operation in turbulent or otherwise troublesome environments, such as search and rescue missions in extreme weather conditions, as well as increasing the usefulness of the UAV by for instance allowing the UAV to fly closer to its targets for inspection purposes.

Autopilots for fixed-wing UAVs are typically designed using cascaded single-variable loops under assumptions of decoupled longitudinal and lateral motion, using classical linear control theory [19]. The dynamics of fixed-wing aircraft including UAVs are however strongly coupled and nonlinear. Nonlinear terms in the equations of motion include kinematic nonlinearities (rotations and Coriolis effects), actuator saturation and aerodynamic nonlinearities, which are also uncertain and difficult to model. The decoupled and linear designs are reliable and well-tested for nominal flight, but also requires conservative safety limits in the allowable range of flight conditions and maneuvers (flight envelope protection), because linear controllers applied to nonlinear systems typically result in a limited region of attraction [100]. This motivates the use of state-of-the-art nonlinear control algorithms.

Examples of nonlinear control methods applied to UAVs include gain scheduling [67], linear parameter varying (LPV) control [161], dynamic inversion (feedback linearization) [98], adaptive backstepping [156], sliding mode control [41], nonlinear model predictive control [129], nonlinear H-infinity control [65], dynamic inversion combined with mu-synthesis [135], model reference adaptive control [110] and L1 adaptive control [97]. Automated agile and aerobatic maneuvering is treated in [112] and [33]. Several of these methods require a more or less accurate aerodynamic model of the UAV. A model-free method based on fuzzy logic can be found in [106]. Fuzzy control falls under the category of intelligent control systems, which also includes the use of neural networks. An adaptive backstepping controller using a neural network to compensate for aerodynamic uncertainties is given in [111], while a genetic neuro-fuzzy approach for attitude control is taken in [48]. The state of the art in intelligent flight control of small UAVs is discussed in [163].

Control of small UAVs requires making very fast control decisions with limited computational power available. Sufficiently sophisticated models incorporating aerodynamic nonlinearities and uncertainties with the necessary accuracy to enable robust real-time control may not be viable under these constraints. Biology suggests that a bottom-up approach to control design might be a more feasible option. Birds perform elegant and marvellous maneuvers and are able to land abruptly with pinpoint accuracy utilizing stall effects. Insects can hover and zip around with astonishing efficiency, in part due to exploiting unsteady, turbulent aerodynamic flow effects [19]. These creatures have developed the skills not through careful consideration and modelling, but through an evolutionary trial-and-error process driven by randomness, with mother nature as a ruthless arbiter of control design proficiency. In similar bottom-up fashion, machine learning (ML) methods have shown great promise in uncovering intricate models from data and representing complex nonlinear relations from its inputs to its outputs. ML can offer an additional class of designs through learning that is not easily accessible through first-principles modelling, exhibiting antifragile properties where unexpected events and stressors provide data to learn and improve from, instead of invalidating the design. It can harbour powerful predictive powers allowing proactive behaviour while meeting the strict computation time budget in fast control systems.

RL [176] is a subfield of ML concerned with how agents should act in order to maximize some measure of utility, and how they can learn this behaviour from interacting with their environment. Control has historically been viewed as a difficult application of RL due to the continuous nature of these problems' state and action spaces. Furthermore, the task has to be sufficiently nonlinear and complex for RL to be an appropriate consideration over conventional control methods in the first place. To apply tabular methods one would have to discretize and thus suffer from the consequences of the curse of dimensionality from a discretization-resolution appropriate to achieve acceptable control. The alternative to tabular methods requires function approximation, which has to be sophisticated enough to handle the dynamics of the task while having a sufficiently stable and tractable training process to offer convergence. NNs are one of few models which satisfy these criteria: they can certainly be made expressively powerful enough for many tasks, but achieving a stable training phase can be a great challenge. Advances in computation capability and algorithmic progress in RL, reducing the variance in parameter updates, have made deep neural networks (DNNs) applicable to RL algorithms, spawning the field of DRL. DNNs in RL algorithms provide end-to-end learning of appropriate representations and features for the task at hand, allowing algorithms to solve classes of problems previously deemed unfit for RL. DRL has been applied to complex control tasks such as motion control of robots [201] as well as other tasks where formalizing a strategy with other means is difficult, e.g. game playing [137].

A central challenge with RL approaches to control is the low sample efficiency of these methods, meaning they need a large amount of data before they can become proficient. Allowing the algorithm full control to learn from its mistakes is often not a viable option due to operational constraints such as safety, and simulations are therefore usually the preferred option. The simulation is merely an approximation of the true environment. The model errors, i.e. the differences between the simulator and the real world, is called the reality gap. If the reality gap is small, then the low sample efficiency of these methods is not as paramount, and the agent might exhibit great skill the first time it is applied to the real world.

The premise of this research was to explore the application of RL methods to low-level control of fixed-wing UAVs, in the hopes of producing a proof-of-concept RL controller capable of stabilizing the attitude of the UAV to a given attitude reference. To this end, an OpenAI Gym environment [32] with a flight simulator tailored to the Skywalker X8 flying wing was implemented, in which the RL controller is tasked with controlling the attitude (the roll and pitch angles) as well as the airspeed of the aircraft. Aerodynamic coefficients for the X8 are given in [82]. The flight simulator was designed with the goal of being valid for a wide array of flight conditions, and therefore includes additional nonlinear effects in the aerodynamic model. The software has been made openly available [34, 35]. Key factors impacting the final performance of the controller as well as the rate of progression during training were identified. To the best of the authors' knowledge, this is the first reported work to use DRL for attitude control of fixed-wing UAVs.

9.2 Related Work

In general, the application of RL to UAV platforms has been limited compared to other robotics applications, due to data collection with UAV systems carrying significant risk of fatal damage to the aircraft. RL have been proposed as a solution to many high-level tasks for UAVs such as the higher level path planning and guidance tasks, alongside tried and tested traditional controllers providing low-level stabilization. In the work of Gandhi et al. [64] a UAV is trained to navigate in an indoor environment by gathering a sizable dataset consisting of crashes, giving the UAV ample experience of how NOT to fly. In [87], the authors tackle the data collection problem by constructing a pseudo flight environment in which a fixed-wing UAV and the surrounding area is fitted with magnets, allowing for adjustable magnetic forces and moments in each degree of freedom (DOF). In this way, the UAV can be propped up as one would do when teaching a baby to walk, and thereby experiment without fear of crashing in a setting more realistic than simulations.

Imanberdiyev et al. [94] developed a model-based RL algorithm called TEXPLORE to efficiently plan trajectories in unknown environments subject to constraints such as battery life. In [202], the authors use an MPC to generate training data for an RL controller, thereby guiding the policy search and avoiding the potentially catastrophic early phase before an effective policy is found. Their controller generalizes to avoid multiple obstacles, compared to the singular obstacle avoided by the MPC in training, does not require full state information like the MPC does, and is computed at a fraction of the time. With the advent of DRL, it has also been used for more advanced tasked such as enabling intelligent cooperation between multiple UAVs [91], and for specific control tasks such as landing: Polvara et al. [153] proposes to employ a hierarchy of deep Q-networks (DQNs) to map from camera data to control input for landing UAVs autonomously in a wide range of conditions. RL algorithms have also been proposed for attitude control of other autonomous vehicles, including satellites [195] and underwater vehicles. Carlucho et al. [40] applies an actor-critic DRL algorithm to low-level attitude control of an autonomous underwater vehicle (AUV), similar to the proposed method in this chapter. They also obtain experimental data from testing their method on the Nessie VII AUV and find satisfactory results in terms of the transferability from simulation training to real-world performance.

Of work addressing problems more similar in nature to the one in this chapter, i.e. low-level attitude control of UAVs, one can trace the application of RL methods back to the works of Bagnell and Schneider [13] and Ng et al. [145], both focusing on helicopter UAVs. Both employed methods based on offline learning from data gathered by an experienced pilot, as opposed to the online self-learning approach proposed in this chapter. The former focuses on control of a subset of the controllable states while keeping the others fixed, whereas the latter work extends the control to all six degrees of freedom. In both cases, the controllers exhibit control performance exceeding that of the original pilot when tested on real UAVs. In [2], the latter work was further extended to include difficult aerobatic maneuvers such as forward flips and sideways rolls, significantly improving upon the state-of-the-art. Cory and Tedrake [46] presents experimental data of a fixed-wing UAV perching maneuver using an approximate optimal control solution. The control is calculated using a value iteration algorithm on a model obtained using nonlinear function approximators and unsteady system identification based on motion capture data. Bou-Ammar et al. [30] compared an RL algorithm using fitted value iteration (FVI) for approximation of the value function, to a non-linear controller based on feedback linearization, on their proficiency in stabilizing a quadcopter UAV after an input disturbance. They find the feedback-linearized controller to have superior performance. Recently, Koch et al. [101] applied three state-of-the-art RL algorithms to control the angular rates of a quadcopter UAV. They found PPO to perform the best of the RL algorithms, outperforming the proportional-integral-derivative (PID) controller on nearly every metric.

9.3 UAV Model

Following [19], the UAV is modeled as a rigid body of mass m with inertia tensor $\boldsymbol{I}$ and a body frame $\{b\}$ rigidly attached to its center of mass, moving relative to a north-east-down (NED) frame assumed to be inertial $\{n\}$. The attitude is represented using Euler angles $\boldsymbol{\Theta} = [\phi \;\; \theta \;\; \psi]^T$, where ϕ, θ, ψ are the roll, pitch and yaw angles respectively. The time evolution of the position $\boldsymbol{p} = [x \;\; y \;\; z]^T$ and attitude $\boldsymbol{\Theta}$ of the UAV is governed by the kinematic equations

$$\dot{\boldsymbol{p}} = \boldsymbol{R}_b^n(\boldsymbol{\Theta})\boldsymbol{v} \tag{9.1}$$

$$\dot{\boldsymbol{\Theta}} = \boldsymbol{T}_\Theta(\boldsymbol{\Theta})\boldsymbol{\omega} \tag{9.2}$$

The matrix $\boldsymbol{T}_{\boldsymbol{\Theta}}(\boldsymbol{\Theta})$ relating the angular velocity $\boldsymbol{\omega} = [p \;\; q \;\; r]^T$ to the time derivative of the Euler angles can be calculated from $\boldsymbol{\Theta}$ as follows:

$$\boldsymbol{T}_{\boldsymbol{\Theta}}(\boldsymbol{\Theta}) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \tag{9.3}$$

The rotation matrix $\boldsymbol{R}_b^n$ transforms vectors from $\{b\}$ to $\{n\}$ and is given by the following sequence of basic rotations:

$$\boldsymbol{R}_b^n(\boldsymbol{\Theta}) = \boldsymbol{R}_{z,\psi}\boldsymbol{R}_{y,\theta}\boldsymbol{R}_{x,\phi} \tag{9.4}$$

The rates of change of the body-fixed velocities $\boldsymbol{v}$ and angular velocities $\boldsymbol{\omega}$ are given by the Newton-Euler equations of motion:

$$m\dot{\boldsymbol{v}} + \boldsymbol{\omega} \times m\boldsymbol{v} = \boldsymbol{R}_b^n(\boldsymbol{\Theta})^T m\boldsymbol{g}^n + \boldsymbol{F}_{prop} + \boldsymbol{F}_{aero} \tag{9.5}$$

$$\boldsymbol{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega} = \boldsymbol{M}_{prop} + \boldsymbol{M}_{aero} \tag{9.6}$$

$\boldsymbol{g}^n = [0 \;\; 0 \;\; g]^T$, where g is the acceleration of gravity. Apart from gravity, the UAV is affected by forces and moments due to aerodynamics and propulsion, that are explained in the next sections. All velocities, forces and moments are represented in the body frame.

9.3.1 Aerodynamic Forces and Moments

The UAV is flying in a wind field decomposed into a steady part $\boldsymbol{v}_{w_s}^n$ and a stochastic part $\boldsymbol{v}_{w_g}^b$ representing gusts and turbulence. The steady part is represented in $\{n\}$, while the stochastic part is represented in $\{b\}$. Similarly, rotational disturbances are modelled through the wind angular velocity $\boldsymbol{\omega}_w$. The relative (to the surrounding air mass) velocities of the UAV is then defined as:

$$\boldsymbol{v}_r = \boldsymbol{v} - \boldsymbol{R}_b^n(\boldsymbol{\Theta})^T \boldsymbol{v}_{w_s} - \boldsymbol{v}_{w_g} = \begin{bmatrix} u_r \\ v_r \\ w_r \end{bmatrix} \tag{9.7}$$

$$\boldsymbol{\omega}_r = \boldsymbol{\omega} - \boldsymbol{\omega}_w = \begin{bmatrix} p_r \\ q_r \\ r_r \end{bmatrix} \tag{9.8}$$

From the relative velocity we can calculate the airspeed V_a, angle of attack α and sideslip angle β:

$$V_a = \sqrt{u_r^2 + v_r^2 + w_r^2} \tag{9.9}$$

$$\alpha = \tan^{-1}\left(\frac{w_r}{u_r}\right) \tag{9.10}$$

$$\beta = \sin^{-1}\left(\frac{v_r}{V_a}\right) \tag{9.11}$$

The stochastic components of the wind, given by $\boldsymbol{v}_{w_g} = [u_{w_g} \;\; v_{w_g} \;\; w_{w_g}]^T$ and $\boldsymbol{\omega}_w = [p_w \;\; q_w \;\; r_w]^T$ are generated by passing white noise through shaping filters given by the Dryden velocity spectra [180, 130], a realization of which is illustrated in Figure 9.1 for severe turbulence conditions.

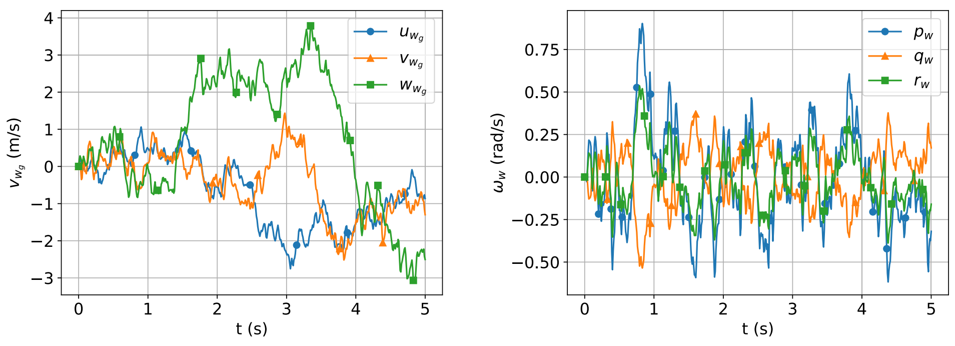
The aerodynamic forces and moments are formulated in terms of aerodynamic coefficients $C_{(*)}$ that are, in general, nonlinear functions of α, β and $\boldsymbol{\omega}_r$, as well as the right and left elevon surface deflections δ_r and δ_l, which acts as control inputs. Note that there is no rudder on the Skywalker X8 UAV.

$$\boldsymbol{F}_{aero} = \boldsymbol{R}_w^b(\alpha, \beta) \begin{bmatrix} -D \\ Y \\ -L \end{bmatrix} \tag{9.12}$$

$$\begin{bmatrix} D \\ Y \\ L \end{bmatrix} = \frac{1}{2}\rho V_a^2 S \begin{bmatrix} C_D(\alpha, \beta, q_r, \delta_r, \delta_l) \\ C_Y(\beta, p_r, r_r, \delta_r, \delta_l) \\ C_L(\alpha, q_r, \delta_r, \delta_l) \end{bmatrix} \tag{9.13}$$

$$\boldsymbol{M}_{aero} = \frac{1}{2}\rho V_a^2 S \begin{bmatrix} bC_l(\beta, p_r, r_r, \delta_r, \delta_l) \\ cC_m(\alpha, q_r, \delta_r, \delta_l) \\ bC_n(\beta, p_r, r_r, \delta_r, \delta_l) \end{bmatrix} \tag{9.14}$$

ρ is the density of air, S is the wing planform area, c is the aerodynamic chord, and b the wingspan of the UAV. The rotation matrix transforming the drag force



(a) Linear velocity **(b)** Angular velocity

Figure 9.1: An example of a realization of the Dryden turbulence model for severe turbulence conditions.

D, side force Y and lift force L from the wind frame to the body frame is given by:

$$\boldsymbol{R}_w^b(\alpha,\beta) = \begin{bmatrix} \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta) & -\sin(\alpha) \\ -\sin(\beta) & \cos(\beta) & 0 \\ \cos(\beta)\sin(\alpha) & \sin(\alpha)\sin(\beta) & \cos(\alpha) \end{bmatrix} \tag{9.15}$$

Aerodynamic coefficients are taken from [82], based on wind tunnel experiments of the Skywalker X8 flying wing. The model has similar structure to the linear coefficients in [19], but has added quadratic terms in α and β to the drag coefficient C_D. In addition, C_D is quadratic in the elevator deflection δ_e. In this chapter, as an attempt to extend the range of validity to a greater range of α and β-values, lift, drag and pitch moment coefficients are made nonlinear using Newtonian flat plate theory from [19] and [79].

9.3.2 Propulsion Forces and Moments

Assuming the propeller thrust is aligned with the x-axis of $\{b\}$, we can write

$$\boldsymbol{F}_{prop} = \begin{bmatrix} T_p \\ 0 \\ 0 \end{bmatrix} \tag{9.16}$$

Where the propeller thrust T_p is given by [58] as presented in [18]:

$$V_d = V_a + \delta_t(k_m - V_a) \tag{9.17}$$

$$T_p = \frac{1}{2}\rho S_p C_p V_d \left(V_d - V_a\right) \tag{9.18}$$

V_d is the discharge velocity of air from the propeller, k_m is a motor constant, S_p is the propeller disc area, and C_p is an efficiency factor. $\delta_t \in [0,1]$ is the throttle. The propeller moments are given by

$$\boldsymbol{M}_{prop} = \begin{bmatrix} -k_Q(k_\Omega \delta_t)^2 \\ 0 \\ 0 \end{bmatrix} \tag{9.19}$$

where k_Ω and k_Q are constants. Gyroscopic moments are assumed negligible.

9.3.3 Actuator Dynamics and Constraints

Note that collective and differential elevon deflections can be mapped to "virtual" aileron and elevator deflections δ_a and δ_e which are used as input to the aerodynamic model.

$$\begin{bmatrix} \delta_a \\ \delta_e \end{bmatrix} = \begin{bmatrix} -0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} \delta_r \\ \delta_l \end{bmatrix} \tag{9.20}$$

Constraints and dynamics are imposed on the transformed aileron and elevator deflections. $\delta_a \in [-30\frac{\pi}{180}, 30\frac{\pi}{180}]$ and $\delta_e \in [-30\frac{\pi}{180}, 35\frac{\pi}{180}]$. Denoting commands with superscript c, the control surface dynamics are given by the following second order transfer function:

$$\frac{\delta_{(*)}(s)}{\delta^c_{(*)}(s)} = \frac{100^2}{s^2 + 100\sqrt{2}s + 100^2} \tag{9.21}$$

The throttle dynamics are given by the following first order transfer function:

$$\frac{\delta_t(s)}{\delta^c_t(s)} = \frac{1}{0.2s + 1} \tag{9.22}$$

9.4 Learning an On-Policy Attitude Controller using PPO

PPO was the chosen RL algorithm for the attitude controller for several reasons: first, PPO was found to be the best performing algorithm for attitude control of quadcopters in [101], and secondly, PPO's hyperparameters are robust for a large variety of tasks, and it has high performance and low computational complexity.

The control objective is to control the UAV's attitude, so a natural choice of controlled variables are the roll, pitch and yaw angles. However, the yaw angle of the aircraft is typically not controlled directly, but through the yaw-rate that depends on the roll angle. In addition, it is desirable to stay close to some nominal airspeed to ensure energy-efficient flight, to avoid stall, and to maintain control surface effectiveness which is proportional to airspeed squared. The RL controller is therefore tasked with controlling the roll and pitch angles, ϕ and θ, and the airspeed V_a to desired reference values. At each time step, the controller receives an immediate reward, and it aims at developing a control law that maximizes the sum of future discounted rewards.

The action space of the controller is three dimensional, consisting of commanded virtual elevator and aileron angles as well as the throttle. Elevator and aileron commands are mapped to commanded elevon deflections using the inverse of the transformation given by (9.20).

The observation vector (i.e. the input to the RL controller) contains information obtained directly from state-feedback of states typically measured by standard sensor suites. No sensor noise is added. To promote smooth actions it also includes a moving average of previous actuator setpoints. Moreover, since the policy network is a feed-forward network with no memory, the observation vector at each time step consists of these values for several previous time steps to facilitate learning of the dynamics.

9.4.1 Action Space

A known issue in optimal control is that while continually switching between maximum and minimum input is often optimal in the sense of maximizing the objective function, in practice, it wears unnecessarily on the actuators. Since PPO during training samples its outputs from a Gaussian distribution, a high variance will generate highly fluctuating actions. This is not much of a problem in a simulator environment but could be an issue if trained online on a real aircraft. PPO's hyperparameters are tuned wrt. a symmetric action space with a small range (e.g. -1 to 1). Adhering to this design also has the benefit of increased generality, training the controller to output actions as a fraction of maximal and minimal setpoints. The actions produced by the controller are therefore clipped to this range, and subsequently scaled to fit the actuator ranges as described in Section 9.3.

9.4.2 Training of Controller

The PPO RL controller was initialized with the default hyperparameters in the OpenAI Baselines implementation [89], and ran with 6 parallel actors. The controller policy is an extended version of the default two hidden layers, 64 nodes multi layer perceptron (MLP) policy: The observation vector is first processed in

Table 9.1: Constraints and ranges for initial conditions and target setpoints used during training of controller.

Variable	Initial Condition	Target
ϕ	$\pm 150°$	$\pm 60°$
θ	$\pm 45°$	$\pm 30°$
ψ	$\pm 60°$	-
ω	$\pm 60\,°/\mathrm{s}$	-
α	$\pm 26°$	-
β	$\pm 26°$	-
V_a	$12 - 30\,\mathrm{m/s}$	$12 - 30\,\mathrm{m/s}$

a convolutional layer with three filters spanning the time dimension for each component, before being fed to the default policy. This allows the policy to construct functions on the time evolution of the observation vector while scaling more favourably in parameter count with increasing observation vector size compared to a fully connected input layer.

The controller is trained in an episodic manner to assume control of an aircraft in motion and orient it towards some new reference attitude. Although the task at hand is not truly episodic in the sense of having natural terminal states, episodic training allows one to adjust episode conditions to suit the agent's proficiency, and also admits greater control of the agent's exploration of the state space. The initial state and reference setpoints for the aircraft are randomized in the ranges shown in Table 9.1. Episode conditions are progressively made more difficult as the controller improves, beginning close to target setpoints and in stable conditions, until finally spanning the entirety of Table 9.1. The chosen ranges allow the RL controller to demonstrate that it is capable of attitude control, and facilitates comparison with the PID controller as it is expected to perform well in this region. According to [19], a typical sampling frequency for autopilots is 100 Hertz, and the simulator therefore advances 0.01 seconds at each time step. Each episode is terminated after a maximum of 2000 time steps, corresponding to 20 seconds of flight time. No wind or turbulence forces are enabled during training of the controller.

In accordance with traditional control theory, where one usually considers cost to be minimized rather than rewards to be maximized, the immediate reward returns to the RL controller are all negative rewards in the normalized range of -1 to 0:

$$
\begin{aligned}
R_\phi &= \text{clip}\left(\frac{|\phi - \phi^d|}{\zeta_1}, 0, \gamma_1\right) \\
R_\theta &= \text{clip}\left(\frac{|(\theta - \theta^d)|}{\zeta_2}, 0, \gamma_2\right) \\
R_{V_a} &= \text{clip}\left(\frac{|V_a - V_a^d|}{\zeta_3}, 0, \gamma_3\right) \\
R_{\delta^c} &= \text{clip}\left(\frac{\sum_{j\in[a,e,t]}\sum_{i=0}^{4}|\delta^c_{j_{t-i}} - \delta^c_{j_{t-1-i}}|}{\zeta_4}, 0, \gamma_4\right) \\
R_t &= -(R_\phi + R_\theta + R_{V_a} + R_{\delta^c})
\end{aligned}
\tag{9.23}
$$

$$
\begin{aligned}
&\zeta_1 = 3.3,\ \zeta_2 = 2.25,\ \zeta_3 = 25,\ \zeta_4 = 60 \\
&\gamma_1 = 0.3,\ \gamma_2 = 0.3,\ \ \gamma_3 = 0.3, \gamma_4 = 0.1
\end{aligned}
$$

In this reward function, L_1 was chosen as the distance metric between the current state and the desired state (denoted with superscript d).[1] Furthermore, a cost is attached to changing the actuator setpoints to address oscillatory control behaviour. Commanded control setpoint of actuator j at time step t is denoted $\delta^c_{j_t}$, where $j \in [a, e, t]$. The importance of each component of the reward function is weighted through the γ factors. To balance the disparate scales of the different components, the values are divided by the variables approximate dynamic range, represented by the ζ factors.

The components of the observation vector are expressed in different units and also have differing dynamic ranges. NNs are known to converge faster when the input features share a common scale, such that the network does not need to learn this scaling itself. The observation vector should therefore be normalized. This is accomplished with the VecNormalize class of [89], which estimates a running mean and variance of each observation component and normalizes based on these estimates.

9.4.3 Evaluation

Representing the state-of-the-art in model free control, fixed-gain PID controllers for roll, pitch and airspeed were implemented to provide a baseline comparison for

[1]The L_1 distance has the advantage of punishing small errors harsher than the L_2 distance, and therefore encourages eliminating small steady-state errors.

the RL controller:

$$\delta_t^c = -k_{p_V}(V_a - V_a^d) - k_{i_V}\int_0^t (V_a - V_a^d)d\tau \tag{9.24}$$

$$\delta_a^c = -k_{p_\phi}(\phi - \phi^d) - k_{i_\phi}\int_0^t (\phi - \phi^d)d\tau - k_{d_\phi}p \tag{9.25}$$

$$\delta_e^c = -k_{p_\theta}(\theta - \theta^d) - k_{i_\theta}\int_0^t (\theta - \theta^d)d\tau - k_{d_\theta}q \tag{9.26}$$

The throttle is used to control airspeed, while virtual aileron and elevator commands are calculated to control roll and pitch, respectively. The PID controllers were manually tuned using a trial-and-error approach until achieving acceptable transient responses and low steady-state errors for a range of initial conditions and setpoints. Wind was turned off in the simulator during tuning. The integral terms in (9.24)-(9.26) are implemented numerically using forward Euler. Controller gains are given in Table 9.2.

Table 9.2: PID controller parameters.

Parameter	Value	Parameter	Value
k_{p_V}	0.5	k_{d_ϕ}	0.5
k_{i_V}	0.1	k_{p_θ}	-4
k_{p_ϕ}	1	k_{i_θ}	-0.75
k_{i_ϕ}	0	k_{d_θ}	-0.1

The same aerodynamic model that is used for training is also used for evaluation purposes, with the addition of disturbances in the form of wind to test generalization capabilities. The controllers are compared in four distinct wind and turbulence regions: light, moderate, severe and no turbulence. Each setting consists of a steady wind component, with randomized orientation and a magnitude of 7 m/s, 15 m/s, 23 m/s and 0 m/s respectively, and additive turbulence given by the Dryden turbulence model [180]. Note that a wind speed of 23 m/s is a substantial disturbance, especially when considering the Skywalker X8's nominal airspeed of 18 m/s. For each wind setting, 100 sets of initial conditions and target setpoints are generated, spanning the ranges shown in Table 9.1. The reference setpoints are set to 20-30 degrees and 3-4 m/s deviation from the initial state for the angle variables and airspeed, respectively. Each evaluation scenario is run for a maximum of 1500 time steps, corresponding to 15 seconds of flight time, which should be sufficient time to allow the controller to regulate to the setpoint.

The reward function is not merely measuring the proficiency of the RL controller but is also designed to facilitate learning. To compare, rank and evaluate different controllers, one needs to define additional evaluation criteria. To this end, the controllers are evaluated on the following criteria: **Success/failure**, whether the controller is successful in controlling the state to within some bound of the setpoint. The state must remain within the bounds for at least 100 consecutive time steps to be counted as a success. The bound was chosen to be $\pm 5°$ for the roll and pitch angles, and $\pm$2m/s for the airspeed. **Rise time**, the time it takes the controller to reduce the initial error from 90 % to 10 %. As these control scenarios are not just simple step responses and may cross these thresholds several times during the episode, the rise time is calculated from the first time it crosses the lower threshold until the first time it reaches the upper threshold. **Settling time**, the time it takes the controller to settle within the success setpoint bounds, and never leave this bound again. **Overshoot**, the peak value reached on the opposing side of the setpoint wrt. the initial error expressed as a percentage of the initial error. **Control variation**, the average change in actuator commands per second, where the average is taken over time steps and actuators. Rise time, settling time, overshoot and control variation are only measured when the episode is counted as a success. When comparing controllers, the success criterion is the most important, as it is indicative of stability as well as achieving the control objective. Secondly, low control variation is important to avoid unnecessary wear and tear on the actuators. While success or failure is a binary variable, rise time, settling time and overshoot give additional quantitative information on the average performance of the successful scenarios.

9.5 Results and Discussion

The controller was trained on a desktop computer with an i7-9700k CPU and an RTX 2070 GPU. The model converges after around two million time steps of training, which on this hardware takes about an hour. This is relatively little compared to other applications of DRL and suggests that the RL controller has additional capacity to master more difficult tasks. Inference with the trained model takes 800 microseconds on this hardware, meaning that the RL controller could reasonably be expected to be able to operate at the assumed autopilot sampling frequency of 100 Hertz in flight.

9.5.1 Key Factors Impacting Training

The choice of observation vector supplied to the RL controller proved to be significant for its rate of improvement during training and its final performance. It was found that reducing the observation vector to only the essential components, i.e. the current airspeed and roll and pitch angles, the current angular velocities of the UAV, and the state errors, helped the RL controller improve significantly faster than other, larger observation vectors.[2] It can be seen from Figure 9.2 that increasing the size of the observation vector both lowers the mean of episode rewards and increases the standard deviation significantly, while the rate of progression is seemingly unaffected. This finding runs counter to the mantra that more data is better with ML methods, but might be explained by fewer factors making it easier for the RL algorithm to determine the effects of its actions on the progress towards the target. Including values for several previous time steps (five was found to be a good choice) further accelerated training, as this makes learning the dynamics easier for the memoryless feed-forward policy.

The reward function is one of the major ways the designer can influence and direct the behaviour of the agent. One of the more popular alternatives to L_1 norm and clipping to achieve saturated rewards are the class of exponential reward functions, and notably the Gaussian reward function as in [40]. Analyzing different choices of the reward function was not given much focus as the original choice gave satisfying results.

It was also discovered that replacing the target setpoint values in the observation vector with the current error of each state helped training speed. Figure 9.2 also shows that normalization helps the rate of progression, and that converged performance is similar after the network has learned to scale the non-normalized numbers.

For complicated tasks, there might be much to gain from starting with simple tasks for the agent and progressively increasing the difficulty as the agent improves, i.e. imposing some form of curriculum learning [60, 131]. This could be for instance having the controller start at the setpoint values for all but one variable, and subsequently add control of more variables as the controller masters the single variable case. This approach was not necessary for the task presented in this chapter, but in attempts at global attitude control over the whole state space with the PPO algorithm, the approach was successful in increasing the proficiency of the con-

[2]Essential here referring to the factors' impact on performance for this specific control task. One would for instance expect α and β to be essential factors when operating in the more extreme and nonlinear regions of the state space.

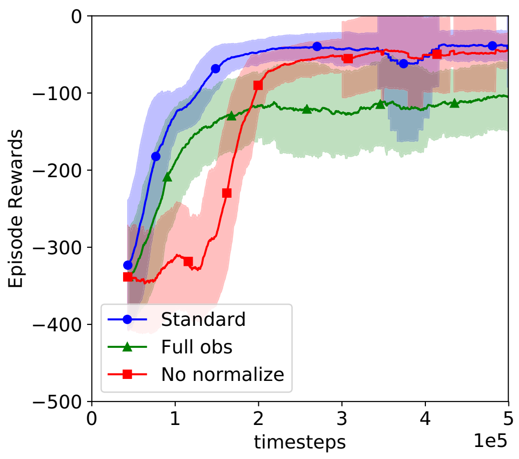


Figure 9.2: The evolution of the training process as influenced by some key factors. Each line and the shaded surrounding area represents the mean episode reward and standard deviation in a rolling window of 100 episodes, respectively. Sudden increases in standard deviations are caused by early terminated episodes with low total reward.

troller. In such scenarios, further constraining states to within reasonable flight envelopes may also help the rate of progression, e.g. constraining the angular rates such that the agent avoids exploring strategies with excessive rotations.

9.5.2 Evaluation of Controller

It is important to note the dependence between pitch and airspeed, and that achieving any arbitrary combination of the two might not be possible due to the fact that lowering the noise of the aircraft typically tend to increase airspeed as potential energy gets converted to kinetic energy. A sizable portion of the failures reported in Table 9.3 stems from conflicting priorities between achieving the target pitch angle or the target airspeed. Moreover, none of the scenarios shows instability. In all cases, the initial errors are reduced in all states, but they might not settle into the specified bound in the allotted time. The results presented should therefore not be interpreted as an indication of the degree of instability of the controller, as no such failure modes could be provoked in the test environment as specified. The rise and settling times reported in Table 9.3 are reasonable and around what one would expect from a good controller. The overshoot values shows that the controller provides underdamped control, but not to a degree where instability and steady oscillations occur.

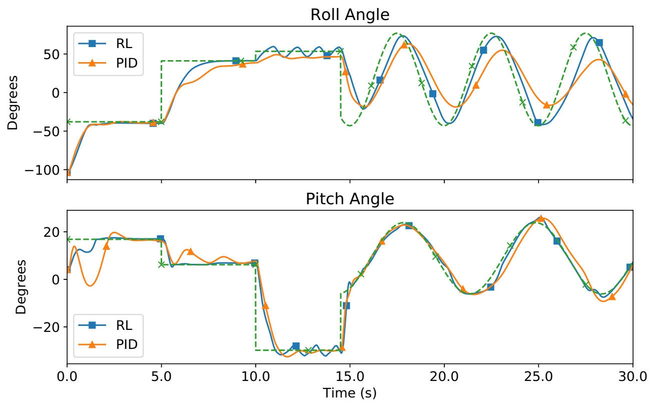


Figure 9.3: Comparison of the PID and RL controllers tasked with tracking the dashed green line.

The RL controller generalizes well to situations and tasks not encountered during training. Even though the controller is trained with a single setpoint for each episode, Figure 9.4 shows that the controller is perfectly capable of adapting to new setpoints during flight. This result was also found by Koch et al. [101] for quadcopters. The generalization capability also holds true for unmodeled wind and turbulence forces. The controller is trained with no wind estimates present in the observation vector, and no wind forces enabled in the simulator, but as Table 9.3 shows it is still able to achieve tracking to the setpoint when steady wind and turbulence is enabled in the test environment. Table 9.3 should be read as a quantitat-

Table 9.3: Performance metrics for the RL controller and the baseline PID controller on the evaluation scenarios. Both controllers exhibit strengths in different aspects — the best value in each circumstance is shown in bold.

		Success (%)				Rise time (s)			Settling time (s)			Overshoot (%)			Control variation
Setting	Controller	ϕ	θ	V_a	All	ϕ	θ	V_a	ϕ	θ	V_a	ϕ	θ	V_a	(s^{-1})
No turbulence	RL	100	100	**100**	**100**	**0.265**	0.661	**0.825**	**1.584**	1.663	2.798	21	24	**31**	0.517
	PID	100	100	98	98	1.344	**0.228**	0.962	2.050	**1.364**	**2.198**	**4**	**17**	35	**0.199**
Light turbulence	RL	100	100	**100**	**100**	**0.210**	0.773	**0.744**	**1.676**	1.806	2.738	28	33	**36**	0.748
	PID	100	100	99	99	1.081	**0.294**	0.863	2.057	**1.638**	**2.369**	**6**	**20**	43	**0.457**
Moderate turbulence	RL	**100**	**100**	**98**	**98**	**0.192**	1.474	0.934	**2.167**	**2.438**	4.085	54	54	74	0.913
	PID	100	93	90	87	0.793	**0.525**	**0.864**	2.764	2.563	**3.460**	**34**	**35**	**70**	**0.781**
Severe turbulence	RL	**100**	**100**	**92**	**92**	**0.166**	1.792	1.585	**2.903**	**3.280**	7.049	122	93	156	**1.083**
	PID	99	96	87	86	0.630	**0.945**	**1.343**	3.576	5.256	**5.470**	**92**	**80**	**122**	1.117

ive analysis of performance in conditions similar to normal operating conditions, while Figure 9.4 and 9.3 qualitatively shows the capabilities of the controllers on significantly more challenging tasks.

Table 9.3 shows that both controllers are generally capable of achieving convergence to the target for the evaluation tasks, with neither controller clearly outperforming the other. The RL controller has an advantage over the PID controller on the success criterion and seems to be more robust to the turbulence disturbance. It is able to achieve convergence in the attitude states in all situations, unlike the PID controller, and is also notably more successful in moderate and severe turbulence conditions. The PID controller has considerably lower control variation for the simple settings with little or no wind, but its control variation grows fast with increasing disturbance. At severe turbulence, the RL controller has the least control variation.

The two controllers perform similarly wrt. settling time and rise time, each having the edge in different states under various conditions, while the PID controller performs favourably when measured on overshoot. All in all, this is an encouraging result for the RL controller, as it is able to perform similarly as the established PID controller in its preferred domain, while the RL controller is expected to make its greatest contribution in the more nonlinear regions of the state space.

A comparison of the two controllers is shown in Figure 9.3 on a scenario involving fairly aggressive maneuvers, which both are able to execute. Figures 9.4 and 9.3 illustrate an interesting result, the RL controller is able to eliminate steady-state errors. While the PID controller has integral action to mitigate steady-state errors, the control law of the RL controller is only a function of the last few states and references. This might suggest that the RL controller has learned some feed-forward action, including nominal inputs in each equilibrium state, thus removing steady-state errors in most cases. Another possibility is that steady-state errors are greatly reduced through the use of high-gain feedback, but the low control variation shown for severe turbulence in Table 9.3 indicates that the gain is not excessive. Future work should include integral error states in the observations and evaluate the implications on training and flight performance.

9.6 Conclusion

The ease with which the proof of concept RL controller learns to control the UAV for the tasks presented in this chapter, and its ability to generalize to turbulent wind conditions, suggests that DRL is a good candidate for nonlinear flight control

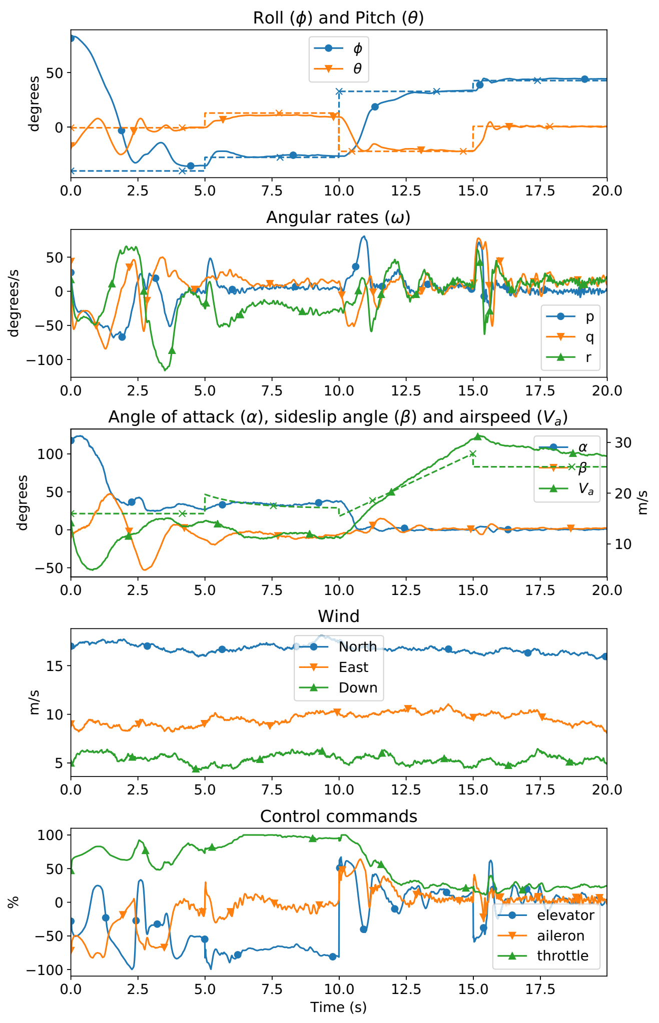


Figure 9.4: The RL controller trained episodically with a single setpoint and no wind or turbulence generalizes well to many wind conditions and continuous tracking of setpoints (shown with dashed lines marked by crosses). Here subjected to severe wind and turbulence disturbances with a magnitude of 20 m/s.

design. A central unanswered question here is the severity of the reality gap, or in other words how transferable the strategies learned in simulations are to real-world flight. Future work should evaluate the controller's robustness to parametric and structural aerodynamic uncertainties; this is essential to do before undertaking any real-life flight experiments. For more advanced maneuvers, e.g. aerobatic flight or recovering from extreme situations, the controller should be given more freedom in adjusting the airspeed, possibly through having it as an uncontrolled state.

There is still much potential left to harness for this class of controllers. The policy network used to represent the control law is small and simple; more complex architectures such as long short term memory (LSTM) could be used to make a dynamic RL controller. Training, experiments and reward structures can be designed to facilitate learning of more advanced behaviour, tighter control or better robustness. Should the reality gap prove to be a major obstacle for the success of the RL controller in the real world, one should look to the class of off-policy algorithms such as SAC. These algorithms are able to learn offline from gathered data, and thus might be more suited for UAV applications.

10

Data Efficient Deep Reinforcement Learning Control of Fixed-Wing UAVs in the Field

This chapter extends the work presented in the preceding chapter, targeting control of the real Skywalker X8 in the field. A controller for the real UAV is developed by training on a simulated version, and therefore a new more data efficient method is developed in order to limit the adverse affects of training in a separate environment from the target environment. Further, several sim-to-real measures are undertaken in order to ensure a smooth transfer. The development process of the work presented in this chapter was highly iterative: Despite all the modeling efforts undertaken, the model is shown to be of poor fit in certain aspects of the real Skywalker X8 system. The behavior of the RL controller in the field was therefore observed, and for each field experiment undertaken, areas of improvement was identified and the simulation was adjusted in order to rectify the discrepancies observed in real flight. The RL controller is compared to the state-of-the-art ArduPlane autopilot, and shows comparable performance. The main contribution of this chapter lies in experimentally verifying dynamic control of fixed-wing UAV using RL. This chapter is based on the following article:

- [27] Eivind Bøhn, Erlend M. Coates, Dirk Reinhardt, and Tor Arne Johansen. Data-efficient deep reinforcement learning for attitude control of fixed-wing uavs validated through field experiments. *IEEE Transactions on Neural Networks and Learning Systems*, 2021. Submitted

Note that the standard notation for the pitch state in the Euler angle representation of the attitude, i.e. θ, clashes with the notation for the parameters of the RL algorithm used throughout this thesis. In this chapter, we will therefore use ϑ to refer to the parameters of the RL algorithm.

10.1 Introduction

Many challenging control problems arise during advanced operation of fixed-wing UAVs, such as aerobatic maneuvering [33], perching [46], deep-stall landing [128], recovery from loss of control [47], flying in strong wind fields [119], or performing VTOL transitions between hover and forward flight [9]. Fixed-wing UAVs, as illustrated in Fig. 8.1, have superior range and endurance when compared to multirotor UAVs. However, the control of such vehicles is challenging due to highly coupled, underactuated nonlinear dynamics, as well as uncertain aerodynamics affected by wind disturbances that make up a large fraction of the vehicle's airspeed.

A class of methods that has shown promising results for challenging control problems, is DRL. RL is an area of machine learning concerned with learning optimal sequential decision making. DRL is the combination of RL algorithms with NNs as function approximators, which is the state-of-the-art approach for many problems requiring complex decision making over long time horizons such as game-playing [137, 166], dexterous in-hand robotic manipulation [8], and object manipulation [165]. It can handle continuous state and action spaces, highly complex and nonlinear system dynamics, and in general does not require a model of the system to be controlled. Furthermore, the online execution of an RL controller is often very computationally efficient. This should make DRL an enticing alternative for problems where accurate identification of the system is difficult and traditional control approaches are unable to yield sufficient control performance. Despite this potential, DRL has yet to be widely adopted for control and notably lacks demonstrations of control applications outside of simulations. One of the main contributing factors to this is the lack of safety guarantees and the ability to formulate operating constraints, both in the exploration and exploitation phase, which is is further complicated by the data-intensive nature of DRL. We will in the rest of this chapter use RL to refer to DRL.

An approach to mitigate the challenges of RL for control is foregoing online exploration entirely and learning the controller exclusively from historical data with no further interaction with the system to be optimized, i.e. offline RL [113]. The latter is a radical alteration of the RL problem introducing new challenges and necessitating its own set of learning algorithms. It could nevertheless be an important tool in the future for problems such as control of UAVs — where data collection carries a high risk and accurate modelling is difficult. A more common approach is performing part of or the whole exploration phase in a simulation of the target system. A downside of this approach is that while RL in principle is a model-free optimization framework, the success of the transfer from the simulated environment to the real target environment is highly affected by the accuracy of the simulation model, the lack of which is referred to as the reality gap in RL. One should therefore take great effort in minimizing the reality gap through sim-to-real measures, which aim at robustifying the learned controller and emulate effects such as latency and measurement noise present in the real control system. For a recent survey on sim-to-real methods in the context of control and robotics, see [204].

For the sim-to-real learning approach to be useful for practical flight control applications, controllers trained in simulation need to transfer well to control of the real UAV. Before attempting advanced problems like e.g. deep-stall landing, it makes sense to first attempt simpler problems and identify what factors are important for controllers to transfer well to reality. In this chapter, we consider the attitude control problem of fixed-wing UAVs. Attitude refers to the orientation of the aircraft, and control of the attitude constitutes the lowest level of flight control deciding how the actuators of the UAV are used to achieve the desired attitude as decided by the guidance components of the control system. This work is a follow-up on our previous work [26] in which we demonstrate the efficacy of DRL for attitude control of fixed-wing UAVs in a simulator environment. We now target control of the real UAV in the field and develop a framework to learn attitude controllers with a focus on data-efficiency, yielding flightworthy controllers with only minutes of learning time. Starting with a UAV model obtained primarily through wind-tunnel experiments, we adopt the method of exploring and learning in a simulator environment and iteratively adjust the model and simulation environment with insights from flight experiments. We extensively apply domain-randomization and other sim-to-real measures in order to reduce the reality-gap. Moreover, the data-efficiency of our method limits overfitting to the simulation model, such that the controller transfers better to the field, and when combined with safe exploration measures the high data-efficiency could enable learning controllers entirely on the real UAV in the field.

The literature on RL-based attitude control of UAVs is dominated by quadcopters, and most works operate exclusively in simulated environments [103, 118, 188, 55, 24]. When it comes to works presenting real-world flight experiments we have identified the following: [107, 194, 102, 44, 59, 186, 140]. Of these, only [44], their follow-up work [59], and [186] use a fixed-wing UAV design. [44] and [59] study the specific problem of controlling a perched landing, [186] fixes the aircraft in a wind-tunnel and limit themselves to controlling the pitch of the UAV, while we study the full attitude control problem. Moreover, the data requirement of their methods are on the order of millions of samples. The other aforementioned works also require millions of samples of data, with the notable exception of [107]. Their method uses model-based RL and involves learning a forward dynamics model that is used in an MPC scheme which controls the quadrotor. While this method is very sample efficient, the MPC is too computationally complex to run on-board the UAV and therefore necessitates continuous communication with an external computer, while our controller can run on a fraction of the computational power embedded in the UAV. For a more general overview of the application of RL to UAVs see [12].

The contributions and novelty of this chapter can be summarized as follows:

- To the best of our knowledge, this is the first work to demonstrate through field experiments the efficacy of RL for attitude controller of fixed-wing UAVs, a class of UAV design generally considered to be significantly more complex to control than the multirotor which is common in the literature.
- The proposed method improves upon the data-efficiency of the existing literature by at least an order of magnitude. We show that our method can develop flightworthy controllers with only 3 minutes of data from interaction with the simulation environment, providing an important step towards enabling the learning of RL controllers entirely on the real UAV.
- We present an analysis of the RL controller in order to better understand how it operates, including a comparison to an industry-standard PID controller.

10.2 Learning an Off-Policy Attitude Controller for the Real UAV using SAC

The control objective of the RL controller is to control the attitude of the aircraft to the desired reference attitude. We use standard aircraft nomenclature and coordinate systems [17], as well as a roll-pitch-yaw Euler angle parameterization of attitude. The heading/yaw angle is typically not controlled directly, but rather through banked-turn maneuvers [17]. Therefore, the natural choice of controlled states are the roll angle ϕ, and the pitch angle θ. We assume that the UAV is equipped with control surfaces such that the roll and pitch angles are controllable (an assumption that is satisfied by most UAV designs). The Skywalker X8 seen in Fig. 8.1 is used in our field experiments. It has two elevon control surfaces, one on each wing, which can be driven together to produce a pitching moment, or driven differentially to produce a rolling moment. In addition, it has a propeller that can produce a thrust force along the longitudinal axis of the UAV. In the simulation environment, the PI-controller described in Section 9.4.3 is used to control the throttle of the propeller.

As a general approach, we tested new ideas in the simulation environment and made extensive use of sim-to-sim experiments where we studied how the controller transferred from simulation with one set of parameters to a simulation with another set of parameters. We then tested the most promising controllers in flight experiments in the field and adjusted our approach based on the insight we gathered from the flight experiments. The simulation environment software is made open-source and is available at [35, 34].

10.2.1 Controller Architecture and State Design

We identified in our previous work that limiting the state vector to only the most useful information and reduce redundancy is important for the rate of convergence, and to prevent the controller from learning spurious relationships. This has also been observed in other research [24]. At every time step we measure the following information:

$$\begin{aligned} m_t = \big[&p_t, q_t, r_t, \alpha_t, \beta_t, V_{a,t}, \delta_{r,t-1}, \delta_{l,t-1}, \\ &I_{\phi,t}, I_{\theta,t}, \phi_t, \theta_t, e_{\phi,t}, e_{\theta,t}\big]^\top \end{aligned} \tag{10.1}$$

$$I_{*,t} = \gamma^I I_{*,t-1} + e_{*,t}, \;\; \gamma^I = 0.99, \;\; I_0 = 0, * \in \{\phi, \theta\} \tag{10.2}$$

where t is the time index, $\omega_t = [p_t, q_t, r_t]^\top$ is the angular velocity in the body-fixed frame, α_t is the angle-of-attack, β_t is the sideslip-angle, $V_{a,t}$ is the airspeed, $\delta_{\{r,l\}}$ represent the previous output of the RL controller, in this case the commanded deflection angles of the right and left elevons, $e_{*,t} = *_t - *_{r,t}$ is the state tracking error where subscript r denotes the state reference, I_* is the integrator of the state error and γ^I is the integrator decay rate. The integration decay scheme follows [194], and facilitates boundedness of the integrator state. The choice of what to include between the coupled state, reference, and error was made on the basis of making the most important information readily available, and we therefore chose the state, which is significant for the aerodynamics, and the error, which is significant for the objective. Lastly, because neural networks are known to converge faster given normalized inputs, the measurements are normalized using running estimates of mean and variance for each component before being fed to the controller.

Due to unmeasured effects such as turbulence and the sim-to-real measures described in Section 10.2.4, the attitude control problem is partially observable. Furthermore, to enable the controller to adapt to the dynamics of the field experiments, we wish to enhance the controller with the capability of inferring the dynamics

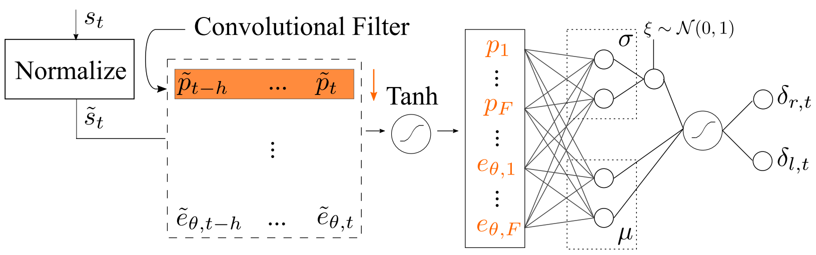


Figure 10.1: Architecture of the RL controller, superscript $\sim$ signifies normalized states.

around the current state. A common approach to achieve this effect is to use a recurrent neural network (RNN) [186, 147]. However, we found that using a one-dimensional convolution over the time dimension as the input layer yielded similar control performance, and therefore prefer it since it is significantly less complex. We therefore include the h last measurements (10.1) in the state vector, where $\hat{m}_t$ indicates a noisy measurement to be defined in Section 10.2.4:

$$s_t = [\hat{m}_t, \hat{m}_{t-1}, \dots, \hat{m}_{t-h}]^\top \tag{10.3}$$

$$[\delta_{r,t}, \delta_{l,t}]^\top = \pi_\vartheta(s_t) + [\delta_{r,\text{trim}}, \delta_{l,\text{trim}}]^\top \tag{10.4}$$

such that the total size of the state vector is $|s_t| = |m_t| \cdot h$. The convolutional input layer scales favorably in number of learned parameters compared to a fully-connected (FC) layer: it scales linearly in $|m_t|$ as opposed to multiplicative for the FC layer, and it is constant for h. The convolutional layers' output size is $F \cdot |m_t|$ where F is the number of learned convolutional filters, and each filter has size h. The memory capacity of the state vector can therefore be increased as required to give sufficient history to infer the dynamics, with only a slight increase in the number of parameters. We found $F = 8$ and $h = 10$ to work well. The complete RL controller architecture is shown in Fig. 10.1.

The output of the RL controller is the commanded states of the controlled system's actuators relative to the trim-point (10.4). The nominal elevon deflection angles $\delta_{r,\text{trim}} = \delta_{l,\text{trim}} = 0.045$ are calculated using a standard trim routine for horizontal, wings-level flight based on the model in Section 10.2.3 [17]. The target UAV for the field experiments, the Skywalker X8, has elevon actuators and we therefore chose to have the controller output the desired deflection angles of these directly, in order to provide RL with as direct control as possible. This choice is fairly arbitrary, however, and experiments showed that outputting virtual elevator and aileron angles (defined by (A.6)-(A.7)) instead yield similar performance.

10.2.2 Reward and Objective Design

We found sparse rewards to yield better results than shaped rewards, both in terms of rate of learning and in terms of asymptotic performance. A sparse reward is one that is nonzero only for some subset of the state space. It has the benefit that it is easier to formulate than hand-crafted shaped rewards, and would therefore be more transferable to other UAVs with fewer adjustments necessary. The reward is formulated as follows:

$$R(s_t, a_t) = \lambda^{\phi} B(e_{\phi,t}) + \lambda^{\theta} B(e_{\theta,t}) + \lambda^{\dot{\phi}} B(\dot{\phi}_t) + \lambda^{\dot{\theta}} B(\dot{\theta}_t) \tag{10.5}$$

$$B(\cdot) = \begin{cases} 1 & \text{if } |\cdot| \leq \cdot^{b} \\ 0 & \text{otherwise} \end{cases} \tag{10.6}$$

$$e^{b}_{\phi_t} = 3^{\circ},\ \ e^{b}_{\theta_t} = 3^{\circ},\ \ \dot{\phi}^{b}_{t} = 4.3^{\circ}/s,\ \ \dot{\theta}^{b}_{t} = 4.3^{\circ}/s \tag{10.7}$$

$$\lambda^{\phi} = 0.5,\ \ \lambda^{\theta} = 0.5,\ \ \lambda^{\dot{\phi}} = 0.167,\ \ \lambda^{\dot{\theta}} = 0.167 \tag{10.8}$$

where superscript b refers to the goal-bound on the variable and the λs are weighting factors. This reward ensures that the controller tracks the setpoints with accuracy as specified by the bound, while the rewards on the derivatives of the controlled states discourage high rates. Our method is not very sensitive to the size of the bound, but generally larger bounds accelerate learning at the expense of tracking accuracy.

When transferring from a simulator environment to the field, it is important to consider how the actuation system impacts the effects of actions. That is, while high-gain bang-bang control may be an optimal strategy in the simulator, frequently changing the setpoints of the actuators introduces considerable wear due to the high currents generated as a result of the switching. In our previous work [26] (and indeed in other works [103]) this problem is addressed through a term in the reward that discourages changing the setpoints. We now take a different approach to this problem, using the conditioning for action policy smoothness (CAPS) method [139]:

$$J^{\text{TS}}(\pi_{\vartheta}) = ||\pi_{\vartheta}(s_t) - \pi_{\vartheta}(s_{t+1})||_2 \tag{10.9}$$

$$J^{\text{SS}}(\pi_{\vartheta}) = ||\pi_{\vartheta}(s_t) - \pi_{\vartheta}(\hat{s}_t)||_2, \qquad \hat{s}_t \sim \mathcal{N}(s_t, 0.01) \tag{10.10}$$

This method adds two regularization terms to the loss, a temporal smoothness term (10.9) and a spatial smoothness term (10.10). As the authors demonstrate, this method is more successful in generating controllers that yield smooth control signals compared to the action reward-term approach. Additionally, removing the action term from the reward simplifies the problem of learning the action-value function since the reward now contains fewer disparate parts, thereby accelerating learning. Instead, the gradient ascent scheme calculating the parameter updates is conditioned towards policies that are smooth in the output. Finally, we add a regularization term on the pre-activation $\pi_\vartheta^{\text{PA}}$ (that is, before applying the hyperbolic tangent in (2.56)) of the output:

$$J^{\text{PA}}(\pi_\vartheta) = ||\pi_\vartheta^{\text{PA}}(s_t)||_2 \tag{10.11}$$

This helps in reducing the gain of the controller, especially for small errors, as it essentially tells the controller that it needs to have a strong benefit to move the actuators away from the trim-point. Additionally, we find it accelerates learning as the controller is biased towards non-aggressive control, which in conjunction with HER means the controller quickly discovers how to achieve the sparse stabilizing objective. Thus, the objective we optimize is defined as:

$$\begin{aligned} J(\pi_\vartheta) = J^{\text{SAC},\pi}(\pi_\vartheta) + \lambda^{\text{TS}} J^{\text{TS}}(\pi_\vartheta) + \lambda^{\text{SS}} J^{\text{SS}}(\pi_\vartheta) + \\ \lambda^{\text{PA}} J^{\text{PA}}(\pi_\vartheta) \end{aligned} \tag{10.12}$$

$$\lambda^{\text{TS}} = 5 \cdot 10^{-2}, \;\; \lambda^{\text{SS}} = 10^{-1}, \;\; \lambda^{\text{PA}} = 10^{-4} \tag{10.13}$$

10.2.3 UAV Model

For the simulated environment, we use a model of the Skywalker X8 UAV based on previous modelling efforts [78, 45, 189, 81]. The model structure is standard in the literature [17, 174] and is based on a single rigid body Newton-Euler formulation affected by forces and moments due to gravity, aerodynamics and propulsion effects. An estimate of the inertia matrix is provided in [81] based on bifilar pendulum experiments. Results from wind-tunnel experiments are provided in [78] for aerodynamic coefficients, and in [45] for the propulsion system model. This data is complemented by computational fluid dynamics (CFD) simulations from [78, 189]. For a more detailed description of the simulation model, see Section 9.3.

We collected a short data series to assess the quality of the dynamic model. To excite the system dynamics, we used the actuator signals from the baseline attitude controller (see Appendix A) and perturbed them with chirp signals before mapping them to the elevon deflections. The start and end frequencies of the chirp signals were 8Hz and 12 Hz, respectively. A dynamic mode analysis of the model indicates that this is the dominant frequency spectrum of the X8. The signal duration was 15 seconds and we used a peak-to-peak amplitude of 20 degrees.

The aerodynamic coefficients that are calculated based on recorded sensor data and the inertia matrix of the vehicle are shown in Fig. 10.2. Following [17], the coefficient subscript L, D, Y, l, m, n denotes lift, drag, side force, roll moment, pitch moment and yaw moment, respectively. These results show that despite the modelling efforts, there are still significant discrepancies between the predicted and measured data, particularly in the pitching moment coefficient, C_m.

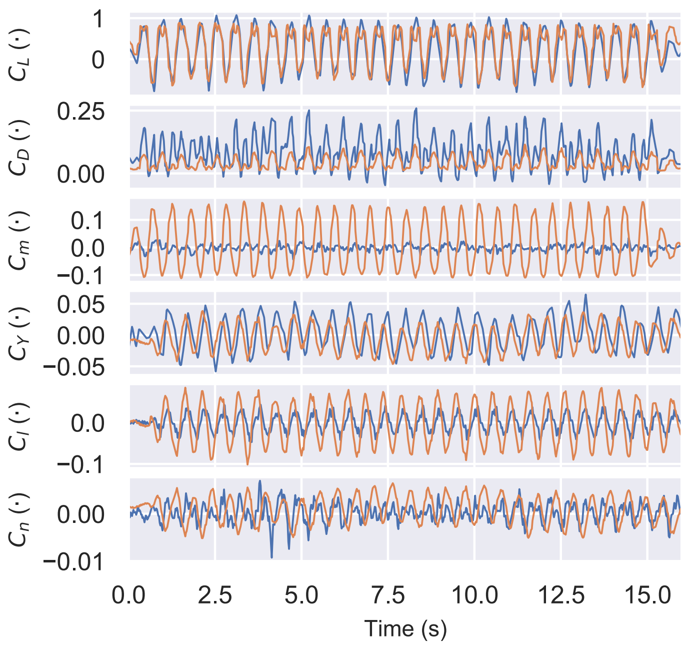


Figure 10.2: The aerodynamic coefficients of the UAV in a longitudinal (top three) and a lateral (bottom three) chirp signal test sequence for elevator and aileron, respectively, based on IMU data (blue) and model prediction (orange).

10.2.4 Sim-to-Real

The main sim-to-real measure employed in the method is domain randomization. As shown in Section 10.2.3, there is a significant reality gap, and as such we want to avoid the RL controller overfitting to the simulation environment. The intuition behind the domain randomization technique is that learning over a distribution of possible UAV models should robustify the controller. To this end, we assess the uncertainty in the estimate of every parameter of the UAV model and use this uncertainty to construct a probability distribution over its range of probable values. The distribution for each parameter is shown in Table 10.1, which shows that we are more uncertain about the coefficients of the rate-dependent terms of the UAV model since these are not estimated based on wind tunnel data but rather on uncertain CFD simulations [78]. The sampled values are also clipped as indicated to avoid extreme unrealistic values. At the start of every episode, we sample a value for each parameter from its distribution, together constituting one realization of the UAV model.

The UAV sensor suite is subject to noise in its measurements. To model these, we first estimated the noise characteristics of the real hardware, and then we emulate this in the simulator environment. We model the measurement noise as an Ornstein-Uhlenbeck (OU) process (10.14), which in addition to white noise gives rise to effects like measurement drift:

$$\begin{aligned} \hat{m}_t &= m_t + w_t, \;\; w_t \sim OU(\mu_m, \sigma_m, \theta_m), \;\; \mu_m = 0, \;\; \theta_m = 1 \\ \sigma_m &= 0.005\,[1.5, 1.5, 1.5, 1, 1, 15, 0, 0, 0, 0, 1, 1, 0, 0]^\top \end{aligned} \tag{10.14}$$

where $\{\mu_m, \sigma_m, \theta_m\}$ are the mean, variance and rate of mean-reversion parameters of the measurement noise. Note that we do not add noise to the error and integrator states, as these are already affected by the noise in the measurement of the state that is used to calculate the error, while the previous output of the controller is by nature free of noise. The sensor suite has an update rate of 50Hz, and we therefore chose this as the control frequency as well. In the simulation environment we add exponentially distributed noise on top of the fixed control frequency in order to simulate sensor timing-variability:

$$\Delta_t = \Delta_0 + z_t, \;\; z_t \sim \mathrm{Exp}(\kappa), \;\; \kappa \sim \mathcal{U}(250, 1000) \tag{10.15}$$

where Δ_t is the simulation step size at step t, $\Delta_0 = 0.02\,\mathrm{s}$ is the base control frequency and κ is the parameter of the exponentially distributed noise which is drawn uniformly at the start of every episode.

Another major effect present in the field is atmospheric disturbances such as wind and turbulence. We model turbulence with the Dryden turbulence model [17], and

Table 10.1: UAV model parameters are sampled at the start of each episode according to these distributions, where ρ refers to the estimated value. Notation is adapted from [17, 78], which is fairly standard in the literature.

Parameter	Distribution	Clipping
$C_{D_0}, C_{D_{\alpha_1}}, C_{D_{\alpha_2}}, C_{D_{\beta_1}}, C_{D_{\beta_2}}, C_{D_{\delta_e}}$ $C_{L_0}, C_{L_\alpha}, C_{L_{\delta_e}}, C_{Y_\beta}, C_{Y_{\delta_e}}, C_{l_\beta}, C_{l_{\delta_a}}$ $C_{m_0}, C_{m_\alpha}, C_{m_{\delta_e}}, C_{m_{\mathrm{fp}}}, C_{n_\beta} C_{n_{\delta_a}}$ $C_{\mathrm{prop}}, J_{\{x,y,z,xz\}}, M, a_0, k_\Omega, k_{T_p}, k_{\mathrm{motor}}$	$\mathcal{N}(\rho, \rho \cdot 0.1)$	$\pm\rho \cdot 0.2$
$C_{D_p}, C_{L_q}, C_{Y_p}, C_{Y_r}, C_{l_p}, C_{l_r}, C_{n_p}, C_{n_r}$	$\mathcal{N}(\rho, \rho \cdot 0.2)$	$\pm\rho \cdot 0.5$
C_{m_q}	$\mathcal{N}(\rho, \rho \cdot 0.5)$	$\pm\rho \cdot 0.95$

a steady wind component whose direction and magnitude between $0\,\mathrm{ms}^{-1}$ and $15\,\mathrm{ms}^{-1}$ is sampled at the start of each episode. The last effect we found was highly impactful for successful transfer was the actuation delay, i.e. the time it takes before the output of the controller is applied to the system, a result which was also found in [186]. The simulator contains a constant actuation delay of $100\,\mathrm{ms}$, while we believe this is a significant overestimation of the delay of the real system, we motivate this choice in Section 10.4.3.

10.2.5 Simulator Episode Design

The standard design of episodes for UAV control in the literature seems to be short episodes with a single constant desired attitude [103, 24]. We found that having constant setpoints accelerates learning, however the operation of the UAV in the field typically sees the navigation system continuously update the desired attitude. To align these considerations, we employ fairly long episodes of length 900 steps where setpoints are kept constant, but resampled every 150 steps. To ensure sufficient diversity of the state trajectories and transitions used to update the parameters of the RL controller, we sample initial conditions as shown in Table 10.2. Considering that the main objective of the simulation environment is to ready the controller for the field, we sample initial conditions mostly from the linear region of the model, as this is where the UAV model is assumed to have the most validity. Note that while the range of initial conditions is somewhat limited, there is nothing stopping the controller from exploring the full state space. Furthermore, since the initial state of the actuators are also randomized the sampled initial conditions

Table 10.2: Initial conditions for the episodes are uniformly sampled from the indicated ranges.

Variable	Range	Unit	Variable	Range	Unit
ϕ	-40, 40	degrees	ϕ_r	-60, 60	degrees
θ	-15, 15	degrees	θ_r	-25, 20	degrees
V_a	13, 26	m/s	α	-8, 8	degrees
ω	-60, 60	degrees	β	-10, 10	degrees
$\delta_{r,l}$	-30, 30	degrees			

could cause instability, such that the controller must learn to recover.

10.2.6 RL Algorithm

To develop the RL controller we use the SAC algorithm and augment the collected data using HER [6], based on the implementation [89], with the hyperparameters listed in Table 10.3. We chose the SAC algorithm because it is off-policy, and therefore has comparatively high data-efficiency among RL methods, and furthermore the policy is explicitly trained to handle perturbations from the inherent randomness, which tends to yield more robust policies that transfer better than non-entropy-regularized algorithms. Note that we employ the technique of initializing the replay buffer of the algorithm with 5k data samples (corresponding to 100 seconds of flight at 50Hz), which is a common technique in RL to help the policy with the initial exploration phase. This data is entirely independent of the learning controller being trained and is obtained by uniformly sampling random actions from the set of possible actions in the simulator environment. This data could also stem from other sources such as historical data gathered by a human pilot or another controller, which might be more suitable when performing exploration exclusively in the field. Since this data is independent of the learning controller, we do not count it towards the data requirement of our method and do not include it in the learning curve graphs.

Table 10.3: Hyperparameters of the RL algorithm

Hyperparameter	Value	Description
γ	0.99	Discount factor of MDP
η	$3 \cdot 10^{-4}$	Learning rate
Buffer size	$5 \cdot 10^{5}$	Size of experience replay buffer
Batch size	128	Number of samples in minibatch
ϱ	0.005	Polyak averaging factor for target networks
χ	auto	Entropy coefficient, learned automatically
Target entropy	-2	Target entropy for the automatic χ learning
Train freq	100	Parameters are updated every train freq steps
Gradient steps	100	Number of gradient steps per parameter update
N goals	4	Number of imagined goals for HER per sample
HER strategy	Future	Goal selection strategy for HER

10.2.7 Experimental Platform

Our custom avionics stack is based on the system architecture developed at the NTNU UAV-Lab, where an early version is described in [206]. It consists of a Cube Orange flight controller running the (industry standard) ArduPlane open-source autopilot [1], and a Raspberry Pi 4 running the DUNE Uniform Navigation Environment [151].

The RL controller is implemented as a DUNE task in C++ with the neural network implemented in TensorFlow. Sensor data and state estimates from ArduPlane are sent through a serial connection to the Raspberry Pi, providing all necessary data for the RL controller. The neural network controller output is converted to PWM duty cycle and sent to the elevon servos using a PCA9685 servo driver, interfaced through I2C from the Raspberry Pi. A PWM multiplexer supports switching between the RL controller output and ArduPlane. This enables the pilot to always take control during testing, either through manual controls, or through ArduPlane's standard autopilot. This switching mechanism enables us to safely engage the RL controller in flight, while takeoff and landing are performed by the pilot operating our tried and tested avionics stack [206].

10.3 Experimental Results

This section presents the main experimental results, collected during two days of flight experiments at Agdenes Airfield, Breivika, Norway in September 2021. During the first day, we enjoyed calm weather and perfect flight conditions, with a mean wind speed (as estimated by ArduPlane's Kalman Filter) of less than $4\,\mathrm{m\,s^{-1}}$. The second day of flight tests, however, presented challenging weather conditions, with a lot of gusts and a mean wind speed of approximately $12.5\,\mathrm{m\,s^{-1}}$ (70 percent of the Skywalker X8's cruise speed of $18\,\mathrm{m\,s^{-1}}$).

We present three types of data, differing mainly by how roll and pitch angle references are provided:

1. References are given by the pilot, mimicking ArduPlane's fly-by-wire-A (fbwa) mode (Section 10.3.1).
2. References are provided by ArduPlane's guidance system, which is set to track a rectangular waypoint pattern (Section 10.3.2).
3. References are set by a predefined, automated series of steps (Section 10.3.3). Similar maneuvers are also performed with an implementation of the ArduPlane PID attitude controller (described in Appendix A), with the response compared to that of the RL controller.

In contrast to the training phase, where the throttle actuator used to control airspeed is operated by a PI controller (see Section 9.4.3 for details on this controller), the throttle is either under manual control by the pilot (fbwa) or controlled by ArduPlane (auto/steps). In every figure presented, the dashed orange line corresponds to state reference, while in the elevon plots, the blue and orange lines correspond to the right and left elevon, respectively.

10.3.1 FBWA Mode

Fig. 10.3 shows an excerpt from the flight experiments where a human pilot decides the desired attitude of the UAV. The RL controller is able to closely track the desired attitude even for the most difficult and aggressive maneuvers, while producing smooth outputs for the actuators. We do however note a consistent steady-state error. Towards the end of the maneuver, we observe that the roll response is non-symmetric, that is, rolling to the left (towards negative roll angles) is slower than rolling in the opposite direction. We investigate and discuss this matter, as well as the steady-state error, in Section 10.4.

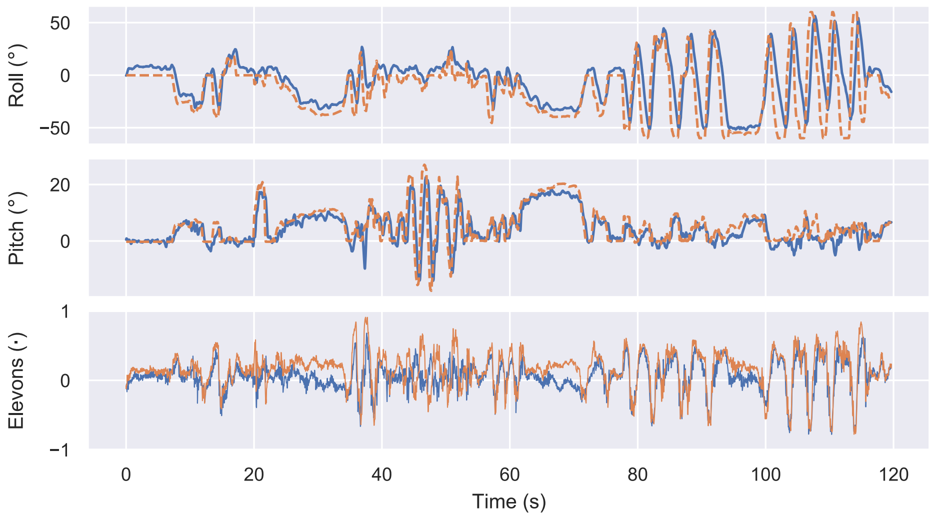


Figure 10.3: Fbwa mode using references (dashed orange line) from the pilot, showing the attitude states and generated right (blue) and left (orange) elevon signals.

10.3.2 Auto mode

Fig. 10.4 shows the results for the RL controller operating with references provided from the ArduPlane guidance system, set to track a square waypoint pattern. Before tracking the square, the UAV loiters in a circular pattern for a while. Despite some steady-state offset, especially for the roll angle error, the UAV successfully completes the specified mission. This is because the outer-loop guidance controller can compensate for this error, and still achieve convergence when faced with disturbances such as wind and offset in inner-loop control. This is similar to how a pilot supplying manual references would offset the references to keep the intended course.

During turns, a certain altitude drop is in Fig. 10.4. This is caused by the aggressive turn radius accompanied by drops in the pitch angle. This effect can be reduced by tuning the guidance system to be less aggressive (e.g. by increasing the turn radius) or reducing the maximum allowable roll angle setpoint, which is set to be 55 degrees. A similar drop in pitch angle is also seen when using the default ArduPlane attitude controller.

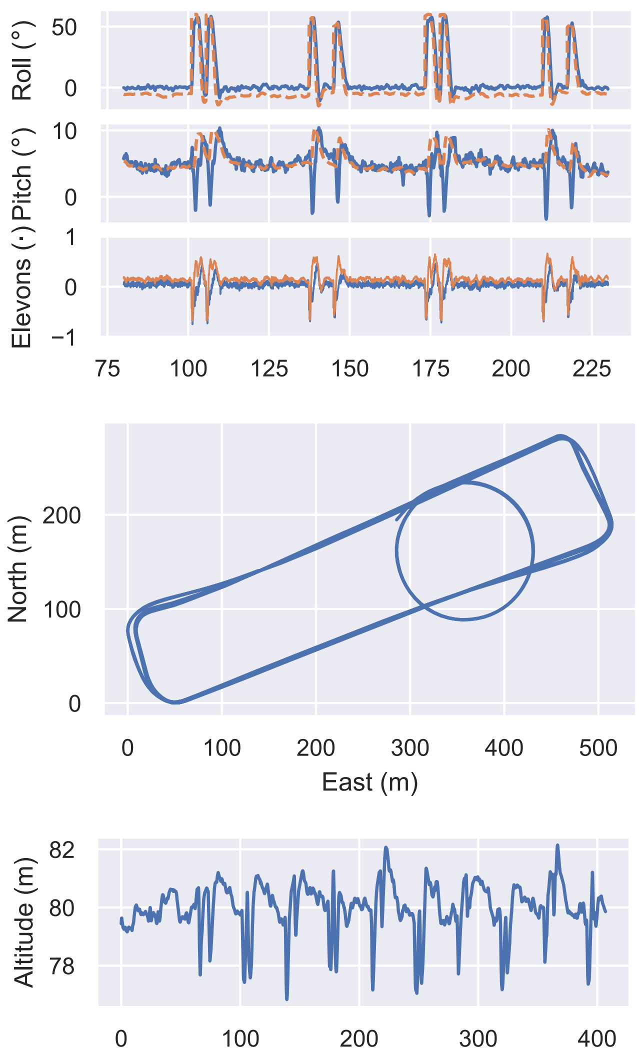


Figure 10.4: A position plot showing how the RL controller is able to take references (dashed orange line) from ArduPlane's guidance system in Auto mode, and effectively follow prescribed paths. First, a loiter, then a square waypoint pattern. Top figure shows the attitude response and elevons control signals produced when following the square pattern, middle figure shows the position, and bottom plot shows the altitude of the UAV.

10.3.3 Step sequences and Comparison with ArduPlane PIDs

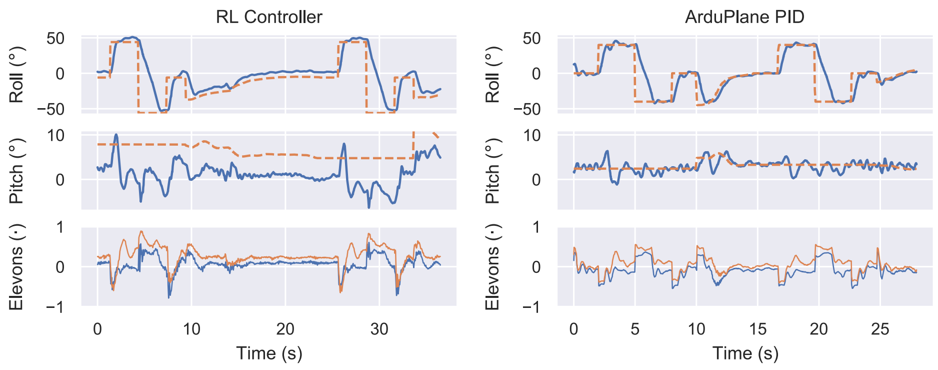


Figure 10.5: Comparison between the RL controller and the ArduPlane PID controller for steps in the roll angle.

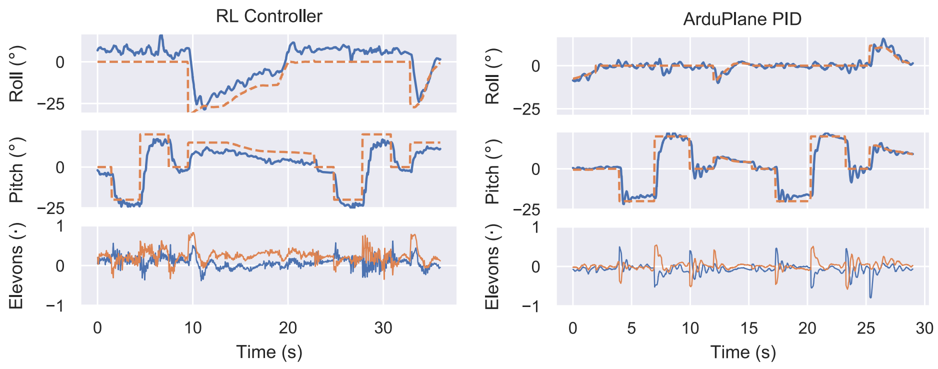


Figure 10.6: Comparison between the RL controller and the ArduPlane PID controller for steps in the pitch angle.

Step responses for the RL controller, as well as the ArduPlane PID controller, are displayed in Figs. 10.5 and 10.6. The RL controller shows comparable performance to that of ArduPlane, the main difference being the steady-state error of the RL controller. Additionally, the pitch response of the PID controller is slightly more aggressive. However, this could be changed by tuning the controller.

The control signals generated by the RL controller are relatively smooth and well-behaved but include some high-frequency components not seen in the PID response. Apart from that, the control input looks qualitatively similar, with similar magnitude. For a quantitative comparison we employ the smoothness metric defined in [139] which jointly considers the amplitudes and frequencies of the control signals:

$$Sm = \frac{2}{nf_s} \sum_{i=1}^{n} M_i f_i \tag{10.16}$$

where M_i is the amplitude of the $i'\mathrm{th}$ frequency component f_i, and f_s is the sampling frequency. On this metric the PID measures at $6.20 \cdot 10^{-4}, 6.30 \cdot 10^{-4}$ for the roll and pitch maneuvers in Fig. 10.5 and 10.6, respectively, while RL measures 2% and 44% higher at $6.30 \cdot 10^{-4}, 9.05 \cdot 10^{-4}$. This metric shows that RL has comparable smoothness in its output with the PID controller, but also indicates the higher frequency components of the RL controller's output in Fig. 10.6. It is not clear why there is such a discrepancy between the two maneuvers for the RL controller, but this data is as mentioned subjected to considerable turbulence and wind, and the discrepancy could therefore be caused by transient gusts.

While the former results were gathered on a calm day with virtually no wind, these maneuvers are executed in harsh wind conditions on day two. The UAV also suffered structural damage (not while under RL control) after collecting the PID data, before redoing the experiments with RL. The vehicle had to be repaired with a new wing and some duct tape, causing a change in the trim point of the UAV. Thus, the presented results demonstrate the RL controller's robustness towards model mismatch and varying wind conditions, including heavy gusts.

To achieve a fair comparison, the ArduPlane PID is implemented in the same software stack and ran with the same hardware architecture as the RL controller (described in Section 10.2.7). In particular, this means that any increased signal latency introduced in our setup does not affect the comparison.

The step maneuvers were executed automatically on the long sides of the rectangular pattern when running in auto mode. During the maneuver, the guidance controller is overridden with a constant reference for the channel not being tested. The steady-state offset of the RL controller caused the UAV's course to change significantly during the maneuver. Therefore, the steady-state error was compensated for as described in Section 10.4.2 such that the original desired attitude is achieved (to avoid turning while performing pitch maneuvers for instance). In the presented figures, however, we display the original non-compensated references to avoid giving the impression that the RL controller is free of steady-state error (except for Fig. 10.8).

10.4 Discussion

The experimental results of Section 10.3 show that the RL controller performs well compared to a state-of-the-art open-source autopilot, and is robust to disturbances caused by harsh wind conditions. The control performance of the RL controller across the various flight modes speaks to its ability to generalize further than just the the maneuvers encountered during training. In particular, no guidance controller was present during training.

Despite the promising results, there is room for improvement. In this section, we further discuss how performance can be improved, the iterative development process, training, and we perform a linear analysis to gain further insight into the behaviour of the RL controller.

10.4.1 Non-symmetric roll response

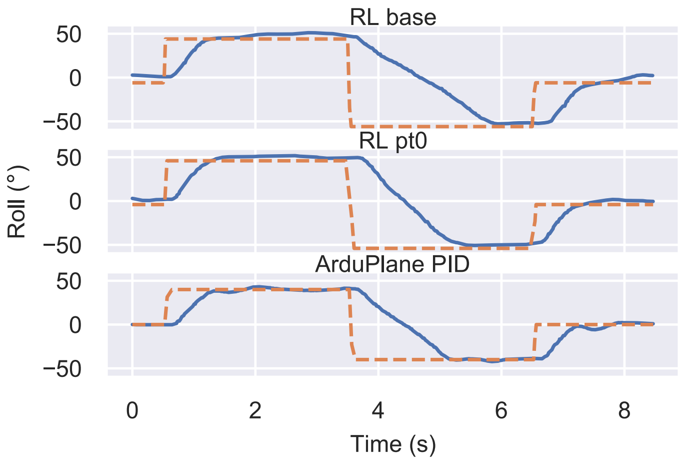


Figure 10.7: The slow roll response to the left of the base RL controller vs a controller trained to address roll symmetry (pt0) vs ArduPlane PID.

As noted in Section 10.3.1, the roll response of the RL controller is non-symmetric, meaning that rolling towards the left wing is slower than rolling to the right. This is supported by the pilot's qualitative assessment during flight. We found that this effect was caused by model mismatch, in particular an overestimation of the propeller torque effect in the simulation model. While the rotating propeller generates a reaction torque, potentially generating roll accelerations that have to be counteracted during flight, this effect was found to be less prominent on the physical UAV than expected, presumably due to the mechanical mounting of the propeller. This causes a bias in the RL controller, which has learned to counteract the propeller torque. When tested in the field, this effect is less prominent and causes a non-symmetric roll response.

To remedy this, we trained a new RL controller where the simulation model had no propeller torque. Fig. 10.7 shows how the roll response of this new controller (pt0) is closer to that of the ArduPlane PID than it is to the original (base).

10.4.2 Steady-State Errors

We experimented with several techniques in order to address the steady-state error of the RL controller observed in flight experiments: pure integrator (no decay), higher decay factor (e.g. 0.999), having integration separate from the neural network controller with learned integration gains, shaped rewards, and training with input disturbances. Whereas some of these measures reduced the steady-state error to some degree, none were successful in entirely eliminating it.

We note that there is no consistent steady-state error in the simulator in the same way we observe in field experiments, i.e. consistently over or under the reference with a consistent magnitude. The controller has learned to use integral action to reduce steady-state error from disturbances in the simulator, but not in a way that transfers to the field. This could be because the controller is overfitting to the simulator, thus the larger tracking errors in the field combined with the hyperbolic tangent saturating functions of the neural network causes the integrator states' effect on the output to saturate prematurely.

An effective way to address this problem is to estimate the steady-state error and then add the estimated value to the references provided to the RL controller, as was done in the flight experiment shown in Fig. 10.8. As can be seen, this simple technique can fully compensate for the steady-state error and may also be automated using an integrator in an outer-loop to estimate the steady-state error [127]. Furthermore, there are compelling arguments for not having integral action in the inner-loop attitude controller, as adding integral action to the controller necessarily reduces the phase margins and the achievable bandwidth [17].

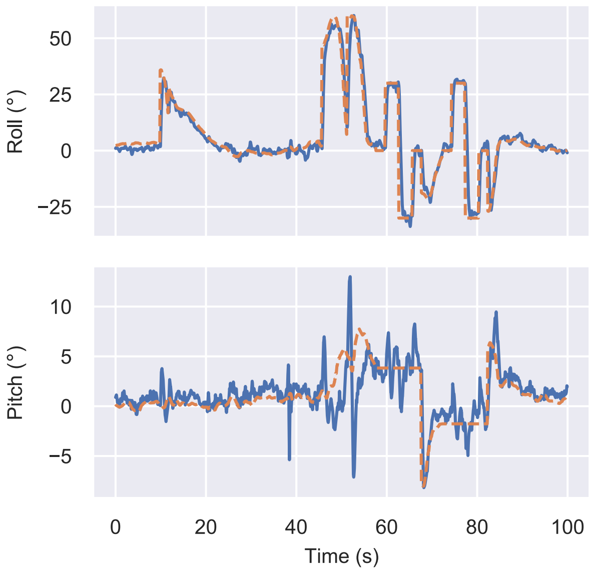


Figure 10.8: Response of the RL controller where the steady-state error has been estimated and references adjusted to compensate.

10.4.3 Oscillations: Illustration of the Iterative Development Process

Initial field experiments were characterized by excessive oscillations in the attitude response of the UAV, especially in pitch, necessitating halving the RL controller's outputs in order to keep the aircraft airborne. These oscillations were not present in the simulator, as such we suspected that this was (at least in part) caused by the simulator overestimating the natural damping present in the aircraft. We therefore reduced the C_{m_q} (pitch damping) parameter by a factor of 10. While this reduced the oscillations somewhat, there were still significant oscillations in the response, see Fig. 10.9.

We initially estimated a typical actuation latency for the system of 10ms. In sim-to-sim experiments, where we raised the latency of the control system during the exploitation phase of a controller trained with 10ms latency, we observed similar oscillatory responses as in the flight experiments and noted its relationship with increasing latency. We then trained an RL controller where the latency was set to 100ms during the learning phase and tested it under the same conditions outlined above. This controller trained with higher latency almost entirely eliminated the oscillations as shown in the simulation environment in Fig. 10.10 and in the field experiments in Fig. 10.3 to 10.11. Moreover, the controller learned with high actuation latency was robust to lower actuation latencies. Favouring robust controller design, we therefore increased the base latency of the simulation environment to 100ms, even though we believe that this is a significant overestimation of the true latency of the real system.

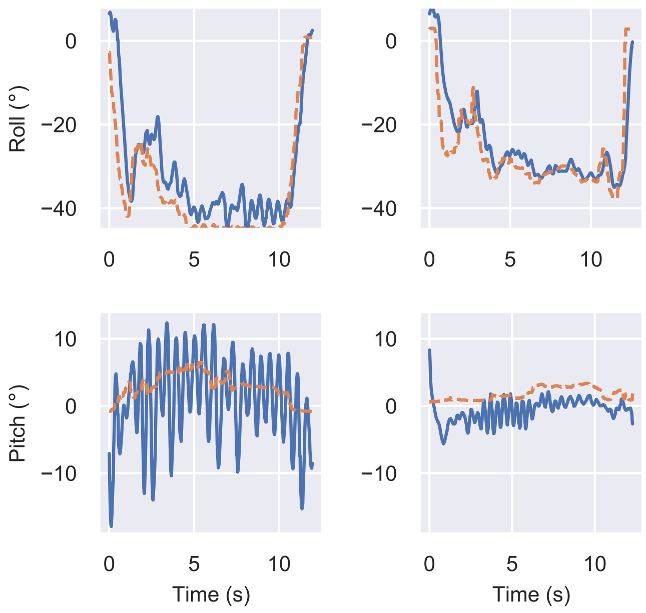


Figure 10.9: Oscillatory attitude response of initial flight experiments, for one controller trained with original (left) and one with reduced (right) C_{m_q}.

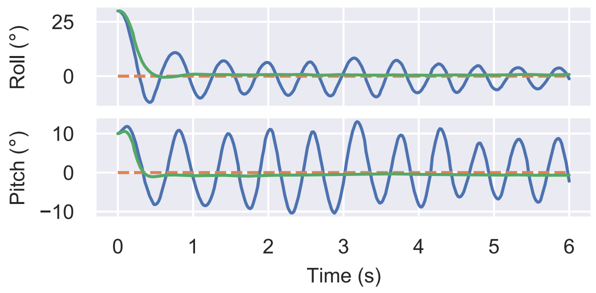


Figure 10.10: Sim-to-sim experiment with an actuation latency of 100ms, for one controller trained with 10ms latency (blue) and one controller trained with 100ms latency (green).

10.4.4 Data Requirement of the RL controller

We investigated how much data is required before the controller learns to stabilize the aircraft and can successfully transfer its strategy to the field. To this end, we evaluated a model that had only trained for 20k steps in the simulator, corresponding to 3 minutes of flight. Fig. 10.11 shows that the controller is successfully able to control the attitude of the real UAV, with acceptable control performance.

With more exposure to the simulation environment, one would expect the learning controller to increasingly adapt and specialize towards the specifics of the simulated dynamics. Since our method only requires a short period of interaction with the simulator, the degree of overfitting is limited, similar to the concept of early stopping in deep learning. With a stabilizing controller learned from the simulator as a basis, our method enables a safer approach to exploring and learning attitude controllers online in the field.

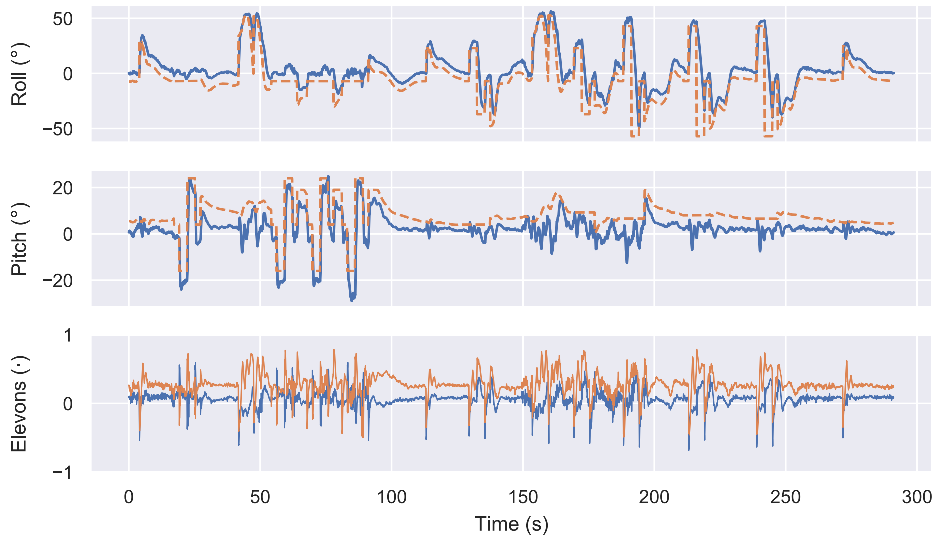


Figure 10.11: Flight experiment for the RL controller that has trained for only 3 minutes of real-time flight in the simulation environment.

10.4.5 Linear Analysis

In order to better understand how the RL attitude controller operates, we analyze its sensitivity to the input variables. In Fig. 10.12 we have plotted the open-loop response of the controller as a function of a single perturbed input. The rest of the state vector is kept constant at the steady-flight value, i.e. zero for all variables except the airspeed V_a which is set to the cruising speed of $18\,\mathrm{m\,s^{-1}}$, and the angle-of-attack α and pitch angle θ, which are kept at the trim values necessary to generate lift for level flight. As for the input values for previous time steps, we experimented with several profiles including constant, linear ramp, cubic etc., finding the qualitative behaviour to be similar for all profiles and therefore settled on the variables being constant in the time dimension. To be able to compare the results with ArduPlane, we translate the elevon outputs into virtual elevator and aileron commands using the inverse of (A.6)-(A.7). The figures and tables are presented in terms of these variables, which also have a more intuitive and straightforward effect on the roll and pitch angles.

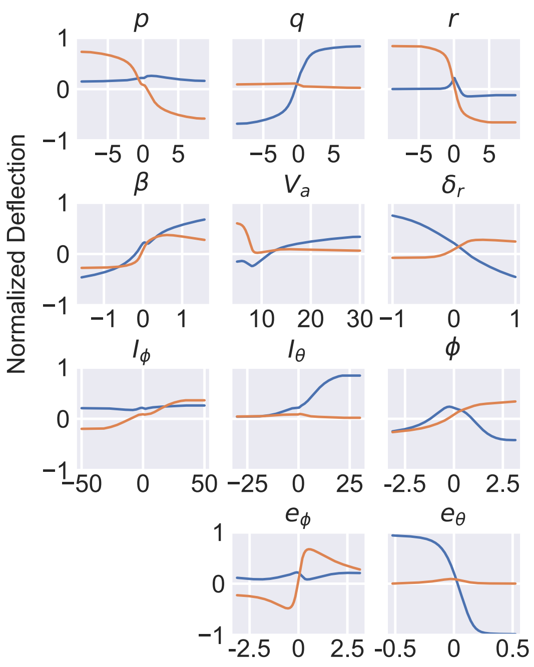


Figure 10.12: Open-loop level-flight response of the RL controller when perturbing one input at a time. The x-axis is in the units of the corresponding state. The lines are elevon outputs mapped to aileron (orange) and elevator (blue).

The saturating effect of the hyperbolic tangent nonlinearity on the RL controller is distinctly present in the responses. This is a desired effect as we know that any input should have a bounded effect on the output, which gives robustness towards possible measurement errors or misalignment of the dynamics of the simulation

Table 10.4: Linearly approximated gains at level-flight, where each input is perturbed in isolation.

Controller	$\frac{\partial \delta_a}{\partial e_\phi}$	$\frac{\partial \delta_e}{\partial e_\theta}$	$\frac{\partial \delta_a}{\partial I_\phi}$	$\frac{\partial \delta_e}{\partial I_\theta}$	$\frac{\partial \delta_a}{\partial p}$	$\frac{\partial \delta_e}{\partial q}$
RL	1.268	-3.320	-0.005	0.006	-0.008	0.223
PID	1.630	-1.081	0.052	-0.052	-0.024	0.031

and real environments. The controller makes use of all its inputs, with the previous outputs of the controller having the most significance for the current output (the typical values for most states in Fig. 10.12 are close to the level-flight value in the center and will thus use a limited range of the response curve, while the previous output of the controller frequently employs the full range). This makes sense as the controller is conditioned towards smooth outputs, as described in Section 10.2.2, which means that a reasonable initial guess of any action is to be similar to the previous action. Moreover, the fact that the previous output (left and right elevons) do not have a symmetric effect on the subsequent output (the elevons have a symmetric effect on the UAV, but there is no mechanism enforcing symmetry in the learning controller) could be a motivating factor to instead employ (virtual) elevator and aileron as outputs of the RL controller.

To get an estimate of the controller gain wrt. an input we take a linear approximation to its response curve by using the slope of the tangent line at the level-flight value as the gain estimate. These gain estimates are shown in Table 10.4, and compared to those of the ArduPlane PID controller. The RL controller is noticeably more aggressive in the pitch error, while simultaneously introducing more pitch damping through the angular velocity component q. This is evident in Fig. 10.6 where the RL controller exhibits less oscillations in the pitch response. The estimated gains for the integrator states in Table 10.4 are not representative of the response curves for these states, as the response curve exhibits cubic characteristics with a small opposing region around the level-flight value. Thus, for these states, a linear approximation over a larger region would be more descriptive. Overall, the gains of the RL controller are similar to those of the PID controller, which increases the trust in the RL controller. On the other hand, the presence of previous time step data in the input and the use of integral states giving a dynamical aspect to the controller increases the complexity of the analysis and thus limits the conclusions that can be drawn from it.

The ArduPlane PID controller (see Appendix A) includes a speed-scaling effect in its gains, motivated by the fact that the control authority of the actuators increases with airspeed, and as such the requisite deflection angle to yield a given acceleration of the UAV decreases as airspeed increases. The RL controller has not learned a similar speed-scaling effect in the sensitivities, however, it has learned to bias the response (essentially shifting the curves in Fig. 10.12 up) as airspeed increases in order to compensate for the change in trim-point with airspeed.

10.4.6 Learning Phase

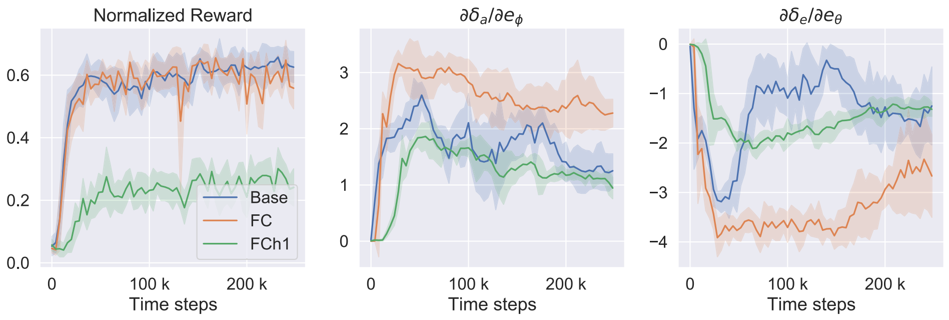


Figure 10.13: The learning phase of the proposed RL controller, showing normalized mean episode reward and error-proportional gains. The solid line represents a rolling average mean value while the shaded region represents one standard deviation over three randomly seeded controllers. Base refers to the method as presented in Section 10.2, the FC version replaces the convolutional layer with an FC layer, and the h1 version has no history in the state input vector.

The evolution of the learning phase for the RL controller as a function of time steps is shown in Fig. 10.13. For every version in Fig. 10.13, we train three controllers each with a different initial random seed, and average the results over the controllers. The rewards are normalized so that 1 corresponds to attaining the maximum reward as defined in (10.5) at every step (although this is not physically achievable) and 0 corresponds to obtaining no rewards at all. Base refers to the RL controller as presented in Section 10.2 that was used in the field experiments. To assess the contribution of the convolutional input layer, we trained one version where the input layer is replaced with an FC layer, and further to test the importance of the history of states in the state vector we train one model with an FC input layer and with $h = 1$ (labelled FCh1).

The base version learns fast, reaching convergent performance after around 40k time steps, corresponding to about 12 minutes of flight time. Moreover, we find that the training method is stable in the sense that the performance differences between controllers with different seeds is small, within a few percent. The FC version without state history is never able to learn to consistently stabilize the UAV at the desired attitude in the time frames we considered. The FC version with state history on the other hand achieves comparable rewards to the base version, showing the importance of history in the RL input state. The base version reaches peak performance slightly faster than the FC version, and further its proportional gains are considerably lower. This is also evidenced by the smoothness metric (10.16) for which the base version scores 50% lower than the FC version. The gains and the smoothness metric indicates that the convolutional input layer provides a superior ability to predict the system response and thus provide smoother response in attitude and control signals, while the FC version is more reactive and oscillatory.

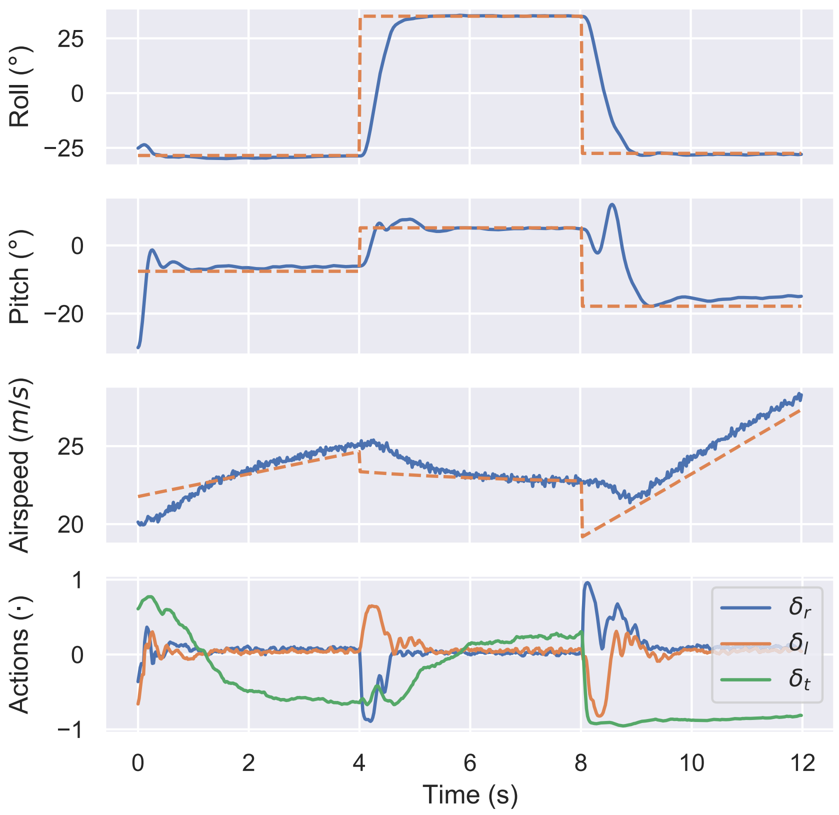


Figure 10.14: The method presented in Section 10.2 can with minor adjustments be made to support controlling the airspeed as well, and learns to solve the coupled pitch and airspeed control problem. Here illustrated in an episode of the simulation environment, where δ_t is the throttle command (-1 corresponds to no throttle, and 1 to maximum throttle).

10.5 Conclusion

This chapter has presented a data-efficient method for learning attitude controllers for fixed wing UAVs using RL. The learning controller is able to operate directly on the nonlinear dynamics, and therefore could extend the flight envelope and capabilities of autopilots. The high data-efficiency of the presented method facilitates transfer to control of the real UAV by limiting overfitting to the simulated model. We demonstrate that the learned controller has comparable performance to the existing state-of-the-art ArduPlane PID autopilot, and is capable of tracking prescribed paths from a guidance system while generating smooth actuation signals and attitude responses.

Following this chapter's demonstration of RL's ability to perform low-level attitude control of fixed-wing UAVs, it should be investigated if RL can handle more complex tasks and longstanding challenges in automatic flight control, such as recovery from loss of control, aerobatic maneuvers, deep-stall landing, and end-to-end path following. Before attempting these tasks, the problem of the limited integral action of the RL attitude controller should be investigated. Moreover, learning from real data, be it historical or generated online by the learning controller, is an intriguing further work that could incentivize integral action and exploration of the true nonlinear regions of the attitude control problem.

One such extension is including the airspeed as a controlled state, as was done in Chapter 9. We chose to focus on the attitude control problem in this chapter, as it is common in flight-control to detach the control of the attitude and pitch angles into separate levels of the control stack due to their coupling. However, simulation experiments show that the SAC RL controller could easily handle the addition of the airspeed as a controlled state as shown in Figure 10.14. The RL controller has learned to use the coupling of the pitch and airspeed to achieve both objectives, as can be seen around 9 seconds in the episode, where it pitches up to decrease its airspeed. We wanted to test this controller also in the field, but regrettably did not get an opportunity to do so.

As in Part I of this thesis we present in this part a dichotomy between two methods where one is based on the on-policy RL algorithm PPO (Chapter 9) and the other is based on the off-policy algorithm SAC (Chapter 10). Again, we observe that the off-policy method has far superior data-efficiency, reaching convergent performance after around 200k time steps for the PPO attitude controller, vs around 40k steps for the SAC attitude controller, albeit with the caveat that the PPO algorithm operates at twice the control frequency such that its real-time total flight time to reach convergence must be halved to be compared to the SAC controller. However, this is even with the SAC controller learning from sparse rewards, which is generally considered a more difficult learning problem than with the dense rewards the PPO controller learns from. Another major difference between the two controllers is the approach taken to ensure smooth outputs from the controller. The PPO controller has an action smoothness term included in the reward signal, while the SAC controller's reward signal is purely a function of the state with the action smoothness enforced through its objective. We found the latter method to work significantly better for the SAC controller, that is, yielding superior control performance, better smoothness, and faster convergence than the reward signal approach. We did not experiment with this approach for the PPO controller, as the method was publicized after our proof of concept work was performed, but we would expect it to yield similar benefits also in this setting.

The better data-efficiency and ability of the SAC controller to learn from previous data (e.g. gathered by pilot or Arduplane autopilot) makes it a natural choice for future work where learning is performed entirely online on the real UAV. However, there is a case to be made for the lower computational complexity of the PPO algorithm, as well as the tendency of policy gradient methods to have smoother behavioural changes between parameter updates than value-based methods, although this difference is less pronounced with DDPG-like algorithms such as SAC (as they also maintain a parameterized policy, rather than simply selecting the maximizing action of the Q-function directly, which can lead to sharp changes from parameter updates when actions are close in value).

Part III

Concluding Remarks

11

Conclusions and Future Possibilities

This thesis has presented methods for enhancing conventional knowledge-based control techniques with model-free RL, as well as using existing control techniques to enhance RL controllers. It has also contributed to RL research by demonstrating a successful application of RL-based low-level control in the field, something which is sorely lacking in RL research. Finally, it has contributed to the attitude control problem of fixed-wing UAVs by proposing and demonstrating the use of RL-based control, which promises to extend the flight envelope into the more nonlinear regions while being more computationally efficient than other nonlinear methods such as NMPC.

In Part I of the thesis, we presented the novel idea of optimizing the meta-parameters of the MPC scheme using RL. We explored this idea in three chapters (Chapters 4, 5, and 6). We demonstrated that optimization of the recomputation problem, i.e. deciding when to compute the solution to the OCP, can be successfully learned under unmeasured disturbances (Chapter 4) and under model mismatch (Chapter 6). We present two methods for learning the optimal prediction horizon of the MPC scheme using RL, one based on the horizon variable modelled as a discretized Gaussian variable and learned with SAC (Chapter 5), and one based on a discrete model of the horizon variable with the GPD using PPO (Chapter 6), each with their own strengths in terms of data-efficiency and adaptivity. Finally, Chapter 6 presented a unified framework in which both of these meta-parameters are jointly optimized, along with any other parameter of the control algorithm that one wishes to optimize. Part I of the thesis is concluded with Chapter 7, in which we presented the improved Q-filter approach to guiding the RL controller's exploration phase, and also highlighted a relationship between the framework of Chapter 6 and the topic of guiding. This relation is something we wanted to look more into given more time for the PhD.

Less is truly more. We started investigating the idea of meta-parameter optimization with the goal of reducing the computational complexity of the MPC scheme, thus increasing the viability of the MPC technique for control applications with limited hardware platforms or limited energy resources. The experiments undertaken confirm that this objective was achieved, with reductions of the processing time of the control algorithms ranging between 30% and 60% in the various experiments. Additionally, we quickly realized that this approach could also significantly improve the control performance of the MPC scheme, by computing it less! We conjecture that the improved control performance is due to the finite-horizon nature of the MPC scheme and how the optimality of the computed solutions interact with the initial conditions that the OCP is computed from, such that by intelligently selecting when to compute the MPC one can receive more optimal control sequences from the MPC. The inverted pendulum control problem is used throughout the thesis to investigate meta-parameter optimization of the MPC scheme. A similar conclusion of non-obvious interaction between the horizon of the NMPC and its control performance is reached in [92] for a physical (i.e. non-simulated) inverted pendulum system, giving credence that our results and conclusions will hold true also in the physical world. A natural extension of this work is therefore looking closer into how and when the optimization of the recomputation meta-parameter leads to increased control performance.

Experimentally verifying the meta-parameter optimization framework in the physical world is something that we wanted to do but did not have time for. The research group at the NTNU UAV-Lab has developed an NMPC that can run in real-time on the Skywalker X8 UAV in flight, which provides an interesting opportunity to test the methods of Part I in the field. In addition to experimental verification and guiding of learning controllers, we also considered other extensions of the meta-parameter optimization framework of Part I. In mixed-integer optimization problems, the computational complexity grows in general exponentially in the horizon [190]. The mixed-integer OCP is therefore typically solved exactly in the original integer variables only for a portion of the prediction horizon, after which the problem is relaxed to its continuous counterpart, which scales more favourably in the horizon [11]. However, this relaxation necessitates a projection back to integer values and subsequent feasibility verification of the solution, further adding to the computational complexity. The choice of at what point of the OCP the problem is relaxed is therefore of great importance for the computational complexity of mixed-integer MPC. Following [73] showing how mixed-integer MPC can be combined with RL, we propose that this relaxation point can be considered a meta-parameter of the mixed-integer MPC scheme, and optimized with the tools presented in Part I of this thesis.

Part II of the thesis presented DRL-based dynamic control of fixed-wing UAVs, including experimental verification of the proposed approach in the field. The first chapter of Part II motivated the utility of the fixed-wing UAV design and highlighted its unique importance for Norway. Chapter 9 then presented the first work we did on this topic, in which we pioneered the idea of using DRL for attitude control of fixed-wing UAVs. In order to investigate this approach, an open-source flight simulator was developed in which the learned attitude controller could explore and later be evaluated on its proficiency. This simulator was made publicly available [34, 35] and has since gained traction in the research community. The chapter goes into detail on the UAV model, the attitude control problem, and the literature and previous research efforts on combining RL with flight control. It then presented the approach we took in formulating the RL attitude controller and the control problem, and presented a comparison to the state-of-the-art approach, demonstrating that RL was at least equally successful in stabilizing the aircraft and exhibited superior robustness towards wind and turbulence disturbances. Following these encouraging results, we targeted control of the real UAV in the field, and developed a new method for learning attitude controllers that focused on data-efficiency, achieving a data-efficiency more that an order of magnitude better than previously reported works. This work was presented in Chapter 10, along with results from the field experiments, showing that the RL controller is competitive with the industry-standard ArduPlane autopilot.

We highlight in Chapter 10 that employing RL control in the real world was an iterative process, especially when the controller is learned in a simulation of the target control system. The model mismatch between the simulation and the real system led to numerous challenges when deploying the RL controller on the real UAV. We were able to address most of these issues, e.g. the oscillatory attitude response and the non-symmetric roll response, however, we were not able to eliminate the steady-state tracking error of the RL controller. While we argue in Chapter 10 that steady-state error in the inner-loop controller is not necessarily a problem as we show that it can effectively be addressed in the outer-loop, it is somewhat frustrating that we despite best efforts were not able to alleviate the steady-state error. Future work on this topic should therefore investigate this issue closer, perhaps even taking a more general look at how RL control interacts with integral action. This issue could also conceivably originate entirely from the model mismatch, such that learning online on the real system could be a solution to this problem that should be examined. Further, as discussed in the conclusion of Chapter 10, the obvious next step after demonstrating RL's suitability for attitude control is to attempt more complex flight control tasks, taking advantage of RL's nonlinear control abilities. We showed that RL can learn to solve the coupled pitch and airspeed control problem, but other longstanding challenges in flight control such as recovery from loss of control are alluring targets for future work.

Appendix

A

The ArduPlane Attitude Controller

This section presents the main equations used for attitude control in ArduPlane [1], which is a state-of-the-art open-source autopilot for fixed-wing UAVs. This controller is used as a baseline comparison for the RL attitude controller in Chapter 10 and to support the discussion in Section 10.4.5. The equations are based on ArduPlane, Release 4.0.9, which is the most recent stable release (as of August 2021).

The ArduPlane attitude controller consists of two cascaded single-input-single-output (SISO) feedback loops. The elevator controls pitch angle, while the ailerons are used for roll control. The inner loop consists of proportional controllers, where desired roll and pitch rates $p_r, q_r \in \mathbb{R}$ are calculated according to

$$p_r = k_\phi \left(\phi_r - \phi\right) \tag{A.1}$$

$$q_r = k_\theta \left(\theta_r - \theta\right) + q_{ct}, \tag{A.2}$$

where $k_\phi, k_\theta > 0$ and q_{ct} is the pitch rate offset needed to maintain height in a coordinated turn, given by

$$q_{ct} = \sin(\phi)\cos(\theta)\frac{g}{V_a}\tan(\phi). \tag{A.3}$$

The rate setpoints are inputs to the inner loop, which consists of proportional-integral (PI) controllers with feedforward action:

$$\delta_a = k_{p,p}\nu^2 \left(p_r - p\right) + \int_0^t k_{i,p}\nu^2 \left(p_r - p\right) d\tau + k_{ff,p}\nu p_r \tag{A.4}$$

$$\delta_e = -k_{p,q}\nu^2 \left(q_r - q\right) - \int_0^t k_{i,q}\nu^2 \left(q_r - q\right) d\tau - k_{ff,q}\nu q_r, \tag{A.5}$$

where $k_{p,*}$, $k_{k_i,*}$ and $k_{ff,*}$ are proportional, integral and feedforward gains, respectively. The variable $\nu = V^*/V_a$, where V^* is some constant reference airspeed, provides airspeed scaling of the controller parameters, accounting for the fact that larger airspeeds give greater aerodynamic control authority. The negative sign in the control law for δ_e is introduced to account for the convention that positive elevator deflections yield a negative pitch moment [17].

For UAVs equipped with a rudder, additional control loops utilize the extra control surface for turn coordination. However, as the Skywalker X8 considered in this thesis is rudderless, this part of the control algorithm is not relevant here.

For an elevon plane like the Skywalker X8, the aileron and elevator deflection angles are virtual control signals that are mapped to elevon control actions using the linear map

$$\delta_l = \delta_e + \delta_a \tag{A.6}$$

$$\delta_r = \delta_e - \delta_a. \tag{A.7}$$

By assuming a constant airspeed $V_a = 18\,\mathrm{m\,s^{-1}}$ and inserting parameters used for the Skywalker X8 UAV at the NTNU UAV-lab, we get the following sensitivities for the elevator and aileron control signals:

$$\left.\frac{\partial \delta_e}{\partial e_\theta}\right|_{\theta=\phi=0} = -1.0813 \qquad \frac{\partial \delta_a}{\partial e_\phi} = 1.6299 \tag{A.8}$$

$$\left.\frac{\partial \delta_e}{\partial q}\right|_{\theta=\phi=0} = 0.0312 \qquad \frac{\partial \delta_a}{\partial p} = -0.0243 \tag{A.9}$$

$$\left.\frac{\partial \delta_e}{\partial I_\theta}\right|_{\theta=\phi=0} = -0.0521 \qquad \frac{\partial \delta_a}{\partial I_\phi} = 0.0521 \tag{A.10}$$

$$\left.\frac{\partial \delta_e}{\partial \theta}\right|_{\theta=\phi=0} = 0.0104 \qquad \frac{\partial \delta_a}{\partial \phi} = -0.0104, \tag{A.11}$$

where $I_\phi = \int_0^t e_\phi d\tau$ and $I_\theta = \int_0^t e_\theta d\tau$ correspond to the (unbounded) integrator states of the RL controller.

Bibliography

[1] ArduPilot open source drone software. `https://ardupilot.org/`. Accessed: 2021-10-09.

[2] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1–8. MIT Press, 2007.

[3] Thivaharan Albin, Dennis Ritter, Dirk Abel, Norman Liberda, Rien Quirynen, and Moritz Diehl. Nonlinear mpc for a two-stage turbocharged gasoline engine airpath. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 849–856. IEEE, 2015.

[4] F. Allgöwer, T. A. Badgwell, J. S. Qin, J. B. Rawlings, and S. J. Wright. Nonlinear predictive control and moving horizon estimation — an introductory overview. In Paul M. Frank, editor, *Advances in Control*, pages 391–449, London, 1999. Springer London. ISBN 978-1-4471-0853-5.

[5] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019. doi: 10.1007/s12532-018-0139-4.

[6] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5055–5065, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[7] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

[8] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[9] André Anglade, Jean-Marie Kai, Tarek Hamel, and Claude Samson. Automatic control of convertible fixed-wing drones with vectorized thrust. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 5880–5887, 2019. doi: 10.1109/CDC40024.2019.9029274.

[10] Anil Aswani, Humberto Gonzalez, S Shankar Sastry, and Claire Tomlin. Provably safe and robust learning-based model predictive control. *Automatica*, 49(5):1216–1226, 2013.

[11] Daniel Axehill, Lieven Vandenberghe, and Anders Hansson. Convex relaxations for mixed integer predictive control. *Automatica*, 46(9):1540–1545, 2010. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2010.06.015.

[12] Ahmad Taher Azar, Anis Koubaa, Nada Ali Mohamed, Habiba A. Ibrahim, Zahra Fathy Ibrahim, Muhammad Kazim, Adel Ammar, Bilel Benjdira, Alaa M. Khamis, Ibrahim A. Hameed, and Gabriella Casalino. Drone deep reinforcement learning: A review. *Electronics*, 10(9), 2021. ISSN 2079-9292. doi: 10.3390/electronics10090999.

[13] J Andrew Bagnell and Jee G Schneider. Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. In *2001 IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[14] Lisanne Bainbridge. Ironies of automation. In *Analysis, design and evaluation of man–machine systems*, pages 129–135. Elsevier, 1983.

[15] Somil Bansal, Roberto Calandra, Ted Xiao, Sergey Levine, and Claire J Tomlin. Goal-driven dynamics learning via bayesian optimization. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 5168–5173. IEEE, 2017.

[16] Dominik Baumann, Jia-Jie Zhu, Georg Martius, and Sebastian Trimpe. Deep reinforcement learning for event-triggered control. In *Proceedings of the 57th IEEE International Conference on Decision and Control (CDC)*, pages 943–950, December 2018.

[17] R. W. Beard and T. W. McLain. *Small Unmanned Aircraft*. Princeton University Press, Princeton, NJ, 2012. ISBN 0691149216.

[18] Randal W. Beard. UAVBOOK Supplement. Additional thoughts on propeller thrust model. Technical report, Princeton University Press, 2014.

[19] Randal W. Beard and Timothy W. McLain. *Small Unmanned Aircraft : Theory and Practice*. Princeton University Press, 2012. ISBN 9780691149219.

[20] Alberto Bemporad and Manfred Morari. Robust model predictive control: A survey. In *Robustness in identification and control*, pages 207–226. Springer, 1999.

[21] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3 – 20, 2002. ISSN 0005-1098. doi: https://doi.org/10.1016/S0005-1098(01)00174-1.

[22] J.D.J. Barradas Berglind, T.M.P. Gommans, and W.P.M.H. Heemels. Self-triggered mpc for constrained linear systems and quadratic costs. *IFAC Proceedings Volumes*, 45(17):342 – 348, 2012. ISSN 1474-6670. doi: https://doi.org/10.3182/20120823-5-NL-3013.00058. 4th IFAC Conference on Nonlinear Model Predictive Control.

[23] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[24] Nicola Bernini, Mikhail Bessa, Rémi Delmas, Arthur Gold, Eric Goubault, Romain Pennec, Sylvie Putot, and François Sillion. A few lessons learned in reinforcement learning for quadcopter attitude control. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2021.

[25] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

[26] Eivind Bøhn, Erlend M Coates, Signe Moe, and Tor Ame Johansen. Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 523–533. IEEE, 2019.

[27] Eivind Bøhn, Erlend M. Coates, Dirk Reinhardt, and Tor Arne Johansen. Data-efficient deep reinforcement learning for attitude control of fixed-wing uavs validated through field experiments. *IEEE Transactions on Neural Networks and Learning Systems*, 2021. Submitted.

[28] Eivind Bøhn, Sebastien Gros, Signe Moe, and Tor Arne Johansen. Optimization of the model predictive control meta-parameters through reinforcement learning. *IEEE Transactions on Cybernetics*, 2021. Submitted.

[29] Eivind Bøhn, Signe Moe, and Tor Arne Johansen. On the effects of properties of the minibatch in reinforcement learning. *Accepted into the 4th International Conference on Intelligent Technologies and Applications*, 2021.

[30] Haitham Bou-Ammar, Holger Voos, and Wolfgang Ertel. Controller design for quadrotor UAVs using reinforcement learning. In *2010 IEEE International Conference on Control Applications*, pages 2130–2135. IEEE, sep 2010. ISBN 978-1-4244-5362-7. doi: 10.1109/CCA.2010.5611206.

[31] Alex Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[32] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, June 2016. arXiv: 1606.01540.

[33] Eitan Bulka and Meyer Nahon. Automatic control for aerobatic maneuvering of agile fixed-wing uavs. *J Intell Robot Syst*, 93:85–100, 2019. doi: https://doi.org/10.1007/s10846-018-0790-z.

[34] Eivind Bøhn. Pyfly. `https://github.com/eivindeb/pyfly`, 2019.

[35] Eivind Bøhn. Fixed-wing aircraft gym environment. `https://github.com/eivindeb/fixed-wing-gym`, 2019.

[36] Eivind Bøhn, Signe Moe, and Tor Arne Johansen. Accelerating reinforcement learning with suboptimal guidance. *IFAC-PapersOnLine*, 53(2):8090–8096, 2020. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2020.12.2278. 21st IFAC World Congress.

[37] Eivind Bøhn, Sebastien Gros, Signe Moe, and Tor Arne Johansen. Optimization of the model predictive control update interval using reinforcement learning. *IFAC-PapersOnLine*, 54(14):257–262, 2021. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2021.10.362. 3rd IFAC Conference on Modelling, Identification and Control of Nonlinear Systems MICNON 2021.

[38] Eivind Bøhn, Sebastien Gros, Signe Moe, and Tor Arne Johansen. Reinforcement learning of the prediction horizon in model predictive control. *IFAC-PapersOnLine*, 54(6):314–320, 2021. ISSN 2405-8963. doi:

https://doi.org/10.1016/j.ifacol.2021.08.563. 7th IFAC Conference on Nonlinear Model Predictive Control NMPC 2021.

[39] R. Cagienard, P. Grieder, E. C. Kerrigan, and M. Morari. Move blocking strategies in receding horizon control. In *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, volume 2, pages 2023–2028 Vol.2, 2004. doi: 10.1109/CDC.2004.1430345.

[40] Ignacio Carlucho, Mariano De Paula, Sen Wang, Yvan Petillot, and Gerardo G. Acosta. Adaptive low-level control of autonomous underwater vehicles using deep reinforcement learning. *Robotics and Autonomous Systems*, 107:71–86, sep 2018. ISSN 0921-8890. doi: 10.1016/J.ROBOT.2018.05.016.

[41] Herman Castañeda, Oscar S. Salas-Peña, and Jesús de León-Morales. Extended observer based on adaptive second order sliding mode control for a fixed wing UAV. *ISA Transactions*, 66:226–232, jan 2017. doi: 10.1016/j.isatra.2016.09.013.

[42] A. Chakrabarty, S. Zavitsanou, F. J. Doyle, and E. Dassau. Event-triggered model predictive control for embedded artificial pancreas systems. *IEEE Transactions on Biomedical Engineering*, 65(3):575–586, 2018. doi: 10.1109/TBME.2017.2707344.

[43] Cheng Chin and Michael Lau. Modeling and testing of hydrodynamic damping model for a complex-shaped remotely-operated vehicle for control. *Journal of Marine Science and Application*, 11(2):150–163, 2012.

[44] Robert J Clarke, Liam Fletcher, Colin Greatwood, Antony Waldock, and Thomas S Richardson. Closed-loop q-learning control of a small unmanned aircraft. In *AIAA Scitech 2020 Forum*, page 1234, 2020.

[45] Erlend M. Coates, Andreas Wenz, Kristoffer Gryte, and Tor Arne Johansen. Propulsion system modeling for small fixed-wing uavs. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 748–757, 2019. doi: 10.1109/ICUAS.2019.8798082.

[46] Rick Cory and Russ Tedrake. Experiments in fixed-wing uav perching. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008. doi: 10.2514/6.2008-7256.

[47] Torbjørn Cunis, Dominic Liao-McPherson, Ilya Kolmanovsky, and Laurent Burlion. Model-predictive spiral and spin upset recovery control for the

generic transport model simulation. In *2020 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1–7, 2020. doi: 10.1109/CCTA41146.2020.9206158.

[48] Hugo Andrade de Oliveira and Paulo Fernando Ferreira Rosa. Genetic neuro-fuzzy approach for unmanned fixed wing attitude control. In *2017 International Conference on Military Technologies (ICMT)*. IEEE, may 2017. doi: 10.1109/miltechs.2017.7988808.

[49] Hakan Demirtas. On accurate and precise generation of generalized poisson variates. *Communications in Statistics - Simulation and Computation*, 46 (1):489–499, 2017. doi: 10.1080/03610918.2014.968725.

[50] Travis Dierks and Sarangapani Jagannathan. Neural network output feedback control of robot formations. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 40(2):383–399, 2009.

[51] Thomas Duriez, Steven L Brunton, and Bernd R Noack. *Machine learning control-taming nonlinear dynamics and turbulence*. Springer, 2017.

[52] William Edwards, Gao Tang, Giorgos Mamakoukas, Todd Murphey, and Kris Hauser. Automatic tuning for data-driven model predictive control. In *International Conference on Robotics and Automation (ICRA)*, 2021.

[53] Jonathan St BT Evans. In two minds: dual-process accounts of reasoning. *Trends in cognitive sciences*, 7(10):454–459, 2003.

[54] Maryam Fazel, Rong Ge, Sham Kakade, and Mehran Mesbahi. Global convergence of policy gradient methods for the linear quadratic regulator. In *International Conference on Machine Learning*, pages 1467–1476. PMLR, 2018.

[55] Fan Fei, Zhan Tu, Dongyan Xu, and Xinyan Deng. Learn-to-recover: Retrofitting uavs with reinforcement learning-assisted flight control under cyber-physical attacks. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7358–7364. IEEE, 2020.

[56] Le Feng, Christian Gutvik, Tor Johansen, Dan Sui, and Alf Brubakk. Approximate explicit nonlinear receding horizon control for decompression of divers. *Control Systems Technology, IEEE Transactions on*, 20:1275–1284, 09 2012. doi: 10.1109/TCST.2011.2162516.

[57] Jaime F Fisac, Anayo K Akametalu, Melanie N Zeilinger, Shahab Kaynama, Jeremy Gillula, and Claire J Tomlin. A general safety framework for

learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control*, 64(7):2737–2752, 2018.

[58] P. Fitzpatrick. Calculation of thrust in a ducted fan assembly for hovercraft. Technical report, Hovercraft Club of Great Britain, 2003.

[59] Liam J Fletcher, Robert J Clarke, Thomas S Richardson, and Mark Hansen. Reinforcement learning for a perched landing in the presence of wind. In *AIAA Scitech 2021 Forum*, page 1282, 2021.

[60] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1515–1528. PMLR, 10–15 Jul 2018.

[61] Marco Forgione, Dario Piga, and Alberto Bemporad. Efficient Calibration of Embedded MPC. In *Proc. of the 21st IFAC World Congress, Berlin, Germany*, 2020.

[62] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

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

[64] D. Gandhi, L. Pinto, and A. Gupta. Learning to fly by crashing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3948–3955, September 2017. doi: 10.1109/IROS.2017.8206247.

[65] Gonzalo A. Garcia, Shawn Kashmiri, and Daksh Shukla. Nonlinear control based on h-infinity theory for autonomous aerial vehicle. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, jun 2017. doi: 10.1109/icuas.2017.7991395.

[66] M. S. M. Gardezi and A. Hasan. Machine learning based adaptive prediction horizon in finite control set model predictive control. *IEEE Access*, 6: 32392–32400, 2018. doi: 10.1109/ACCESS.2018.2839519.

[67] Chowdhary Vinayak Girish, Frazzoli Emilio, How P. Jonathan, and Liu Hugh. Nonlinear flight control techniques for unmanned aerial vehicles. In *Handbook of Unmanned Aerial Vehicles*, pages 577–612. Springer Netherlands, aug 2014. doi: 10.1007/978-90-481-9707-1_87.

[68] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2015.

[69] Gregor Goebel and Frank Allgöwer. A simple semi-explicit mpc algorithm. *IFAC-PapersOnLine*, 48(23):489 – 494, 2015. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2015.11.326. 5th IFAC Conference on Nonlinear Model Predictive Control NMPC 2015.

[70] Ravi Gondhalekar, Eyal Dassau, and Francis J. Doyle III. Tackling problem nonlinearities & delays via asymmetric, state-dependent objective costs in mpc of an artificial pancreas. *IFAC-PapersOnLine*, 48(23):154 – 159, 2015. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2015.11.276. 5th IFAC Conference on Nonlinear Model Predictive Control NMPC 2015.

[71] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(9), 2004.

[72] Sébastien Gros and Mario Zanon. Data-driven economic nmpc using reinforcement learning. *IEEE Transactions on Automatic Control*, 65(2):636–648, 2019.

[73] Sebastien Gros and Mario Zanon. Reinforcement learning for mixed-integer problems based on mpc. *IFAC-PapersOnLine*, 53(2):5219–5224, 2020.

[74] Sébastien Gros, Rien Quirynen, and Moritz Diehl. Aircraft control based on fast non-linear mpc & multiple-shooting. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 1142–1147. IEEE, 2012.

[75] Sébastien Gros and Mario Zanon. Bias correction in reinforcement learning via the deterministic policy gradient method for mpc-based policies. In *2021 American Control Conference (ACC)*, pages 2543–2548, 2021. doi: 10.23919/ACC50511.2021.9483016.

[76] Sébastien Gros and Mario Zanon. Reinforcement learning based on mpc and the stochastic policy gradient method. In *2021 American Control Conference (ACC)*, pages 1947–1952, 2021. doi: 10.23919/ACC50511.2021.9482765.

[77] L. Grüne and J. Pannek. *Nonlinear Model Predictive Control.* Springer-Verlag, London, 2011.

[78] Kristofer Gryte, Richard Hann, Mushfiqul Alam, Jan Roháč, Tor Arne Johansen, and Thor I. Fossen. Aerodynamic modeling of the skywalker x8 fixed-wing unmanned aerial vehicle. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 826–835, 2018. doi: 10.1109/ICUAS.2018.8453370.

[79] Kristoffer Gryte. High Angle of Attack Landing of an Unmanned Aerial Vehicle. Master's thesis, Norwegian University of Science and Technology, 2015.

[80] Kristoffer Gryte. Precision control of fixed-wing uav and robust navigation in gnssdenied environments. 2020.

[81] Kristoffer Gryte. *Precision control of fixed-wing UAV and robust navigation in GNSSdenied environments*. PhD thesis, Norwegian University of Science and Technology, 2020.

[82] Kristoffer Gryte, Richard Hann, Mushfiqul Alam, Jan Roháč, Tor Arne Johansen, and Thor I Fossen. Aerodynamic modeling of the Skywalker X8 Fixed-Wing Unmanned Aerial Vehicle. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018.

[83] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, November 2016.

[84] Vijaykumar Gullapalli, Judy A Franklin, and Hamid Benbrahim. Acquiring robot skills via reinforcement learning. *IEEE Control Systems Magazine*, 14(1):13–24, 1994.

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

[86] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[87] Jae-Hung Han, Dong-Kyu Lee, Jun-Seong Lee, and Sang-Joon Chung. Teaching micro air vehicles how to fly as we teach babies how to walk.

Journal of Intelligent Material Systems and Structures, 24(8):936–944, 2013. doi: 10.1177/1045389X13478270.

[88] W. P. M. H. Heemels, J. H. Sandee, and P. P. J. Van Den Bosch. Analysis of event-driven controllers for linear systems. *International Journal of Control*, 81(4):571–590, 2008. doi: 10.1080/00207170701506919.

[89] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[90] Bing-Qiang Huang, Guang-Yi Cao, and Min Guo. Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance. In *2005 International Conference on Machine Learning and Cybernetics*, volume 1, pages 85–89. IEEE, 2005.

[91] Shao-Ming Hung and Sidney N. Givigi. A Q-Learning Approach to Flocking With UAVs in a Stochastic Environment. *IEEE Transactions on Cybernetics*, 47(1):186–197, jan 2017. ISSN 2168-2267. doi: 10.1109/TCYB.2015.2509646.

[92] Ricus Husmann and Harald Aschemann. Comparison and benchmarking of nmpc for swing-up and side-stepping of an inverted pendulum with underlying velocity control. *Accepted to the 3rd IFAC Conference on Modelling, Identification and Control of Nonlinear Systems MICNON*, 2021.

[93] Y. Iino, T. Hatanaka, and M. Fujita. Event-predictive control for energy saving of wireless networked control system. In *2009 American Control Conference*, pages 2236–2242, 2009. doi: 10.1109/ACC.2009.5160732.

[94] Nursultan Imanberdiyev, Changhong Fu, Erdal Kayacan, and I-Ming Chen. Autonomous navigation of UAV by using real-time model-based reinforcement learning. In *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1–6. IEEE, nov 2016. ISBN 978-1-5090-3549-6. doi: 10.1109/ICARCV.2016.7838739.

[95] T. A. Johansen. Toward dependable embedded model predictive control. *IEEE Systems Journal*, 11:1208–1219, 2017.

[96] Shivaram Kalyanakrishnan, Siddharth Aravindan, Vishwajeet Bagdawat, Varun Bhatt, Harshith Goka, Archit Gupta, Kalpesh Krishna, and Vihari Piratla. An analysis of frame-skipping in reinforcement learning. *arXiv preprint arXiv:2102.03718*, 2021.

[97] Isaac Kaminer, Antonio Pascoal, Enric Xarga, Naira Hovakimyan, Chengyu Cao, and Vladimir Dobrokhodov. Path Following for Small Unmanned Aerial Vehicles Using L1 Adaptive Augmentation of Commercial Autopilots. *Journal of Guidance, Control, and Dynamics*, 33:550–564, 2010.

[98] Yoshifumi Kawakami and Kenji Uchiyama. Nonlinear controller design for transition flight of a fixed-wing UAV with input constraints. In *AIAA Guidance, Navigation, and Control Conference*. American Institute of Aeronautics and Astronautics, jan 2017. doi: 10.2514/6.2017-1041.

[99] Eleni Kelasidi, Kristin Ytterstad Pettersen, and Jan Tommy Gravdahl. A control-oriented model of underwater snake robots. In *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pages 753–760. IEEE, 2014.

[100] Hassan K. Khalil. *Nonlinear Systems (3rd Edition)*. Pearson, 2001. ISBN 9780130673893.

[101] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for UAV attitude control. *CoRR*, abs/1804.04154, 2018.

[102] William Koch, Renato Mancuso, and Azer Bestavros. Neuroflight: Next generation flight control firmware. *arXiv preprint arXiv:1901.06553*, 2019.

[103] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for uav attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2):1–21, 2019.

[104] Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. Learning-based model predictive control for safe exploration. In *2018 IEEE conference on decision and control (CDC)*, pages 6059–6066. IEEE, 2018.

[105] Arthur J Krener. Adaptive Horizon Model Predictive Control. *IFAC-PapersOnLine*, 51(13):31–36, January 2018. ISSN 2405-8963. doi: 10.1016/j.ifacol.2018.07.250.

[106] Sefer Kurnaz, Omer Cetin, and Okyay Kaynak. Fuzzy logic based approach to design of flight control and navigation tasks for autonomous unmanned aerial vehicles. *Journal of Intelligent and Robotic Systems*, 54(1-3):229–244, oct 2008. doi: 10.1007/s10846-008-9263-0.

[107] Nathan O Lambert, Daniel S Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer SJ Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.

[108] Mark Lau, S Yue, Keck Ling, and Jan Maciejowski. A comparison of interior point and active set methods for fpga implementation of model predictive control. *Proc. European Control Conference*, 03 2015.

[109] Alan Laub. A schur method for solving algebraic riccati equations. *IEEE Transactions on automatic control*, 24(6):913–921, 1979.

[110] Eugene Lavretsky and Kevin A. Wise. *Robust and Adaptive Control With Aerospace Applications*. Springer London, 2012.

[111] Taeyoung Lee and Youdan Kim. Nonlinear adaptive flight control using backstepping and neural networks controller. *Journal of Guidance, Control, and Dynamics*, 24(4):675–682, jul 2001. doi: 10.2514/2.4794.

[112] Joshua M. Levin, Aditya A. Paranjape, and Meyer Nahon. Agile maneuvering with a small fixed-wing unmanned aerial vehicle. *Robotics and Autonomous Systems*, 116:148–161, jun 2019. ISSN 0921-8890. doi: 10.1016/J.ROBOT.2019.03.004.

[113] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[114] Caihong Li, Jingyuan Zhang, and Yibin Li. Application of artificial neural network based on q-learning for mobile robot path planning. In *2006 IEEE International Conference on Information Acquisition*, pages 978–982. IEEE, 2006.

[115] Huiping Li and Yang Shi. Event-triggered robust model predictive control of continuous-time nonlinear systems. *Automatica*, 50(5):1507 – 1513, 2014. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2014.03.015.

[116] Shuo Li and Osbert Bastani. Robust model predictive shielding for safe reinforcement learning with stochastic dynamics. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7166–7172. IEEE, 2020.

[117] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016*, September 2015.

[118] Xiaobo Lin, Yao Yu, and Changyin Sun. Supplementary reinforcement learning controller designed for quadrotor uavs. *IEEE Access*, 7:26422–26431, 2019.

[119] Cunjia Liu and Wen-Hua Chen. Disturbance rejection flight control for small fixed-wing unmanned aerial vehicles. *J Guid Control Dyn*, 39(12): 2804–2813, 2016. doi: https://doi.org/10.2514/1.G001958.

[120] Lennart Ljung. Perspectives on system identification. *Annual Reviews in Control*, 34(1):1–12, 2010. ISSN 1367-5788. doi: https://doi.org/10.1016/j.arcontrol.2009.12.001.

[121] Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. *arXiv preprint arXiv:1811.01848*, 2018.

[122] Sergio Lucia, Alexandru Tătulea-Codrean, Christian Schoppmeyer, and Sebastian Engell. Rapid development of modular and sustainable nonlinear model predictive control solutions. *Control Engineering Practice*, 60:51–62, 2017.

[123] Nguyen Tan Luy. Reinforecement learning-based optimal tracking control for wheeled mobile robot. In *2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 371–376. IEEE, 2012.

[124] Michael Maiworm, Daniel Limon, Jose Maria Manzano, and Rolf Findeisen. Stability of gaussian process learning based output feedback model predictive control. *IFAC-PapersOnLine*, 51(20):455–461, 2018. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2018.11.047. 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018.

[125] Michael Maiworm, Daniel Limon, and Rolf Findeisen. Online learning-based model predictive control with gaussian process models and stability guarantees. *International Journal of Robust and Nonlinear Control*, 2021.

[126] Alonso Marco, Philipp Hennig, Jeannette Bohg, Stefan Schaal, and Sebastian Trimpe. Automatic lqr tuning based on gaussian process global optimization. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 270–277. IEEE, 2016.

[127] Andreas Bell Martinsen. End-to-end training for path following and control of marine vehicles. Master's thesis, NTNU, 2018.

[128] Siri Mathisen, Kristoffer Gryte, Sebastien Gros, and Tor Arne Johansen. Precision deep-stall landing of fixed-wing uavs using nonlinear model predictive control. *J Intell Robot Syst*, 101(24), 2021. doi: 10.1007/s10846-020-01264-3.

[129] Siri H. Mathisen, Kristoffer Gryte, Tor Johansen, and Thor I. Fossen. Nonlinear model predictive control for longitudinal and lateral guidance of a small fixed-wing UAV in precision deep stall landing. In *AIAA Infotech @ Aerospace*. American Institute of Aeronautics and Astronautics, jan 2016. doi: 10.2514/6.2016-0512.

[130] MathWorks. Dryden wind turbulence model (continuous). `https://se.mathworks.com/help/aeroblks/drydenwindturbulencemodelcontinuous.html`. Accessed: 2019-02-18.

[131] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems*, 2019. doi: 10.1109/TNNLS.2019.2934906.

[132] David Q Mayne, María M Seron, and SV Raković. Robust model predictive control of constrained linear systems with bounded disturbances. *Automatica*, 41(2):219–224, 2005.

[133] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000. ISSN 0005-1098. doi: https://doi.org/10.1016/S0005-1098(99)00214-9.

[134] 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.

[135] Michail G. Michailidis, Konstantinos Kanistras, Mohammed Agha, Matthew J. Rutherford, and Kimon P. Valavanis. Robust nonlinear control of the longitudinal flight dynamics of a circulation control fixed wing UAV. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, dec 2017. doi: 10.1109/cdc.2017.8264236.

[136] H. Michalska and D.Q. Mayne. Robust receding horizon control of constrained nonlinear systems. *IEEE Transactions on Automatic Control*, 38(11):1623–1633, November 1993. ISSN 1558-2523. doi: 10.1109/9.262032. Conference Name: IEEE Transactions on Automatic Control.

[137] 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):529–533, 2015.

[138] Signe Moe, Anne Marthine Rustad, and Kristian G Hanssen. Machine learning in control systems: An overview of the state of the art. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 250–265. Springer, 2018.

[139] Siddharth Mysore, Bassel Mabsout, Renato Mancuso, and Kate Saenko. Regularizing action policies for smooth control with reinforcement learning. In *IEEE International Conference on Robotics and Automation*, 2021.

[140] Siddharth Mysore, Bassel Mabsout, Kate Saenko, and Renato Mancuso. How to train your quadrotor: A framework for consistently smooth and responsive flight control via reinforcement learning. *ACM Transactions on Cyber-Physical Systems (TCPS)*, 5(4):1–24, 2021.

[141] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566, 2018. doi: 10.1109/ICRA.2018.8463189.

[142] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming Exploration in Reinforcement Learning with Demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, September 2017.

[143] Hossein Nejatbakhsh Esfahani, Arash Bahari Kordabad, and Sebastien Gros. Reinforcement learning based on mpc/mhe for unmodeled and partially observable dynamics. pages 2121–2126, 05 2021. doi: 10.23919/ACC50511.2021.9483399.

[144] Andrew Y. Ng and Stuart J. Russell. Algorithms for Inverse Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, June 2000. ISBN 978-1-55860-707-1.

[145] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.

[146] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto,

Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv:1912.06680*, 2019.

[147] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[148] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225. IEEE, 2006.

[149] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682 – 697, 2008. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2008.02.003. Robotics and Neuroscience.

[150] Dario Piga, Marco Forgione, Simone Formentin, and Alberto Bemporad. Performance-oriented model learning for data-driven mpc design. *IEEE Control Systems Letters*, 3(3):577–582, 2019. doi: 10.1109/LCSYS.2019.2913347.

[151] José Pinto, Paulo S. Dias, Ricardo Martins, João Fortuna, Eduardo Marques, and João Sousa. The lsts toolchain for networked vehicle systems. In *2013 MTS/IEEE OCEANS - Bergen*, pages 1–9, 2013. doi: 10.1109/OCEANS-Bergen.2013.6608148.

[152] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. *arXiv preprint arXiv:1802.09464*, February 2018.

[153] Riccardo Polvara, Massimiliano Patacchiola, Sanjay Sharma, Jian Wan, Andrew Manning, Robert Sutton, and Angelo Cangelosi. Toward End-to-End Control for UAV Autonomous Landing via Deep Reinforcement Learning. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 115–123. IEEE, jun 2018. ISBN 978-1-5386-1354-2. doi: 10.1109/ICUAS.2018.8453449.

[154] Christopher V Rao, Stephen J Wright, and James B Rawlings. Application of interior-point methods to model predictive control. *Journal of optimization theory and applications*, 99(3):723–757, 1998.

[155] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, WI, 2017.

[156] Wei Ren and Ella Atkins. Nonlinear trajectory tracking for fixed wing UAVs via backstepping and parameter adaptation. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*. American Institute of Aeronautics and Astronautics, aug 2005. doi: 10.2514/6.2005-6196.

[157] S. Richter, C. N. Jones, and M. Morari. Computational complexity certification for real-time mpc with input constraints based on the fast gradient method. *IEEE Transactions on Automatic Control*, 57(6):1391–1403, 2012. doi: 10.1109/TAC.2011.2176389.

[158] John W Roberts, Ian R Manchester, and Russ Tedrake. Feedback controller parameterizations for reinforcement learning. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 310–317. IEEE, 2011.

[159] Ugo Rosolia and Francesco Borrelli. Learning model predictive control for iterative tasks. a data-driven control framework. *IEEE Transactions on Automatic Control*, 63(7):1883–1896, 2017.

[160] Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS) 2011*, 2011.

[161] Damiano Rotondo, Andrea Cristofaro, Kristoffer Gryte, and Tor Arne Johansen. LPV model reference control for fixed-wing UAVs. *IFAC-PapersOnLine*, 50(1):11559–11564, jul 2017. doi: 10.1016/j.ifacol.2017.08.1640.

[162] Ihab Samy, Ian Postlethwaite, Da-Wei Gu, and John Green. Neural-network-based flush air data sensing system demonstrated on a mini air vehicle. *Journal of aircraft*, 47(1):18–31, 2010.

[163] Fendy Santoso, Matthew A. Garratt, and Sreenatha G. Anavatti. State-of-the-art intelligent flight control systems in unmanned aerial vehicles. *IEEE Transactions on Automation Science and Engineering*, 15(2):613–627, apr 2018. doi: 10.1109/tase.2017.2651109.

[164] Tom Schaul, Dan Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[165] Gerrit Schoettler, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5548–5555, 2020. doi: 10.1109/IROS45743.2020.9341714.

[166] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[167] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[168] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

[170] P. O. M. Scokaert and D. Q. Mayne. Min-max feedback model predictive control for constrained linear systems. *IEEE Transactions on Automatic Control*, 43(8):1136–1142, 1998. doi: 10.1109/9.704989.

[171] P. O. M. Scokaert, D. Q. Mayne, and J. B. Rawlings. Suboptimal model predictive control (feasibility implies stability). *IEEE Transactions on Automatic Control*, 44(3):648–654, 1999. doi: 10.1109/9.751369.

[172] Leila Sedghi, Zohaib Ijaz, Md. Noor-A-Rahim, Kritchai Witheephanich, and Dirk Pesch. Machine learning in event-triggered control: Recent advances and open issues, 2020.

[173] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine

Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687.

[174] B. L. Stevens, F. L. Lewis, and E. N. Johnson. *Aircraft Control and Simulation*. Wiley-Blackwell, Hoboken, NJ, 2016. ISBN 1118870980.

[175] Wen Sun, Arun Venkatraman, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. Deeply AggreVaTeD: Differentiable Imitation Learning for Sequential Prediction. In *Proceedings of the 34th International Conference on Machine Learning*, March 2017.

[176] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249. doi: 10.5555/3312046.

[177] Nassim Nicholas Taleb. *Antifragile: Things that gain from disorder*, volume 3. Random House Incorporated, 2012.

[178] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Oct 2012.

[179] Sebastian Trimpe, Alexander Millane, Simon Doessegger, and Raffaello D'Andrea. A self-tuning lqr approach demonstrated on an inverted pendulum. *IFAC Proceedings Volumes*, 47(3):11281–11287, 2014. ISSN 1474-6670. doi: https://doi.org/10.3182/20140824-6-ZA-1003.01455. 19th IFAC World Congress.

[180] U.S. Department of Defense. U.S. Military Specification MIL-F-8785C. *Washington, D.C.: U.S. Department of Defense*, 1980.

[181] Pepijn WJ Van De Ven, Tor A Johansen, Asgeir J Sørensen, Colin Flanagan, and Daniel Toal. Neural network augmented identification of underwater vehicle models. *Control Engineering Practice*, 15(6):715–725, 2007.

[182] Harm van Seijen. Effective multi-step temporal-difference learning for non-linear function approximation, 2016.

[183] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *arXiv:1707.08817 [cs]*, October 2018.

[184] Renato Vidoni, Giovanni Carabin, Alessandro Gasparetto, and Fabrizio Mazzetto. Stability analysis of an articulated agri-robot under different central joint conditions. In *Robot 2015: Second Iberian Robotics Conference*, pages 335–346. Springer, 2016.

[185] Kim P Wabersich and Melanie N Zeilinger. Linear model predictive safety certification for learning-based control. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 7130–7135. IEEE, 2018.

[186] Daichi Wada, Sergio A. Araujo-Estrada, and Shane Windsor. Unmanned aerial vehicle pitch control under delay using deep reinforcement learning with continuous action in wind tunnel test. *Aerospace*, 8(9), 2021. ISSN 2226-4310. doi: 10.3390/aerospace8090258.

[187] Weiren Wang and Felix Famoye. Modeling household fertility decisions with generalized poisson regression. *Journal of population economics*, 10 (3):273–283, 1997.

[188] Yuanda Wang, Jia Sun, Haibo He, and Changyin Sun. Deterministic policy gradient with integral compensator for robust quadrotor control. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(10):3713–3725, 2020. doi: 10.1109/TSMC.2018.2884725.

[189] Adrian Winter, Richard Hann, Andreas Wenz, Kristofer Gryte, and Tor Arne Johansen. Stability of a flying wing uav in icing conditions. In *8th European Conference for Aeronautics and Aerospace Sciences (EUCASS)*, 2019. doi: 10.13009/EUCASS2019-906.

[190] Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

[191] Yilei Wu, Qing Song, and Xulei Yang. Robust recurrent neural network control of biped robot. *Journal of Intelligent and Robotic Systems*, 49(2): 151–169, 2007.

[192] Andreas Wächter and Lorenz Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 03 2006. doi: 10.1007/s10107-004-0559-y.

[193] Linhai Xie, Sen Wang, Stefano Rosa, Andrew Markham, and Niki Trigoni. Learning with Training Wheels: Speeding up Training with a Simple Controller for Deep Reinforcement Learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6276–6283, May 2018.

[194] Jie Xu, Tao Du, Michael Foshey, Beichen Li, Bo Zhu, Adriana Schulz, and Wojciech Matusik. Learning to fly: computational controller design for hybrid uavs with reinforcement learning. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[195] Ke Xu, Fengge Wu, and Junsuo Zhao. Model-based deep reinforcement learning with heuristic search for satellite attitude control. *Industrial Robot: An International Journal*, pages IR–05–2018–0086, oct 2018. ISSN 0143-991X. doi: 10.1108/IR-05-2018-0086.

[196] Guo-Sheng Yang, Er-Kui Chen, and Cheng-Wan An. Mobile robot navigation using neural q-learning. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826)*, volume 1, pages 48–52. IEEE, 2004.

[197] X. Yang, H. He, and D. Liu. Event-triggered optimal neuro-controller design with reinforcement learning for unknown nonlinear systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(9):1866–1878, 2019. doi: 10.1109/TSMC.2017.2774602.

[198] J. Yoo and K. H. Johansson. Event-triggered model predictive control with a statistical learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–11, 2019. doi: 10.1109/TSMC.2019.2916626.

[199] Mario Zanon and Sébastien Gros. Safe reinforcement learning using robust mpc. *IEEE Transactions on Automatic Control*, 2020.

[200] Mario Zanon, Greg Horn, Sebastien Gros, and Moritz Diehl. Control of dual-airfoil airborne wind energy systems based on nonlinear mpc and mhe. In *2014 European Control Conference (ECC)*, pages 1801–1806. IEEE, 2014.

[201] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter I. Corke. Towards vision-based deep reinforcement learning for robotic motion control. *CoRR*, abs/1511.03791, 2015.

[202] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. *CoRR*, abs/1509.06791, 2015.

[203] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 528–535. IEEE, 2016.

[204] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744. IEEE, 2020.

[205] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov. Value function approximation and model predictive control. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 100–107, 2013. doi: 10.1109/ADPRL.2013.6614995.

[206] Artur Zolich, Tor Arne Johansen, Krzysztof Cisek, and Kristian Klausen. Unmanned aerial system architecture for maritime missions. design amp; hardware description. In *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 342–350, 2015. doi: 10.1109/RED-UAS.2015.7441026.

[207] K. J. Åström and B. M. Bernhardsson. Comparison of riemann and lebesgue sampling for first order stochastic systems. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, volume 2, pages 2011–2016 vol.2, 2002. doi: 10.1109/CDC.2002.1184824.