Optimal Control Theoretic Value Function Learning

Daniel Layeghi

THE UNIVERSITY
of EDINBURGH

Thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy
to the
University of Edinburgh — 2024

Declaration

I declare that this thesis has been composed solely by myself and that it has not been submitted, either in whole or in part, in any previous application for a degree. Except where otherwise acknowledged, the work presented is entirely my own.

Daniel Layeghi
May 2024

Abstract

Generating behaviours to complete complex tasks can be viewed under the paradigm of controlling dynamical systems. To solve such tasks, most approaches fall under two paradigms: Reinforcement Learning (RL) and Optimal Control (OC) theoretic approaches. OC theoretic solutions are mostly local and can only provide global controllers for special cases. As a result of this, locality solutions become trajectories. Synthesising these trajectories in the deterministic setting is formalised under the calculus of variations. This paradigm imposes strict constraints on the objective landscape: differentiability and continuity. In the stochastic setting, OC theoretic solutions have been proposed to remove the burden of these constraints and infer the optimal trajectory through sampling. RL has very similar theoretical groundings but diverges significantly in its approach. For example, RL parametersises value and/or policy functions instead of trajectories, allowing generalisation to new initial conditions. Additionally, in its model-free setting, which is our focus in this thesis, there is no need for constraints such as differentiability on the objective. RL is capable of estimating the gradients via sampling. However, these gradient estimates come at the high price of noisy solutions and slow convergence. To this end, defining methods that can leverage the best of both approaches is desirable. Our thesis aims to derive methods that greatly remove the burden of cost function design on the user while enabling generalisation by efficiently learning approximate global controllers.

As our initial attempt at this formalisation, we introduce a local method that combines the efficiency of derivative-based OC-theoretic approaches with the flexibility of local solutions based on sampling. To this end, we propose a hybrid approach that aims for consensus between derivative-based solution iterative Linear Quadratic Regulator (iLQR) and sampling-based method Path Integral (PI) path integral control. We define an objective that enables us to sample when derivatives vanish and follow optimised trajectories when derivatives arise. We use the Kullback Leibler (KL) control interpretation of PI control to formulate an inference problem that computes the optimal controls constrained by an adaptive distribution defined by the solution of iLQR. Our results show better convergence on manipulation and obstacle avoidance tasks than sampling strategy, path integral control and gradient-based strategy iLQR.

In the second segment of this thesis, we evaluate the widely used RL algorithms and its core gradient estimation machinery, policy gradients, without the typical

convergence strategies. Our results are obtained on simple nonlinear continuous control problems. We show that RL still requires extensive tuning, even on simple nonlinear problems and the flexibility gained by zeroth-order derivative estimation is paid for by hyperparameter tuning. In turn, we propose an OC-theoretic approach based on Bellman optimality that leverages differentiable dynamics and first-order gradients. Our approach can learn approximate time-varying value functions and robustly converge with minimal tuning. We further verify the ability of our method by relaxing the objective and obtaining first-order approximations of time-varying Lyapunov constraints. We further verify our approach by satisfying this first-order constraint over a compact set of initial conditions. When comparing our method to Soft Actor-Critic (SAC) and Proximal Policy Optimisation (PPO) we show faster convergence and outperform PPO and SAC in task cost by at least 2 and 4 orders of magnitude, respectively.

In the third part of the thesis, we combine our findings from the previous sections to create a method that can handle discontinuities using stochasticity, ensure convergence with differentiability, and generalise with function parameterisation. To achieve this, we approach the problem using stochastic optimal control and robustness. We use the stochastic Bellman equation, differentiable dynamics, and the natural smoothing induced by stochastic first-order gradients. Our results demonstrate that the policies based on learned value functions outperform SAC and PPO in task cost by factors of up to 1076.02 and 8, respectively. Moreover, we observe that adding noise to the dynamics smoothens the curvature of the value function. This effect is especially noticeable in our obstacle navigation task with discontinuous dynamics and costs, where the value functions learned under noisier dynamics follow wider paths around obstacles, making them more robust. Finally, we show that our learned value functions can also be integrated into local methods, reducing their effective search horizon by a factor of 15.

Acknowledgements

The manuscript that follows would not have been possible without the help of many. This passage is too short to serve justice to the people who have supported me along the way. There are a few who stand out, however. First, I would like to thank Prof. Michael Mistry, my supervisor, for providing me with the opportunity to pursue this PhD and trusting me with the intellectual freedom driven by my curiosity. I owe great gratitude to Dr Steve Tonneau, my second supervisor, who encouraged me to persevere and deeply engaged with my work at times when progress seemed to halt. This journey has not been free of despair, but with my dear friends Mohammadreza, Keyhan, Saeed and Houman, I found much joy and laughter to spur me on. I want to thank Aoife, who walked much of this path with me providing support while reminding me of much that is important in life. Finally, I consider myself very lucky to have parents who have shown me time and time again unconditional support with any path that I chose to take in life, and this PhD has been no exception.

برای پدرم کامران و مادرم ستاره

Notation

notation	meaning
$\mathbf{x}_t := \mathbf{x}(t) \in \mathbb{R}^n$	n-dimensional time-varying state vector
$\mathbf{u}_t := \mathbf{u}(t) \in \mathcal{U} \subseteq \mathbb{R}^m$	m-dimensional time-varying control vector
$\mathbf{x}_n := \mathbf{x}(t_n) \in \mathbb{R}^n$	discrete-time state vector at discrete time step n
$\mathbf{u}_n := \mathbf{u}(t_n) \in \mathcal{U} \subseteq \mathbb{R}^m$	discrete-time control vector at discrete time step n
$\dot{\mathbf{x}}_t = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$	dynamics drift term
$\pi(\mathbf{x}_t) := \mathbf{u}(\mathbf{x}_t) \in \mathcal{U}$	policy function
$\pi^*(\mathbf{x}_t) := \mathbf{u}^*(\mathbf{x}_t) \in \mathcal{U}$	optimal policy function
$\ell(\mathbf{x}_t, \mathbf{u}_t)$	cost function
$\ell_T(\mathbf{x}_T)$	terminal cost function
$J(\mathbf{x}_t)$	summation of running cost from time t to terminal cost
$v(\mathbf{x}_t)$	cost-to-go or value of starting at state $\mathbf{x}_t$
$v^*(\mathbf{x}_t)$	the optimal cost-to-go, i.e. under optimal policy
$\nabla_{\mathbf{x}_t}$	spatial first order derivative operator
$\nabla^2_{\mathbf{x}_t\mathbf{x}_t}$	spatial second order derivative operator

Contents

Declaration iii

Abstract v

Acknowledgements vii

Notation ix

1 Motivation and Introduction 1
- 1.1 Prologue: Decision making and behaviour . . . 1
- 1.2 Computational view . . . 1
- 1.3 Optimal control and reinforcement learning . . . 4
- 1.4 Thesis outline . . . 6

2 Background and relevant work 9
- 2.1 Discrete optimal decision making . . . 9
  - 2.1.1 Dynamic programming . . . 10
  - 2.1.2 Value iteration . . . 12
- 2.2 Continous state . . . 13
  - 2.2.1 Deterministic setting . . . 13
  - 2.2.2 Stochastic setting . . . 24
- 2.3 Conclusion . . . 41

3 Optimal Control via Combined Inference and Numerical Optimisation 43
- 3.1 Introduction . . . 43
- 3.2 Related work . . . 44
- 3.3 The optimal control setting . . . 45
  - 3.3.1 Numerical optimisation of the optimal control . . . 46
  - 3.3.2 Approximate inference of optimal control . . . 46
- 3.4 Combining sampling with second order method . . . 47
- 3.5 Experimental results and discussion . . . 50
  - 3.5.1 Implementation details . . . 51
  - 3.5.2 Cartpole task . . . 51
  - 3.5.3 Finger-Spinner task . . . 51

3.5.4 Push Object task . . . 52
3.5.5 Obstacle Avoidance task . . . 52
3.6 Discussion . . . 53
3.7 Conclusion and future work . . . 54
3.8 Appendix . . . 54
3.8.1 KL control derivation . . . 54

4 Neural Lyapunov and Optimal Control 57
4.1 Introduction . . . 57
4.2 Related work . . . 59
4.3 Preliminaries . . . 61
4.3.1 Optimal control . . . 61
4.3.2 Neural ODEs . . . 62
4.4 Learning Lyapunov and value functions . . . 63
4.4.1 Value functions . . . 63
4.4.2 Approximate Lyapunov constraint . . . 64
4.5 Empirical results . . . 67
4.5.1 Value function results . . . 68
4.5.2 Lyapunov function results . . . 71
4.6 Discussion and future work . . . 71
4.7 Conclusion . . . 73

5 Neural Stochastic Optimal Control 75
5.1 Introduction . . . 75
5.2 Background . . . 76
5.3 Method . . . 78
5.4 Results . . . 82
5.4.1 Comparison to model-free RL . . . 85
5.4.2 Effects of smoothing . . . 86
5.4.3 Generalisation . . . 87
5.5 Discussion . . . 90
5.5.1 Convergence . . . 91
5.5.2 Smoothing . . . 91
5.5.3 Generalisation . . . 92
5.6 Conclusion . . . 92

6 Conclusion and future work 95

References 99

List of Tables

3.1 Total trajectory cost and success rate for each method per environment. The costs are computed across the length of trajectories until the completion of tasks. Column Time represents the total computation time until task completion in seconds. For the number of samples and hyperparameters, refer to each task in section B. 48

4.1 Training statistics and performance comparison against SAC and PPO 70

5.1 Training statistics and performance comparison against SAC and PPO 84
5.2 Task parameters 85
5.3 Tracking MSE error 90

List of Figures

1.1 Innovations in Robotics and Artificial Intelligence 2

2.1 Enumeration strategy Liberzon (2011) 10
2.2 DP strategy Liberzon (2011) 11
2.3 Continous time principle of optimality 14
2.4 Interpretation of influence of $\mathbf{p}_t$ or $\nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t)$ on the dynamics as a description for the Hamiltonian 22
2.5 First vs zeroth order gradients Bach (2020) 40

3.1 Left to right: Cartpole scenario: The objective is to swing the pole to an upwards position. Finger-Spinner: The objective is for the fully actuated 2-link manipulator to rotate the object to the desired location marked as transparent. Push Object: The goal is for the 3-link manipulator to move the orange sphere to the desired location (green). Obstacle Avoidance: The goal is for the 3-link manipulator to move to the desired position (red) without contacting spherical obstacles. Results video at link 49

4.1 Compact stability region for a double integrator, computed by Neural Lyapunov Control. 58
4.2 Constraint satisfaction loss for value and Lyapunov function constraints. 69
4.3 Trajectory cost using our method. Bottom three rows: SAC and PPO trajectory cost. **Due to high values, SAC costs are scaled for visualisation**. 70
4.4 Cartpole balancing Lyapunov trajectories. 72

5.1 Top and middle rows: Constraint satisfaction loss and trajectory cost for value function constraints using our method. 83
5.2 Middle and bottom rows: Trajectory cost and SAC/PPO trajectory cost for value function constraints, scaled for visualisation. . 84
5.3 Impact of noise on value function curvature. 86
5.4 Navigation under different levels of noisy dynamics. From left to right, top to bottom: $\mathbf{\Sigma} = 0$, $\mathbf{\Sigma} = 2$, $\mathbf{\Sigma} = 4$, $\mathbf{\Sigma} = 8$. The large green area is the goal location, the green dots are the starting point of the trajectories, and the red dots are the final points in the trajectories. 88

5.5 Cost vs noisy policy. 89
5.6 Effect of value function on trajectory tracking. 90
5.7 Real hardware experiment setup and trajectory tracking. 90

Chapter 1

Motivation and Introduction

1.1 Prologue: Decision making and behaviour

Decision-making has been a central part of the existence of nearly all living organisms. Independent of their complexity, all living beings continuously decide on a series of actions that serve their existence. Bacteria navigate their surrounding environments based on nutrient information. Crows have developed adaptive behaviour to use moving cars as tools to crack nuts for sustenance. Gazelles use a zigzag pattern of movement to escape predators. Humans navigate traffic while driving by conceding or deciding to take charge in merging scenarios and avoid accidents. Decision-making can be observed in all species and at all layers of execution. For example, the strategic decision of a gazelle to run in a certain pattern is separate from the decisions which drive its gait. Similarly, the decisions which direct human negotiations in traffic are separate from processes controlling the actions performed by the limbs. The generation of these behaviours can be studied under many paradigms. A biological view may aim to describe the biomechanical machinery necessary to create a behaviour, or it may concern itself with the neural information circuits that regulate decision-making. An ethological perspective may describes how these decision-making behaviours have evolved to function within the context of an animal's natural environment and social structures. In this thesis, we focus on the computational view. We do not examine the biological or societal generators of behaviours, but we study the general behaviour generation under the paradigm of optimal decision-making. We aim to study the principles of synthetic optimal action or control generation under the Reinforcement Learning (RL) and Optimal Control (OC) theory framework.

1.2 Computational view

For centuries, humans have attempted to mimic the natural decision-making process at the biomechanical and cognitive levels. The advent of automata was one of the earliest signs of significant effort toward synthetic decision-making.

Al-Khwārizmī formalised the reasoning process by inventing the notion of an algorithm, and the Greeks enhanced this tradition with automated sculptures that mimic animal movements. The persistence of this interest in decision-making behaviour has brought us to today's achievements and the popular paradigms which underpin them. Optimal control-based solutions allow robots to perform impressive athletic feats, and deep reinforcement learning enables computers to beat experts at complex games. To provide intuition for the popular paradigms which underpin these achievements, we explore two specific examples: the Boston Dynamics robot performing impressive athletic behaviour and the incredible achievement of AlphaGo beating world champion Lee Sedol at the game of Go.

Of course, we have no concrete insight into the methods used by Boston Dynamics. Still, an educated guess based on academic presentations given by the engineers suggests a heavy use of OC. We can assume an optimisation-based approach that uses a model of the robot to predict control decisions based on hand-designed objectives that define the motion of behaviours, such as a backflip. Intuitively, a backflip is a rotation of the torso concluded by contact of the foot with the ground. Unfortunately, this intuitive description does not define the motion of a backflip. This discrepancy is generally reflected in the process of designing guiding functions for the generation of such motions. It is with trial and error and refinement of the cost functions that a satisfactory backflip is achieved.

(a) Robot Acrobatics Carter (2021)

(b) AlphaGo vs. Lee Sedol Borowiec (2016)

Figure 1.1: Innovations in Robotics and Artificial Intelligence

Could a similar approach be applied to our second example, the game of Go? Are guiding functions intuitive to define in this context? The game of Go is simple: create territories using a linked group of your stones until you capture the majority of the territory. However, as many Go players would reflect, the simplicity of the game does not lend itself to a simple strategy that would guide you to win. This notion leads to the impression that many expert Go players rely on intuition and instinct to perform well. Yet the AlphaGo program succeeded with only one objective: winning the game. So how does AlphaGo achieve this? Or maybe, more importantly, why can Go be solved to such a degree?

What AlphaGo does differs from traditional solvers typically used in games which involve exhaustive search approaches which would simply break down in a complex game like Go due to the curse of dimensionality. The search space of Go is extremely large, with approximately 250 legal moves and a game length of 150 iterations—famously, this gives a search space larger than the number of atoms in the universe: 250^{150}. At its heart, AlphaGo tackles both these dimensions using function approximations with neural nets. The depth of the game of Go, with 150 iterations, is approximated using a value function, an approximation of the goodness of the current board state. The choice of 250 actions is reduced by a policy approximation, a probability distribution that defines the likely action given the current board state. In AlphaGo, these approximations are leveraged inside Monte Carlo Tree Search to find moves with the best outcome Silver et al. (2016). This simple recipe of shrinking the depth and selection complexity of search with approximations is what achieves superhuman performance. But is this paradigm readily applicable to dynamical systems like our back-flipping robot? If yes, why are robots far from human dexterity, even in simulation? Why has beating Lee Sedol at the game of Go, a task for which we have very little strategic knowledge, been solved while performing dexterous manipulation and athletic feats intuitive to us all remains an open problem?

Unlike winning the game of Go, what defines complex biomechanical behaviour, such as a good backflip or dexterous manipulation of a cluttered environment, is difficult to formalise, but describing a winning performance at a game like Go is simple: whatever you do, you must win. The reward or cost function of the game of Go is concrete but sparse, and AlphaGo discovers what to do to win. On the other hand, athletic behaviour is difficult to describe concretely; this is partially a credit assignment problem.

Dynamical systems like robots also differ in their dimensionality from a game like Go. Robots have a continuous state and control space. This essentially makes their search space infinitely large in comparison. A discretised approach to searching this space is impossible, even for low-dimensional problems. All hope is not lost however, continuous dynamical systems are described by differential forms, which means their sensitivity to inputs is measurable on any metric up to machine precision. With dynamical systems, we are able to clearly answer how my choice of action locally changes my system under a specific metric. Calculus is what comes to the rescue of this large space, and the calculus of variations formalises the synthesis of optimal actions or controls. However, the cost of this formalisation is the cause of the trial and error and engineering effort required for cost function design. Optimal control imposes strict constraints such as differentiability and continuity regarding the objective. The engineer is then burdened with smoothing the objective landscape for gradient-based solvers. Here, by smoothing, we mean objectives that have dense derivative information. It is this process that makes or breaks the performance of a high-quality backflip. Essentially, the descriptive power of the cost functions guides

the solution. Although many tasks seem intuitive to us, our descriptions of their rewards or costs are often highly contrived.

This description starkly contrasts what was achieved with the game of Go. AlphaGo beats Lee Sedol with one objective: winning the game. This reward is sparse and certainly does not guide the process of gameplay step by step. AlphaGo is left to discover winning moves. This is the promise of deep reinforcement learning or evidence for the bitter lesson: the discovery of behaviours or strategies that are unknown to us. It is, therefore, tempting to extend this paradigm and its components to robotics. The notion has not been lost in the robotics scientific community, where many model-free reinforcement learning-based algorithms have been applied. Although there have been successful results, the same groundbreaking discoveries are yet to be seen. Optimal control theoretic approaches remain more capable of performing intricate and clean behaviours, while motions generated by RL-based approaches remain somewhat unrefined and extremely inefficient to learnSutton (2019). This phenomenon and its causal factors are at the core of this thesis.

1.3 Optimal control and reinforcement learning

Optimal control theoretic frameworks are capable of generating complex, refined motions. However, the key to this outcome is the operation of differentiation and the differentiability of the objective. Differentiation is a fundamental tool of calculus that describes the sensitivity of a function—in this case, our objective—to its arguments. The objective here comprises two components: the model of our environment or dynamics and the metric by which we measure the evolution of our environment. In optimal control, we typically use costs to define this metric. The dynamics function is our best approximation of the relationship between the inputs of our environment—state and action—and the cost function is our best approximation of what we believe to be of utility given our current state. For optimal control to work, both of these functions should be differentiable. Given that these constraints are satisfied and the initial condition of our environment is defined, OC-theoretic approaches provide us with very fast solutions and, in special cases, with the globally optimal sequence of controls.

Deep Reinforcement Learning, on the other hand, specifically its model-free variant, which is our focus, imposes much looser constraints on the problem definition. Given a model of our environment and a metric that defines the goodness of our states—which we typically call rewards in deep RL—deep RL is able to provide approximations of functions that output useful actions. At first glance, both approaches seem very similar, and this would not be an inaccurate assessment—many theories within OC provide grounding for RL. However, there are fundamental differences between OC-based approaches and model-free RL, differences that can illuminate the causal factors for the contrast between what has been achieved by either approach. Differentiation and differentiability are

key components of OC, but the same isn't true for model-free Reinforcement Learning. In fact, rewards are free to be defined in any form, and the environment must simply explain state transitions, whether these transitions are defined by differential form is not important. RL maximizes the reward without directly differentiating the objectives and instead by leveraging sampling of the objective. Additionally, the OC theoretic framework provides solutions unique to our environments' initial conditions or states. Conversely, RL approximates a policy or controller that is valid for a general set of initial conditions.

The framing and the methodology of RL we have outlined here naturally lend themselves to settings where derivatives and differentiable approximations are unavailable, for example, in the game of Go. Although navigating any objective landscape requires measurements of change, or in its most clear form, derivatives, RL navigates the objective landscape through sampling, which in the limit of very large samples, represents the gradient of a highly smoothed version of the original objective. However, historically, the primary reason for this choice has been the lack of access to models and, hence, derivatives, settings typically seen in games. Additionally, computational approaches to solving games require a deep search into the game horizon and approximation of the future goodness of states or values is another approach that deep RL methods leverage to shrink the search depth. As a result, the immediate application of RL to dynamical systems like robotics will have benefits and shortcomings. For example, in many cases, dynamical systems like robots have sufficiently good differentiable approximations; therefore, leveraging direct gradients can reduce the need for sample-based gradient estimates. However, not all dynamics components can have differentiable approximations, such as the description of objects touching or what is known as contact phenomena. The same can also be said about costs. As explained, we do not have well-defined differentiable and smooth costs for many complex behaviours, such as manipulation or locomotion. It is then desirable to be able to provide a method that combines the best of both RL and OC. For example, use sample-based derivative approximations for scenarios where derivatives are sparse and direct gradients where derivative information is available. Additionally, as we explained, RL has been able to provide a framing that can shrink the search depth required to minimise any objective. We face the same problem in OC-based solvers, where the horizon of our optimisation problem is crucial for estimating the future costs or goodness, or in more formal terms, the values of trajectories. In such cases, function approximations used in RL can inspire OC-theoretic methods that are more efficient in optimisation depth.

The effective combination of both approaches can enable us to efficiently solve tasks with various dynamics and more flexible definitions of utility. To achieve this, we must be able to define an effective combination based on concrete theoretical groundings. This is our goal in this thesis; we aim to improve our understanding of what constitutes an effective combination of the merits of both reinforcement learning and OC-theoretic approaches.

In section 1, we asked the question: Why is it that, despite all the embedded human intuition, biomechanical behaviours such as dexterous manipulation and other athletic feats remain an open problem? In this thesis, we do not aim to solve this grand challenge of generating behaviour. Instead, we concern ourselves with the question: Can the efficiency of OC-theoretic methods be combined with RL's flexibility and ability to discover and improve the synthesis of actions that lead to desirable behaviour?

1.4 Thesis outline

At its core, this thesis is concerned with effectively combining deep RL and OC components to advance the current state of the art. The following chapters dive into our attempt at achieving this goal.

- In Chapter 2, we define the necessary background and discuss recent developments underpinning our contributions. We begin with the optimal decision-making paradigm in a discrete setting, introducing the Bellman optimality principle and global approaches based on this optimality. Next, we explore the deterministic continuous representation of Bellman optimality and present local trajectory optimisation methods relevant to OC-theoretic approaches.

  We then discuss the optimality principle in the continuous stochastic setting, the foundation of model-free RL, and provide an understanding of the fundamental components of solving RL. We highlight the importance and implications of these components through a high-level derivation of recent model-free RL approaches. Finally, we extend our analysis to the OC-theoretic continuous stochastic setting, expanding on local methods that provide solutions.

  Throughout our discussion of the theoretical background, we emphasise recent advancements relevant to each component, aligning with our goal: effectively combining RL components with OC.

- In Chapter 3, we present our first contribution, which enhances local methods by combining the flexibility of sampling with the faster convergence of derivative-based optimisation. We utilise the Kullback-Leibler control framework and the second-order optimisation method, Differential Dynamic Programming, to infer optimal control trajectories. Our approach enables vanilla path integral solutions when there is no derivative information and guides samples towards the approximate distribution of our second-order method when derivatives are available. Our results demonstrate that guided sampling outperforms typical path integral solutions in manipulation and navigation tasks with discontinuous costs and dynamics.

Optimal Control via Combined Inference and Numerical Optimisation (ICRA 2022)
Daniel Layeghi, Steve Tonneau, Michael Mistry
Video: Contribution 1

- In Chapter 4, we present our second contribution, which combines the efficiency of OC-theoretic approaches with the flexibility of function approximation in RL. Our method uses the Hamilton-Jacobi-Bellman equation and first-order gradients to learn approximate value and Lyapunov functions. Our results demonstrate that this approach significantly outperforms model-free reinforcement learning in convergence while retaining the benefits of function approximation.

  Neural Lyapunov and Optimal Control (in submission)
  Daniel Layeghi, Steve Tonneau, Michael Mistry
  Video: Contribution 2

- In Chapter 5, we present our final contribution: a method that combines RL-like generalisation through function approximation with the efficiency of OC-theoretic solutions capable of handling objectives with discontinuous dynamics and costs. Using the stochastic Hamilton-Jacobi-Bellman equation and perturbed gradients, we learn approximate value functions. Our results demonstrate that this approach outperforms RL on continuous control tasks and effectively manages discontinuous dynamics and cost functions, showing good generalisation to hardware.

  Neural Stochastic Optimal Control (in submission)
  Daniel Layeghi, Mohammadreza Kasaei, Mohsen Khadem, Steve Tonneau, Michael Mistry
  Video: Contribution 3

- In Chapter 6, we conclude our work with remarks about extensions and future research directions.

Chapter 2

Background and relevant work

Our goal was to provide methods that enable efficient learning of generalisable policies while reducing the user's burden of objective design. Our contributions rely on combining ingredients of OC theory and model-free RL. This segment introduces a comprehensive review of the mathematical machinery that defines each approach. We ground this review in the theory of optimal decision-making. We then describe the global and local solutions to optimal decision-making in continuous space. Our evaluation of relevant work is detailed in each corresponding chapter of our contributions. However, throughout this segment, we also mention works that are either seminal or immediately relevant to our review.

2.1 Discrete optimal decision making

We start by considering an agent with state $\mathbf{x} \in X$ part of a finite set $\mathbf{X} = \{\mathbf{x}_0, ..., \mathbf{x}_N\}$ describing its place within the environment. We then define the set of decisions available to the agent as $\mathbf{u} \in \mathbf{U}$ part of a finite set $\mathbf{U} = \{\mathbf{u}_0, ..., \mathbf{u}_M\}$. We name these decision controls or actions. The objective of this agent is to choose the correct series of decisions that will allow it to complete a task. To formalise this objective, we require two additional components: a state transition function, which describes the next stage of the agent $\mathbf{x}' \in \mathbf{X}$ of the environment given a decision $\mathbf{u}$, $\mathbf{x}' = \text{next}(\mathbf{x}, \mathbf{u})$. To achieve the specified task, we also need to define a notion of utility: what is the badness or the goodness of our state and control? We define a notion of badness as cost $\ell(\mathbf{x}, \mathbf{u})$ and goodness as reward $r(\mathbf{x}, \mathbf{u})$. This distinction between the choice of terms is arbitrary; however, it defines the difference between optimal control theory and reinforcement learning terminology. Given these components, we can define and formalise the decision-making problem

$$J(\mathbf{x}_0) = \ell_T(\mathbf{x}_T) + \sum_{k=0}^{T-1} \ell(\mathbf{x}_k, \mathbf{u}_k) \tag{2.1}$$

Where given a sequence of controls $(\mathbf{u}_0, ..., \mathbf{u}_{T-1})$ over $T-1$ iterations, a sequence of initial and resulting states $(\mathbf{x}_0, ..., \mathbf{x}_T)$ over T iterations and utility function defining the cost of the last or terminal state $\ell(\mathbf{x}_N)$, we can compute the overall cost of our controls and resulting states. The goal of optimal decision-making is to choose this control sequence such that it incurs minimal cost or, depending on the terminology, maximises reward. We can also view this as a shortest path problem where each state is a node, and each action is an edge connecting nodes. if we assign costs to each node, our goal is to traverse this path using edges or, in our case, controls that take the path of states or nodes with the least cost.

2.1.1 Dynamic programming

Finding the sequence of controls that minimise the total cost at a given initial state $J(\mathbf{x}_0)$ is an optimisation problem that is optimally solvable using the Dynamic Programming (DP) paradigm. The simplest view to solving this graphical problem is to start from our initial state $\mathbf{x}_0$ and enumerate all possible paths up to T iterations, and then choose the path with the lowest $J(\mathbf{x}_0)$, a graphical representation of this is shown in figure 2.1. This approach is computationally

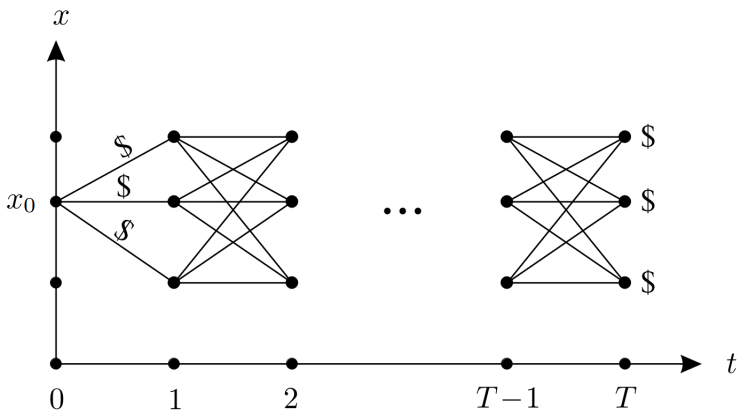


Figure 2.1: Enumeration strategy Liberzon (2011)

inefficient as we must evaluate the utility of all of our M actions for every single iteration k over our full horizon of the problem T. This leads to the complexity of $O(M^T T)$. this complexity grows linearly with the number of states. If we were to calculate the optimal path for another initial condition, we would have to recompute, leading to a complexity of $O(NM^T T)$.

The DP approach reverses this paradigm. Instead of enumerating all paths from the initial condition, it starts from the goal state and works backwards. A simple intuition for the difference between DP and the naive approach is that if we were to trust the forward path with the lowest immediate cost, although we may be incurring low costs in the short term, we may never reach our goal. DP alleviates this problem by starting from the goal. Therefore, given a cost for all

states, including the goal, we start from our $k = T - 1$ states and choose the forward path $k + 1$ with minimal cost. We then mark this path and move to $k = T - 2$. We repeat this process until we reach our initial states; our solution is to traverse the minimally marked cost paths, as shown in the figure 2.2. The

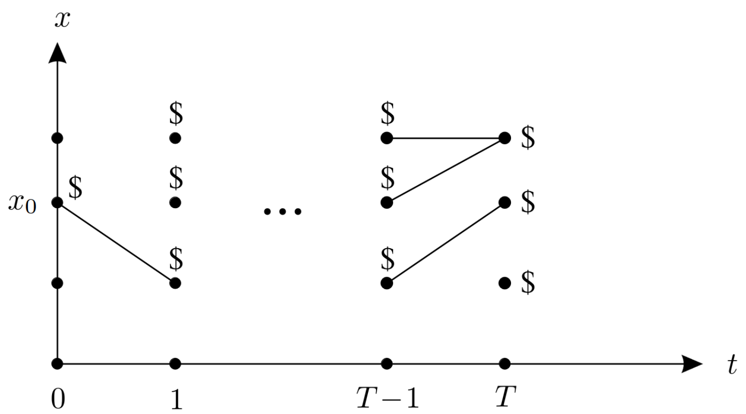


Figure 2.2: DP strategy Liberzon (2011)

complexity of DP is much lower than the naive approach. In this case, we do not need to compute the total cost at every given state, e.g. $J(\mathbf{x}_k)$. Due to the backward nature of the scheme at every iteration k we have already calculated the future cost $J(\mathbf{x}_{k+1})$. This results in a total complexity of $O(NMT)$. This reduction in complexity is due to Bellman's principle of optimality, which suggests that starting from the tail, one can recover the total optimal cost path by solving the smallest tail problem and then the next larger smallest tail problem until we cover all paths. The gain in efficiency is due to the implicit transfer of the optimal cost-to-go of the previous subproblem to the current. This optimal cost-to-go is the value function $v(\mathbf{x})$. which defines the minimal cost incurred by reaching a goal, starting from state $\mathbf{x}$. This notion is formally defined through the Bellman equation

$$v(\mathbf{x}) = \min_{\mathbf{u} \in \mathbf{U}} \left[\ell(\mathbf{x}, \mathbf{u}) + v\left(\text{next}(\mathbf{x}, \mathbf{u})\right) \right] \tag{2.2}$$

where the optimal control can be recovered from

$$\mathbf{u}^*(\mathbf{x}) = \arg\min_{\mathbf{u} \in \mathbf{U}} \left[\ell(\mathbf{x}, \mathbf{u}) + v\left(\text{next}(\mathbf{x}, \mathbf{u})\right) \right] \tag{2.3}$$

The two equations (2.2) and (2.3) formally define the principle of optimality and the recursive dependence of the value functions. We can observe that the optimal control $\mathbf{u}^*(\mathbf{x})$ is recovered by simply choosing the control with the minimum sum of the cost of the current state and control with the value of the resulting future state. This is a greedy selection criterion, yet it results in an optimal policy. The value function is crucial to enable greedy selection as it encodes all information about the optimality of the future states. It is essentially an oracle that evaluates

the optimal utility of our decision. This is essentially the core ingredient in our framework of the decision-making paradigm.

Our motivation was concerned with the generation of synthetic behaviour. Here, we see that under the abstraction of Bellman optimality, it is the value function that guides us, and without its existence, optimality does not exist. In the next segment of this section, we describe approaches that aim to solve for this optimal value function.

2.1.2 Value iteration

When we're concerned with systems with discrete dynamics, the two main approaches capable of solving for the optimal value functions are policy and value iteration. Here, we primarily focus on describing the value iteration algorithm 1.

Algorithm 1: Value Iteration

```
Initialise v^(0)(x) arbitrarily
repeat
    forall x ∈ X do
        v^(k+1)(x) = min_{u∈U} [ℓ(x, u) + v^(k)(next(x, u))]
    end
    k ← k + 1
until ‖v^(k+1) − v^(k)‖_∞ ≤ ϵ;
```

In short, value iteration estimates an initial value function and uses the discounted (λ) Bellman operation in equation (2.2) to update its estimate. This simple process is guaranteed to converge to the optimal value function. Here we simply provide a sketch of a proof based on an important property of the bellman operation (2.2), as contraction mappings with unique fixed points.

Convergence

We rely on contraction mapping and the fixed point theorem to prove the value iteration algorithm's convergence. We show that the Bellman operation is a contraction mapping, and as a result, convergence to a fixed point follows. We define the Bellman operator $\mathcal{T}$ as:

$$(\mathcal{T}v)(\mathbf{x}) = \min_{\mathbf{u}\in\mathbf{U}} [\ell(\mathbf{x}, \mathbf{u}) + v(\text{next}(\mathbf{x}, \mathbf{u}))] \tag{2.4}$$

Which means $v_{k+1} = \mathcal{T}v_k$. To show that operation $\mathcal{T}$ is a contraction mapping we need to show that $\|(\mathcal{T}v_k)(\mathbf{x}) - (\mathcal{T}v_{k+1})(\mathbf{x})\|_\infty \leq \|v_k(\mathbf{x}) - v_{k+1}(\mathbf{x})\|_\infty$. Using the definition of the Bellman operator and the difference between two value functions

v_k and v_{k+1}, we have:

$$\begin{aligned}
&|(\mathcal{T}v_k)(\mathbf{x}) - (\mathcal{T}v_{k+1})(\mathbf{x})| \\
&\leq \max_{\mathbf{x}\in\mathbf{X}} \left| \min_{\mathbf{u}\in\mathbf{U}} [\ell(\mathbf{x},\mathbf{u}) + \lambda v_{k+1}(\text{next}(\mathbf{x},\mathbf{u}))] - \min_{\mathbf{u}\in\mathbf{U}} [\ell(\mathbf{x},\mathbf{u}) + \lambda v_k(\text{next}(\mathbf{x},\mathbf{u}))] \right| \\
&\leq \lambda \max_{\mathbf{x}\in\mathbf{X}} \max_{\mathbf{u}\in\mathbf{U}} |v_{k+1}(\text{next}(\mathbf{x},\mathbf{u})) - v_k(\text{next}(\mathbf{x},\mathbf{u}))| \\
&= \lambda \|v_{k+1} - v_k\|_\infty
\end{aligned} \tag{2.5}$$

Thus $\|(\mathcal{T}v_k)(\mathbf{x}) - (\mathcal{T}v_{k+1})(\mathbf{x})\|_\infty \leq \|v_k(\mathbf{x}) - v_{k+1}(\mathbf{x})\|_\infty$ and $\mathcal{T}$ is a contraction mapping. Following this step based on the Banach fixed point theorem, the bellman operation will converge to its unique fixed point v^*, the optimal value function as $k \to \infty$.

Remarks

Although value iteration converges to the fixed point optimal value function, this paradigm has obvious problems. For one, the environments considered up to this point are all discrete in both state and action. Our motivation was based on dynamic behaviour. In those cases, the dynamics are continuous. Additionally, we computed the complexity of the DP paradigm to be $O(NMT)$ as a result, even if discretisation of the state space will still result in large values of N and M, which renders DP inapplicable. This phenomenon is what Bellman called the curse of dimensionality. In the next segments, we move to approximate paradigms tackling continuous and stochastic settings.

2.2 Continous state

The following sections of this chapter are concerned with the control of systems in a continuous state. We first introduce the deterministic setting and the methodologies for exact and approximate solutions. We then move to the stochastic setting and provide a similar background. Throughout each section, we discuss recent relevant progress.

2.2.1 Deterministic setting

In the continuous setting, we follow a dynamical system paradigm. Let our dynamics' state be a real n-dimensional continuous vector $\mathbf{x}_t \in \mathbb{R}^n$ and control with similar representation where $\mathbf{u} \in \mathbb{R}^m$. In the continuous setting, our state transition is defined by an Ordinary Differential Equation (ODE), where

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) \tag{2.6}$$

As a result, we compute the state evolution of our system by integrating the ODE (2.6)

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \int_t^{t+\Delta t} \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) dt \tag{2.7}$$

where $t \in (t_0, t_1]$ and $\Delta t \in (0, t_1 - t]$ under this assumption, we will now define the continuous analogue of the principle of optimality. In this case, the differential dynamics of environment (2.6) results in a value function that also varies with time. Therefore, we can write the continuous principle of optimality as:

$$v(\mathbf{x}_t, t) = \min_{\mathbf{u}_t} \left[\int_t^{t+\Delta t} \ell(\mathbf{x}_t, \mathbf{u}_t)\, dt + v(\mathbf{x}_{t+\Delta t}, t + \Delta t) \right] \tag{2.8}$$

The continuous principle of optimality is similar in intuition to one described in Section 2.1.1. The key difference, however, comes down to finding the optimal control that minimises the cost of a short time horizon (Δt) plus the optimal cost-to-go.

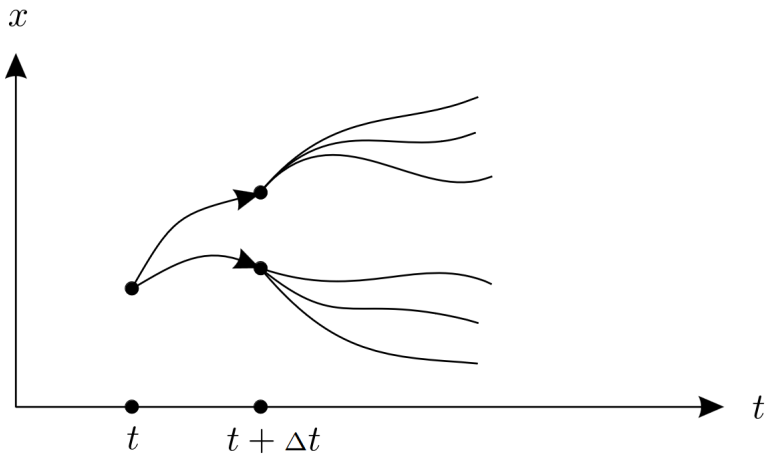


Figure 2.3: Continous time principle of optimality

Hamilton-Jacobi-Bellman

Equation (2.8) is an integral equation describing the evolution of the optimal value function v with respect to $\mathbf{x}_t$ and t. In this segment, we derive the compact representation of this equation in the form of a partial differential equation (PDE) known as the Hamilton-Jacobi-Bellman (HJB) equation. This compact form provides further insight into the continuous setting and, under special cases, provides analytical solutions to the optimal policy $\mathbf{u}_t^*(\mathbf{x}_t)$. First we approximate

the integral over $\ell(\mathbf{x}_t, \mathbf{u}_t)$ using left Riemann sum:

$$\int_{t}^{t+\Delta t} \ell(\mathbf{x}_t, \mathbf{u}_t)\, dt = \ell(\mathbf{x}_t, \mathbf{u}_t)\Delta t + O(\Delta t) \tag{2.9}$$

We then perform first-order Taylor expansion of $v(\mathbf{x}_{t+\Delta t}, t + \Delta t)$ over both of its arguments

$$v(\mathbf{x}_{t+\Delta t}, t+\Delta t) = v(\mathbf{x}_t, t) + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)\Delta t + \frac{\partial v(\mathbf{x}_t, t)}{\partial t}\Delta t + O(\Delta t) \tag{2.10}$$

we rewrite equation (2.8) using equations (2.9) and (2.10) and cancelling out common term $v(\mathbf{x}_{t+\Delta t}, t + \Delta t)$

$$0 = \min_{\mathbf{u}_t} \left[\Delta t \left(\ell(\mathbf{x}_t, \mathbf{u}_t) + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) + \frac{\partial v(\mathbf{x}_t, t)}{\partial t} \right) + O(\Delta t) \right] \tag{2.11}$$

dividing both sides by Δt and taking limit $t \to 0$ we see that the higher order terms in $O(\Delta t)/\Delta t$ go to zero; therefore, we arrive at the final HJB equation:

$$-\frac{\partial v(\mathbf{x}_t, t)}{\partial t} = \min_{\mathbf{u}_t} \left[\ell(\mathbf{x}_t, \mathbf{u}_t) + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) \right] \tag{2.12}$$

Equation (2.12) is the compact PDE description of the continuous principle of optimality (2.8). An upside of this representation is that it allows us to compute the optimal control law analytically. Taking the derivative with respect to $\mathbf{u}_t$ given a convex strictly positive cost in $\mathbf{u}_t$

$$\mathbf{u}^*(\mathbf{x}_t) = -(\nabla_{\mathbf{u}_t} \ell_{\text{ctrl}}(\mathbf{u}_t))^{-1} \left(\nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \nabla_{\mathbf{u}_t} \mathbf{f}(\mathbf{x}_t, u_t) \right) \tag{2.13}$$

Linear Quadratic Control

When deriving the compact form of the HJB (2.12) we mentioned that this representation provides further insight into the optimal controller or policy under certain settings. The Linear Quadratic (LQ) setting is an example. The LQ setting is described by a problem with linear dynamics or an environment with a task definition defined by quadratic costs. we can define this minimisation problem as:

$$J(\mathbf{x}_0) = \min_{\mathbf{u}_{[t,T]} \in \mathbf{U}} \left[\int_{t}^{T} \left(\mathbf{x}_t^\top \mathbf{Q} \mathbf{x}_t + \frac{1}{2} \mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t \right) dt + \mathbf{x}_T^\top \mathbf{Q}_T \mathbf{x}_T \right] \tag{2.14}$$

$$\text{subject to:} \quad \dot{\mathbf{x}} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t$$

Here, dynamics are linear in both state and control where $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{B} \in \mathbb{R}^{m \times m}$. Our state cost and control regularisation is also quadratic, with

running state cost gain $\mathbf{Q} \in \mathbb{R}^{n\times n}$ control regulariser $\mathbf{R} \in \mathbb{R}^{m\times m}$ and terminal cost term $\mathbf{Q}_T \in \mathbb{R}^{m\times m}$. However, we will present this problem under the one-step HJB equation view (2.12) under the LQ assumptions.

$$-\frac{\partial v(\mathbf{x}_t, t)}{\partial t} = \min_{\mathbf{u}_t} \left[\mathbf{x}_t^\top \mathbf{Q}\mathbf{x}_t + \frac{1}{2}\mathbf{u}_t^\top \mathbf{R}\mathbf{u}_t + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \cdot (\mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t) \right] \tag{2.15}$$

An important caveat of conversion of any optimal control problem such as (2.14) is that the HJB equations, just like any other PDE, require boundary conditions in order to be entirely defined. As a result, we will always need to define the terminal boundary condition on the value function, $v(\mathbf{x}_T, T)$. Under the LQ case, we define this as the terminal cost $v(\mathbf{x}_T, T) = \mathbf{x}_t^\top \mathbf{Q}_T \mathbf{x}_t$. As a result, assuming all cost terms $\mathbf{Q}, \mathbf{Q}_T$ are positive semi-definite (PSD), and $\mathbf{R}$ is PD we know that $v(\mathbf{x}_T, T)$ is also PSD. We guess $v(\mathbf{x}_t, t) = \mathbf{x}_t^\top \mathbf{S}_t \mathbf{x}_t$. We can rewrite equation (2.15) as:

$$-\frac{\partial v(\mathbf{x}_t, t)}{\partial t} = \min_{\mathbf{u}_t} \left[\mathbf{x}_t^\top \mathbf{Q}\mathbf{x}_t + \frac{1}{2}\mathbf{u}_t^\top \mathbf{R}\mathbf{u}_t + \mathbf{x}_t^\top \mathbf{S}_t \cdot (\mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t) \right] \tag{2.16}$$

minimising the LQR HJB (2.16) in controls $\mathbf{u}$ gives us an analytic description of the optimal control

$$\mathbf{u}_t^*(\mathbf{x}) = -\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{S}_t \mathbf{x}_t \tag{2.17}$$

Substituting the optimal policy (2.17) into equation (2.16), we arrive at

$$-\mathbf{x}_t^\top \dot{\mathbf{S}}_t \mathbf{x}_t = \mathbf{x}_t^\top \left(\mathbf{Q} + \mathbf{A}^\top \mathbf{S}_t + \mathbf{S}_t \mathbf{A} - \mathbf{S}_t \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{S}_t \right) \mathbf{x}_t \tag{2.18}$$

Leading us to a differential equation known as the Riccati equation that describes the evolution of the value function in time

$$\dot{\mathbf{S}}_t = -\mathbf{Q} - \mathbf{A}^\top \mathbf{S}_t - \mathbf{S}_t \mathbf{A} = \mathbf{S}_t \mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{S}_t \tag{2.19}$$

Since we already know the boundary condition for $v(\mathbf{x}_T, T)$ as defined by the terminal cost. We can numerically solve for $\mathbf{S}_t$. The only matter that remains is whether our value function is quadratic. Below, we show a simple sketch of proof based on the Bellman equation (2.2). Reversing the steps for deriving the HJB, we add $v(\mathbf{x}_t, t)$ from both sides and move $\frac{\partial v(\mathbf{x}_t, t)}{\partial t}$. The now present Taylor expansion of the right-hand side can be converted to the discrete value at the representation of the value at the next state $v(\mathbf{x}_{n+1})$:

$$\begin{aligned} v(\mathbf{x}_n, n) &= \min_{\mathbf{u}_n \in \mathbb{R}^m} \left[\mathbf{x}_n^\top \mathbf{Q}\mathbf{x}_n + \frac{1}{2}\mathbf{u}_n^\top \mathbf{R}\mathbf{u}_n + v\left(\mathbf{A}\mathbf{x}_n + \mathbf{B}\mathbf{u}_n, n+1\right) \right] \\ &= \min_{\mathbf{u}_n \in \mathbb{R}^m} \left[\mathbf{x}_n^\top \mathbf{Q}\mathbf{x}_n + \frac{1}{2}\mathbf{u}_n^\top \mathbf{R}\mathbf{u}_n + \left(\mathbf{A}\mathbf{x}_n + \mathbf{B}\mathbf{u}_n\right)^\top \mathbf{S}_{n+1} \left(\mathbf{A}\mathbf{x}_n + \mathbf{B}\mathbf{u}_n\right) \right] \end{aligned} \tag{2.20}$$

the optimal control in this setting becomes

$$\mathbf{u}_n = -\left(\mathbf{R} + \mathbf{B}^\top \mathbf{S}_{n+1}\mathbf{B}\right)^{-1} \mathbf{B}^\top \mathbf{S}_{n+1}\mathbf{A}\mathbf{x}_n \tag{2.21}$$

This differs from continuous optimal control (2.17). Substituting back into

$$\begin{aligned} v(\mathbf{x}_n, n) &= \mathbf{x}_n^\top \left(\mathbf{Q} + \mathbf{A}^\top \mathbf{S}_{n+1} \mathbf{A}\right) \mathbf{x}_n + \mathbf{u}_n^\top \left(\mathbf{R} + \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{B}\right) \mathbf{u}_n + 2\mathbf{u}_n^\top \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{A} \mathbf{x}_n \\ &= \mathbf{x}_n^\top \left(\mathbf{Q} + \mathbf{A}^\top \mathbf{S}_{n+1} \mathbf{A} - \mathbf{A}^\top \mathbf{S}_{n+1} \mathbf{B} \left(\mathbf{R} + \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{B}\right)^{-1} \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{A}\right) \mathbf{x}_n \end{aligned}$$

Let $\mathbf{S}_n = \mathbf{Q} + \mathbf{A}^\top \mathbf{S}_{n+1} \mathbf{A} - \mathbf{A}^\top \mathbf{S}_{n+1} \mathbf{B} \left(\mathbf{R} + \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{B}\right)^{-1} \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{A}$. To show $\mathbf{S}_n$ is PSD, we start by considering the matrix $\mathbf{S}_{n+1}$. Given the boundary condition that $\mathbf{S}_{n+1}$ is PSD. We can write $\mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{B} - \mathbf{B}^\top \mathbf{S}_{n+1} \left(\mathbf{S}_{n+1} + \epsilon \mathbf{I}\right)^{-1} \mathbf{S}_{n+1} \mathbf{B} \geq 0$ with its Schur complement as:

$$\begin{bmatrix} \mathbf{S}_{n+1} & \mathbf{S}_{n+1} \mathbf{B} \\ \mathbf{B}^\top \mathbf{S}_{n+1} & \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{B} \end{bmatrix} \geq 0 \tag{2.22}$$

Next factorise $\mathbf{A}$ in $\mathbf{S}_n$

$$\mathbf{S}_n = \mathbf{Q} + \mathbf{A}^\top \left(\mathbf{S}_{n+1} - \mathbf{S}_{n+1} \mathbf{B} \left(\mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{B}\right)^{-1} \mathbf{B}^\top \mathbf{S}_{n+1}\right) \mathbf{A} \tag{2.23}$$

If we wrote the Schur complement of (2.23) we would arrive at a matrix very similar to (2.22) with the additional matrix $\mathbf{R}$. But since $\mathbf{R}$ is by design PD and therefore above 0, we can write:

$$\begin{bmatrix} \mathbf{S}_{n+1} & \mathbf{S}_{n+1} \mathbf{B} \\ \mathbf{B}^\top \mathbf{S}_{n+1} & \mathbf{R} + \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{B} \end{bmatrix} \geq 0 \tag{2.24}$$

Since $\mathbf{Q}$ is chosen to be PSD, we can conclude that $\mathbf{S}_n$ is PSD, and by induction, $\mathbf{S}_n$ is PSD for all n. We can also "feel" an intuition for this through the Bellman operation as a contraction discussed in section 7, which means that the DP principle converges to a fixed point. Thus, there is one unique value function, and since our boundary condition is PSD and all components of (2.21) are quadratic, then the value function for all n must be PSD.

Algorithm 2: Finite Horizon LQR (Discrete)

Initialisation: Set $\mathbf{S}_T = \mathbf{Q}_T$, $\mathbf{R}$, $\mathbf{Q}$
forall $n \in \{N-1, N-2, \ldots, 0\}$ **do**
  $\mathbf{S}_n = \mathbf{Q} + \mathbf{A}^\top \mathbf{S}_{n+1} \mathbf{A} - \mathbf{A}^\top \mathbf{S}_{n+1} \mathbf{B} (\mathbf{R} + \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{B})^{-1} \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{A}$
end
forall $n \in \{0, \ldots, N-1\}$ **do**
  $\mathbf{u}_n = -(\mathbf{R} + \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{B})^{-1} \mathbf{B}^\top \mathbf{S}_{n+1} \mathbf{A} \mathbf{x}_n$
  $\mathbf{x}_{n+1} = \mathbf{A} \mathbf{x}_n + \mathbf{B} \mathbf{u}_n$
end

Remarks

The LQR algorithm efficiently computes a linear system's globally optimal feedback law. We can synthesise optimal behaviour if our system is linear. Unfortunately, linear state transitions cannot explain nearly any dynamic behaviour of interest. However, we take two lessons, one that under certain dynamics, namely linear or affine in controls, and given a convex regularisation term of the control or action, we can analytically derive the optimal policy based on the optimal value function. The less optimistic lesson, however, is that we only know this optimal value function for linear systems.

Maximum Principle

We have now discussed methods capable of solving for a globally optimal feedback policy with DP for the discrete setting and LQR for the linear quadratic setting. DP is burdened by the curse of dimensionality and inapplicable, and LQR is only valid for systems with linear dynamics and quadratic costs. In short, HJB-based approaches primarily rely on the existence of a value function. In many cases, we are unaware of this oracle, and we solve the problem for a single initial condition, or more formally, in the tangent space. Here, we start with the Pontryagin Maximum Principle (PMP). Let us define the Bellman equation again.

$$v(\mathbf{x}_t) = \min_{\mathbf{u}_t \in \mathbf{U}} \left[\underbrace{\ell(\mathbf{x}_t, \mathbf{u}_t) + v(\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t))}_{Q(\mathbf{x}_t, \mathbf{u}_t)} \right] \tag{2.25}$$

Our goal is to find the optimal control sequence that minimises the Bellman equation along a trajectory. We will now evaluate this objective along a single trajectory. We assume $\mathbf{x}_t$ and $\mathbf{u}_t$ as the nominal point in trajectory and $\delta\mathbf{x}_t$ and $\delta\mathbf{u}_t$ as perturbation around the normal trajectory. We then perform a first-order Taylor expansion of this point, resulting in

$$Q(\mathbf{x}_t + \delta\mathbf{x}_t, \mathbf{u}_t + \delta\mathbf{u}_t) \approx Q(\mathbf{x}_t, \mathbf{u}_t) + \begin{bmatrix} \nabla_{\mathbf{x}_t} Q & \nabla_{\mathbf{u}_t} Q \end{bmatrix} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} \tag{2.26}$$

where the expansion of each composing element is:

$$\begin{aligned} \ell(\mathbf{x}_t + \delta\mathbf{x}_t, \mathbf{u}_t + \delta\mathbf{u}_t) &\approx \ell(\mathbf{x}_t, \mathbf{u}_t) + \nabla_{\mathbf{x}_t}\ell \cdot \delta\mathbf{x}_t + \nabla_{\mathbf{u}_t}\ell \cdot \delta\mathbf{u}_t \\ v(\mathbf{f}(\mathbf{x}_t + \delta\mathbf{x}_t, \mathbf{u}_t + \delta\mathbf{u}_t)) &\approx v(\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)) + (\nabla_{\mathbf{x}_t} v \cdot \nabla_{\mathbf{x}_t}\mathbf{f}) \cdot \delta\mathbf{x}_t + (\nabla_{\mathbf{x}_t} v \cdot \nabla_{\mathbf{u}_t}\mathbf{f}) \cdot \delta\mathbf{u}_t \end{aligned} \tag{2.27}$$

We can then define each element of the expansion as:

$$\begin{aligned} \nabla_{\mathbf{x}_t} Q &= \nabla_{\mathbf{x}_t}\ell + (\nabla_{\mathbf{x}_t} v \cdot \nabla_{\mathbf{x}_t}\mathbf{f}) \\ \nabla_{\mathbf{u}_t} Q &= \nabla_{\mathbf{u}_t}\ell + (\nabla_{\mathbf{x}_t} v \cdot \nabla_{\mathbf{u}_t}\mathbf{f}) \end{aligned} \tag{2.28}$$

given the above definitions, we can then find the optimal minimiser of the Q function

$$\begin{aligned} \delta\mathbf{u}_t^* &= \arg\min_{\delta\mathbf{u}_t^*} \left[Q(\mathbf{x}_t + \delta\mathbf{x}_t, \mathbf{u}_t + \delta\mathbf{u}_t) \right] \\ \delta\mathbf{u}_t^* &= \nabla_{\mathbf{u}_t} Q = \nabla_{\mathbf{u}_t}\ell + \nabla_{\mathbf{x}_t} v \cdot \nabla_{\mathbf{u}_t}\mathbf{f} \end{aligned} \tag{2.29}$$

We can observe that the optimal policy $\delta \mathbf{u}_t^*$ is the gradient of the right-hand side of the HJB equation that we introduced before (2.12). The only matter that remains is the computation $\nabla_{\mathbf{x}_t} v(\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t))$. This is the gradient of the optimal value at the next state. So, the control at the current time is dependent on the value of the future state. We see a notion of dynamic programming here; however, the main difference is that this value only needs to exist over the next trajectory state. We now formalise the PMP. Defining the right hand side of the HJB (2.12) as H (Hamiltonian) and substituting $\mathbf{p}_t$ for the value of the $\nabla_{\mathbf{x}_t} v(\mathbf{x}_t)$ induced by a control trajectory $\delta \mathbf{u}_{t:T-1}$ we can rewrite the Hamiltonian as

$$H(\mathbf{x}_t, \mathbf{u}_t, \mathbf{p}_t) = \ell(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{f}^\top(\mathbf{x}_t, \mathbf{u}_t) \cdot \mathbf{p}_t \tag{2.30}$$

under this definition, we can define our state transition differential $\dot{\mathbf{x}}$ as

$$\dot{\mathbf{x}}_t = \nabla_{\mathbf{p}_t} H(\mathbf{x}_t, \mathbf{u}_t, \mathbf{p}_t) \tag{2.31}$$

We can also define our optimal control in terms of $\mathbf{p}_t$

$$\delta \mathbf{u}_t^* = \nabla_{\mathbf{u}_t} \ell(\mathbf{x}_t, \mathbf{u}_t) + (\nabla_{\mathbf{u}_t} \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t))^\top \cdot \mathbf{p}_t \tag{2.32}$$

Finally, because we need to know the value under the next state, we can define a differential equation based on $\mathbf{p}_t$. for brevity, here we drop the arguments $\mathbf{x}_t, \mathbf{u}_t$ and the time dependence subscript t

$$\begin{aligned}
-\dot{\mathbf{p}}^* &= \frac{d}{dt}\left(\frac{\partial v(\mathbf{x})}{\partial \mathbf{x}}\right) = \frac{\partial}{\partial t}\left(\frac{\partial v}{\partial \mathbf{x}}\right) + \frac{\partial}{\partial \mathbf{x}}\left(\frac{\partial v}{\partial \mathbf{x}}\right) \cdot \dot{\mathbf{x}}^* \\
&= \frac{\partial}{\partial \mathbf{x}}\left(\frac{\partial v}{\partial t}\right) + \frac{\partial}{\partial \mathbf{x}}\left(\frac{\partial v}{\partial \mathbf{x}}\right) \cdot \mathbf{f} \\
&= -\frac{\partial}{\partial \mathbf{x}}\left(\frac{\partial v}{\partial t} + \frac{\partial v}{\partial \mathbf{x}} \cdot \mathbf{f}\right) + \left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)^\top \frac{\partial v}{\partial \mathbf{x}} \\
&= \frac{\partial \ell}{\partial \mathbf{x}} + \left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)^\top \mathbf{p}^*
\end{aligned} \tag{2.33}$$

In final form, we get an ODE that describes the evolution of $\mathbf{p}_t^*$ given the corresponding boundary condition:

$$\begin{aligned}
\dot{\mathbf{p}}_t^* &= \nabla_{\mathbf{x}_t} \ell(\mathbf{x}_t, \mathbf{u}_t) + (\nabla_{\mathbf{x}_t} \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t))^\top \mathbf{p}_t^* \\
\dot{\mathbf{p}}_t^* &= \nabla_{\mathbf{x}_t} H(\mathbf{x}_t, \mathbf{u}_t, \mathbf{p}_t)
\end{aligned} \tag{2.34}$$

Finally, the maximum principle defines a three system of ODEs

$$\begin{aligned}
\dot{\mathbf{x}}_t &= \nabla_{\mathbf{p}_t} H(\mathbf{x}_t, \mathbf{u}_t, \mathbf{p}_t) \\
\dot{\mathbf{p}}_t^* &= \nabla_{\mathbf{x}_t} H(\mathbf{x}_t, \mathbf{u}_t, \mathbf{p}_t) \\
\mathbf{u}_t^* &= \arg\min_{\mathbf{u}_t^*} \left[H(\mathbf{x}_t, \mathbf{u}_t, \mathbf{p}_t)\right]
\end{aligned} \tag{2.35}$$

Algorithm 3: PMP (discrete)

Input: Initial state $\mathbf{x}_0$, time horizon T and its discretisation N and control input $\mathbf{u}_{[0,N-1]}$, and cost function $\ell(\mathbf{x}_t, \mathbf{u}_t)$

Output: Optimal control sequence $\mathbf{u}_{[n:N-1]}$, state trajectory $\mathbf{x}_{[n:N-1]}$

begin
  forall $n \in \{0, ..., N-1\}$ **do**
    $\mathbf{x}_{n+1} = \nabla_{\mathbf{p}_n} H(\mathbf{x}_n, \mathbf{u}_n, \mathbf{p}_n)$;
  end
  Initialise $\nabla_{\mathbf{x}_N} \mathbf{p}_N = \nabla_{\mathbf{x}_N} \ell(\mathbf{x}_N)$;
  forall $n \in \{N, ..., 0\}$ **do**
    $\mathbf{p}^*_{n-1} = \nabla_{\mathbf{x}_n} H(\mathbf{x}_n, \mathbf{u}_n, \mathbf{p}_n)$
  end
  forall $n \in \{0, ..., N-1\}$ **do**
    $\mathbf{u}^*_n = \arg\min_{\mathbf{u}^*_n} [H(\mathbf{x}_n, \mathbf{u}_n, \mathbf{p}_n)]$
  end
end

With the boundary condition $\mathbf{p}_T = \nabla_{\mathbf{x}_T} \ell_T(\mathbf{x}_T)$. Finally, for the sake of brevity, we do not derive the discrete version of this method but provide the analogous algorithm for it.

Remarks

We have now seen how we move away from the dependence of the known value function and solve Bellman optimality along a trajectory instead or, more formally, along the tangent space. This is an optimisation-based approach, and its primary shortcoming is the locality of solutions and the algorithm's convergence. The results from PMP are unique to a single initial condition $\mathbf{x}_0$, and the convergence from this point depends on the optimisation landscape defined by our cost function $\ell(\mathbf{x}_t, \mathbf{u}_t)$. We can now see what we referred to in our introduction 1. To do a backflip or dexterous manipulation under this paradigm, we must guide the optimiser by defining cost functions that are not only continuous and, in this case, first-order differentiable but also must encode enough information to define our goal. This is a difficult task.

Differential dynamic programming

Differential Dynamic Programming (DDP) is essentially the second-order approach to PMP. In PMP, we worked on the first-order approximation of the $Q(\mathbf{x}_t, \mathbf{u}_t)$ and minimised its change using gradient descent. DDP is concerned with the second-order approximation of the $\mathbf{Q}(\mathbf{x}_t, \mathbf{u}_t)$ and therefore uses second-order derivative information to minimise. Let us first define the second-order

approximation of the Q function.

$$Q(\mathbf{x}_t + \delta\mathbf{x}_t, \mathbf{u}_t + \delta\mathbf{u}_t) \approx Q(\mathbf{x}_t, \mathbf{u}_t) + \frac{1}{2}\begin{bmatrix}1\\ \delta\mathbf{x}_t\\ \delta\mathbf{u}_t\end{bmatrix}^\top \begin{bmatrix}0 & \nabla_{\mathbf{x}_t}Q^\top & \nabla_{\mathbf{u}_t}Q^\top\\ \nabla_{\mathbf{x}_t}Q & \nabla^2_{\mathbf{x}_t\mathbf{x}_t}Q & \nabla^2_{\mathbf{x}_t\mathbf{u}_t}Q\\ \nabla_{\mathbf{u}_t}Q & \nabla^2_{\mathbf{u}_t\mathbf{x}_t}Q & \nabla^2_{\mathbf{u}_t\mathbf{u}_t}Q\end{bmatrix}\begin{bmatrix}1\\ \delta\mathbf{x}_t\\ \delta\mathbf{u}_t\end{bmatrix} \tag{2.36}$$

Similar to (2.28), we now define each term of this approximation

$$\begin{aligned}
\nabla_{\mathbf{x}_t}Q &= \nabla_{\mathbf{x}_t}\ell + (\nabla_{\mathbf{x}_t}\mathbf{f})^\top \nabla_{\mathbf{x}_t}v\\
\nabla_{\mathbf{u}_t}Q &= \nabla_{\mathbf{u}_t}\ell + (\nabla_{\mathbf{u}_t}\mathbf{f})^\top \nabla_{\mathbf{x}_t}v\\
\nabla^2_{\mathbf{x}_t\mathbf{x}_t}Q &= \nabla^2_{\mathbf{x}_t\mathbf{x}_t}\ell + (\nabla_{\mathbf{x}_t}\mathbf{f})^\top \nabla^2_{\mathbf{x}_t\mathbf{x}_t}v\cdot\mathbf{f} + (\nabla_{\mathbf{x}_t}v)^\top \nabla^2_{\mathbf{x}_t\mathbf{x}_t}\mathbf{f}\\
\nabla^2_{\mathbf{u}_t\mathbf{u}_t}Q &= \nabla^2_{\mathbf{u}_t\mathbf{u}_t}\ell + (\nabla_{\mathbf{u}_t}\mathbf{f})^\top \nabla^2_{\mathbf{u}_t\mathbf{u}_t}v\cdot\mathbf{f} + (\nabla_{\mathbf{u}_t}v)^\top \nabla^2_{\mathbf{u}_t\mathbf{u}_t}\mathbf{f}\\
\nabla^2_{\mathbf{u}_t\mathbf{x}_t}Q &= \nabla^2_{\mathbf{u}_t\mathbf{x}_t}\ell + (\nabla_{\mathbf{u}_t}\mathbf{f})^\top \nabla^2_{\mathbf{x}_t\mathbf{x}_t}v\cdot\mathbf{f} + (\nabla_{\mathbf{u}_t}v)^\top \nabla^2_{\mathbf{u}_t\mathbf{x}_t}\mathbf{f}
\end{aligned} \tag{2.37}$$

We see that the first two terms are exactly the ones from PMP. Next, similar to the PMP case (2.29)

$$\begin{aligned}
\delta\mathbf{u}_t^* &= \arg\min_{\delta\mathbf{u}_t}\left[Q(\mathbf{x}_t + \delta\mathbf{x}_t, \mathbf{u}_t + \delta\mathbf{u}_t)\right]\\
\delta\mathbf{u}_t^* &= -\left(\nabla^2_{\mathbf{u}_t\mathbf{u}_t}Q\right)^{-1}\left(\nabla_{\mathbf{u}_t}Q + \nabla^2_{\mathbf{u}_t\mathbf{x}_t}Q\cdot\delta\mathbf{x}_t\right)
\end{aligned} \tag{2.38}$$

This update is analogous to the Newton's method. In short, PMP uses the gradient of the q function, but DDP uses the hessian information $\nabla^2_{\mathbf{u}_t\mathbf{u}_t}Q(\mathbf{x}_t, \mathbf{u}_t)$. Similarly, DDP optimal control depends on the value function's derivative information at the next state. As this method is local, we do not need the actual value function but the approximate value over the states induced by our control trajectory. So, similar to PMP, we need a series of different equations that compute this dependency based on the boundary condition $\ell_T(\mathbf{x}_T)$.

$$\begin{aligned}
\nabla_{\mathbf{x}_n}v^n &= \nabla_{\mathbf{x}_n}Q^{n+1} - \nabla_{\mathbf{u}_n}Q\left(\nabla^2_{\mathbf{u}_n\mathbf{u}_n}Q\right)^{-1}\nabla^2_{\mathbf{u}_n\mathbf{x}_n}Q\\
\nabla^2_{\mathbf{x}_n\mathbf{x}_n}v^n &= \nabla^2_{\mathbf{x}_n\mathbf{x}_n}Q^{n+1} - \nabla^2_{\mathbf{u}_n\mathbf{x}_n}Q\left(\nabla^2_{\mathbf{u}_n\mathbf{u}_n}Q\right)^{-1}\nabla^2_{\mathbf{u}_n\mathbf{x}_n}Q
\end{aligned} \tag{2.39}$$

These canonical equations of DDP (2.39) define the evolution of the first and second-order derivatives of the value function. These equations are the discrete second-order analogue of (2.34)

Remarks

We can see that the DDP approach has deep similarities to PMP. Because of this, it also has many of its shortcomings. For example, DDP is a local algorithm. As a result, optimisation per every initial condition is necessary. Returning to our original motivation of synthesising dynamic behaviour, we see that the user is again burdened with designing cost functions that are, in this case, second-order differentiable. This gives us faster convergence than PMP but also imposes much

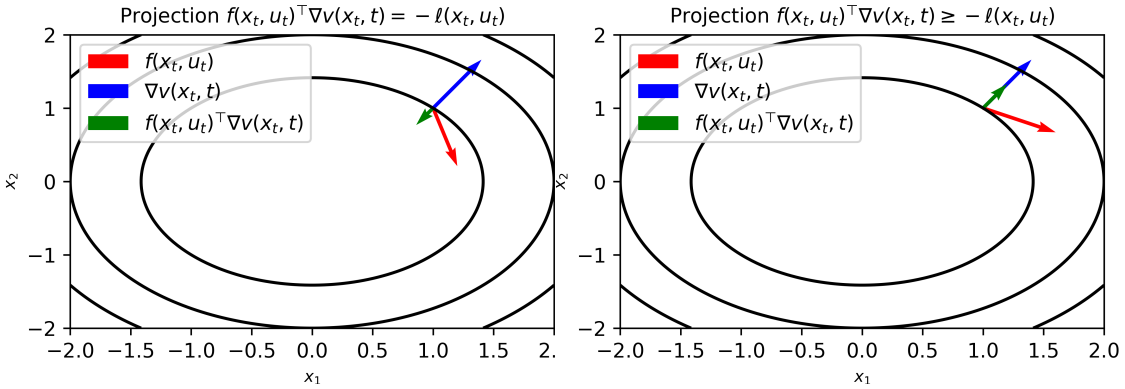
Algorithm 4: DDP (discrete)

Input: Initial state $\mathbf{x}_0$, time horizon N, initial control sequence $\mathbf{u}_{[n:N-1]}$
Output: Optimal control sequence $\mathbf{u}_{[n:N-1]}$, state trajectory $\mathbf{x}_{[n:N-1]}$

begin
  Backward Pass:
  Initialise $\nabla_{\mathbf{x}_N} v = \nabla_{\mathbf{x}_N} \ell_T(\mathbf{x}_N)$, $\nabla^2_{\mathbf{x}_N\mathbf{x}_N} v = \nabla^2_{\mathbf{x}_N\mathbf{x}_N} \ell_T(\mathbf{x}_N)$
  forall $n \in \{N-1, ...0\}$ **do**
    Compute $\nabla_{\mathbf{u}_n} Q$, $\nabla_{\mathbf{x}_n} Q$, $\nabla^2_{\mathbf{u}_n\mathbf{u}_n} Q$, $\nabla^2_{\mathbf{u}_n\mathbf{x}_n} Q$, $\nabla^2_{\mathbf{x}_n\mathbf{x}_n} Q$
    Update $\nabla_{\mathbf{x}_n} v = \nabla_{\mathbf{x}_n} Q - \nabla_{\mathbf{u}_n} Q \left(\nabla^2_{\mathbf{u}_n\mathbf{u}_n} Q\right)^{-1} \nabla^2_{\mathbf{u}_n\mathbf{x}_n} Q$
    Update $\nabla^2_{\mathbf{x}_n\mathbf{x}_n} v = \nabla^2_{\mathbf{x}_n\mathbf{x}_n} Q - \nabla^2_{\mathbf{u}_n\mathbf{x}_n} Q \left(\nabla^2_{\mathbf{u}_n\mathbf{u}_n} Q\right)^{-1} \nabla^2_{\mathbf{u}_n\mathbf{x}_n} Q$
  Control Update:
  forall $n \in \{0, ..., N-1\}$ **do**
    Update $\mathbf{u}^*_n = \mathbf{u}_n - \left(\nabla^2_{\mathbf{u}_n\mathbf{u}_n} Q\right)^{-1} \left(\nabla_{\mathbf{u}_n} Q + \nabla^2_{\mathbf{u}_n\mathbf{x}_n} Q \cdot \delta\mathbf{x}_n\right)$

stricter constraints. This is the curse of local methods; they help us escape the curse of dimensionality by only providing solutions along trajectories. Optimising along these trajectories is guided by our approximation of the value function through the cost function.

Our derivation of both PMP and DDP shows the presence of one term, in pmp the costate vector $\mathbf{p}_t = \nabla_{\mathbf{x}_t} v(x_t, t)$ or in DDP the additional second order derivative $\nabla^2_{\mathbf{x}_t\mathbf{x}_t} v(x_t, t)$. An additional observation tells us that these terms never appear independently and always as inner products with dynamics $\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$ or, in other words, the costate vector acts on velocity. Figure 2.4 below shows an intuitive representation of this idea. This interpretation tells us that



(a) Optimal flow on Hamiltonian **(b)** Non-optimal flow on Hamiltonian

Figure 2.4: Interpretation of influence of $\mathbf{p}_t$ or $\nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t)$ on the dynamics as a description for the Hamiltonian

the Hamiltonian essentially encodes the notion of optimality by "pushing" the

dynamics down the level-sets of the value function with distances equal to our cost function. Another interpretation of this is seen through the total derivative of the value function given by the HJB 2.12, where we can write

$$\begin{aligned} \frac{\partial v(\mathbf{x}_t, t)}{\partial t} + \frac{\partial v(\mathbf{x}_t, t)}{\partial \mathbf{x}_t}\frac{d\mathbf{x}_t}{dt} &= -\ell(\mathbf{x}_t, \mathbf{u}_t) \\ \frac{dv(\mathbf{x}_t, t)}{dt} &= -\ell(\mathbf{x}_t, \mathbf{u}_t) \end{aligned} \tag{2.40}$$

In other words, the evolution of the state along the value function should be negative (or converging) at a rate equal to our cost function. One can see that this essentially is a stricter form of the time-varying Lyapunov condition $\frac{dv(\mathbf{x}_t,t)}{dt} \leq \ell(\mathbf{x}_t)$. Therefore, we can view the value function as a lower bound on the optimal value function.

The observation above and the locality of methods like DDP and PMP have implications for our original objective: synthesising optimal behaviour or control sequences without encoding all the necessary information in the cost. Our derivation of the PMP and DDP showed us that our solutions are local and, in the PMP case, a linear approximation of our value function. Similarly, in the DDP case, we compute a solution along a quadratic approximation of our value function. It is, therefore, clear that the generation of behaviour would be very sensitive to such approximation error. As a consequence, our cost function must be differentiable along the path. A burden that is difficult to satisfy. Finally, local methods work on the approximation of the value function given the trajectory horizon, a notion based on the principle of optimality where the approximate value function $\tilde{v}(\mathbf{x}_t, t)$ is

$$\begin{aligned} \tilde{v}(\mathbf{x}_{t_0}, t_0) &= \ell_T(\mathbf{x}_T) + \int_{t_0}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)dt \\ &\geq v(\mathbf{x}_T) + \int_{t_0}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)dt \\ &\geq v(\mathbf{x}_{t_0}, t_0) \end{aligned} \tag{2.41}$$

Essentially, our approximation of this value function depends on the horizon T and our terminal $\ell_T(x_T)$. This is an error-prone formulation as we define all the intricacies of our value space by hand and approximate them further by first or second-order methods for tractability.

Relevant work

We refer the reader to Liberzon (2011); Bertsekas (2012) for a comprehensive overview of optimal control theory. The concept of differentiable dynamic programming (DDP) was initially introduced by Jacobson (1968). The method

has since been widely adapted, for example, iterative quadratic control (iLQR) Todorov and Li (2005), which simply relies on a first-order approximation of the dynamics. The iLQR-based solutions enable fast synthesis of behaviour, which have been applied in the context of simulationsTassa et al. (2012) and real hardwareKumar et al. (2016). Combining learning with OC theory has also been explored by authors in Jin et al. (2020), propose an end-to-end differentiable framework that differentiates through PMP to obtain analytical derivatives of trajectories with respect to tunable parameters, enabling end-to-end learning of dynamics, policies, and control objectives. hybrid approaches where DDP or ILQR solutions are used for data generation have also been explored. Authors in Viereck et al. (2022) improve the computational efficiency of iLQR gradients and hessian by defining a regression problem on the data collected from the iLQR trajectories. Another offline approach Mansard et al. (2018) proposed building a database based on trajectory optimisers to then use as a reference to train neural net policies. Optimisation-based controllers have also been embedded in differentiable computation graphs; authors Amos et al. (2017) embed quadratic program solvers with parameterised neural networks, enabling end-to-end optimisation.

2.2.2 Stochastic setting

In this segment, we define stochastic settings, the local solutions, and their inherent benefits in contrast to the shortcomings we have discussed in solutions for the deterministic setting. Finally, we give a general overview of deep model-free reinforcement learning and its specific advantages.

Control of Markov decision processes

Consider a given Markov process described by the transition probabilities $p(\mathbf{x}_{n+1}|\mathbf{x}_n)$ where $\mathbf{x}_n \in \mathbb{R}^n$. This describes the dynamics of our uncontrolled system. We also consider the controlled transition probabilities given by $q(\mathbf{x}_{n+1}|\mathbf{x}_n, \mathbf{u}_n)$ where $\mathbf{u}_n \in \mathbb{R}^m$ under this definition, we can define similar Bellman optimality condition similar to (2.2).

$$v(\mathbf{x}_n) = \min_{\mathbf{u}_n} \left[\ell(\mathbf{x}_n) + D(\mathbf{q}(\mathbf{x}_{n+1}|\mathbf{x}_n, \mathbf{u}_n) \parallel \mathbf{p}(\mathbf{x}_{n+1}|\mathbf{x}_n)) + \mathbb{E}_{\mathbf{x}_{n+1}\sim q(\cdot|\mathbf{x}_n,\mathbf{u}_n)}\left[v(\mathbf{x}_{n+1})\right]\right] \tag{2.42}$$

This essentially encodes the Bellman equation for the MDP setting. In this case, $D(\mathbf{q}(\mathbf{x}'|\mathbf{x}, \mathbf{u}) \parallel \mathbf{p}(\mathbf{x}'|\mathbf{x}))$ defines the KL divergence between the passive and the controlled distribution. Intuitively, we can view this as regularising the control distribution close to the passive one. Expanding and combining expectations, we can rewrite (2.42)

$$v(\mathbf{x}_n) = \min_{\mathbf{u}_n} \left[\ell(\mathbf{x}_n) + \mathbb{E}_{\mathbf{x}_{n+1}\sim q(\cdot|\mathbf{x}_n,\mathbf{u}_n)} \left[\log\left(\frac{q(\mathbf{x}_{n+1}|\mathbf{x}_n, \mathbf{u}_n)}{p(\mathbf{x}_{n+1}|\mathbf{x}_n)}\right) + v(\mathbf{x}_{n+1})\right]\right] \tag{2.43}$$

Finally, by combining logs, we can arrive at

$$v(\mathbf{x}_n) = \min_{\mathbf{u}_n} \left[\ell(\mathbf{x}_n) + \mathbb{E}_{\mathbf{x}_{n+1} \sim q(\cdot|\mathbf{x}_n, \mathbf{u}_n)} \left[\log \left(\frac{q(\mathbf{x}_{n+1}|\mathbf{x}_n, \mathbf{u}_n)}{p(\mathbf{x}_{n+1}|\mathbf{x}_n) \exp(-v(\mathbf{x}_{n+1}))} \right) \right] \right] \tag{2.44}$$

From the definition of p and q, we know they are probabilities summing to 1. However, $p(\mathbf{x}_{n+1}|\mathbf{x}_n) \exp(-v(\mathbf{x}_{n+1}))$ is not and for correctness requires to be re-normalised to sum to 1. Define the desirability function $\psi(\mathbf{x}_n)$ as:

$$\psi(\mathbf{x}_n) = \exp(-v(\mathbf{x}_n)) \tag{2.45}$$

Introduce the normalisation term $\eta[\psi](\mathbf{x}_n)$ as:

$$\eta[\psi](\mathbf{x}_n) = \sum_{\mathbf{x}_{n+1}} p(\mathbf{x}_{n+1}|\mathbf{x}_n)\psi(\mathbf{x}_{n+1}) = \mathbb{E}_{\mathbf{x}_{n+1} \sim p(\cdot|\mathbf{x}_n)}[\psi(\mathbf{x}_{n+1})] \tag{2.46}$$

Now, multiply and divide the denominator in the expectation by η Simplify the expression inside the logarithm:

$$\mathbb{E}_{\mathbf{x}_{n+1} \sim q(\cdot|\mathbf{x}_n, \mathbf{u}_n)} \left[\log \left(\frac{q(\mathbf{x}_{n+1}|\mathbf{x}_n, \mathbf{u}_n)}{p(\mathbf{x}_{n+1}|\mathbf{x}_n)\psi(\mathbf{x}_{n+1})\eta[\psi](\mathbf{x}_n)/\eta[\psi](\mathbf{x}_n)} \right) \right] \tag{2.47}$$

Separate the logarithms and simplify we express the resulting expectation in terms of KL divergence; we can rewrite the Bellman equation (2.44):

$$v(\mathbf{x}_n) = \min_{\mathbf{u}_n} \left[\ell(\mathbf{x}_n) - \log \eta[\psi](\mathbf{x}_n) + D(q(\cdot|\mathbf{x}_n, \mathbf{u}_n) \parallel \frac{p(\cdot|\mathbf{x}_n)\psi(\cdot)}{\eta[\psi](\mathbf{x}_n)}) \right] \tag{2.48}$$

Finally, we aim to find the optimal control $\mathbf{u}_n^*$. Observing (2.48), we can see that the only term dependent on control is the KL divergence. For the KL divergence to be minimised, $q(\cdot|\mathbf{x}_n, \mathbf{u}_n) = \frac{p(\cdot|\mathbf{x}_n)\psi(\cdot)}{\eta[\psi](\mathbf{x}_n)}$. Therefore, we can define the optimal policy probabilities by

$$q^*(\mathbf{x}_{n+1}|\mathbf{x}_n, \mathbf{u}_n) = \frac{p(\mathbf{x}_{n+1}|\mathbf{x}_n)\psi(\mathbf{x}_{n+1})}{\eta[\psi](\mathbf{x}_n)} \tag{2.49}$$

Remarks

Equation (2.49) has an intuitive interpretation: sample several next states under the uncontrolled probabilities and weight these samples by their corresponding values to find the optimal transition probabilities. However, the main drawback of this approach is its inefficiency. We rely on the uncontrolled transition to get us where we want. And also, once again, the value function is not known in many cases. In the next segment, we introduce local methods that provide solutions to the stochastic setting.

Control of Brownian Motions

Here, we use stochastic differential equations (SDEs) to define our dynamics.

$$d\mathbf{x}_t = \mathbf{f}\left(\mathbf{x}_t, \mathbf{u}_t\right) dt + \mathbf{B}\left(\mathbf{x}_t\right) d\mathbf{w}_t \tag{2.50}$$

Where state $\mathbf{x}_t \in \mathbb{R}^m$ and control $\mathbf{u}_t \in \mathbb{R}^n$. Here, $\mathrm{d}\mathbf{w}_t$ represents a Brownian motion with variance $\mathbf{\Sigma}\Delta t$. Under this SDE we can define transition probabilities sampled from a normal distribution $\mathbf{x}_{t+\Delta t} \sim \mathcal{N}\left(\mathbf{x}_t + \mathbf{f}\left(\mathbf{x}_t, \mathbf{u}_t\right)\Delta t, \mathbf{\Sigma}\right)$. Using Ito's lemma, expanding using a Taylor series:

$$\begin{aligned} v(\mathbf{x}_t + d\mathbf{x}_t, t + dt) = v(\mathbf{x}_t, t) + \frac{\partial v(\mathbf{x}_t, t)}{\partial t} dt + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) d\mathbf{x}_t \\ + \frac{1}{2} d\mathbf{x}_t{}^\top \nabla^2_{\mathbf{x}_t \mathbf{x}_t} v(\mathbf{x}_t, t) d\mathbf{x}_t + \ldots \end{aligned} \tag{2.51}$$

For the first-order in dt and second-order in $d\mathbf{x}$, we have:

$$\begin{aligned} dv = v(\mathbf{x}_{t+\Delta t}, t + \Delta t) - v(\mathbf{x}_t, t) \approx \frac{\partial v(\mathbf{x}_t, t)}{\partial t} dt + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \cdot d\mathbf{x}_t \\ + \frac{1}{2} d\mathbf{x}_t{}^\top \nabla^2_{\mathbf{x}_t \mathbf{x}_t} v(\mathbf{x}_t, t) d\mathbf{x}_t \end{aligned} \tag{2.52}$$

In the deterministic case, we only considered first-order terms because $d\mathbf{x}d\mathbf{x}^\top$ would include terms of order dt^2. This results in the HJB (2.12) without the limit

$$dv(\mathbf{x}_t, t) = \frac{\partial v(\mathbf{x}_t, t)}{\partial t} dt + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) d\mathbf{x}_t \tag{2.53}$$

For the stochastic setting, $d\mathbf{x}_t d\mathbf{x}_t{}^\top$ is potentially of order dt since $d\mathbf{w}_t d\mathbf{w}_t{}^\top$ is of the same order. Under this intuition, we retain the second-order terms:

$$dv(\mathbf{x}_t, t) = \frac{\partial v(\mathbf{x}_t, t)}{\partial t} dt + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) d\mathbf{x}_t + \frac{1}{2} d\mathbf{x}_t{}^\top \nabla^2_{\mathbf{x}_t \mathbf{x}_t} v(\mathbf{x}_t, t) d\mathbf{x}_t \tag{2.54}$$

As we were interested in the expected value over the next state under the process (2.50), we denoted any expected value of this process as $\mathbb{E}_{\mathbb{Q}}[\cdot]$. Therefore, taking expectations of $\mathbb{E}_{\mathbb{Q}}[dv(\mathbf{x}_t, t)]$ with the knowledge that $\mathbb{E}_{\mathbb{Q}}[\mathbf{B}(\mathbf{x}_t) d\mathbf{w}_t] = 0$ and $\mathbb{E}_{\mathbb{Q}}[d\mathbf{x}_t d\mathbf{x}_t{}^\top] = \mathbf{\Sigma}(\mathbf{x}_t) dt$ where $\mathbf{\Sigma}(\mathbf{x}_t) = \mathbf{B}(\mathbf{x}_t)\mathbf{B}(\mathbf{x}_t)^\top$ we can define the expected derivative as

$$\mathbb{E}_{\mathbb{Q}}[dv(\mathbf{x}_t, t)] = \frac{\partial v(\mathbf{x}_t, t)}{\partial t} dt + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) dt + \frac{1}{2} \mathrm{trace}\left(\nabla^2_{\mathbf{x}_t \mathbf{x}_t} v(\mathbf{x}_t, t) \mathbf{\Sigma}(\mathbf{x}_t)\right) dt \tag{2.55}$$

Equation (2.55) gives a formal definition of the expected derivative of the value function. We now have the opportunity to discretise and derive what is known as the general infinitesimal generator. Approximation the differentials with differences we get

$$\begin{aligned} \mathbb{E}_{\mathbb{Q}}[v(\mathbf{x}_{t+\Delta t}, t + \Delta t)] - v(\mathbf{x}_t, t)] = \frac{\partial v(\mathbf{x}_t, t)}{\partial t} \Delta t + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) \Delta t \\ + \frac{1}{2} \mathrm{trace}\left(\nabla^2_{\mathbf{x}_t \mathbf{x}_t} v(\mathbf{x}_t, t) \mathbf{\Sigma}(\mathbf{x}_t)\right) \Delta t \end{aligned} \tag{2.56}$$

Dividing both sides by Δt gives us the infinitesimal generator as $\Delta t \to 0$. However, deriving the stochastic Bellman equation is important to us. In a similar fashion to (2.8), we can write

$$\min_{\mathbf{u}_t} \left[\mathbb{E}_{\mathbb{Q}}[v(\mathbf{x}_{t+\Delta t}, t+\Delta t) - v(\mathbf{x}_t, t)] + \Delta t \ell(\mathbf{x}_t, \mathbf{u}_t)\right] = 0 \tag{2.57}$$

dividing both sides by Δt and substituting (2.56) for the resulting equation gives us

$$-\frac{\partial v(\mathbf{x}_t, t)}{\partial t} = \min_{\mathbf{u}_t} \left[\ell(\mathbf{x}_t, \mathbf{u}_t) + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t)\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) + \frac{1}{2}\text{trace}\left(\nabla^2_{\mathbf{x}_t\mathbf{x}_t} v(\mathbf{x}_t, t)\mathbf{\Sigma}(\mathbf{x}_t)\right)\right] \tag{2.58}$$

This is the original deterministic HJB (2.12), with an additional term incorporating the incremental variance of our process and the curvature of our value function. Like before, we can now compute the optimal control by minimising with respect to $\mathbf{u}_t$. Given a convex strictly positive cost in $\mathbf{u}_t$

$$\mathbf{u}^*(\mathbf{x}_t, t) = -(\nabla_{\mathbf{u}_t} \ell_{\text{ctrl}}(\mathbf{u}_t))^{-1} (\nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \nabla_{\mathbf{u}_t} \mathbf{f}(\mathbf{x}_t, u_t)) \tag{2.59}$$

which under control affine dynamics $(\mathbf{f}(\mathbf{x}_t) + \mathbf{g}(\mathbf{x}_t)\mathbf{u}_t)\, dt + \mathbf{B}(\mathbf{x}_t)\, d\mathbf{w}_t$ and quadratic control regularisation cost $\frac{1}{2}\mathbf{u}_t^\top \mathbf{R}\mathbf{u}_t$ becomes

$$\mathbf{u}^*(\mathbf{x}_t, t) = -\mathbf{R}^{-1} \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \mathbf{g}(\mathbf{x}_t) \tag{2.60}$$

Remarks

Interestingly, the optimal policy is the same as the deterministic case (2.13). However, intuitively, this seems incorrect; we certainly should not perform the same sequence of actions for a noise-free environment in a noisy environment. However, the key is to realise that the differential equations that define the value function in the stochastic case (2.58) are different to the deterministic one (2.12). Specifically, the stochastic HJB evolves with effects of noise $\mathbf{\Sigma}(\mathbf{x}_t)$ and curvature $\nabla^2_{\mathbf{x}_t\mathbf{x}_t} v(\mathbf{x}_t, t)$. So, once again, the value function encodes all necessary information. The stochastic setting has also been tackled with local approaches similar to DDP and PMP. In the next segment, we derive one core method based on the linearisation of the stochastic HJB (2.58) known as the Path Integral (PI) control.

Linear HJB

We consider the control affine case with quadratic control cost. Substituting the optimal policy (2.60) into (2.58) we arrive at:

$$\begin{aligned} -\frac{\partial v(\mathbf{x}_t, t)}{\partial t} = {} & \ell_{\text{state}}(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t)\mathbf{f}(\mathbf{x}_t) + \frac{1}{2}\text{trace}\left(\nabla^2_{\mathbf{x}_t\mathbf{x}_t} v(\mathbf{x}_t, t)\mathbf{\Sigma}(\mathbf{x}_t)\right) \\ & - \frac{1}{2}\left(\nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t)\right)^\top \mathbf{g}(\mathbf{x}_t)\mathbf{R}^{-1}\mathbf{g}(\mathbf{x}_t)^\top \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \end{aligned} \tag{2.61}$$

We then perform Fleming's logarithmic transformation of the value function . This is similar to what we showed in (2.45), and we will see how section 2.2.2 is the discrete form of PI control.

$$\psi(\mathbf{x}_t, t) = \exp(-\frac{1}{\lambda} v(\mathbf{x}_t, t)) \tag{2.62}$$

Under this transformation $v(\mathbf{x}, t) = -\lambda \log(\psi(\mathbf{x}_t, t))$, we can redefine the terms that define (2.61).

$$\begin{aligned} \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) &= -\lambda \frac{\nabla_{\mathbf{x}_t} \psi(\mathbf{x}_t, t)}{\psi(\mathbf{x}_t, t)}, \quad \frac{\partial v(\mathbf{x}_t, t)}{\partial t} = \frac{-\lambda \partial \psi(\mathbf{x}_t, t)}{\psi(\mathbf{x}_t, t) \partial t} \\ \nabla^2_{\mathbf{x}_t \mathbf{x}_t} v(\mathbf{x}_t, t) &= -\lambda \frac{\nabla^2_{\mathbf{x}_t \mathbf{x}_t} \psi(\mathbf{x}_t, t)}{\psi(\mathbf{x}_t, t)} + \lambda \frac{\left(\nabla_{\mathbf{x}_t} \psi(\mathbf{x}_t, t)\right)^\top \nabla_{\mathbf{x}_t} \psi(\mathbf{x}_t, t)}{\psi(\mathbf{x}_t, t)^2} \end{aligned} \tag{2.63}$$

We now plug equations (2.63) into the (2.61) with the critical assumption that $\mathbf{g}(\mathbf{x}_t)\mathbf{R}^{-1}\mathbf{g}(\mathbf{x}_t)^\top = \mathbf{B}(\mathbf{x}_t)\mathbf{B}(\mathbf{x}_t)^\top = \mathbf{\Sigma}(\mathbf{x}_t)$

$$\begin{aligned} \frac{\lambda \partial \psi(\mathbf{x}_t, t)}{\psi(\mathbf{x}_t, t) \partial t} = \ell_{\text{state}}(\mathbf{x}_t) - \frac{\lambda}{\psi(\mathbf{x}_t, t)} \Bigg(& \nabla_{\mathbf{x}_t} \psi(\mathbf{x}_t, t) \mathbf{f}(\mathbf{x}_t) + \frac{1}{2} \left(\nabla^2_{\mathbf{x}_t \mathbf{x}_t} \psi(\mathbf{x}_t, t) \mathbf{\Sigma}(\mathbf{x}_t) \right) \\ & - \frac{\lambda}{2\psi(\mathbf{x}_t, t)} \left(\nabla_{\mathbf{x}_t} \psi(\mathbf{x}_t, t) \right)^\top \mathbf{\Sigma}(\mathbf{x}_t) \nabla_{\mathbf{x}_t} \psi(\mathbf{x}_t, t) \\ & + \frac{1}{2\psi(\mathbf{x}_t, t)} \left(\nabla_{\mathbf{x}_t} \psi(\mathbf{x}_t, t) \right)^\top \mathbf{\Sigma}(\mathbf{x}_t) \nabla_{\mathbf{x}_t} \psi(\mathbf{x}_t, t \Bigg) \end{aligned} \tag{2.64}$$

Simplifying, we arrive at

$$-\frac{\partial \psi(\mathbf{x}_t, t)}{\partial t} = -\frac{\ell_{\text{state}}(\mathbf{x}_t)}{\lambda} \psi(\mathbf{x}_t, t) + \nabla_{\mathbf{x}_t} \psi(\mathbf{x}_t, t) \mathbf{f}(\mathbf{x}_t) + \frac{1}{2} \text{trace} \left(\nabla^2_{\mathbf{x}_t \mathbf{x}_t} \psi(\mathbf{x}_t, t) \mathbf{\Sigma}(\mathbf{x}_t) \right) \tag{2.65}$$

We made critical assumptions that allowed us to arrive at (2.65). Namely, we require control and noise to act in the same subspace and control cost to be inversely proportional to noise. Under this assumption, we get a linearisation of the original nonlinear HJB (2.61). The optimal policy of this linearised setting is of the form

$$\mathbf{u}^*(\mathbf{x}_t, t) = \lambda \mathbf{R}^{-1} \frac{\nabla_{\mathbf{x}_t} \psi(\mathbf{x}_t, t)}{\psi(\mathbf{x}_t, t)} \mathbf{g}(\mathbf{x}_t) \tag{2.66}$$

We see that as (2.66) is generally similar to (2.60); however, the major change is the difference in the sign of the optimal control. This is simply because the logarithmic transformation converts the cost-to-go to reward-to-gos essentially converting the problem to a maximisation setting.

We have been able to define a definition of optimal control (2.66) under the linearised stochastic setting. At the final steps of the Path Integral, we need to define the terms $\psi(\mathbf{x}_t, t)$ and $\nabla_{\mathbf{x}_t} \psi(\mathbf{x}_t, t)$ independently of the value function and its transformation. Here, we leverage the Feynman-Kac lemma to arrive at the

definitions. To prove this application's correctness, we start from the Feynman-Kac theorem and work backward to (2.65). The application of the Feynman-Kac lemma to our problem states that the linear PDE (2.64) can be converted to

$$\psi(\mathbf{x}_t, t) = \mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}\int_t^T \ell_{\text{state}}(\mathbf{x}_t, t)\right)\psi(\mathbf{x}_T, T)\right] \tag{2.67}$$

Essentially, this states that our current reward-to-go function $\psi(\mathbf{x}_t, t)$ can be computed by a path integral of all paths at the current state until the terminal time T under the passive dynamics described by the distribution $\mathbb{P}$. This choice of passive dynamics results from our linearisation and the critical assumption that essentially gives us a linear PDE (2.64) that is not driven by control. It is essential to note that this is now a forward process, and the approximation of the reward to go depends on our choice of $\psi(\mathbf{x}_T, T)$ Let us now discretise the (2.67) over Δt increment and rewrite the expectation

$$\psi(\mathbf{x}_t, t) = \exp\left(-\frac{1}{\lambda}\ell_{\text{state}}(\mathbf{x}_t, t)\Delta t\right)\mathbb{E}_{\mathbb{P}}\left[\psi(\mathbf{x}_{t+\Delta t}, t+\Delta t)\right] \tag{2.68}$$

dividing both sides by $\exp\left(-\frac{1}{\lambda}\ell_{\text{state}}(\mathbf{x}_t, t)\Delta t\right)$ then subtracting $\psi(\mathbf{x}_t, t)$ and dividing by Δt

$$\frac{\exp\left(\frac{1}{\lambda}\ell_{\text{state}}(\mathbf{x}_t, t)\Delta t\right) - 1}{\Delta t}\psi(\mathbf{x}_t, t) = \frac{\mathbb{E}_{\mathbb{P}}\left[\psi(\mathbf{x}_{t+\Delta t}, t+\Delta t) - \psi(\mathbf{x}_t, t)\right]}{\Delta t} \tag{2.69}$$

taking a limit of the left-hand side

$$\begin{aligned}
\frac{\exp\left(\frac{1}{\lambda}\ell_{\text{state}}(\mathbf{x}_t, t)\Delta t\right) - 1}{\Delta t} &= \frac{\left[1 + \frac{1}{\lambda}\ell_{\text{state}}(\mathbf{x}_t, t)\Delta t + \frac{(\frac{1}{\lambda}\ell_{\text{state}}(\mathbf{x}_t, t)\Delta t)^2}{2!} + \cdots\right] - 1}{\Delta t} \\
\lim_{\Delta t \to 0} \frac{\exp\left(\frac{1}{\lambda}\ell_{\text{state}}(\mathbf{x}_t, t)\Delta t\right) - 1}{\Delta t} &= \frac{1}{\lambda}\ell_{\text{state}}(\mathbf{x}_t, t)
\end{aligned} \tag{2.70}$$

by recognising that the right-hand side of the equation (2.69) is the generalised infinitesimal generator as shown before in (2.56), taking its limit, we can rewrite (2.69) as

$$\frac{\ell_{\text{state}}(\mathbf{x}_t)}{\lambda}\psi(\mathbf{x}_t, t) = \frac{\partial\psi(\mathbf{x}_t, t)}{\partial t} + \nabla_{\mathbf{x}_t}\psi(\mathbf{x}_t, t)\mathbf{f}(\mathbf{x}_t) + \frac{1}{2}\text{trace}\left(\nabla^2_{\mathbf{x}_t\mathbf{x}_t}\psi(\mathbf{x}_t, t)\boldsymbol{\Sigma}(\mathbf{x}_t)\right) \tag{2.71}$$

this is the same equation as (2.67), proving the validity of Feynman-Kac in this application. We will show the connection of the (2.67) to the stochastic MDP setting (2.42). Let us start by defining the distributions $\mathbb{Q}$ by its transition densities as

$$\begin{aligned}
\mathbf{x}_{t+\Delta t} &= \mathbf{x}_t + \left(\mathbf{f}(\mathbf{x}_t) + \mathbf{G}(\mathbf{x}_t, t)\mathbf{u}_t\right)\Delta t + \mathbf{B}(\mathbf{x}_t, t)\epsilon\sqrt{\Delta t} \\
q(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t) &\sim \mathcal{N}\left(\mathbf{x}_t + \left(\mathbf{f}(\mathbf{x}_t) + \mathbf{G}(\mathbf{x}_t)\mathbf{u}_t\right)\Delta t, \Sigma\right)
\end{aligned} \tag{2.72}$$

and the uncontrolled dynamics $\mathbb{P}$ by its density

$$\begin{aligned} \mathbf{x}_{t+\Delta t} &= \mathbf{x}_t + (\mathbf{f}(\mathbf{x}_t, t))\,\Delta t + \mathbf{B}(\mathbf{x}_t, t)\epsilon\sqrt{\Delta t} \\ p(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t) &\sim \mathcal{N}\left(\mathbf{x}_t + (\mathbf{f}(\mathbf{x}_t))\,\Delta t, \Sigma\right) \end{aligned} \tag{2.73}$$

We now discretise (2.67) by a Δt increment and use the likelihood trick to move from the expectation under the passive distribution to the control distribution.

$$\psi(\mathbf{x}_t, t) = \exp\left(-\frac{1}{\lambda}\ell_{\text{state}}(\mathbf{x}_t, t)\Delta t\right) \mathbb{E}_{\mathbf{x}_{t+\Delta t}\sim q(\cdot|\mathbf{x}_t,\mathbf{u}_t)}\left[\psi(\mathbf{x}_{t+\Delta t}, t+\Delta t)\frac{p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t)}{q(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t, \mathbf{u}_t)}\right] \tag{2.74}$$

Taking the log of both sides cancelling $-\frac{1}{\lambda}$ and applying Jensen's inequality and using the definition of KL divergence, we arrive at

$$\begin{aligned} v(\mathbf{x}_t, t) \leq\ & \ell_{\text{state}}(\mathbf{x}_t, t)\Delta t + D(\mathbf{q}(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t, \mathbf{u}_t) \parallel \mathbf{p}(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t)) \\ & + \mathbb{E}_{\mathbf{x}_{t+\Delta t}\sim q(\cdot|\mathbf{x}_t,\mathbf{u}_t)}\left[\phi(\mathbf{x}_{t+\Delta t}, t+\Delta t)\right] \end{aligned} \tag{2.75}$$

The equation (2.75) is essentially bounded below by the right-hand side of (2.42). This is because Feynman-Kac converts the HJB to a forward process in which if our choice of terminal cost $\phi(\mathbf{x}_{t+\Delta t}, t)$ is the same as $v(\mathbf{x}_{t+\Delta t}, t+\Delta t)$ the inequality becomes an equality. To derive the Path Integral controller, we use the MDP case.

Remarks

We make a small remark about (2.67); up to now, all our estimates of the value function in continuous time relied on a form of the HJB equation. the HJB equation in stochastic and deterministic cases relies on $\nabla_{\mathbf{x}}\mathbf{f}(\mathbf{x}, \mathbf{u})$. This essentially requires the differentiability of the dynamics. The Feynman-Kac theorem removes this requirement and uses path integrals to estimate the reward-to-go. In the next section, we see how control under this formulation removes the differentiability requirements of the calculus of variation that we have seen up to now in cases such as DDP and PMP. This has an implication for our original motivation: to move away from describing smooth cost function and transfer this burden to our algorithms. Path Integral control is an instance of this, albeit at the cost of many samples.

Path Integral control

The Path Integral (PI) control method solves the stochastic optimal control problem. As the discretised Feynman-Kac approach (2.75) results in the same objective as the (2.42), we can induce that the optimal policy remains (2.49) unchanged;

$$q^*(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t, \mathbf{u}_t) = \frac{p(\mathbf{x}_{t+1}|\mathbf{x}_t)\psi(\mathbf{x}_{t+\Delta t})}{\mathbb{E}_{\mathbf{x}_{t+\Delta t}\sim p(\cdot|\mathbf{x}_t)}[\psi(\mathbf{x}_{t+\Delta t})]} \tag{2.76}$$

Path Integral control aims to compute the optimal policy to sample optimal trajectories. In this sense, PI is a policy optimisation paradigm. Therefore in

order to compute the optimal policy $q^*(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t, \mathbf{u}_t)$ we introduce a candidate policy $q(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t, \mathbf{u}_t)$ and minimise the KL divergence between the two

$$\arg\min_{\mathbf{u}_t} \left[\mathbb{E}_{\mathbf{x}_{t+\Delta t} \sim q^*} \left[\log \left(\frac{q^*(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t)}{q(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t)} \right) \right] \right] \tag{2.77}$$

we multiply by $\frac{p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t)}{p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t)}$:

$$\arg\min_{\mathbf{u}_t} \left[\mathbb{E}_{\mathbf{x}_{t+\Delta t} \sim q^*} \left[\log \left(\frac{q^*(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t) p(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t) p(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t)} \right) \right] \right] \tag{2.78}$$

separating the terms, we get:

$$\arg\min_{\mathbf{u}_t} \left[\mathbb{E}_{\mathbf{x}_{t+\Delta t} \sim q^*} \left[\log \left(\frac{q^*(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t)}{p(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t)} \right) - \log \left(\frac{q(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t)}{p(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t)} \right) \right] \right] \tag{2.79}$$

since $p(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t)$ does not depend on $\mathbf{u}_t$, and based on the definition of the optimal policy (2.76) optimal control policy is also independent $\mathbf{u}_t$, the first term can be removed from the arg min:

$$\arg\min_{\mathbf{u}_t} \mathbb{E}_{\mathbf{x}_{t+\Delta t} \sim q^*} \left[\log \left(\frac{q(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t)}{p(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t)} \right) \right] \tag{2.80}$$

in order to minimise with respect to $\mathbf{u}_t$ we must compute the log term based on the definitions (2.73) and (2.72). We now compute the term inside the expectation.

$$\mathbf{u}_t^* = \mathbb{E}_{\mathbf{x}_{t+\Delta t} \sim q^*} \left[\left(\mathbf{g}^\top \boldsymbol{\Sigma}(\mathbf{x}_t)^{-1} \mathbf{g}(\mathbf{x}_t) \right)^{-1} \mathbf{g}^\top(\mathbf{x}_t) \boldsymbol{\Sigma}(\mathbf{x}_t)^{-1} \mathbf{B}(\mathbf{x}_t) \epsilon \sqrt{\Delta t} \right] \tag{2.81}$$

Removing parts independent of the expectation, we can write optimal control as

$$\mathbf{u}_t^* = \mathbf{H}(\mathbf{x}_t) \mathbb{E}_{\mathbf{x}_{t+\Delta t} \sim q^*} \left[\frac{\Delta \mathbf{x}_t}{\Delta t} - \mathbf{f}(\mathbf{x}_t) \right] \tag{2.82}$$

Where $\mathbf{H}(\mathbf{x}_t) = \left(\mathbf{g}^\top \boldsymbol{\Sigma}(\mathbf{x}_t)^{-1} \mathbf{g}(\mathbf{x}_t) \right)^{-1} \mathbf{g}^\top(\mathbf{x}_t) \boldsymbol{\Sigma}(\mathbf{x}_t)^{-1}$The equation above would be useful if we could sample from the optimal control transition density q^*. Since this is impossible, we have two choices: importance sample from q or p. This choice will also define our parameter $\mathbf{B}(\mathbf{x}_t)\epsilon\sqrt{\Delta t}$. Next, we show how we can importance sample from q

$$\mathbf{u}_t^* = \mathbf{H}(\mathbf{x}_t) \mathbb{E}_{\mathbf{x}_{t+\Delta t} \sim q} \left[\frac{q^*(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t)}{q(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t)} \left(\frac{\Delta \mathbf{x}_t}{\Delta t} - \mathbf{f}(\mathbf{x}_t) \right) \right] \tag{2.83}$$

substituting (2.76)

$$\mathbf{u}_t^* = \mathbf{H}(\mathbf{x}_t) \mathbb{E}_{\mathbf{x}_{t+\Delta t} \sim q} \left[\frac{p(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t) \psi(\mathbf{x}_{t+\Delta t})}{q(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t) \mathbb{E}_{\mathbf{x}_{t+\Delta t} \sim p}[\psi(\mathbf{x}_{t+\Delta t})]} \left(\frac{\Delta \mathbf{x}_t}{\Delta t} - \mathbf{f}(\mathbf{x}_t) \right) \right] \tag{2.84}$$

Since we have changed our measure of expected performance to be under the controlled transition, we must also change the measure of $\mathbb{E}_{\mathbf{x}_{t+\Delta t}\sim p}[\psi(\mathbf{x}_{t+\Delta t})]$. Therefore, we rewrite it as $\mathbb{E}_{\mathbf{x}_{t+\Delta t}\sim q}[\frac{p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t)}{q(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t,\mathbf{u}_t)}\psi(\mathbf{x}_{t+\Delta t})]$. In the final step of this derivation, we must also define the ratio between p and q. Using the Gaussian density functions:

$$p(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t) = \frac{1}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}(\mathbf{x}_t)|}} \exp\left(-\frac{1}{2}(\mathbf{x}_{t+\Delta t} - \mu_{p_t})^\top \boldsymbol{\Sigma}(\mathbf{x}_t)^{-1}(\mathbf{x}_{t+\Delta t} - \mu_{p_t})\right) \tag{2.85}$$

$$q(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t) = \frac{1}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}(\mathbf{x}_t)|}} \exp\left(-\frac{1}{2}(\mathbf{x}_{t+\Delta t} - \mu_{q_t})^\top \boldsymbol{\Sigma}(\mathbf{x}_t)^{-1}(\mathbf{x}_{t+\Delta t} - \mu_{q_t})\right) \tag{2.86}$$

Where: $\mu_{p_t} = \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t)\Delta t$ and $\mu_{q_t} = \mathbf{x}_t + (\mathbf{f}(\mathbf{x}_t) + \mathbf{g}(\mathbf{x}_t)\mathbf{u}_t)\,\Delta t$. If we define $\mathbf{v}_t = \mathbf{x}_{t+\Delta t} - \mathbf{x}_t - \mathbf{f}(\mathbf{x}_t)\Delta t$. Then:

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t)\Delta t + \mathbf{v}_t \tag{2.87}$$

Substituting $\mathbf{x}_{t+\Delta t}$ in the exponents:

$$\mathbf{x}_{t+\Delta t} - \mu_{p_t} = \mathbf{v}_t \tag{2.88}$$

$$\mathbf{x}_{t+\Delta t} - \mu_{q_t} = \mathbf{v}_t - \mathbf{g}(\mathbf{x}_t)\mathbf{u}_t\Delta t \tag{2.89}$$

the ratio becomes:

$$\begin{aligned}&\frac{p(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t)} \\ &= \exp\left(-\frac{1}{2}\mathbf{v}_t^\top \boldsymbol{\Sigma}(\mathbf{x}_t)^{-1}\mathbf{v}_t + \frac{1}{2}(\mathbf{v}_t - \mathbf{g}(\mathbf{x}_t)\mathbf{u}_t\Delta t)^\top \boldsymbol{\Sigma}(\mathbf{x}_t)^{-1}(\mathbf{v}_t - \mathbf{g}(\mathbf{x}_t)\mathbf{u}_t\Delta t)\right)\end{aligned} \tag{2.90}$$

Simplifying the terms inside the exponent, we arrive at

$$\begin{aligned}&\frac{p(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t+\Delta t} \mid \mathbf{x}_t, \mathbf{u}_t)} \\ &= \exp\left(-\mathbf{v}_t^\top \boldsymbol{\Sigma}(\mathbf{x}_t)^{-1}\mathbf{g}(\mathbf{x}_t)\mathbf{u}_t\Delta t + \frac{1}{2}(\mathbf{g}(\mathbf{x}_t)\mathbf{u}_t\Delta t)^\top \boldsymbol{\Sigma}(\mathbf{x}_t)^{-1}(\mathbf{g}(\mathbf{x}_t)\mathbf{u}_t\Delta t)\right)\end{aligned} \tag{2.91}$$

We can view the term inside the exponent as an important sampling cost C, plugging into (2.84) and keeping in mind that $\psi\mathbf{x}_{t+\Delta} = \exp(-\frac{1}{\lambda}\phi(\mathbf{x}_{t+\Delta t}))$ we can write the numerator as

$$\mathbf{u}_t^* = \mathbf{H}(\mathbf{x}_t)\mathbb{E}_{\mathbf{x}_{t+\Delta t}\sim q}\left[\frac{\exp\left(C_t - \frac{1}{\lambda}\phi(\mathbf{x}_{t+\Delta t})\right)}{\mathbb{E}_{\mathbf{x}_{t+\Delta t}\sim q}[\frac{p(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t)}{q(\mathbf{x}_{t+\Delta t}|\mathbf{x}_t,\mathbf{u}_t)}\psi(\mathbf{x}_{t+\Delta t})]}\left(\frac{\Delta\mathbf{x}_t}{\Delta t} - \mathbf{f}(\mathbf{x}_t)\right)\right] \tag{2.92}$$

rewriting the previous definition

$$\mathbf{u}_t^* = \mathbf{H}(\mathbf{x}_t)\mathbb{E}_{\mathbf{x}_{t+\Delta t}\sim q}\left[\frac{\exp\left(C_t - \frac{1}{\lambda}\phi(\mathbf{x}_{t+\Delta t})\right)}{\mathbb{E}_{\mathbf{x}_{t+\Delta t}\sim q}[\exp\left(C_T - \frac{1}{\lambda}\phi(\mathbf{x}_{t+\Delta t})\right)]}\left(\frac{\Delta \mathbf{x}_t}{\Delta t} - \mathbf{f}(\mathbf{x}_t)\right)\right] \tag{2.93}$$

we now make the observation that under the state transition q, $\frac{\Delta \mathbf{x}_t}{\Delta t} - \mathbf{f}(\mathbf{x}_t)$ based on SDE (2.72) becomes $\mathbf{g}(\mathbf{x}_t)\mathbf{u}_t + \mathbf{B}(\mathbf{x}_t)\frac{\epsilon}{\sqrt{\Delta t}}$ substituting this into (2.93)

$$\mathbf{u}_t^* = \mathbf{H}(\mathbf{x}_t)\left(\mathbf{g}(\mathbf{x}_t)\mathbf{u}_t + \mathbb{E}_{\mathbf{x}_{t+\Delta t}\sim q}\left[\frac{\exp\left(C_t - \frac{1}{\lambda}\phi(\mathbf{x}_{t+\Delta t})\right)}{\mathbb{E}_{\mathbf{x}_{t+\Delta t}\sim q}[\exp\left(C_t - \frac{1}{\lambda}\phi(\mathbf{x}_{t+\Delta t})\right)]}\left(\mathbf{B}(\mathbf{x}_t)\frac{\epsilon}{\sqrt{\Delta t}}\right)\right]\right) \tag{2.94}$$

This concludes our derivation of the optimal control under the Path Integral control paradigm. Here, our derivation was concerned with the one-step discretisation. The conversion to longer horizons is trivial; it simply involves performing a discrete integral over the terms inside the *exp* up to iteration N. Where our terminal cost is defined by $\psi(\mathbf{x}_N, N)$. Equation (2.94) has an intuitive interpretation. Given an initial state and control, draw several sample controls and simulate the SDE (2.72) to the desired horizon. Then proceed to compute the importance weights C state cose $\ell_{\textbf{state}}$ and the terminal cost $\phi(\mathbf{x}_N)$. Proceed to compute costs and compute the weighted average over the samples, which we call $S(\epsilon)$. Update the previous controls using this weighted average.

Algorithm 5: PI (discrete)

Input: Initial state $\mathbf{x}_0$, time horizon T and its discretisation N, number of samples K and control input $\mathbf{u}_{[0,N-1]}$, and cost function $\ell(\mathbf{x}_n)$ and $\phi(\mathbf{x}_N)$

Output: Optimal control sequence $\mathbf{u}_{[0,N-1]}$

while *not done* **do**
  Draw random samples $\epsilon^{(K)}_{[0,N-1]}$
  forall $n \in \{0, ..., N-1\}$ **do**
    $\mathbf{x}^{(K)}_{n+1} = \mathbf{x}^{(K)}_n + \left(\mathbf{f}(\mathbf{x}^{(K)}_n, n) + \mathbf{G}(\mathbf{x}^{(K)}_n)\mathbf{u}^{(K)}_n(\mathbf{x}^{(K)}_n)\right)\Delta t + \mathbf{B}(\mathbf{x}^{(K)}_n)\epsilon^{(K)}_n\sqrt{\Delta t}$
    $S_{\epsilon^{(K)}} += C^{(K)}_{n+1} + \ell(\mathbf{x}^{(K)}_{n+1})$
  end
  $S_{\epsilon^{(K)}} += \ell(\mathbf{x}^{(K)}_N)$
  forall $n \in \{0, ..., N-1\}$ **do**
    $\mathbf{u}_n +=$ equation (2.93) given $S_{\epsilon^{(K)}_n}$
  end
end

Remarks

We started the lengthy derivation of the PI method by considering the stochastic dynamics. We remarked that the Feynman-Kac theorem allows us to represent the stochastic Bellman equation through a forward process. This segment shows that the optimal trajectory computed by PI (2.94) differs from the optimal trajectories computed in the deterministic case by DDP or PMP. Local solutions in the deterministic case required first or second-order approximation of the HJB equation. The PI solution, on the other hand, requires access to a dynamics model and computes the optimal policy by weighted sampling of this dynamics model under predefined cost functions. This has important implications for our goal. We aimed to reduce the burden of cost function design; here, we see that local stochastic solutions allow the user to define non-differentiable cost functions. All our solutions up to this point have either been local or global under discrete or linear dynamics. It is unclear whether the exact global policies for nonlinear dynamical systems are solvable, but approximation strategies have been developed. In the next segment, deep model-free Reinforcement Learning (RL) will be described as a candidate for this strategy.

Relevant work

For a comprehensive overview of stochastic optimal control and stochastic differential equations, we refer the reader to Fleming and Soner (2006) and Särkkä and Solin (2019). The original work introducing the linear HJB method can be traced back to Fleming and Soner (2006) and Kappen (2005). Authors in Todorov (2007) extend the linear HJB to discrete settings. The general PI controller was introduced by Theodorou et al. (2010). The results from this work were then extended to enable PI to be used in Model Predictive Control (MPC) settings Williams et al. (2015), which were then applied to a number of real-world tasks Williams et al. (2016, 2017, 2018). Similar to the deterministic setting, approaches to leveraging function approximation within the PI framework have been explored. Authors Lowrey et al. (2018) introduce an offline approach that uses the trajectory information from a PI controller to learn a value function using the Bellman backup equation for regression. The authors show that this approach reduces the PI controller's search horizon. Authors Carius et al. (2022) introduce a similar approach but use control data to fit an optimal policy. The stochastic HJB equation and its corresponding Forward and Backward SDEs have also been embedded as neural architectures Exarchos and Theodorou (2018); Pereira et al. (2019). These methods are able to provide solutions to stochastic HJB again through end-to-end optimisation.

Deep model-free reinforcement learning

This section focuses on deep model-free Reinforcement Learning (D-RL). However, we do so from the perspective of our goal, or what enables RL to learn approximate global policies without the need to define differentiable cost functions.

To this end, we analyse two of the relevant DRL's core ingredients: function approximation and policy gradient theorem. We consider a similar MDP setting to 2.2.2 where the next state is defined by transition densities $p(\mathbf{x}_{n+1}|\mathbf{x}_n, \mathbf{u}_n)$. Deep RL aims to find an approximation of a policy $\pi(\mathbf{x}_n)$ that maximises the expected future reward of our trajectories $R(\tau)$. Let us first formalise this setting. We can compute the likelihood of our trajectory τ given our initial state $\mathbf{x}_0$ and a policy $\pi(\mathbf{x}_n)$. First, we define the total reward over a horizon N as $R(\tau)$ given state action trajectory under the policy $\tau = \{(\mathbf{x}_0, \mathbf{u}_0), ..., \mathbf{x}_N\}$:

$$R(\tau) = \sum_{n=0}^{N-1} r(\mathbf{x}_n, \mathbf{u}_n) + \phi(\mathbf{x}_N) \tag{2.95}$$

where $\phi(\mathbf{x}_N)$ represents our terminal reward. The likelihood of the trajectory $P(\tau|\theta)$ is given by:

$$p(\tau|\pi(\cdot|\mathbf{x}_0)) = \rho_0(\mathbf{x}_0) \prod_{n=0}^{N-1} p(\mathbf{x}_{n+1}|\mathbf{x}_n, \mathbf{u}_n)\pi(\mathbf{u}_n|\mathbf{x}_n) \tag{2.96}$$

Here, we aim to find a policy that maximises the reward under the likelihood of all possible trajectories. In other words, we want to maximise the expected total reward given our policy π. Therefore, the optimal policy maximises this notion.

$$\pi^* = \arg\max_{\pi} \left[\mathbb{E}_{\tau\sim P(\tau|\pi)} \left[R(\tau)\right]\right] \tag{2.97}$$

Searching over the space of all possible policies is an impossible task. Deep RL assumes an approximation of π using a parameterised kernel such as a neural net. This converts our problem to maximisation under the parameters θ

$$J^*(\theta) = \min_{\theta} \left[\mathbb{E}_{\tau\sim P(\tau|\pi_\theta)} \left[R(\tau)\right]\right] \tag{2.98}$$

to minimise this objective, we need to take the gradient with respect to the parameters of our policy approximation θ. First we express this expectation as an integral:

$$\nabla_\theta J(\theta) = \int R(\tau)\nabla_\theta P(\tau|\theta)d\tau \tag{2.99}$$

We then apply the same likelihood trick that we have seen before but this time without changing the underlying density

$$\nabla_\theta J(\theta) = \int R(\tau) \left(\frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta)}\right) P(\tau|\theta)d\tau \tag{2.100}$$

Using the definition of log derivative

$$\nabla_\theta J(\theta) = \int R(\tau)\nabla_\theta \log P(\tau|\theta)P(\tau|\theta)d\tau \tag{2.101}$$

converting back to the expectation form the gradient of the expected reward with respect to the policy parameters θ is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim P(\tau|\theta)}[R(\tau)\nabla_\theta \log P(\tau|\theta)] \tag{2.102}$$

Equation (2.102) is known as the REINFORCE trick or the policy gradient theorem.

Remarks

We observe that (2.102) computes the direction of the parameter update of our policy approximation π_θ without actually computing any gradient information with respect to our utility $R(\tau)$. This is relevant to our original goal: to remove the cost or reward function engineering burden. The policy gradient method essentially removes any constraint on the objective.

Vanilla policy gradient method

We now define the core method behind many variants of model-free RL: the vanilla policy gradient algorithm. To do so, we first need to define a number of other concepts, many of which should be familiar to us as they rely on the Bellman equation (2.2). Let us first define the familiar value function under an approximate policy parameterised by θ:

- The value Function, $v^{\pi_\theta}(\mathbf{x})$, according to policy π_θ:

$$v^{\pi_\theta}(\mathbf{x}) = \mathbb{E}_{\tau\sim P(\tau|\theta)}\left[R(\tau) \mid \mathbf{x}_0 = \mathbf{x}\right] \tag{2.103}$$

- The Q Function, $Q^{\pi_\theta}(\mathbf{x}, \mathbf{u})$. The expected reward given, starting from state $\mathbf{x}_0$ with the first action $\mathbf{u}_0$ chosen arbitrarily, while following on under the policy π_θ:

$$Q^{\pi_\theta}(\mathbf{x}, \mathbf{u}) = \mathbb{E}_{\tau\sim P(\tau|\theta)}\left[R(\tau) \mid \mathbf{x}_0 = \mathbf{x}, \mathbf{u}_0 = \mathbf{u}\right] \tag{2.104}$$

- The optimal value Function, $v^*(\mathbf{x})$. The expected reward starting from state $\mathbf{x}_0$ under the optimal policy π_{θ^*}:

$$v^*(\mathbf{x}) = \mathbb{E}_{\tau\sim P(\tau|\theta^*)}\left[R(\tau) \mid \mathbf{x}_0 = \mathbf{x}\right] \tag{2.105}$$

- The advantage function, $A^{\pi_\theta}(\mathbf{x}, \mathbf{u})$. The "benefit" of taking action $\mathbf{u}$ in state $\mathbf{x}$ over randomly selecting an action according to $\pi_\theta(\cdot|\mathbf{x})$, while following π_θ after:

$$A^{\pi_\theta}(\mathbf{x}, \mathbf{u}) = Q^{\pi_\theta}(\mathbf{x}, \mathbf{u}) - v^{\pi_\theta}(\mathbf{x}) \tag{2.106}$$

The policy gradient algorithm incorporates another element, which is value function approximation. To approximate any function, we need to find a penalty

to minimise. Equation (2.103) is the candidate function in this case. Here, we approximate the value function with v_ψ leading to a penalty

$$\min_{\psi} \mathbb{E}_{\tau\sim P(\tau|\theta)} \left[\left(R(\tau) - v_\psi(\mathbf{x}) \right)^2 \right] = 0 \tag{2.107}$$

To convert the ideas mentioned into an implementable algorithm, we must compute expected trajectories for (2.102) and (2.107). We estimate these expectations through sample means by collecting a set of trajectories $\mathcal{D} = \{\tau_0, ..., \tau_K\}$. We arrive at the following approximations

$$\begin{aligned} \mathbb{E}_{\tau\sim P(\tau|\theta)}[R(\tau)\nabla_\theta \log P(\tau|\theta)] &\approx \frac{1}{|\mathcal{D}|} \sum_{\tau\in\mathcal{D}} \sum_{n=0}^{N-1} R(\tau)\nabla_\theta \log \pi_\theta(\mathbf{u}_n|\mathbf{x}_n) \\ \mathbb{E}_{\tau\sim P(\tau|\theta)} \left[\left(\sum_{n=0}^{N} R(\tau) - v_\psi(\mathbf{x}) \right)^2 \right] &\approx \frac{1}{|\mathcal{D}|N} \sum_{\tau\in\mathcal{D}} \sum_{n=0}^{N} \left(v_\psi(\mathbf{x}_n) - \hat{R}_t \right)^2 \end{aligned} \tag{2.108}$$

We can now put together the vanilla policy gradient algorithm.

Algorithm 6: Vanilla Policy Gradient Algorithm

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
forall $k \in \{0, 1, 2, \ldots\}$ **do**
  Collect $\mathcal{D}_k = \{\tau_i\}$ under policy $\pi_k = \pi(\theta_k)$.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (Equation (2.106)).
  Estimate policy gradient as:
  $\hat{g}_{\theta_k} = \frac{1}{|\mathcal{D}_k|} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{N} \nabla_\theta \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\big|_{\theta_k} \hat{A}_t$.
  Update policy using gradient descent:
  $\theta_{k+1} = \theta_k + \alpha_k \hat{g}_{\theta_k}$
  Fit value function by regression on mean-squared error:
  $\hat{g}_\psi = \nabla_\phi \frac{1}{|\mathcal{D}_k|N} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{N} \left(v_\phi(\mathbf{x}_t) - \hat{R}_t \right)^2$
  Update value using gradient descent:
  $\psi_{k+1} = \psi_k + \alpha_k \hat{g}_{\psi_k}$

Remarks

We showed that model-free RL relies on the policy gradient method to estimate the gradient of the objective. We have explained throughout that any problem in which we aim to minimise an objective is an optimisation problem that requires some form of descent along the objective landscape. In local deterministic methods like DDP or PMP, we saw that this gradient was computed directly.

In the stochastic setting, we saw that the local gradient of the value function is approximated with path integrals. Policy gradients in model-free RL are no different and are essentially a zeroth order (or derivative-free) optimisation method. Derivative-free optimisers do not require the objective to conform to any constraints. At this point, we might assume that this completes our goal: to remove the burden of cost function design. Unfortunately, there is no free lunch; policy gradient style derivatives provide many upsides such as flexibility and smoothing; however, they are also incredibly high variance, and usable gradient estimates require many samples.

0^{th} order gradients and function approximation

In this segment, we make mathematical remarks about the impact of zeroth-order gradient estimates on the objective landscape. Given the expected total reward $J^*(\theta)$ as defined in equation (2.98), we aim to minimise the objective using the policy gradient method. The zeroth-order gradient estimate, also known as the REINFORCE trick, computes the gradient as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P(\tau|\theta)}[R(\tau) \nabla_\theta \log P(\tau|\theta)] \tag{2.109}$$

To analyse the variance of the zeroth-order gradient estimator, we express it as a sample mean over a set of trajectories $\mathcal{D}$:

$$\begin{aligned} \nabla_\theta J(\theta) &\approx \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} R(\tau) \nabla_\theta \log P(\tau|\theta) \\ &= \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} R(\tau) \sum_{n=0}^{N-1} \nabla_\theta \log \pi_\theta(\mathbf{u}_n|\mathbf{x}_n) \end{aligned} \tag{2.110}$$

For a more formal representation, we define the per-sample zeroth-order gradient estimate $\nabla^{[0]} \hat{J}_i(\theta)$ and the batched zeroth-order gradient $\nabla^{[0]} \bar{J}(\theta)$ as follows:

$$\nabla^{[0]} \hat{J}_i(\theta) := \frac{1}{\sigma^2} R(\tau_i) \left[\sum_{n=0}^{N-1} \nabla_\theta \log \pi(\mathbf{u}_n^i | \mathbf{x}_n^i, \theta) \right] \tag{2.111}$$

$$\nabla^{[0]} \bar{J}(\theta) := \frac{1}{K} \sum_{i=1}^{K} \nabla^{[0]} \hat{J}_i(\theta) \tag{2.112}$$

The empirical variance of the zeroth-order gradient estimator is given by:

$$\hat{\sigma}_0^2 = \frac{1}{K-1} \sum_{i=1}^{K} \left\| \nabla^{[0]} \hat{J}_i(\theta) - \nabla^{[0]} \bar{J}(\theta) \right\|^2 \tag{2.113}$$

To derive the variance bound for the zeroth-order gradient estimator, we slightly modify the steps followed in Suh et al. (2022): first, we express the variance of

$\nabla^{[0]}\hat{J}_i(\theta)$:

$$\text{Var}\left[\nabla^{[0]}\hat{J}_i(\theta)\right] = \text{Var}\left[\frac{1}{\sigma^2}R(\tau_i)\left[\sum_{n=0}^{N-1}\nabla_\theta \log \pi(\mathbf{u}_n^i|\mathbf{x}_n^i,\theta)\right]\right] \tag{2.114}$$

Using the inequality $\text{Var}[z] \leq \mathbb{E}[z^2]$ and assuming $R(\tau_i)$ and $\nabla_\theta \log \pi(\mathbf{u}_n^i|\mathbf{x}_n^i,\theta)$ are bounded by B_R and B_π respectively, we proceed as follows:

$$\begin{aligned}
\text{Var}\left[\frac{1}{\sigma^2}R(\tau_i)\left[\sum_{n=0}^{N-1}\nabla_\theta \log \pi(\mathbf{u}_n^i|\mathbf{x}_n^i,\theta)\right]\right] &\leq \frac{1}{\sigma^4}\mathbb{E}\left[\left(R(\tau_i)\sum_{n=0}^{N-1}\nabla_\theta \log \pi(\mathbf{u}_n^i|\mathbf{x}_n^i,\theta)\right)^2\right] \\
&\leq \frac{B_R^2}{\sigma^4}\mathbb{E}\left[\left(\sum_{n=0}^{N-1}\nabla_\theta \log \pi(\mathbf{u}_n^i|\mathbf{x}_n^i,\theta)\right)^2\right] \\
&\leq \frac{B_R^2}{\sigma^4}\sum_{n=0}^{N-1}\mathbb{E}\left[\left(\nabla_\theta \log \pi(\mathbf{u}_n^i|\mathbf{x}_n^i,\theta)\right)^2\right] \\
&\leq \frac{B_R^2}{\sigma^4}\sum_{n=0}^{N-1}B_\pi^2 \\
&= \frac{B_R^2 B_\pi^2 N}{\sigma^4}
\end{aligned} \tag{2.115}$$

Since this is for a single trajectory, for K trajectories, the variance of the batched zeroth-order gradient estimator $\nabla^{[0]}\bar{J}(\theta)$ is:

$$\text{Var}[\nabla^{[0]}\bar{J}(\theta)] = \frac{1}{K}\text{Var}[\nabla^{[0]}\hat{J}_i(\theta)] \leq \frac{B_R^2 B_\pi^2 N n}{\sigma^2 K} \tag{2.116}$$

where: - B_R is the bound on the reward $R(\tau_i)$. - B_π is the bound on the policy gradients $\nabla_\theta \log \pi_\theta(\mathbf{u}_n|\mathbf{x}_n)$. - N is the horizon length. - n is the dimensionality of the state space. - σ is the standard deviation of the noise. - K is the number of samples (trajectories) in $\mathcal{D}$.

From (2.116), we can see that the variance of REINFORCE is directly proportional to three dominant factors: The value of episode cost, which is directly related to our value function performance $v_(\psi)(\tau)$ and our current policy π. Our episode length also directly drives up the variance of our gradient estimate. Finally, the value of our control output based on our policy is directly proportional to the variance. This bound also tells us that the only way for us to reduce this variance is to increase our sample size, K, or tune our policy variance σ^2.

Although high in variance, zeroth-order methods can also inherently smooth the objective landscape. In the case of policy optimisation, REINFORCE allows for learning policies that correspond to the highly smoothed version of the original landscape. This, however, is not entirely unique to complete 0th-order gradient estimates. Averaged perturbed gradients also provide similar smoothing benefits.

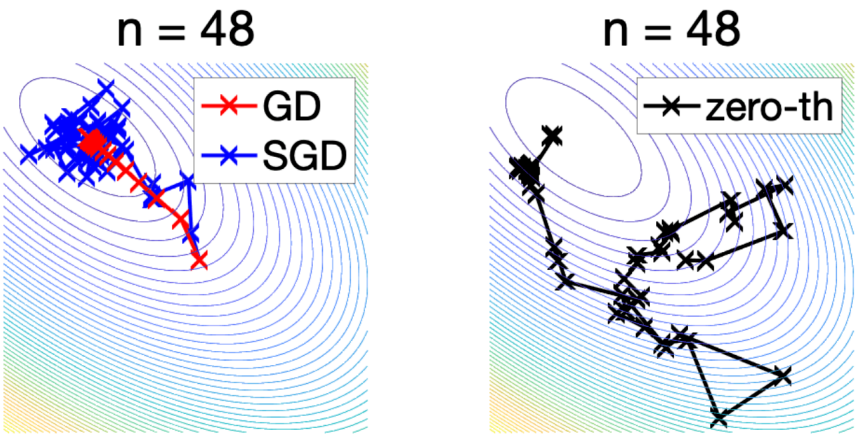


Figure 2.5: First vs zeroth order gradients Bach (2020)

We can imagine a discontinuous objective c as a utility measure for our dynamics $\mathbf{f}$ noisy evaluation of this utility through perturbations of our controls gives us an expected gradient $\nabla_{\mathbf{u}} c\left(\mathbf{f}(\mathbf{x}, \mathbf{u})\right) = \frac{1}{K}\left(\sum_{i=1}^{K} \nabla_{\mathbf{u}} c\left(\mathbf{f}(\mathbf{x}, \mathbf{u} + \epsilon_i)\right)\right)$. This is also similar to the notion of pathwise gradients.

Remarks

Our derivation of the REINFORCE method showed that one can estimate gradients of objective functions without any requirements such as differentiability or continuity. However, further analysis revealed that there is no free lunch, and what we gain from 0th-order gradients is paid for by high variance estimates, which can make convergence unstable. There are, however, important elements to 0th-order gradients that we can leverage, namely, noise. Finally, we also see that policy gradient methods do not suffer from the same pathologies as local methods such as MPPI, DDP, or PMP. Policy gradients parameterise the function space instead of trajectories and, therefore, learn controllers that do not require reoptimisation. Besides the computational upside, this approach allows for the generalisation to unseen states.

Relevant work

For a comprehensive overview of Deep RL, we refer the reader to the seminal references Sutton and Barto (2018) and Bertsekas and Tsitsiklis (1996). The large variance of the REINFORCE method Williams (1992) has been widely discussed, and many solutions have been proposed. For example, Trust Region Policy Optimisation Schulman et al. (2015) minimises the policy gradient objective while constraining the policy update to a region. Proximal Policy Optimsition Schulman et al. (2017) relaxes this constraint into a penalty. Both these methods essentially bound the high variance update of the zeroth order gradients. Suh et al. (2022) and Berahas et al. (2022) compare the variance of the zeroth and

first-order gradients and outline the benefits and drawbacks. Pang et al. (2023) and Le Lidec et al. (2024) explore benefits such as smoothing in robotics settings with discontinuous dynamics.

2.3 Conclusion

The goal of this thesis was to derive methods that can emulate optimal decision-making while providing flexibility in cost-function design without being inefficient in convergence. We now conclude our comprehensive review of that background method relevant to our goal. Our takeaways that we develop use to develop our contributions are:

- Global continuous optimal policies are only computable for linear systems under quadratic or exponential utility.
- Derivative-based local methods can provide detailed solutions at the cost of strict differentiability and continuity requirements on the objective.
- Sampling-based local methods provide flexibility at the cost of less refined solutions.
- All local solutions parameterise trajectories and require re-optimisation for new initial conditions.
- Model-free RL imposes no requirements on the objective landscape but suffers from high variance.
- Deep RL parameterises function space, which enables generalisation to new initial conditions.

Our *first contribution* in section 3 provides a local method that combines the flexibility of cost function design in sampling with the faster convergence of derivative-based methods. Our approach combines the modifies the Linear HJB objective (2.67) with additional divergence penalty based on an adaptive distribution representing the solution of the second order method iLQG (2.38). The resulting policy samples widely when there is no derivative information and follows optimised trajectories when derivatives arise.

Our *second contribution* in section 4 is focused on generalisation while maintaining learning efficiency. To achieve this, we first evaluate the effectiveness of model-free RL on continuous control environments without engineering the dynamics. We formulate our method based on the deterministic HJB equation (2.12). Using the HJB equation as a constraint, we are able to parameterise function spaces and learn the corresponding approximate value function for different tasks. The OC-theoretic nature of this formulation allows us to estimate gradients of our loss function similarly to the canonical equation of PMP (2.34).

For our *third* and final *contribution* in section 5, we focus on combining the ability to generalise with the ability to handle non-differentiable objectives while maintaining learning efficiency. Our approach relies on the stochastic HJB equation (2.58). Similar to the previous contribution, we convert this equation into an instantaneous penalty(2.57), enabling us to learn the corresponding value function. We show that this approach is able to handle non-differentiable objectives, and additionally, we provide empirical evidence that the inherent noise in our stochastic dynamics provides smoothing, which can be interpreted as robustness.

Chapter 3

Optimal Control via Combined Inference and Numerical Optimisation

Derivative-based optimisation methods are efficient at solving optimal control problems near local optima. However, their ability to converge halts when derivative information vanishes. The inference approach to optimal control does not have strict requirements on the objective landscape. However, sampling, the primary tool for solving such problems, tends to be much slower in computation time. We propose a new method that combines second-order methods with inference. We utilise the Kullback Leibler (KL) control framework to formulate an inference problem that computes the optimal controls from an adaptive distribution approximating the solution of the second-order method. Our method allows for the combination of simple convex and non-convex cost functions. This simplifies the process of cost function design and leverages the strengths of inference and second-order optimisation. We compare our method to Model Predictive Path Integral (MPPI) and iterative Linear Quadratic Gaussian controller (iLQG), outperforming both in sample efficiency and quality on manipulation and obstacle avoidance tasks.

3.1 Introduction

The goal of minimising an objective is a general one that applies to many areas. In the simplest cases, the age-old solution is to solve for the minimisation analytically. The analytical method becomes intractable in many interesting cases, and we must resort to computational methods. Methods for computing the minimisation generally fall into two categories: numerical and sampling-based optimisation. Both methods have advantages when applied to the correct domain. Numerical methods are better suited to domains where the objective of concern is smooth and derivative information is available. On the other hand, for domains in which the

objective function is highly discontinuous and multi-modal, sampling is a better choice due to its inherent exploitative capabilities and looser requirements on the objective. Interesting optimisation problems, however, are not only restricted to one of the mentioned domains. As a result, it may seem desirable to combine both methods to work in tandem. However, the abstraction that combines both methods and the formulation of a consensus between both methods is nontrivial. This paper aims to naturally combine numerical methods to refine when dense derivative information is available and sample to explore when derivatives vanish.

Here, we focus on the optimal control setting and its application to interesting, robotic settings where the governing dynamics are discontinuous, and the exact structure of the objective function is unknown. For example, humanoid locomotion is a task with the higher-level objective to move from point A to point B, using contacts with the ground. The exact objective function that generates a natural gait is unknown, and a naive objective based on the distance between point A and point B with respect to the span of controls is usually sparse. As a result, making and breaking contacts to solve the problem creates discontinuous optimisation landscapes. One can imagine the same principles in dexterous manipulation problems or problems where avoiding contact is key, such as obstacle avoidance.

3.2 Related work

One approach to remove the problem of sparse derivatives is to relax the optimisation problem. Mordatch et al. (2012) show impressive results on locomotion and manipulation tasks. The key insight is the relaxation of the contact constraints, resulting in dynamics where contact forces exist at a distance. A drawback of this approach is the problem of local minima. Once the optimisation landscape is sufficiently smoothed, many local optima may exist where finding the feasible is hard. In addition, physically unrealistic solutions are also a problem. To mediate the problem of local optimum Toussaint et al. (2020) introduced an adaption with additional logical decision variables over action modes such as sliding and flying. This allows for selective smoothing of dynamics, reducing the total number of local optima. However, the choice modes create a limiting range of dynamics. In addition, the problem remains deterministic, leading to extremely intricate behaviours that are not robust.

To tackle the sparsity problem, Posa et al. (2014) introduces linear complementary constraints to the optimisation problem. This allows the optimiser to decide whether bodies are in contact. This formulation underpins many formulations around locomotion. Pardo et al. (2017); Winkler et al. (2018); Cebe et al. (2020) use the same framework with additional task-specific parametrization. The formulation in this approach is specific to contact dynamics, and additionally, the optimisation generated is large and slow.

Aceituno-Cabezas et al. (2017); Kuindersma et al. (2016) proposed a separate

approach to reducing this problem of sparsity. They decompose the problem of planning and control and use integer variables and Mixed Integer Programming to select the dynamic modes of systems, for example, in or out of contact. Subsequently, a trajectory optimisation problem is solved to follow the planned points. The major drawback of this method is the loose relationship between the two solvers. As a result, significant work has been done to introduce dynamic feasibility into the planning segment Tonneau et al. (2020). Such feasibility constraints can be highly task-dependent and difficult to formulate.

Apart from numerical methods, Kappen et al. (2012); Todorov (2007) proposed an entirely different approach inspired by Kalman's duality Stengel (1986) that treats the optimal control problem as an inference. In this formalisation, the dynamics are treated as graphical models, and the optimal control distribution is solved through approximate inference using a Monte Carlo method variant. Toussaint (2009) extended this approach by introducing Gaussian approximations to solve nonlinear problems. Theodorou et al. (2010) proposed a path integral (PI) formulation of this approach and Rajamäki et al. (2016) proposed a PI-inspired approach modified by insights from differential dynamic programming (DDP) Jacobson (1968). The inference approach results in desirable outcomes such as clear formulations for exploration and the relaxation of the requirement on the objective, such as continuity and smoothness, and, therefore, a simpler cost function design for achieving goals. However, the major drawback of such methods is sampling, i.e. inferring the unknown distribution is computationally costly.

We aim to combine the strengths of inference and second-order trajectory optimisation. We aim to formulate a natural combination that can efficiently sample when no derivative information is available and follow optimised trajectories when derivatives arise.

3.3 The optimal control setting

Let us consider the generic continuous optimal control setting. An agent at state $\mathbf{x}_t \in \mathbb{R}^n$ is required to choose an action or control $\mathbf{u}_t \in \mathbb{R}^m$ such that the resulting sequence of states and actions $\{(\mathbf{x}_0, \mathbf{u}_0), (\mathbf{x}_1, \mathbf{u}_1)...(\mathbf{x}_N)\}$ minimises the summation associated to the total cost of each state and action pair. This problem can be solved by using the *principle of optimality*. The optimal solution is the combined optimal solutions to the subproblems of the original problem. This principle is formalised in the Bellman equation Bellman (1966), which underpins most optimal control theory.

$$v(\mathbf{x}, t) = \min_{\mathbf{u}} \left[\ell(\mathbf{x}, \mathbf{u}, t) + v\left(\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)\right)\right]$$

$v(\mathbf{x}_t, t)$ represents the value function, $\ell(\mathbf{x}_t, \mathbf{u}_t)$ the running cost and $\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$ the deterministic system dynamics all evaluated at $\mathbf{x}_t$ and $\mathbf{u}_t$. Solving this equation in its vanilla form suffers from curse dimensionality; the computation grows

exponentially with the number of states. In the next sections, we focus on the numerical and inference-based solution to the problem.

3.3.1 Numerical optimisation of the optimal control

The most established numerical solution to the Bellman solves the problem in its tangent space using a second-order method. This formulation is known as Differential Dynamic Programming (DDP) Jacobson (1968). Rewriting the Bellman equation as a function of state and control and using $Q(\mathbf{x_t}, \mathbf{u_t})$ to represent the summation of running cost with the value function

$$Q(\delta\mathbf{x}, \delta\mathbf{u}) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix}^\intercal \begin{bmatrix} 0 & \nabla_{\mathbf{x}} Q^\intercal & \nabla_{\mathbf{u}} Q^\intercal \\ \cdot & \nabla_{\mathbf{xx}} Q & \nabla_{\mathbf{xu}} Q \\ \cdot & \cdot & \nabla_{\mathbf{uu}} Q \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix}$$

In the quadratic approximation $Q(\delta\mathbf{x}, \delta\mathbf{u})$, the matrix is symmetric. Thus, only the upper triangular components are displayed. The terms $\nabla_{\mathbf{x}} Q$, $\nabla_{\mathbf{u}} Q$ represent gradients, while $\nabla_{\mathbf{xx}} Q$, $\nabla_{\mathbf{xu}} Q$, and $\nabla_{\mathbf{uu}} Q$ represent the respective Hessians.

Minimising the approximation with respect to the perturbed control $\delta\mathbf{u}$ and solving for the optimal control update

$$\begin{aligned} &\nabla_{\mathbf{u}} Q + \nabla_{\mathbf{ux}} Q \delta\mathbf{x} + (\hat{\mathbf{u}} - \mathbf{u}) \nabla_{\mathbf{uu}} Q = 0 \\ &\hat{\mathbf{u}} = (\mathbf{u} - \nabla_{\mathbf{u}} Q - \nabla_{\mathbf{ux}} Q \delta\mathbf{x}) \, (\nabla_{\mathbf{uu}} Q)^{-1} \end{aligned}$$

The above is identical to Newton's method where $\nabla_{\mathbf{u}} Q + \nabla_{\mathbf{ux}} Q \delta\mathbf{x}$ is the control gradient direction and $\nabla_{\mathbf{uu}} Q$ is the Hessian with respect to control used as curvature information. In DDP the update step is defined through a feed-forward gain $\mathbf{k} = \nabla_{\mathbf{u}} Q (\nabla_{\mathbf{uu}} Q)^{-1}$ and a feedback gain $\mathbf{K} = \nabla_{\mathbf{ux}} Q (\nabla_{\mathbf{uu}} Q)^{-1}$. Tassa et al. (2012) comprehensively explains the Gauss-Newton version of DDP known as iterative Linear Quadratic Gaussian controlle (iLQG). In this case, the second-order derivatives with respect to dynamics are ignored, and additional regularization terms are introduced for the robustness of Hessian approximations. From here onwards we refer to iLQG when referring to the numerical solution to the optimal control.

3.3.2 Approximate inference of optimal control

The inference approach to the optimal control problem has been tackled under many formulations. The KL control approach is grounded in the Bellman equation; therefore, drawing comparisons with the numerical approach is simpler. As a result, our focus in this paper is the KL-control approach and its dualities with path integral control. One can view the KL control approach to be grounded in the stochastic version of the Bellman equation shown in section 3.3.1

$$\min_{\mathbf{u}} \left[\ell(\mathbf{x}) + D(\mathbf{p}(\mathbf{x}'|\mathbf{x}, \mathbf{u}) \parallel \mathbf{p}(\mathbf{x}'|\mathbf{x})) + \mathbb{E}_{\mathbf{x}' \sim \mathbf{u}} \left[v(\mathbf{x}') \right] \right]$$

Where $\mathbf{x}'$ represent the next discrete state and $\mathbf{p}(\mathbf{x}'|\mathbf{x},\mathbf{u})$ and $\mathbf{p}(\mathbf{x}'|\mathbf{x})$ represent the controlled and uncontrolled probabilities, D represents a KL divergence operator. KL divergence measures the information difference between two distributions. v is the typical value function. Under this formulation, the probability of the optimally controlled dynamics becomes

$$\mathbf{p}^*(\mathbf{x}'|\mathbf{x},\mathbf{u}) = \mathbf{p}(\mathbf{x}'|\mathbf{x}) \exp\left(-v(\mathbf{x}')\right) \eta^{-1}$$

Therefore, the optimal transition probability $\mathbf{p}^*(\mathbf{x}'|\mathbf{x},\mathbf{u})$ becomes the transition probability of the passive dynamics adjusted by the exponential values of states and normalised by η.

KL control's central assumption is that the stochasticity of dynamics is entirely the function of noise over the control input. Meaning, in our case with normal input noise, an input $\mathbf{u}_t$ becomes $\mathbf{w}_t \sim \mathcal{N}(\mathbf{u}_t, \Sigma)$. As a result, if the control input trajectory $\mathbf{W} = (\mathbf{w}_0, \mathbf{w}_1, ..., \mathbf{w}_{T-1})$ and the initial state $\mathbf{x}_0$ are known, the resulting states trajectory can be computed. This effect allows for the representation of state distributions given the control distributions. We can represent the state distributions by computing the likelihood, where $\mathbf{X} = (\mathbf{x}_0, ..., \mathbf{x}_t, ..., \mathbf{x}_T)$

$$\begin{aligned}\mathbf{p}^*(\mathbf{X}) &= \prod_{t=0}^{T-1} \mathbf{p}(\mathbf{x}_{t+1}|\mathbf{x}_t) \exp\left(-v(\mathbf{x}_{t+1})\right) \eta^{-1} \\ &= \mathbf{p}(\mathbf{X}) \exp\left(-v(\mathbf{X})\right) \eta^{-1}\end{aligned}$$

Using the same state-control mappings defined in Williams et al. (2017), optimal state distribution $\mathbf{p}^*(\mathbf{X})$ can be represented in terms of optimal control trajectory $\mathbf{q}^*(\mathbf{W})$ under passive and candidate control distributions; $\mathbf{p}(\mathbf{W}) := \mathbf{w}_t \sim \mathcal{N}(0, \mathbf{\Sigma})$, $\mathbf{q}(\mathbf{W}) := \mathbf{w}_t \sim \mathcal{N}(\mathbf{u}_t, \mathbf{\Sigma})$

$$\mathbf{q}^*(\mathbf{W}) = \mathbf{p}(\mathbf{W}) \exp\left(-v(\mathbf{W})\right) \eta^{-1}$$

Theodorou and Todorov (2012); Williams et al. (2017); Theodorou (2015) show comprehensive derivation and a Model Predictive Control (MPC) formulation that allows an importance sample from this distribution. Our final algorithmic implementation uses the overall MPC structure of this method shown in Williams et al. (2017).

3.4 Combining sampling with second order method

To combine second-order optimisation and approximate inference, we start by reformulating the Bellman equation and introduce a secondary divergence term with the solution of the Bellman equation in the tangent space. That is the optimal trajectory obtained by the DDP/iLQG algorithm. Modifying the KL

Environment %	Ours			MPPI			Naive Combination			iLQG		
	Cost	% Success	Time	Cost	% Success	Time	Cost	% Success	Time	Cost	% Success	Time
Cartpole	6.69×10^3	100	15.05	8.20×10^3	100	10.03	2.07×10^4	100	33.13	2.5×10^3	100	1.84
Finger-Spinner	1.45×10^8	100	7.19	1.97×10^8	100	7.68	3.52×10^8	100	10.56	N/A	0	N/A
Push Object	2.12×10^{11}	100	15.87	4.92×10^{11}	100	73.93	N/A	0	N/A	N/A	0	N/A
Obstacle Avoidance	6.50×10^5	100	12.95	8.79×10^5	25	11.376	N/A	0	N/A	N/A	0	N/A

Table 3.1: Total trajectory cost and success rate for each method per environment. The costs are computed across the length of trajectories until the completion of tasks. Column Time represents the total computation time until task completion in seconds. For the number of samples and hyperparameters, refer to each task in section B.

objective, we can write

$$\begin{aligned} &\min_{\mathbf{u}}[\ell(\mathbf{x}) + (1-k)D(\mathbb{Q} \parallel \mathbb{P}) + kD(\mathbb{Q} \parallel \mathbb{C}) + \mathbb{E}_{\mathbb{Q}}[J(\mathbf{x}')]] \\ &D(\mathbb{Q} \parallel \mathbb{P}) = \mathbb{E}_{\mathbb{Q}}\left[\log\left[\frac{\mathbf{p}(\mathbf{x}'|\mathbf{x},\mathbf{u})}{\mathbf{p}(\mathbf{x}'|\mathbf{x})}\right]\right] \\ &D(\mathbb{Q} \parallel \mathbb{C}) = \mathbb{E}_{\mathbb{Q}}\left[\log\left[\frac{\mathbf{p}(\mathbf{x}'|\mathbf{x},\mathbf{u})}{\mathbf{p}(\mathbf{x}'|\mathbf{x},\mathbf{u}_{\text{iLQG}})}\right]\right] \end{aligned} \tag{3.1}$$

where $\mathbb{P}$ is the distribution of the passive dynamics for which the input v has a mean of 0 with variance $\mathbf{\Sigma}$. $\mathbb{C}$ represents the distribution of the controlled dynamics under iLQG where the input has a mean of $\mathbf{u}_{\text{iLQG}}$ with variance $\mathbf{\Sigma}_{\text{iLQG}}$. $\mathbb{Q}$ represents our control distribution which has a mean of u and variance Σ under the input v. $k \in [0,1]$ is the importance given to minimising each KL. Combining expectations and rearranging leads to the optimal control trajectory (complete derivation in the Appendix 3.8.1)

$$\mathbf{q}^*(\mathbf{W}) = \mathbf{p}(\mathbf{W})^{1-k}\mathbf{c}(\mathbf{W})^k \exp\big(-\frac{1}{\lambda}v(\mathbf{W})\big)\eta^{-1} \tag{3.2}$$

Where λ is a hyperparameter discounting trajectory costs. Equation 3.2 is very similar to the optimal transition described in section 3.3.2. However, in this case, the optimal trajectory is a factor of the joint distributions $\mathbb{P}$ and $\mathbb{C}$ scaled by the importance k and adjusted by the exponentiated negative cost of trajectories $\exp\big(-\frac{1}{\lambda}v(\mathbf{W})\big)$. In similar fashion to Williams et al. (2017) we use equation 3.2 to importance sample from the optimal distribution $\mathbb{Q}^*$

$$\mathbf{u}^* = \int \mathbf{q}(\mathbf{W}) \underbrace{\frac{\mathbf{q}^*(\mathbf{W})}{\mathbf{p}(\mathbf{W})^{1-k}\mathbf{c}(\mathbf{W})^k} \frac{\mathbf{p}(\mathbf{W})^{1-k}\mathbf{c}(\mathbf{W})^k}{\mathbf{q}(\mathbf{W})}}_{w(\mathbf{W})} \mathbf{w}_t \mathrm{d}\mathbf{W} \tag{3.3}$$

Computing the importance sampling weights $w(\mathbf{W})$ with respect to the distri-

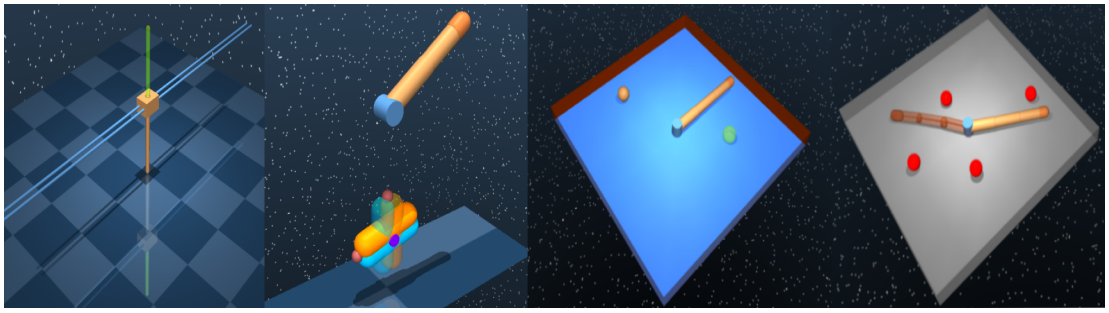


Figure 3.1: Left to right: Cartpole scenario: The objective is to swing the pole to an upwards position. Finger-Spinner: The objective is for the fully actuated 2-link manipulator to rotate the object to the desired location marked as transparent. Push Object: The goal is for the 3-link manipulator to move the orange sphere to the desired location (green). Obstacle Avoidance: The goal is for the 3-link manipulator to move to the desired position (red) without contacting spherical obstacles. Results video at link

butions (see Appendix) for a trajectory in the length of time T.

$$
\begin{aligned}
w(\mathbf{W}) = & \frac{1}{\eta} \exp\Bigg(-\frac{1}{2}\frac{1}{\lambda} v(\mathbf{W}) \\
& + \frac{1}{2} \sum_{t=0}^{T} \Big(\mathbf{u}_t^\top \boldsymbol{\Sigma}_t^{-1} \mathbf{u}_t + 2\mathbf{w}_t^\top \boldsymbol{\Sigma}_t^{-1} \mathbf{u}_t \\
& + k \Big((\mathbf{w}_t - \mathbf{u}_{\mathrm{iLQG}_t})^\top \boldsymbol{\Sigma}_{\mathrm{iLQG}_t}^{-1} (\mathbf{w}_t - \mathbf{u}_{\mathrm{iLQG}_t}) \Big) \\
& - k(\mathbf{w}_t{}^\top \boldsymbol{\Sigma}_t^{-1} \mathbf{w}_t) \Big) \Bigg)
\end{aligned}
\tag{3.4}
$$

From equation 3.4 $D(\mathbb{Q} \parallel \mathbb{C})$ translates into a cost term within $w(\mathbf{W})$ that penalises the deviation of the sampled control $\mathbf{v}_t$ from the solution obtained by iLQG, $\mathbf{u}_{\mathrm{iLQG}_t}$ where the cost gain is the covariance of the iLQG solution $\boldsymbol{\Sigma}_{\mathrm{iLQG}_t}$. Additionally, the parameter k dictates the importance of this cost and if set to 0 the weights become the same as the KL control case. $\boldsymbol{\Sigma}_{\mathrm{iLQG}_t}$ is a proxy measure of our confidence in the solution provided by the second-order method. To compute the variance of $\mathbb{C}$, the distribution that represents the solution of the second order method, we know that the trajectory $\mathbf{u}_{\mathrm{iLQG}\ t}$ minimises the quadratic approximation of the instance of the Q function $\delta Q(\mathbf{x}_t, \mathbf{u}_t)$ at $\mathbf{u}_{\mathrm{iLQG}_t}$. As a result, the mean and the variance for the normal distribution $\mathbb{C}$ can be defined by maximizing the log-likelihood:

$$
\ln \exp(-\delta Q(\mathbf{x}_t, \mathbf{u}_t)) \tag{3.5}
$$

maximising the log-likelihood leads to the same problem shown in section 3.3.1. Therefore, $\frac{\delta}{\delta \mathbf{u}} \ln \exp(-\delta Q_t(\mathbf{x}_t, \mathbf{u}_t))$ is at a maximum if $\mathbf{u}_t = \mathbf{u}_{\mathrm{iLQG}_t}$. We can obtain

the variance by computing the second derivative, $\frac{\delta^2}{\delta \mathbf{u}^2} \ln \exp(-\delta Q_t(\mathbf{x}_t, \mathbf{u}_t)) = (\nabla_{\mathbf{uu}} Q)^{-1} = \mathbf{\Sigma}_{\text{iLQG}_t}$. As a result input trajectories sampled form $\mathbb{C}$ take the form $\mathbf{w}_t \sim \mathcal{N}(\mathbf{u}_{\text{iLQG}_t}, (\nabla_{\mathbf{uu}} Q)^{-1})$. In our final formulation, we scale this variance approximation with a hyperparameter β.

Algorithm 7 shows the final structure of our proposed method. KL control method here represents the algorithm in Williams et al. (2017) with updated importance weights 3.4.

Algorithm 7: Inference with optimisation (Ours)

K: Number of samples and T: Number of time-steps
$q_{\text{iLQG}_r}(\mathbf{x}_t)$ and $q_r(\mathbf{x}_t)$: Running costs
$q_{\text{iLQG}_t}(\mathbf{x}_T)$ and $q_t(\mathbf{x}_T)$: Terminal costs
$\mathbf{u}_{0:T-1}$ and $\mathbf{u}_{\text{iLQG}_{0:T-1}}$: Initial trajectories
λ, k, β, $\mathbf{\Sigma}$: Hyper parameters for eq. 3.4
while *not done* **do**
  $\mathbf{x}_0 = \text{EstimateState}()$
  $[\mathbf{u}_{\text{iLQG}_{0:T-1}}, \nabla_{\mathbf{uu_{0:T-1}}} Q] = \text{iLQG}(\mathbf{x}_0)$
  $\mathbf{\Sigma}_{\text{iLQG}_{0:T-1}} = \beta \, (\nabla_{\mathbf{uu_{0:T-1}}} Q)^{-1}$
  $\mathbf{u}_{0:T-1} = \text{KLControl}(\mathbf{u}_{\text{iLQG}_{0:T-1}}, \mathbf{\Sigma}_{\text{iLQG}_{0:T-1}}, \mathbf{x}_0)$
  apply $\mathbf{u}_0$
  $\mathbf{u}_{0:T-1} = \mathbf{u}_{1:T-1}$
  $\mathbf{u}_{\text{iLQG}_{0:T-1}} = \mathbf{u}_{0:T-1}$
end

3.5 Experimental results and discussion

The results were generated from environments designed to test our method's validity and benefits. Therefore, key attributes of these environments shown in figure 3.1 are

- Discontinuous dynamics.
- Tasks where making or avoiding contact is necessary for completion.
- Tasks that allow for designing simple non-convex cost functions that induce sparsity in derivative information.

In this section, we evaluate whether our algorithm can combine trajectory optimisation and inference in a manner that allows for following the optimised trajectory when derivative information is available and resorting to unbiased inference when derivatives vanish. We compare our method to 3 other algorithms: MPPI, iLQG, and the naive combination, where the output of iLQG is used as

an initial trajectory for MPPI and vice versa. The results obtained are obtained across 20 random seeds. For each task, the lengths of the trajectories between solvers are the same. Table 3.1 summarises these results.

3.5.1 Implementation details

The implementation was developed in C++. The environments were simulated using Mujoco 2.0 Todorov et al. (2012). We have tuned the contact dynamics in Mujoco to be realistic. Additionally, our iLQG implementation uses derivative information at contacts. Some of the environments were modified versions of models developed by Tassa et al. (2018). The results were obtained on a PC with an Intel Core i9 CPU running Ubuntu 18.04.

3.5.2 Cartpole task

The Cartpole is a task we use to evaluate and compare the performance of our method in its simplest case. In this task, the state $\mathbf{x} \in \mathbb{R}^4$ consists of the cart and pole position and velocities. The desired state is $\mathbf{x}_{\text{goal}} = [0, 0, 0, 0]$. The running cost for this task is $\mathbf{x}_t^\top \mathbf{Q}_t \mathbf{x}_t$ and a terminal cost of $\mathbf{x}_T^\top \mathbf{Q}_T \mathbf{x}_T$ where $\mathbf{Q}_t = \text{diag}(1 \times 10^3, 5 \times 10^2, 0, 0)$ and $\mathbf{Q}_T = \text{diag}(1 \times 10^5, 5 \times 10^4, 5 \times 10^2, 5 \times 10^2)$ with horizon $T = 0.75s$ and number of samples $K = 3$. The hyperparameters β, λ and k were set to 1×10^{-3}, 0.1 and 1 respectively.

The results show that iLQG outperforms all other methods, beating ours by a factor of 2.7. This is not surprising as the cost function is not sparse, and dynamics are continuous, leading to a very smooth optimisation landscape where any derivative-based method will do well. Our method, however, outperforms MPPI by a factor of 1.2. This is achieved with a very low number of samples and biasing the samples towards the iLQG trajectory by setting a low value of β.

3.5.3 Finger-Spinner task

This is a manipulation task where the states $\mathbf{x} \in \mathbb{R}^6$ are the joint positions of the manipulator and the passive spinner, followed by their velocities. The desired goal state is $\mathbf{x}_{\text{goal}} = [0, 0, 0, 0, 0, 0]$. The cost function is quadratic with respect to the states where the gain terms act on the state of the spinner. $\mathbf{Q}_t = \text{diag}(0, 0, 1.5 \times 10^4, 0, 0, 50)$ and $\mathbf{Q}_T = \text{diag}(0, 0, 1.5 \times 10^4, 0, 0, 50)$. An additional binary scaling term $\mathbb{1}(\mathbf{x}_t) \times \mathbf{x}_t$ is also added to the running cost, where

$$\mathbb{1}(\mathbf{x}_t) = \begin{cases} 0, & \text{if contact between spinner and finger} \\ 1, & \text{otherwise.} \end{cases}$$

This term tracks the global positions of bodies and encourages contact until state errors are low. Since this term is non-differentiable, it is only added to the inference cost in equation 3.4. The choice of the hyperparameters are $[\beta, \lambda, k] = [0.25, 0.5, 1]$ and $T = 0.75$ and $K_{\text{MPPI}} = 30$ and $K_{\text{Ours}} = 10$

Our method outperforms all other methods, beating the second best (MPPI) by a factor of 1.3 with a third of the number of samples and a faster compute cycle. The derivative-based method (iLQG) entirely fails to solve this task. The cost function is sparse, and the dynamics are highly discontinuous. As a result, very few control inputs will create the contacts and the derivative direction required to solve the task. The naive combination can solve this problem by randomising the control input until contact is made. This results in a suboptimal trajectory due to the loose coupling between inference and numerical optimisation.

3.5.4 Push Object task

This is another manipulation task similar to the previous one. In this task, the object to be manipulated has higher degrees of freedom, and the exploration domain for finding contact is much larger. The states $\mathbf{x} \in \mathbb{R}^{10}$ are the joint positions of the planar manipulator in addition to the global Cartesian position of the object that is manipulated, followed by their derivatives. The desired state is $\mathbf{x}_{\text{goal}} = [0, 0, 0, 0.25, -0.22, 0.022, 0, 0, 0, 0]$. The cost function is quadratic with respect to the states and only defined by a term on the state of the object where $\mathbf{Q}_t = \text{diag}(0, 0, 0, 1 \times 10^5, 1 \times 10^5, 0, 0, 0, 5 \times 10^2, 5 \times 10^2)$ and $\mathbf{Q}_T = \text{diag}(0, 0, 0, 1 \times 10^5, 1 \times 10^5, 0, 0, 0, 0, 0)$. To encourage contact similar to the previous case, an additional binary term $\mathbf{1}(\mathbf{x}_t)\mathbf{x}_t$ is added to the running cost in equation 3.4. The choice of the hyperparameters are $[\beta, \lambda, k] = [1, 0.25, 1]$ and $T = 0.75$, $K_{\text{MPPI}} = 80$ and $K_{\text{Ours}} = 50$

Our method performs better than all other methods. Beating the second-best (MPPI) by a factor of 2.3. Our results show that contact dynamics is the major contributing factor to this. Our method can push the object much more intricately by staying close to the solution of the second-order method. iLQG, on its own, cannot solve this task due to the same sparsity of derivatives and discontinuity reasons as the previous task. The naive combination also fails 25% of the time and otherwise computes a trajectory that is an order of magnitude worse.

3.5.5 Obstacle Avoidance task

This task is essentially the opposite of the manipulation task. For successful completion, contact with the environment has to be avoided. The states $\mathbf{x} \in \mathbb{R}^6$ are the joint positions and velocities. The desired state is $\mathbf{x}_{\text{goal}} = [9.72, 0, 0, 0, 0, 0]$. The cost function is quadratic with respect to the states, defined by a term on the state of the manipulator where $\mathbf{Q}_t = \text{diag}(100, 10, 10, 0, 0, 0)$ and $\mathbf{Q}_T = \text{diag}(100, 100, 100, 10, 1, 1)$. To discourage contact, a binary term penalising contacts $\mathbb{1}(\mathbf{x}_t) \times \mathbf{x}_t$ is added to the running cost in equation 3.4. The choice of the hyperparameters are $[\beta, \lambda, k] = [20, 0.1, 1]$ and $T = 0.75$, $K_{\text{MPPI}} = 100$ and $K_{\text{Ours}} = 50$

Similar to the previous case, we outperformed all other algorithms, on average beating MPPI by a factor of 1.4 compared to its best case. We can achieve a 100% success rate in comparison to the 25% success rate of MPPI with half the number of samples. By choosing lower values of λ, our algorithm performs complex manoeuvres to escape local optima produced by the obstacles where MPPI cannot generate such motions. iLQG and the naive combination cannot avoid obstacles or solve this task.

3.6 Discussion

The relationship between the covariance and the inverse Hessian $\beta(\nabla_{\mathbf{u}\mathbf{u}_t} Q)^{-1} = \mathbf{\Sigma}_{\text{iLQG}_t}$ plays a central role in our method. We use this information to define a confidence measure. However, this measure reflects the convergence rate and not the quality of the solution. This poses problems where derivative-based optimisation will pay high costs for fast convergence.

The results show that our method can find derivative dense locations and follow optimised trajectories. This notion is tightly coupled with the choice of hyperparameters, specifically λ and β. This choice becomes apparent for environments with flat optimisation landscapes, such as the ones related to the tasks shown in section 3.5.4. These cases require exploration through high values of λ to allow for the sparse spread of the probability mass across trajectories. This, however, results in discounting the KL divergence cost, which can be detrimental to high-confidence optimised trajectories. Therefore, choosing the hyperparameters in our method is not trivial and can result in unsuccessful trajectories.

The ability to introduce non-differentiable costs to the inference part of our algorithm introduces a desirable formulation where we can remove the need to define complex convex costs for the derivative-based part of our algorithm. For example, in section 3.5.5, we remove the requirement for a distance cost to avoid obstacles. Another benefit of this formulation is encouraging contacts with similar cost structures and removing requirements such as defining contact locations or introducing relaxations to guide the derivative-based method towards contacts.

We have obtained the results for cases where the mixing term k between the passive and the controlled distribution is set to 1. This is because k induces a weighted joint distribution between the probabilities $\mathbb{P}$ and $\mathbb{C}$. This can become problematic when the distance between the two distributions is large. In that case, the joint can exist in a domain without useful control inputs. An automatic adaption of the mixing term can prove useful when the control distribution is uninformative.

We show that we can outperform MPPI in our tasks regarding the computation time. This is because of the sample efficiency of our algorithm through the combination with the second-order method. This is especially apparent in cases where contact and reasoning about their dynamics are required. The second-order

method can provide more information about task completion in such scenarios. Our push object task shows this. We perform on similar wall clock times when compared to the naive implementation. This is expected as the complexity of both algorithms is the same.

3.7 Conclusion and future work

We present a method that combines approximate inference with second-order trajectory optimisation. We formulate the approximate inference through a variant of KL control that introduces a divergence term from the distribution of optimised trajectories. Our derivation shows that this distribution can be approximated by the trajectory optimisation solution and its inverse Hessian. This natural combination removes the design of complex convex cost functions. We demonstrate this effect on our obstacle avoidance and manipulation tasks, where we only use binary cost functions to encourage or discourage making contacts. We test our methods on environments with discontinuities and a mix of simple differentiable and non-differentiable cost functions that nonetheless induce sparsity in derivative information. We compare our results to those obtained from vanilla iLQG, iLQG with randomized warmstarts and MPPI. In cases where derivative information is not immediately present, our method outperforms all others by exploring low-cost trajectories and following the optimised trajectories when derivative information appears. Some avenues can extend this work. For example, the hyperparameters for temperature, scaling the Hessian and the mixing term between two distributions are central to efficient exploration and refinement. A systematic approach to adapting these parameters can bring efficiency gains. The KL control formulation of the approximate inference is based on the assumption of constant input noise. Introducing theoretically sound variance adaption strategies can provide better convergence, especially when combined with derivative-based optimisation.

3.8 Appendix

3.8.1 KL control derivation

To obtain the final optimal distribution obtained in section 3.4 we start by

$$\min_{\mathbf{u}}[\ell l(\mathbf{x}) + (1-k)D(\mathbb{Q} \parallel \mathbb{P}) + kD(\mathbb{Q} \parallel \mathbb{C}) + \mathbb{E}_{\mathbb{Q}}[v(\mathbf{x}')]]$$

$$D(\mathbb{Q} \parallel \mathbb{P}) = \mathbb{E}_{\mathbb{Q}}\left[\log\left[\frac{\mathbf{p}(\mathbf{x}'|\mathbf{x},\mathbf{u})}{\mathbf{p}(\mathbf{x}'|\mathbf{x})}\right]\right]$$

$$D(\mathbb{Q} \parallel \mathbb{C}) = \mathbb{E}_{\mathbb{Q}}\left[\log\left[\frac{\mathbf{p}(\mathbf{x}'|\mathbf{x},\mathbf{u})}{\mathbf{p}(\mathbf{x}'|\mathbf{x},\mathbf{u}_{\text{iLQG}})}\right]\right]$$

Combining expectations gives (only minimisable terms of interest shown)

$$\min_{\mathbf{u}} \mathbb{E}_{\mathbb{Q}} \left[\log \frac{\mathbf{p}(\mathbf{x}'|\mathbf{x}, \mathbf{u})}{\mathbf{p}(\mathbf{x}'|\mathbf{x})^{(1-k)} \mathbf{p}(\mathbf{x}'|\mathbf{x}, \mathbf{u}_{\text{iLQG}})^k \exp(-v(\mathbf{x}'))} \right]$$

The exponentiated negative values are not transition probabilities, therefore introducing a normalisation constant

$$\phi(\mathbf{x}) = \sum \mathbf{p}(\mathbf{x}'|\mathbf{x})^{(1-k)} \mathbf{p}(\mathbf{x}'|\mathbf{x}, \mathbf{u}_{\text{iLQG}})^k \exp(-v(\mathbf{x}'))$$

Adding the normalization constant to the original problem and assuming $z(\mathbf{x}') = \exp(-v(\mathbf{x}'))$we can write

$$\min_{\mathbf{u}} \left[D(\mathbf{p}(\mathbf{x}'|\mathbf{x}, \mathbf{u}) \parallel \frac{\mathbf{p}(\mathbf{x}'|\mathbf{x})^{(1-k)} \mathbf{p}(\mathbf{x}'|\mathbf{x}, \mathbf{u}_{\text{iLQG}})^k z(\mathbf{x}')}{\phi(\mathbf{x})}) \right]$$

The above KL is minimised when the two probabilities are equal:

$$\mathbf{p}^*(\mathbf{x}'|\mathbf{x}, \mathbf{u}) = \frac{\mathbf{p}(\mathbf{x}'|\mathbf{x})^{(1-k)} \mathbf{p}(\mathbf{x}'|\mathbf{x}, \mathbf{u}_{\text{iLQG}})^k z(\mathbf{x}')}{\phi(\mathbf{x})}$$

Moving to likelihoods

$$\mathbf{p}^*(\mathbf{X}) = \prod_{t=0}^{T-1} \frac{\mathbf{p}(\mathbf{x}_{t_{j+1}}|\mathbf{x}_t)^{1-k} \mathbf{p}(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_{\text{iLQG}_t})^k z(\mathbf{x}_{t1})}{\phi(\mathbf{x}_t)}$$

Using the same state-control mappings defined in Williams et al. (2017), we write the above state distribution in terms of their density functions over a trajectory of length T with control input $\mathbf{W}$

$$\mathbf{p}(\mathbf{W}) = Z^{-1} \exp\left(-\frac{1}{2} \sum_{t=0}^{T-1} \mathbf{w}_t^\top \boldsymbol{\Sigma}^{-1} \mathbf{w}_t \right)$$

$$\mathbf{q}(\mathbf{W}) = Z^{-1} \exp\left(-\frac{1}{2} \sum_{t=0}^{T-1} (\mathbf{w}_t - \mathbf{u}_t)^\top \boldsymbol{\Sigma}^{-1} (\mathbf{w}_t - \mathbf{u}_t) \right)$$

$$\mathbf{c}(\mathbf{W}) = Z_{\text{iLQG}}^{-1} \exp\left(-\frac{1}{2} \sum_{t=0}^{T-1} (\mathbf{w}_t - \mathbf{u}_{\text{iLQG}_t})^\top \boldsymbol{\Sigma}_{\text{iLQG}_t}^{-1} (\mathbf{v}_t - \mathbf{u}_{\text{iLQG}_t}) \right)$$

We represent optimal state distribution as a function of the above distributions. As a result, the final optimal distribution becomes

$$\mathbf{q}^*(\mathbf{W}) = \mathbf{p}(\mathbf{W})^{1-k} \mathbf{c}(\mathbf{W})^k \exp\big(-\frac{1}{\lambda} v(\mathbf{W}) \big) \eta^{-1}$$

where η is the normalisation constant over trajectories and λ is the temperature an adjustable hyperparameter.

Chapter 4

Neural Lyapunov and Optimal Control

Despite impressive results, reinforcement learning (RL) suffers from slow convergence and requires many tuning strategies. In this paper, we investigate the ability of RL algorithms on simple continuous control tasks. We show that RL suffers from poor convergence without reward and environment tuning. In turn, we introduce an optimal control (OC) theoretic learning-based method that can solve the same problems robustly with simple parsimonious costs. We use the Hamilton-Jacobi-Bellman (HJB) and first-order gradients to learn optimal time-varying value functions and policies. We show the first-order relaxation of our objective results in approximate time-varying Lyapunov functions. We further verify our approach by satisfying this constraint over a compact set of initial conditions. We compare our method to Soft Actor Critic (SAC) and Proximal Policy Optimisation (PPO). In this comparison, we solve all tasks, never underperform in task cost, and show that at the point of our convergence, we outperform SAC and PPO in the best case by 4 and 2 orders of magnitude.

4.1 Introduction

Finding the optimal law to control a non-linear dynamical system with respect to an objective function is an open challenge for many systems. In recent years, reinforcement learning (RL) algorithms have empirically demonstrated an ability to solve complex continuous control tasks Hwangbo et al. (2019); Andrychowicz et al. (2020). RL does so by leveraging concepts within optimal control (OC) by parameterising and learning the policy or the value function space. Many of these impressive feats, however, rely on strategies to improve convergence and stability, such as; linearising dynamics regimes via Proportional-Derivative (PD) control or avoiding non-linear space regimes by terminating the episode outside of locally linear state space Katz et al. (2019) Tassa et al. (2012). Additionally, they rely on complex reward-shaping and extensive hyperparameter tuning Henderson

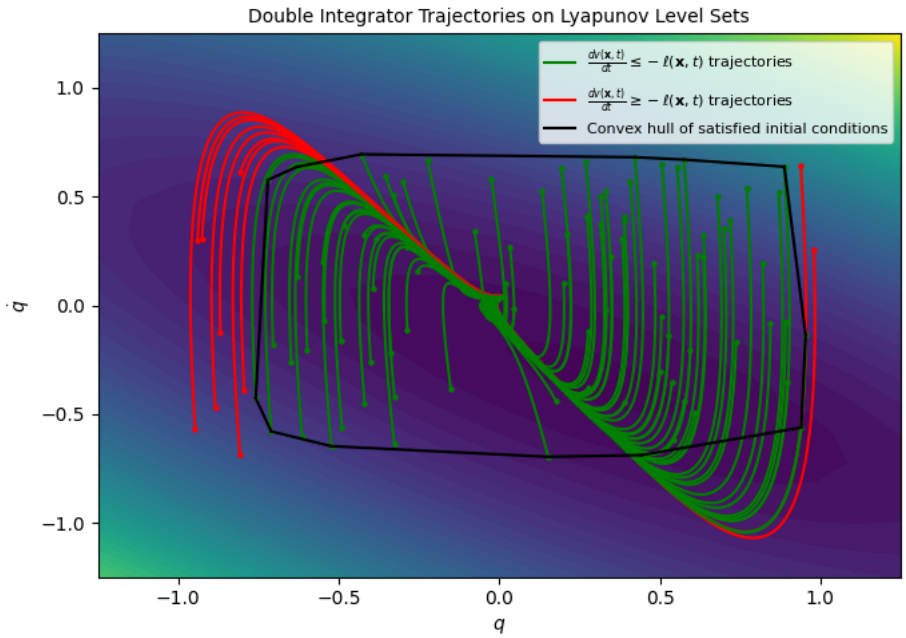


Figure 4.1: Compact stability region for a double integrator, computed by Neural Lyapunov Control.

et al. (2018). Part of the reason RL requires these strategies is that many of the algorithms use zeroth-order gradient estimates to minimise objectives. These estimates are high in variance and can descend into regions of the state space that impair convergence Suh et al. (2022).

Trajectory optimisation (TO) is another class of OC that has proven effective and efficient in problems where dynamics are nonlinear and unstable Tassa et al. (2012); Kuindersma et al. (2016). The efficiency of TO methods comes from leveraging first or higher-order derivatives of the objective. This results in more stable optimisation that does not require extensive hyperparameters, complex cost design or termination strategies. However, TO methods parametrise trajectory spaces and, importantly, require reoptimisation per new initial condition.

In this paper, we investigate the possibility of leveraging OC-theoretic tools to learn an optimal policy that does not require online reoptimisation for new inital conditions, while avoiding the drawbacks of RL. This requires finding a way to leverage OC theory within massively parallel machine learning frameworks. We ask two questions:

- Can RL algorithms solve simple continuous control tasks without the need for dampening the dynamics, shaping rewards or avoiding regions of the state space?

- Can we solve these problems with minimal hyperparameter optimisation and reward shaping, while embracing the entirety of the state and control space?

We believe the above are important problems to consider in order to improve the robustness, applicability, and reproducibility of learning-based approaches. To this end, we formulate a new OC theoretic function parameterised approach to learn OC policies. We utilise differentiable dynamics and the Hamilton-Jacobi-Bellman (HJB) optimality constraint to formulate mathematical programs that enable learning value functions and optimal policies. Additionally, we show that a specific relaxation of this objective allows us to learn Lyapunov functions, further verifying our method's first-order stability over a compact set of initial conditions.

To solve these programs, we leverage neural ODEs Sandoval et al. (2022), a new gradient estimator, and the parallel optimisation capabilities of deep learning. Finally, we compare our method to Soft Actor-Critic (SAC) Haarnoja et al. (2018) and Proximal Policy Optimisation (PPO) Schulman et al. (2017) on selected linear and nonlinear control affine tasks. We employ minimal cost shaping by using simple, parsimonious quadratic costs. We do not restrict the landscape of the state space by early termination. Additionally, we use identical network architectures, episode horizons and boundaries of initial conditions. We empirically show the following:

- RL suffers from poor convergence in environments where minimal reward shaping and environment tuning are used.
- In our experiments, our proposed method solves the tasks with significantly faster convergence and variance in random seeds. Outperforming SAC and PPO by at least a factor of 74 and 2, respectively.

4.2 Related work

RL robustness

Despite RL's effectiveness, several works have studied the shortcomings of the popular approaches. Authors in Henderson et al. (2018) performed a comprehensive study on the reproducibility of policy gradient algorithms such as PPO, TRPO and DDPG Schulman et al. (2017, 2015); Lillicrap et al. (2015). Their results showed that these algorithms are very sensitive to various parameters. For example, choosing a random seed may lead to significant outperformance on the same solver. The reward is also crucial to the performance, and authors show that simply scaling the rewards may lead to ineffective policies. The choice of network architecture and activation function have also been shown to be consequential. The sample complexity of RL-based methods has also been studied. In Recht (2019), authors show that policy gradient methods require many samples to converge even on simple linear LQR problems. They also show that applying LQR to simple learnt models outperforms policy gradient-based methods by orders of magnitude. The source of this sample complexity is also investigated by Suh et al. (2022). Authors show zeroth-order gradient estimates, the underlying

estimator in policy gradient methods, are very high variance and inefficient for various continuous control tasks. Thus, this raises an important drawback of RL, namely its sample complexity and sensitivity to hyperparameters. As a result, many approaches have been proposed to mitigate these problems.

OC theoretic policy/value learning

Approaches to learning OC policy and value functions have been previously explored. Many seminal works exist in offline settings, where off-the-shelf solvers are used to solve OC problems, and the generated data is then used for training. For example, Mordatch et al. (2015) applied this formulation to character control. The authors used second-order nonlinear OC to generate state trajectories. A neural network was then used to train a policy to generate these trajectories and perform complex dynamic behaviour. However, the regression does not consider the implicit consequential constraints within the trajectory optimisation problem, such as inverse dynamics. Additionally, this process is computationally expensive and requires compute clusters to solve multiple OC problems simultaneously. Other works within the offline setting focus on the value function space. Wang et al. (2022) formulate a supervised approach that uses neural networks to learn to fit approximate cost-to-gos, collected from trajectory optimisation. The authors tackle the complex task of humanoid foothold selection. This approach, inspired by Li et al. (2021), is based on the notion that informed terminal constraints and value functions can reduce planning horizons. However, due to the complexity of the task and the difficulty of the data generation process, the authors only considered two initial conditions. Additionally, as they solve a supervised problem, the regression does not typically consider implicit constraints such as Bellman backup.

Work done by Lutter et al. (2021b) addresses the problem of implicit constraints. They do so by approaching the problem from a value iteration perspective. In this case, the one-step HJB policy is used to rollout and compute cost-to-go data that is fitted to the value function using Bellman backup loss. Authors of Lowrey et al. (2018) also employ a similar approach; however, the HJB policy is replaced with a trajectory optimiser. However, neither author considers time parameterisation of the value function, even though the data is collected over a finite time. Overall, offline approaches are computationally intensive and primarily rely on supervised training methods that overlook the implicit dynamics of data being learnt.

Other online and more unified approaches have formulated methods that learn Lyapunov and control barrier functions. These methods train neural networks to inherently satisfy Lyapunov stability by minimising the penalty form of Lyapunov constraints. Dawson et al. (2022) jointly optimises over Lyapunov and a policy network, whilst Xiao et al. (2023) additionally optimises over a barrier function. Authors in Chang et al. (2019) also use a similar approach whilst introducing a verification procedure for certifying the learnt functions. However, these methods

do not apply to finite time horizon problems such as trajectory optimisation, as they do not solve for time-varying Lyapunov functions. Similarly, the finite horizon formulation of this approach is also investigated in Xiao et al. (2021). However, similar to before, this approach requires offline data generation by nominal controllers. Perhaps the closest line of work is done by Ainsworth et al. (2021), where the authors exploit the structure of neural ODEs with application to policy learning, given known dynamics. Similarly to Ainsworth et al. (2021), Sandoval et al. (2022) explores the constrained version of the policy-based formulation. Zhang et al. (2023) considers the time-parameterised version of the policy whilst introducing further regularisation using offline data. However, the above methods are formulated in policy space and cannot be applied to value and Lyapunov functions synthesis.

More specific HJB-based approaches also exist. Authors in Bansal and Tomlin (2021) focus on reachability analysis, wherein they relax the Hamilton-Jacobi (HJ) differential equation and employ neural networks to approximate the corresponding value function, effectively mitigating the curse of dimensionality typically associated with classical grid-based solvers. The results, however, are not compared to any baselines, and it is unclear whether backpropagation through time is feasible for high-dimensional systems. Similarly, Engin and Isler (2023) utilises a composite loss involving the HJB equation, Hamiltonian, and trajectory cost. This loss is minimised by learning both value and policy functions. The efficacy of this method is demonstrated through comparisons with model-based RL. To the best of our understanding, this method learns both a policy and a value function independently, so the effect of each network is unclear. In Zhong et al. (2020), authors also consider a similar domain where they learn the costate vector in the maximum principle formulation. However, this vector only considers the value function in its time differential space, limiting the learnt function to the trajectories on which it was trained.

This work introduces a unified approach that solves finite-time parameterised OC problems while leveraging neural ODEs to learn the corresponding time-varying Lyapunov and value functions. We test our method on linear and nonlinear control-affine systems. We compare our method to RL baselines, and outperform all by orders of magnitude.

4.3 Preliminaries

4.3.1 Optimal control

Optimal control is grounded within the Hamilton-Jacobi-Bellman (HJB) equation.

$$-\frac{\partial v}{\partial t}\left(\mathbf{x}_t, t\right) = \ell\left(\mathbf{x}_t, \mathbf{u}_t\right) + \mathbf{f}\left(\mathbf{x}_t, \mathbf{u}_t\right)^\top \nabla_{\mathbf{x}_t} v\left(\mathbf{x}_t, t\right) \tag{4.1}$$

where $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$ represents the deterministic dynamics, given state at time t $\mathbf{x}_t \in \mathbb{R}^n$ and control at time t $\mathbf{u}_t \in \mathbb{R}^m$. $\ell(\mathbf{x}, \mathbf{u})$ represents the state-control cost,

$\frac{\partial v}{\partial t}(\mathbf{x}_t, t)$ is the partial derivative of the value function with respect to time and $\nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t)$ is the spatial derivative of the value function with respect to state. This is a Partial Differential Equation (PDE) defining the time evolution of the value function. In other words, it is a compact description of the relationship between optimal costs for different states and times. In the case of the existence of a C^1 value function, cost and affine dynamics, one can analytically compute a *closed loop* feedback law optimal for all initial conditions $\mathbf{x}(0)$. The following shows this optimal control for a control-affine dynamical system where $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}_t) + \mathbf{g}(\mathbf{x}_t)\mathbf{u}_t$. Given a quadratic regularisation of control $\|\mathbf{u}\|_{\mathbf{R}}$. Where $\mathbf{R}$ is a positive definite matrix.

$$\begin{aligned} \min_{\mathbf{u}\in\mathbf{U}} \left[-\frac{\partial v}{\partial t}(\mathbf{x}_t, t) = \ell(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)^\top \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t) \right] \\ \pi^*(\mathbf{x}_t, t) = -\frac{1}{2}\mathbf{R}^{-1}\mathbf{g}(\mathbf{x}_t)^\top v_{\mathbf{x}}(\mathbf{x}_t, t) \end{aligned} \tag{4.2}$$

The above policy gives us a time-varying optimal policy $\pi^*(\mathbf{x}_t, t)$ for any initial condition. This is an important property of the HJB equation, which relies on the strong assumption that a known value function exists. On the other hand, the Pontryagin Minimum Principle (PMP) alleviates this problem by operating in the tangential space. It converts this PDE into an Ordinary Differential Equation (ODE) by interpreting $v_{\mathbf{x}}$ the value gradient as a stand-alone vector $\mathbf{p}$ known as a costate vector. Referring to the right-hand side of equation 4.2 as Hamiltonian H, the PMP aims to minimise H over a horizon T using an optimal control trajectory $\mathbf{u}^*$. This trajectory is determined given an initial condition $\mathbf{x}(0)$ and a terminal condition $\ell_{\mathbf{x}}(\mathbf{x}_T) = \mathbf{p}(T)$, subject to the following:

$$\begin{aligned} \dot{\mathbf{x}} = \nabla_{\mathbf{p}_t} H|_*, \qquad \dot{\mathbf{p}}^* = -\nabla_{\mathbf{x}_t} H|_* \\ \mathbf{u_t}^* = \arg\max_{\mathbf{u}\in\mathbf{U}} H(\mathbf{x}_t^*, \mathbf{u}, \mathbf{p}_t^*) \end{aligned} \tag{4.3}$$

Due to its tractability, this framework underpins the majority of trajectory optimisation algorithms such as iterative LQR Tassa et al. (2012) or differential dynamic programming (DDP) Jacobson (1968). However, it only generates an open loop trajectory valid for a single initial condition. This can be a significant limitation as new initial conditions require reoptimisation. We refer the reader to Liberzon (2011) for further discussion.

4.3.2 Neural ODEs

Neural ODEs are both practically and theoreticallyChen et al. (2018) fundamental to our work. In this section, we briefly describe neural ODEs from an OC perspective. The general objective for neural ODEs is defined as:

$$L(\mathbf{x}_T) = L\left(\mathbf{x}(0) + \int_{t_0}^{T} \mathbf{f}(\mathbf{x}_t, \mathbf{u}, t)\, dt\right) \tag{4.4}$$

Neural ODEs differ from traditional OC methods in two ways. Firstly. the control parameter $\mathbf{u}$ is a time-invariant variable interpreted as network weights. Secondly, the loss is evaluated at the end of the trajectory due to the analogous nature of the integration time step to hidden layers in a standard neural network. As a result, the canonical equations of neural ODEs are a special case of PMP, with the costate vector $\dot{\mathbf{p}}^* = -\nabla_{\mathbf{x}} H|_* = -\nabla_{\mathbf{x}} \mathbf{f}^\top(\mathbf{x}_t, \theta, t)\mathbf{p}_t$ ignoring the effects of running loss $L_{\mathbf{x}}(\mathbf{x}_t)$. Therefore, although neural ODEs create a familiar grounding to OC, they cannot be immediately applied to problems where a state change over time matters. The authors of Ainsworth et al. (2021) provide a reformulation of neural ODE's gradient estimator for policy learning, which we show can be extended to the value and Lyapunov space in the next section.

4.4 Learning Lyapunov and value functions

Our approach provides a straightforward and effective framework for learning value and Lyapunov parameterised OC problems while respecting implicit OC constraints.

4.4.1 Value functions

Let us focus on the HJB equation 4.2 under the optimal policy $\pi^*(\mathbf{x}_t, t)$. By moving the inner product between $v_{\mathbf{x}}$ and the dynamics $\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t$ or $\frac{d\mathbf{x}_t}{dt}$ to the left-hand side we can rewrite HJB as a definition for the total rate of change of the value function

$$\begin{aligned} \frac{\partial v(\mathbf{x}_t, t)}{\partial t} + \frac{\partial v(\mathbf{x}_t, t)}{\partial \mathbf{x}_t}\frac{d\mathbf{x}_t}{dt} &= -\ell(\mathbf{x}_t, \pi^*(\mathbf{x}_t)) \\ \frac{dv(\mathbf{x}_t, t)}{dt} &= -\ell(\mathbf{x}_t, \pi^*(\mathbf{x}_t)) \end{aligned} \tag{4.5}$$

The above can be interpreted as a constraint on the rate of change of the value function under the optimal policy. Integrating over a horizon T allows us to evaluate the consistency of optimal policy with respect to the above constraint. This leads to:

$$\begin{aligned} \int_0^T \frac{dv(\mathbf{x}_t, t)}{dt} &= -\int_0^T \ell(\mathbf{x}_t, \pi^*(\mathbf{x}_t)) \\ v(\mathbf{x}_T, T) - v(\mathbf{x}(0), 0) &= -\int_0^T \ell(\mathbf{x}_t, \pi^*(\mathbf{x}_t)) \end{aligned} \tag{4.6}$$

Equation 4.6 provides an equality that defines the evolution of the value function over the horizon T as a function of the running loss. However, we aim to learn the granular temporal and spatial change in the value function over the full horizon.

We discretise both integrals over Δt increments to capture this effect and apply the left Riemann sum approximation.

$$\begin{aligned} \int_{t}^{t+\Delta t} \frac{dv(\mathbf{x}_t, t)}{dt} &= -\int_{t}^{t+\Delta t} \ell(\mathbf{x}_t, \pi^*(\mathbf{x}_t, t)) \\ v(\mathbf{x}_{t+\Delta t}, t+\Delta t) - v(\mathbf{x}_t, t) &\simeq -\Delta t \times \ell(\mathbf{x}_t, \pi^*(\mathbf{x}_t, t)) \end{aligned} \tag{4.7}$$

equation 4.7 provides an approximate temporal and spatial instantaneous constraint on the value function. By rearranging and squaring, we can convert this constraint into a soft penalty P_v where:

$$\begin{aligned} P_v = \big(&v\left(\mathbf{x}_{t+\Delta t}, t+\Delta t\right) - v\left(\mathbf{x}_t, t\right) \\ &+ \Delta t \times \ell\left(\mathbf{x}_t, \pi^*\left(\mathbf{x}_t, t\right)\right)\big)^2 \end{aligned} \tag{4.8}$$

The mathematical program 4.9 leverages this penalty to learn an approximate value function $\tilde{v}(\mathbf{x}, t; \theta)$ that minimises the integration of this penalty over the horizon T and a compact set of initial conditions $\mathbf{x}_0 \in X_0$ where $|X_0| = K$. Program 4.9 uses the optimal feedback policy shown in 4.2 and is therefore valid only for control-affine dynamics and quadratic regularisation of controls. However, analytical optimal policies may be computed under any convex and PSD control constraints and are easily incorporated within this framework. This is further discussed in Section 4.6.

$$\begin{aligned} \min_{\theta} \Bigg[\frac{1}{K} \sum_{k=0}^{K} \sum_{n=0}^{N-1} \Big(&\tilde{v}\left(\mathbf{x}_{n+1,k}, n+1; \theta\right) \\ &- \tilde{v}\left(\mathbf{x}_{n,k}, n; \theta\right) + \Delta t \times \left(\|\mathbf{u}^*_{n,k}\|_{\mathbf{R}} + \ell\left(\mathbf{x}_{n,k}\right) \right) \Big)^2 \Bigg] \\ \text{s.t:} \\ \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}_t) + \mathbf{g}(\mathbf{x}_t)\mathbf{u}^*(\mathbf{x}, t) \\ \mathbf{u}^*(\mathbf{x}_t, t) &= -\frac{1}{2}\mathbf{R}^{-1}\mathbf{g}(\mathbf{x}_t)^\top v_{\mathbf{x}_t}(\mathbf{x}_t, t; \theta) \end{aligned} \tag{4.9}$$

The above mathematical program defines a finite-time parameterised OC problem that aims to learn a value function that approximately minimises the Bellman backup condition in equation 4.7 over horizon T, where $N = \frac{T - t_0}{\Delta t}$ represents the number of discrete time steps in the interval from t_0 to T. Each $t_n = n\Delta t + t_0$ and $t_{n+1} = (n+1)\Delta t + t_0$ specify the time instances at the n-th and $n+1$-th time steps, respectively. Here, $\mathbf{x}_{n,k}$ and $\mathbf{u}^*_{n,k}$ denote the state and control input at time t_n.

4.4.2 Approximate Lyapunov constraint

We briefly motivate constructing Lyapunov programs. In program 4.9, our objective is to minimise the square Bellman error. While it is possible to minimise,

approximation errors may lead to an unquantifiable performance loss in trajectory cost. The Lyapunov function relaxes this constraint to an inequality, guaranteeing a negative rate of change. In our case, we do not aim to synthesise true Lyapunov functions over the region of state space. We aim to satisfy the Lyapunov condition up to a first order over a compact set of initial conditions. As a result, we can provide first-order performance guarantees on compact regions of the state space, which are not possible for program 4.9.

We now focus on the derivation of the Lyapunov program. Finite-time Lyapunov analysis states that if we have a continuously differentiable positive definite function $v(\mathbf{x}_t, t)$, where $v(\mathbf{x}_t, t) > 0$ for $\mathbf{x}_t \neq 0$ and $v(0, t) = 0$ then $\mathbf{f}(\mathbf{x}_t, t)$ is stable if: $\dot{v}(\mathbf{x}_t, t) = \frac{\partial v(\mathbf{x}_t, t)}{\partial \mathbf{x}_t}\mathbf{f}(\mathbf{x}) + \frac{\partial v(\mathbf{x}_t, t)}{\partial t} < 0,\ \dot{v}(0, t) = 0$. This condition must hold for all $\mathbf{x}$ and all t. However, finding such Lyapunov functions is not trivial and is unknown apriori. Equation 4.5 and the conditions above impose constraints on the rate of change of function v in both the contexts of value and Lyapunov functions. However, the Lyapunov condition is a relaxed version of the HJB equality constraint. Constraint 4.5 is a lower bound where satisfying it transforms v from a Lyapunov function into an optimal value function. We aim to learn an approximate Lyapunov function that allows for generating controls that can sub-optimally complete a task and satisfy Lyapunov constraints within a compact set of initial conditions. To formulate this, we relax the HJB condition and define the Lyapunov constraint with respect to a task loss $\ell(\mathbf{x}) \geq 0$.

$$\begin{aligned} \frac{\partial v(\mathbf{x}_t, t)}{\partial t} + \frac{\partial v(\mathbf{x}_t, t)}{\partial \mathbf{x}_t}\mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) &\leq -\ell(\mathbf{x}_t, \mathbf{u}_t) \\ \frac{dv(\mathbf{x}_t, t)}{dt} &\leq -\ell(\mathbf{x}_t, \mathbf{u}_t) \end{aligned} \tag{4.10}$$

Performing the same integration over discrete time step Δt we obtain an instantaneous inequality constraint that can be converted to a soft penalty P_L:

$$\begin{aligned} P_L = \max\big(&v\left(\mathbf{x}_{t+\Delta t}, t + \Delta t\right) - v\left(\mathbf{x}_t, t\right) \\ &+ \Delta t \times \ell\left(\mathbf{x}_t, \mathbf{u}\left(\mathbf{x}_t, t\right)\right), 0\big) \end{aligned} \tag{4.11}$$

The program 4.12 uses this penalty to formulate an objective function that aims to learn an approximate Control Lyapunov Function (CLF) that enforces this Lyapunov constraint over a horizon T and K set of initial conditions.

$$\begin{aligned} \min_{\theta} \Bigg[\frac{1}{K} \sum_{k=0}^{K} \sum_{n=0}^{N-1} \max\Big(&\tilde{v}\left(\mathbf{x}_{n+1,k}, n+1; \theta\right) \\ &- \tilde{v}\left(\mathbf{x}_{n,k}, n; \theta\right) + \Delta t \times \left(\|\mathbf{u}^*_{n,k}\|_{\mathbf{R}} + \ell\left(\mathbf{x}_{n,k}\right)\right), 0\Big) \Bigg] \end{aligned} \tag{4.12}$$

s.t:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}_t) + \mathbf{g}(\mathbf{x}_t)\mathbf{u}^*(\mathbf{x}_t, t)$$

$$\mathbf{u}^*(\mathbf{x}_t, t) = -\frac{1}{2}\mathbf{R}^{-1}\mathbf{g}(\mathbf{x})^\top v_{\mathbf{x}}(\mathbf{x}_t, t; \theta)$$

We make use of the same HJB feedback policy used in value learning. The HJB policy provides the absolute lower bound on the Lyapunov condition. As a result it is a valid policy given a CLF. Subscripts n and k represent the same parameters as program 4.9. To parameterise the Lyapunov function while respecting Lyapunov constraints, we make use of input convex neural networks (ICNNs) Amos et al. (2017) with an additional positive constant term to approximate the positive definite Lyapunov function: $v(\mathbf{x}, t; \theta) = v_{\text{ICNN}}(\mathbf{x}, t; \theta) + \epsilon\|\mathbf{x}\|_2^2$.

Our penalty is defined as the first-order approximation of the Lyapunov condition, as shown below.

$$v(\mathbf{x}_{t+\Delta t}, t + \Delta t) = v(\mathbf{x}_t, t) + \left(\frac{\partial v}{\partial \mathbf{x}}\Delta\mathbf{x} + \frac{\partial v}{\partial t}\Delta t\right) + O((\Delta t)^2) \tag{4.13}$$

As a result, our penalty suffers from a local truncation error of $O((\Delta t)^2)$ and captures the first-order dominant effect, leading to the first-order satisfaction of the Lyapunov constraint over a compact set.

Neural ODE

The mathematical programs 4.9 and 4.12 have a bounded time horizon. While the neural ODE gradient estimator is suitable for finite horizon optimisation, it cannot be directly applied to these programs as the task loss needs to be evaluated over every timestep of the entire horizon T. We follow the same modifications shown in Ainsworth et al. (2021) to alleviate this. These modifications aim to convert the canonical equations of neural ODEs to the PMP case. Given the control policy and boundary conditions:

Given the control policy and boundary conditions:

$$\begin{aligned}
\mathbf{u}^*(\mathbf{x}_t, t, \theta) &= -\frac{1}{2}\mathbf{R}^{-1}\mathbf{g}(\mathbf{x}_t)^\top \nabla_{\mathbf{x}} v(\mathbf{x}_t, t; \theta), \\
\mathbf{a}_T &= \nabla_{\mathbf{x}} v(\mathbf{x}_T, T; \theta), \\
\mathbf{a}_t &= \nabla_{\mathbf{x}} \ell(\mathbf{x}_t, \mathbf{u}_t^*),
\end{aligned}$$

the canonical equations of the modified PMP can be formulated as:

$$\begin{aligned}
\dot{\mathbf{a}}_t &= -\mathbf{a}_t^\top \nabla_{\mathbf{x}_t} \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t^*) \\
&\quad - \nabla_{\mathbf{x}_t} \ell(\mathbf{x}_t, \mathbf{u}_t^*), \\
\nabla_\theta \dot{\ell} &= -\mathbf{a}_t^\top \nabla_{\mathbf{u}_t} \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t^*) \nabla_\theta \mathbf{u}_t^* \\
&\quad - \nabla_{\mathbf{u}_t} \ell(\mathbf{x}_t, \mathbf{u}(\mathbf{x}_t, t; \theta)) \nabla_\theta \mathbf{u}_t^*.
\end{aligned} \tag{4.14}$$

These canonical equations differ from vanilla neural ODEs mentioned in Chen et al. (2018). The resulting gradient estimator is less efficient in complexity compared to its vanilla form. However, it is a necessary addition. We have shown how parameterised OC problems can be formulated using the mathematical programs 4.12 and 4.9. In the next segment, we will demonstrate how these problems can be solved using the gradient estimator 4.14. We obtain these results on systems with control-affine dynamics.

4.5 Empirical results

Criteria

To evaluate our method, we apply it to tasks with linear or locally linear dynamics, such as the Double Integrator and Cartpole stabilisation. We also consider nonlinear problems like Cartpole swing-up and the planar reacher environment. Our method successfully learns time-varying Lyapunov and value functions that approximately satisfy the Bellman and Lyapunov constraints through minimising programs 4.9 and 4.12. For a comprehensive assessment, we compare the effectiveness of the learned functions against Soft Actor Critic (SAC) Haarnoja et al. (2018) and Proximal Policy Optimisation (PPO) Schulman et al. (2017) using the Stable-Baselines3 implementation Raffin et al. (2021). The hyperparameters for the baseline experiments can be found in our supplementary material. The results are presented in the accompanying table 5.1 and figure 4.3. We implemented the neural ODEs gradient estimator defined in equation **??** for all tasks. The results were obtained on a core i9 Intel processor with Nvidia GeForce RTX 4070 GPU. The Cartpole and the Reacher models are adopted from Tedrake (2023)Todorov and Li (2003)

Environments and solver setting

Dynamics: Since our method leverages first-order gradients, we require gradients to be available within the PyTorch graph. As a result, we implement our own dynamics.

In the introduction, we mentioned convergence strategies such as dampening dynamics and avoiding areas of the state and control space. Rigid body dynamics is defined by $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{b}(\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \tau$. However, in the case of very high friction: $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} \ll \mathbf{b}(\dot{\mathbf{q}})$, and the resulting dynamics becomes $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{b}(\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q})$. Thus,, one can practically remove a significant nonlinearity by tuning the environment with high friction coefficients, as done in the Mujoco Reacher environment used in OpenAI's Gym Brockman et al. (2016a). Regarding the state space, for example, in CartPole-V1, the control space is discrete, where 0 is the pushcart to the left, and 1 is the pushcart to the right. Also, the termination condition is active not far from the upright where $|\mathbf{q}_{pole}| \geq 0.2$ rads and $|\mathbf{q}_{cart}| \geq 2.4$. For the Inverted Double Pendulum, the control space is constrained within $-1 \leq \mathbf{u} \geq 1$. The termination condition is active when the pole length falls below 1 and the upright height is 1.196; $l_1 \cos(\mathbf{q}_{pole1}) + l_2 \cos(\mathbf{q}_{pole1} + \mathbf{q}_{pole2}) \leq 1$. Similar termination conditions are also used on the continuous control CartPole-V4 environments. Although such strategies can help convergence, they lead to policies that are trained on small regions of the state space. Additionally, boundaries for such constraints are not clear apriori. For instance, control constraints can directly impact the behaviour

of the policy, and in many cases, we do not know the control bounds at which the optimal solution is achieved. Additionally, dampening dynamics can also linearise and simplify the problem. However, it can lead to policies that are trained on unrealistic dynamics. We avoid such strategies in our experiments to evaluate the complete effectiveness of the methods. In the next section, we explain the details of our reward/cost design, control and space boundaries and termination conditions.

Solver setting: We aim to keep the settings identical for solvers. As a result, timestep Δt, horizon/episode length T, number of timestep, and rewards/cost are equal between solvers. Additionally, we make the control space unbounded $\mathbf{u} \in \mathbb{R}$; however, we regularise this space with quadratic regularisation $\|\mathbf{u}\|_{\mathbf{R}}$, $\mathbf{R}$ is also identical between solvers per task. Our reward design is negative for the cost function. Where our cost functions are the simple sum of running quadratic functions, cost $= \mathbf{x}^\top(t)\mathbf{Q}\mathbf{x}_t + \mathbf{x}^\top(T)\mathbf{Q}_T\mathbf{x}_T$. We also do not employ state-based termination conditions and rely only on the horizon to terminate. However, we try to tune the hyperparameters of each baseline solver to the best of our ability, starting from the parameters mentioned in the original papers Schulman et al. (2017) Haarnoja et al. (2018).

4.5.1 Value function results

Reacher

For this fully actuated system, the goal state is defined as $\mathbf{x} = [0, 0, 0, 0]$. It is important to mention the dynamics here follow the low friction definition mentioned in Section 4.5. The cost function is quadratic with respect to $\mathbf{x} \in \mathbb{R}^4$ and control $\mathbf{u} \in \mathbb{R}^1$ with $\mathbf{Q} = \text{diag}(1, 1, 0, 0)$ and $\mathbf{R} = \mathbf{M}^{-1}(\mathbf{q})$ where $\mathbf{M}^{-1}(\mathbf{q})$ is the inverse inertial matrix. Additionally we use a terminal cost $\mathbf{Q}_T = \text{diag}(100, 100, 1, 1)$. The negative of this cost is used as a reward. The value function is fully connected neural network (FCN) $\tilde{V} : \mathbb{R}^{4+1} \to \mathbb{R}^1$: (5-128-128-1 FCN). All discretisation timesteps are 0.01s, and the Adam optimiser for training Kingma and Ba (2014). As shown in table 5.1, we satisfy the HJB constraint 4.5 with an error of 0.25 $\pm$0.10. Constraint satisfaction approximately converges at 5e4. However, small errors result in task cost converging at approximately 1e5 timesteps. Our average final cost outperforms SAC and PPO by a factor of 7.43 and 1836.65.

Cartpole Swing Up

The Cartpole provides an underactuated environment obtained from Tedrake (2023). In this, the goal is defined at the unstable equilibrium $\mathbf{x} = [0, 0, 0, 0]$. A the quadratic cost function is used with respect to $\mathbf{x} \in \mathbb{R}^4$ and control $\mathbf{u} \in \mathbb{R}^1$ with $\mathbf{Q} = \text{diag}(0, 0, 0, 0)$ and $\mathbf{R} = \mathbf{M}^{-1}(\mathbf{q})$. Additionally, we use a terminal cost $\mathbf{Q}_T = \text{diag}(80, 600, .8, 4.5)$. The negative of this cost was used

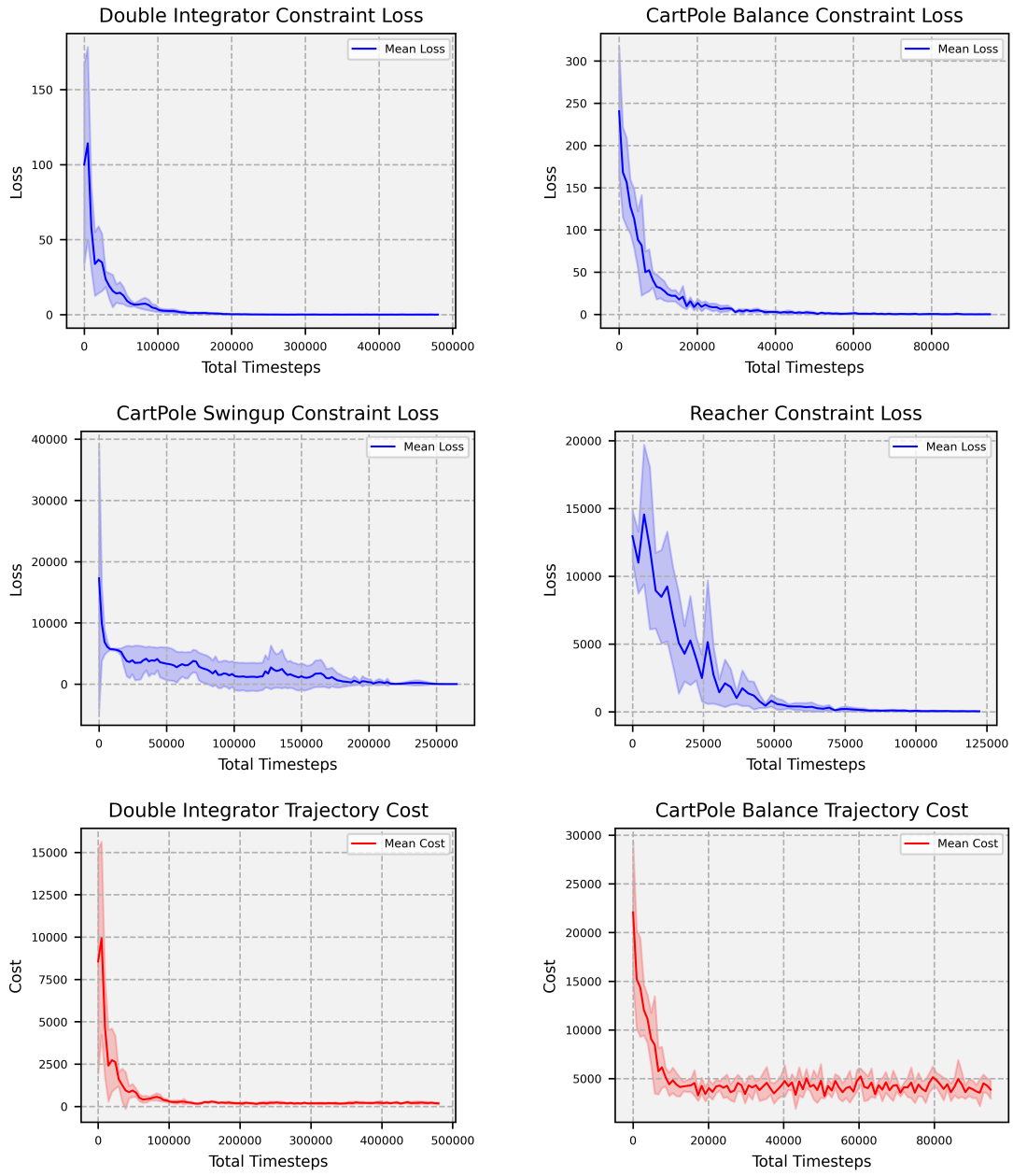


Figure 4.2: Constraint satisfaction loss for value and Lyapunov function constraints.

for baselines. The value function is parameterised by $\tilde{V}_{tl} : \mathbb{R}^{4+1} \to \mathbb{R}^1$: (5-128-128-1 FCN). The program was solved using the same timestep and optimiser as Reacher. The program satisfied HJB constraint by 24.65. Constraint satisfaction approximately converges at 2e5 with small errors causing cost convergence at under 2.5e5 timesteps. As shown in table 5.1 SAC and PPO show no convergence at this time.

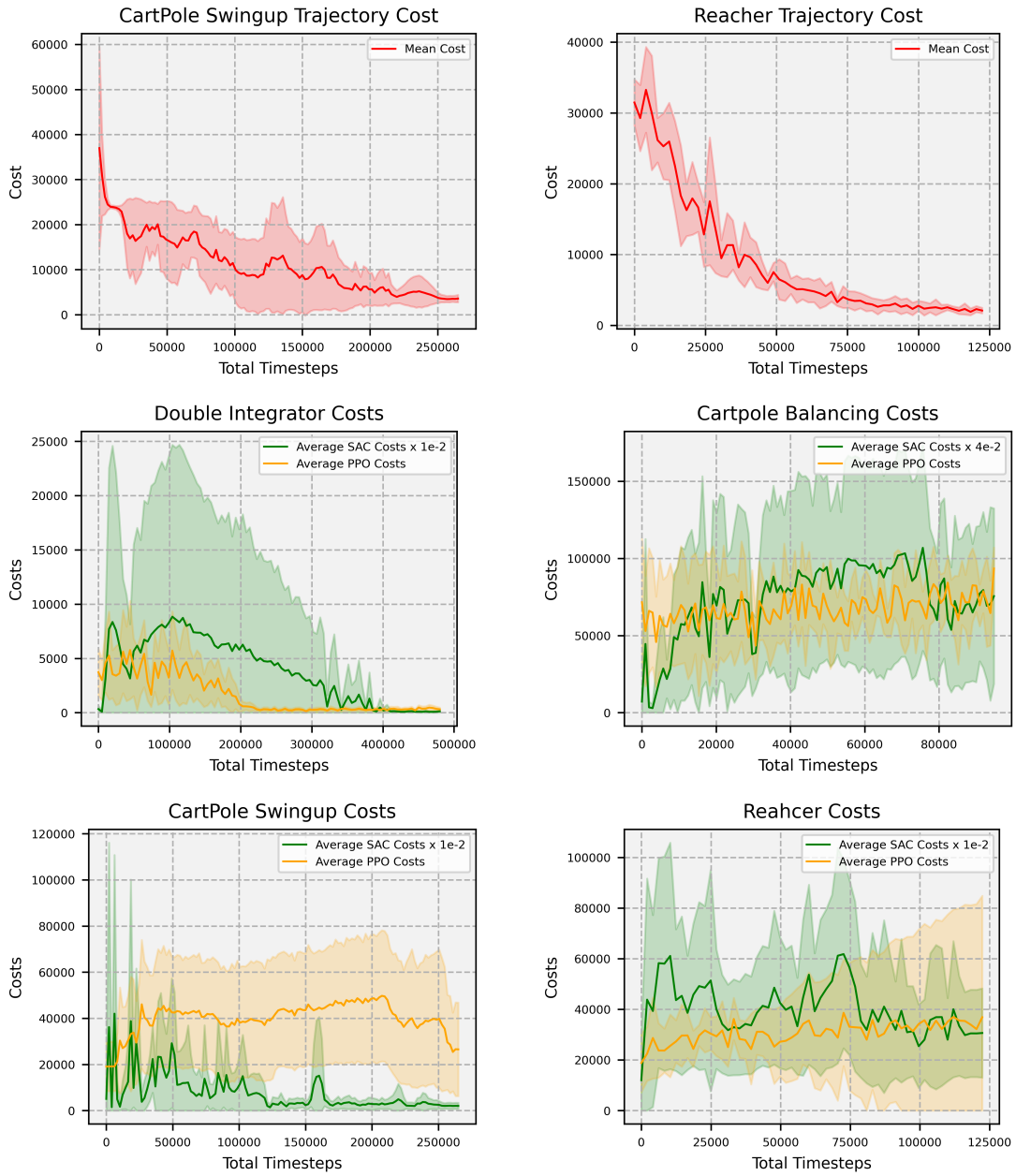


Figure 4.3: Trajectory cost using our method. Bottom three rows: SAC and PPO trajectory cost. **Due to high values, SAC costs are scaled for visualisation**.

Environment	Constraint Loss	Trajectory Cost			Cost Improvement		Horizon
		Ours	PPO	SAC	PPO	SAC	
Reacher	33.61 ± 20.25	2086.24 ± 419.84	$15514.49, \pm 4465.52$	3831267.56 ± 471973.23	7.43	1836.65	170
Swing up	24.65 ± 16.29	3568.74 ± 887.16	33147.12 ± 16041.53	$255854.68, \pm 81662.83$	144.47	10379.47	171
Balancing	0.07 ± 0.12	3868.80 ± 1012.27	93109.16 ± 15629.57	$2360774.63, \pm 1098978.99$	24.07	284.12	79
Double Integrator	0.03 ± 0.03	178.92 ± 77.63	$359.81, \pm 128.76$	$13311.58, \pm 7681.00$	2.01	74.78	400

Table 4.1: Training statistics and performance comparison against SAC and PPO

4.5.2 Lyapunov function results

Double Integrator

We also assess the ability of program 4.12 to satisfy trajectory stability Lyapunov constraints on locally linear systems up to first order. We consider a fully linear double integrator with the goal state at $\mathbf{x} = [0, 0]$. For this problem we require the rate of change of the stable trajectories to be upper bounded by the quadratic cost with respect to $\mathbf{x} \in \mathbb{R}^2$ and control $\mathbf{u} \in \mathbb{R}^1$ with $\mathbf{Q} = \text{diag}(10, 0.1)$, $\mathbf{Q}_T = \text{diag}(10, 0.1)$ and $\mathbf{R} = \mathbf{M}^{-1}(\mathbf{q})$ with the Lyapunov function $\tilde{V} : \mathbb{R}^{2+1} \to \mathbb{R}^1$: (3-64-64-1 ICNN). Our results show that we satisfy the Lyapunov constraint 4.10 with error 0.03 ± 0.03. Additionally, we show that if we keep the set of initial conditions $\mathbf{X}_0$, we are able to satisfy 1st order Lyapunov stability of 90% of the compact set of K = 100 initial conditions. This is also shown in the figure 4.1. Similar to the previous cases, constraint satisfaction loss converges more quickly than trajectory cost. Surprisingly, SAC significantly underperforms on this task. We outperform SAC and PPO by a factor of 74.78 and 2.01.

Cartpole Balancing

We apply program 4.12 locally linear task, Cartpole balancing. Again, we do not terminate episodes based on any state 4.5. Our termination is only based on the end of Horizon. The goal state is at $\mathbf{x} = [0, 0]$ with initial conditions $\mathbf{X}_0 \in (-0.6, 0.6)$. Similarly, rate of change of the stable trajectories to be upper bounded by the quadratic cost with respect to $\mathbf{x} \in \mathbb{R}^4$ and control $\mathbf{u} \in \mathbb{R}^1$ with $\mathbf{Q} = \text{diag}(0, 25, 0.5, 0.1)$, $\mathbf{Q}_T = \text{diag}(0, 25, 0.5, 0.1)$ and $\mathbf{R} = \mathbf{M}(\mathbf{q})^{-1}$ with the Lyapunov function $\tilde{V} : \mathbb{R}^{4+1} \to \mathbb{R}^1$: (5-200-500-1 ICNN). In this, the first order Lyapunov constraint is satisfied 4.10 with error 0.17 ± 0.16. The same program can also satisfy Lyapunov stability of 85% of the compact set of K = 100 constant initial conditions. Constraint and task cost converge at approximately 3e4 timesteps, with the final cost reaching 3868.80, outperforming SAC and PPO by a factor of 284.12 and 24.07.

4.6 Discussion and future work

The results show that our proposed programs notably outperform both SAC and PPO regarding convergence, costs associated with generated solutions, variance in results, and training stability. This difference is largely attributed to leveraging dynamics and its derivative information within training, analytically encoding dynamic structure within the policy, deterministic dynamics, and time parameterisation of the policy. In our evaluation, we treated the dynamics models as exact. This is a widespread practice within OC because of the assumption of certainty equivalence. This is where state feedback and re-optimisation through methods like Model Predictive Control can compensate for surprising amounts

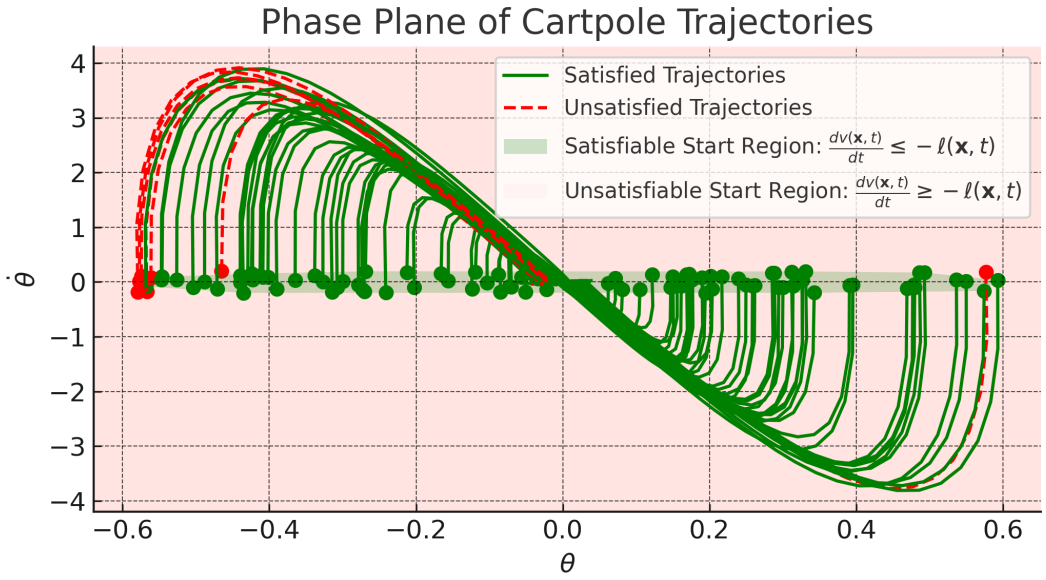


Figure 4.4: Cartpole balancing Lyapunov trajectories.

of error Recht (2019). Derivative information through optimisation allows for considering dynamic effects when minimising task costs. Additionally, the analytical inclusion of model parameters compensates for the system's dynamic effects and reduces the complexity of the search space. Typically, RL algorithms are optimised over fixed horizons or episode lengths but do not consider this time dependency within the policy. This essentially results in using an infinite-horizon policy for a finite-time task, which is provably sub-optimal.

There are also challenges with our proposed programs. The HJB constraint and its relaxation are central to our proposed approach. The programs 4.9 and 4.12 essentially aim to learn the underlying governing function of the differential equation 4.5 and its relaxation. Our results show we can satisfy these constraints over a compact set, albeit up to a margin. The existence of this margin can have non-trivial impacts. For example, small approximation errors within the value function can impact the task cost more. We hypothesise that the slower convergence of the task cost results from this phenomenon. Additionally, we only satisfy the Lyapunov constraint up to the first order; this essentially ignores intermediary effects between trajectories and does not guarantee the entire trajectory.

The credit assignment problem, in our case, remains an issue. Our results define the HJB constraint through the assigned cost function. This cost function essentially encodes the optimality constraint and, if defined poorly, can result in poor task performance. The choice of time horizon is also non-trivial. In our experiments, we initially perform a line search over several horizons and select the best one. However, we find our method is robust to various horizons so long

as values are not chosen to be very high or low. The discretisation time Δt is also chosen by considering both simulation efficiency and approximation errors.

Finally, our method is online, leading to typical problems associated with online algorithms, such as non-stationary learning, which can result in forgetting, especially in long-horizon tasks.

We aim to extend this work in other further practical directions. For instance, the problem formulation and the environments used were all deterministic. This work can be extended to the stochastic domain where the benefits of sampling, for example, smoothing or exploration, can be evaluated. Additionally, in this work we primarily focused on problem formulation and empirical verification of the proposal. However, further work aims to evaluate this method in more complex discontinuous environments with contact dynamics. It is important to mention the ability to do this requires the availability of differentiable GPU-based simulators, which are currently in the early stages of development. Finally, we derived the HJB policy under quadratic regularisation using our method. This can be a limiting factor in cases where different input constraints are required. In future work, we aim to show that this extension can be relatively trivial as analytical policies are computable for any convex positive semi-definite function in $\mathbf{u}$ Lutter et al. (2021b).

4.7 Conclusion

We started by asking two questions. Firstly, can widely used RL algorithms such as SAC and PPO solve simple continuous control problems with minimal reward and dynamics shaping? Our results show that RL still requires significant effort in tuning to achieve reasonable performance. Our second question asked for an alternative. We answer this by introducing two mathematical programs that use the HJB equation and first-order gradients to learn the value and Lyapunov functions. We demonstrate our effectiveness empirically by comparing PPO and SAC on linear and nonlinear control-affine tasks. Our results show that we can outperform both in terms of quality of the generated solution, task cost, variance in results and training stability.

Chapter 5

Neural Stochastic Optimal Control

Reinforcement Learning (RL) excels at learning generalisable policies and value functions for non-differentiable objectives, while Optimal Control (OC) optimisers achieve rapid convergence for unique initial conditions with differentiable objectives. We integrate these strengths through Stochastic OC and the discrete stochastic Hamilton-Jacobi-Bellman (HJB) equation to formulate a value learning problem. Our method allows us to leverage stochasticity to induce robustness and smoothing of the value function curvature. In addition, we use perturbed first-order gradients of differentiable dynamics to enhance convergence. We outperform model-free RL methods like Soft Actor-Critic (SAC) and Proximal Policy Optimisation (PPO) on classic continuous control tasks, reducing task costs by factors of up to 1076.02 and 8, respectively. We also show that noise enhances robustness and smoothing, achieving a fivefold reduction in task cost for navigation tasks with discontinuous objectives compared to deterministic settings. Finally, we learn a value function for a continuum robot on point-to-point tasks with obstacles. We show that a one-step greedy planner using this value function halves the tracking error on novel trajectories compared to a 15-step planner without it. We also successfully apply this planner to real hardware, achieving low tracking errors on unseen trajectories.

5.1 Introduction

Current approaches to controlling dynamical systems primarily employ two paradigms: Trajectory Optimization (TO) and Reinforcement Learning (RL), both grounded in optimal control (OC) theory. TO directly parameterises trajectories to minimise objectives using gradient-based solvers; this imposes strict differentiability and continuity requirements on the cost functions and dynamics. This results in fast convergence near local optima at the cost of detailed objective engineering. In contrast, deep RL typically uses differentiable function approximation to learn policies and/or value functions that maximise rewards. In many cases, the requirements on the objective landscape are removed.

Specifically, in model-free contexts, rewards are maximised through zeroth-order gradient estimates that are unbiased and inherently smooth objective landscapes. This, however, is at the cost of many noisy samples.

In short, TO achieves fast convergence by utilising derivative information from smooth objectives but doesn't generalise to new initial conditions. Conversely, deep RL smooths objectives through sampling and learns functions that adapt to new initial conditions, though at the cost of slow convergence. Therefore, it is desirable to develop a method that combines the best of both approaches: a generalisable method with fast convergence and relaxed requirements on the objective.

To achieve this, we approach the problem through the Stochastic Optimal Control (SOC) lens. We first provide an intuitive interpretation of smoothing via the stochastic Hamilton-Jacobi-Bellman (HJB) equation. We then establish a framework based on Neural Stochastic Differential Equations (NSDEs) that employs first-order gradients to learn the value function, which minimises the corresponding HJB equation. To validate our method's effectiveness, we initially benchmark it against established deep RL methods; Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO) on simple continuous control tasks. We show that we can outperform both by 4 and 2 orders of magnitude, respectively. We further demonstrate the benefits of smoothing and its impact on robustness in a navigation environment with discontinuous costs and dynamics. Our empirical results show that introducing noise enhances robustness and reduces overall task cost by a factor of 5 compared to the deterministic case. Lastly, we apply our method to learnt dynamics in a hardware experiment. Here, a one-step look-ahead MPC-style controller using the learned value function compared to the same setting; however, without the value function, can achieve an order of magnitude better performance in trajectory error. All dynamics are affine in control.

5.2 Background

Our work leverages the stochastic HJB equation to learn the corresponding value function. We achieve this using differentiable dynamics, OC theoretical framing, stochasticity, and perturbed gradients for effective smoothing. In this section, we review the literature relevant to each of these components.

Recent work has incorporated OC theory into learning policy and value functions. For example, Ainsworth et al. (2021) presents a continuous-time policy gradient formulation that minimises the total sum of running and terminal costs. This is achieved by learning a policy relative to a differentiable simulator, with scaling improvements made by refining the neural ODEs gradient estimator to incorporate the effects of the running cost. Similarly, Xu et al. (2022) advances the actor-critic framework using differentiable simulation and backpropagating the loss derivative through the dynamics. Another approach by Bansal and Tomlin (2021) focuses on reachability analysis, inspired by Physics-Informed

Neural Networks (PINN); the authors generate backwards reachable tubes by learning and minimising the task-specific Hamilton-Jacobi (HJ) equation. In Engin and Isler (2023), the authors introduce a composite loss derived from the HJB equation, incorporating distinct penalties for the Hamiltonian, trajectory, and terminal costs. They minimise this composite loss by learning the policy and value function separately. Similarly, Layeghi et al. (2023) formulates HJB-based loss to learn the value and Lyapunov function that uses analytical policies based on the value function to solve optimal trajectory and stabilisation problems. The authors propose efficient gradient estimators to improve scaling. Lutter et al. (2021a) adopts a comparable approach for learning infinite-horizon value functions. In Tassa and Erez (2007), the authors apply the second-order stochastic HJB equation in SOC, noting that minimising this equation naturally results in smoother function approximations due to inherent regularisation on the Hessian of the value function. However, in this setting, the effects of noise are ignored as the underlying dynamics remain deterministic. Another class of methods incorporating the stochastic HJB equation utilises Forward-Backwards Stochastic Differential Equations (FBSDEs), recently explored in various studies Exarchos and Theodorou (2018); Pereira et al. (2019). Inspired by PINNs Raissi et al. (2019); Cuomo et al. (2022), this approach features a novel network architecture that simulates FBSDEs and learns the value function. The key innovation is its ability to approximate the backpropagation of a conditional expectation, although this remains computationally intensive and unsuitable for high-dimensional systems. Despite these advances in scaling, the impact of smoothing from noisy perturbations remains an under-explored area in scaling OC theoretic formulations.

The concept of randomised smoothing dates back to the Gumbel-Max trick Gumbel (1954), wherein Gumbel noise is added to a categorical distribution to enable differentiable maximisation in discrete spaces Pogančić et al. (2019). This technique finds significant application where backpropagation through discrete choice spaces is required Maddison et al. (2016). Another setting where randomised smoothing plays a role is when a function of interest is non-differentiable. An example is model-free RL, where indirect estimation is necessary due to a lack of access to the world model. The REINFORCE trick is the main tool that enables this estimation Williams (1992). This allows computing the expected derivative of our function to be estimated by scaling its value by the negative of a selected score function. This is a zeroth-order optimisation technique that is the backbone of nearly all model-free RL algorithms, both policy gradient and actor-critic methods Schulman et al. (2015, 2017); Silver et al. (2014); Fujimoto et al. (2018); Haarnoja et al. (2018). It is unclear whether smoothing benefits are considered part derivation or a byproduct of the assumption of lack of access to model derivatives.

Perhaps the closest setting to our work is perturbed gradients, where the stochasticity induced in the optimisation process is produced by either noise in

dynamics, a noisy policy class or averaging over a minibatch of data Berthet et al. (2020). In this setting, the function of interest is differentiable but additionally perturbed by noise. This phenomenon is explored in Metz et al. (2021), where they show that increasing randomness by increasing the number of seeds leads to a smoother loss landscape in a rigid body dynamics task. Within dynamical systems, this idea has been demonstrated for tasks with discontinuities. For example, Le Lidec et al. (2024) uses random perturbation to smooth and solve TO problems by numerically smoothing system dynamics. Layeghi et al. (2022) approaches the problem from the opposite view and uses second-order gradient information to constrain sampling-based TO problems with complex contact dynamics. Work done by Suh et al. (2022) also combines zeroth and first-order gradient estimates to achieve a partially unbiased estimation with lower variance.

In this work, we develop a scalable OC solver that parameterises the function space for generalisation and leverages stochasticity to manage non-smooth objectives through randomised smoothing. We employ stochastic first-order gradients to maintain efficiency.

5.3 Method

Stochastic HJB

We first consider the nonlinear control affine stochastic differential equations (SDE).

$$\mathrm{d}\mathbf{x}_t = \left(\mathbf{f}\left(\mathbf{x}_t, t\right) + \mathbf{G}\left(\mathbf{x}_t, t\right)\mathbf{u}_t\right)\mathrm{d}t + \mathbf{B}\left(\mathbf{x}_t, t\right)\mathrm{d}\mathbf{w}_t \tag{5.1}$$

where $\mathbf{x}_t \in \mathbf{R}^n$ is an n dimensional state vector and $\mathbf{u}_t \in \mathbf{R}^m$ is the control vector. $\mathbf{f}(\mathbf{x}_t$, $\mathbf{w}_t \in \mathbf{R}^k$ is the Brownian motion of k-dimension, generator of the diffusion process. $\mathbf{B}(\mathbf{x}_t) \in \mathbb{R}^{n\times n}$ is the selection matrix representing the control authority at the current state. Given the equation (5.1), we aim to solve the following stochastic optimal control problem

$$\begin{aligned} v(\mathbf{x}, t_0) = \min_{\mathbf{u}\in\mathbb{R}^m} \quad & \mathbb{E}_{\mathbb{Q}}\left[v\left(\mathbf{x}_T, T\right) + \int_{t=t_0}^{T} \ell\left(\mathbf{x}_t, t\right)\,\mathrm{d}t\right] \\ \text{s.t.} \quad & \text{equation (5.1)} \end{aligned} \tag{5.2}$$

where $\mathbb{Q}$ represents control distribution $v(\mathbf{x})$ is the value function $l(\mathbf{x})$ is the state cost and $\mathbf{R}$ is the quadratic regulariser of the control input. Truncating the above problem into one step and applying the Ito' s lemma, we can derive the finite time stochastic HJB equation Fleming and Soner (2006). Below, for the sake of clarity,

we suppress the time dependence of $\mathbf{x}(t)$ and $\mathbf{u}(t)$:

$$
\begin{aligned}
-\frac{\partial v}{\partial t}\left(\mathbf{x}_t, t\right) = \min_{\mathbf{u}_t \in \mathbb{R}^m} \Big[& \ell\left(\mathbf{x}_t, t\right) + \frac{1}{2}\mathbf{u}_t^\top \mathbf{R}\mathbf{u}_t \\
& + \mathbf{f}\left(\mathbf{x}_t, \mathbf{u}_t, t\right)^\top \nabla_{\mathbf{x}_t} v\left(\mathbf{x}_t, t\right) \\
& + \left(\mathbf{G}\left(\mathbf{x}_t, t\right)\mathbf{u}_t\right)^\top \nabla_{\mathbf{x}_t} v\left(\mathbf{x}_t, t\right) \\
& + \frac{1}{2}\mathrm{Tr}\left(\mathbf{\Sigma}\left(\mathbf{x}_t, t\right) \nabla^2_{\mathbf{x}_t} v\left(\mathbf{x}_t, t\right)\right) \Big]
\end{aligned}
\tag{5.3}
$$

Equation 5.3 represents the ordinary differential equation that defines the evolution of the value function v given any initial time and state, with boundary condition $v(\mathbf{x}_T, T)$ defined by the terminal cost. Here, subscripts of t and $\mathbf{x}$ represent the derivatives' Legendre notation. $\mathbf{\Sigma}(\mathbf{x}_t, t) = \mathbf{B}(\mathbf{x}_t, t)\mathbf{B}(\mathbf{x}_t, t)^\top$. Minimising for $\mathbf{u}$, one can compute the optimal feedback controller

$$
\mathbf{u}^*(\mathbf{x}_t, t) = -\mathbf{R}^{-1}\mathbf{G}(\mathbf{x}_t)^\top \nabla_{\mathbf{x}_t} v(\mathbf{x}_t, t). \tag{5.4}
$$

Interestingly, the stochastic optimal feedback control law is the same as that of the deterministic case. Substituting this controller 5.4 into the HJB equation 5.3, we can write this ode in writing it in terms of the state where $\mathbf{K}(\mathbf{x}_t, t) = \mathbf{G}(\mathbf{x}_t, t)\mathbf{R}^{-1}\mathbf{G}(\mathbf{x}_t, t)^\top$

$$
\begin{aligned}
-\frac{\partial v}{\partial t}\left(\mathbf{x}_t, t\right) = & \ell\left(\mathbf{x}_t, t\right) \\
& + \nabla_{\mathbf{x}_t} v\left(\mathbf{x}_t, t\right)^\top \mathbf{f}\left(\mathbf{x}_t, \mathbf{u}_t, t\right) \\
& + \frac{1}{2}\mathrm{Tr}\left(\mathbf{\Sigma}\left(\mathbf{x}_t, t\right) \nabla^2_{\mathbf{x}_t} v\left(\mathbf{x}_t, t\right)\right) \\
& - \frac{1}{2}\nabla_{\mathbf{x}_t} v\left(\mathbf{x}_t, t\right)^\top \mathbf{K}\left(\mathbf{x}_t, t\right) \nabla_{\mathbf{x}_t} v\left(\mathbf{x}_t, t\right)
\end{aligned}
\tag{5.5}
$$

Solving the above partial differential equation (PDE) allows us to compute the optimal value function. Next, we aim to gain intuition about the properties of equation (5.5). To do so, we rearrange for the total rate of change of the value function.

$$
\frac{dv}{dt} = -\underbrace{\ell(\mathbf{x}_t, \mathbf{u}_t^*, t)}_{\text{cost term}} - \underbrace{\frac{1}{2}\mathrm{Tr}\left(\mathbf{\Sigma}(\mathbf{x}_t, t)\nabla^2_{\mathbf{x}_t} v(\mathbf{x}_t, t)\right)}_{\text{curvature-variance term}} \tag{5.6}
$$

From equation (5.6), one can see that the cost term is one which is also present in the deterministic case. This term implies that controls that incur lower costs at any given state will lessen the rate at which the value function declines. The curvature-variance term essentially increases the sensitivity of the rate of change of the value function to its local curvature, with this sensitivity further amplified by the variance. This interpretation is important in the context of value learning, as minimising the HJB equation under noisy, high variance, uncertain dynamics

tends to result in learning smoother value functions with less pronounced local curvature. We will empirically demonstrate the implications of this for robustness and smoothing in Section 5.4.2. Our goal now is to learn the value function that satisfies the PDE 5.6. One approach is to discretise and integrate both sides of the PDE to obtain instantaneous constraints. However, this can be difficult due to second-order partial derivatives $\nabla^2_{\mathbf{x}_t} v(\mathbf{x}_t, t)$.

To alleviate this, we use the Focker-Planck-Kolmogorov equation and its infinitesimal generator. The Fokker-Planck-Kolmogorov (FPK) equation defines the solution of an SDE (equation (5.1) under the optimal policy) through a partial differential equation defining the evolution of a corresponding probability density (HJB equation (5.5)). One can intuitively think of the SDE to define the evolution of a single trajectory of a stochastic process. In contrast, the FPK equation defines how the "density" of a sample of trajectories evolves in time. Next, we denote the definition of the infinitesimal generator of the FPK equation (5.5). To do so, we start from equation (5.3) under the optimal policy; we take the partial derivative of the value function with respect to time $\frac{\partial v}{\partial t}(x_t, t)$ to the left-hand side, then multiply by a discretisation timestep Δt and add $v(x_t, t)$ to both sides arriving at

$$\begin{aligned} v(\mathbf{x}_t, t) =& \ell\left(\mathbf{x}_t, \mathbf{u}_t^*, t\right) \Delta t + v\left(\mathbf{x}_t, t\right) + \frac{\partial v}{\partial t}\left(\mathbf{x}_t, t\right) \Delta t \\ &+ \left(\mathbf{f}\left(\mathbf{x}_t, \mathbf{u}_t, t\right) + \mathbf{G}\left(\mathbf{x}_t, t\right) \mathbf{u}_t\right)^\top \nabla_{\mathbf{x}_t} v\left(\mathbf{x}_t, t\right) \Delta t \\ &+ \frac{1}{2} \mathrm{Tr}\left(\mathbf{\Sigma}\left(\mathbf{x}_t, t\right) \nabla^2_{\mathbf{x}_t} v\left(\mathbf{x}_t, t\right)\right) \Delta t \end{aligned} \tag{5.7}$$

We can then collect the terms on the right-hand side by using Ito's lemma; we can rewrite the (5.7) as

$$v\left(\mathbf{x}_t, t\right) = \ell\left(\mathbf{x}_t, \mathbf{u}_t^*, t\right) \Delta t + \mathbb{E}_{\mathbb{Q}}\left[v\left(\mathbf{x}_{t+\Delta t}, t + \Delta t\right)\right] \tag{5.8}$$

Equation (5.8) represents what is known as the stochastic Bellman backup. It states that the value of the current state is the expected value of the next state in addition to the cost of the current state under the optimal policy. This represents an instantaneous constraint on the value function. We convert this into a penalty to learn a detailed spatial and temporal approximation of this. The next state probability measures $\mathbb{Q}$ is defined such that under $\mathbb{Q}$, the next state $\mathbf{x}_{t+\Delta t}$ follows a normal distribution centred around optimally control dynamics with variance $\mathbf{\Sigma}$ scaled by discretisation. This is written as $\mathbf{x}_{t+\Delta t} \sim \mathcal{N}\left(\mathbf{x}_t + \left(\mathbf{f}\left(\mathbf{x}_t\right) + \mathbf{G}\left(\mathbf{x}_t, t\right) \mathbf{u}_t^*\right) \Delta t, \mathbf{\Sigma} \Delta t\right)$.

Learning the value function

Equation (5.12) provides us with a discrete temporal and spatial constraint of the stochastic HJB 5.7. To learn the corresponding value function, we first convert this constraint to a penalty shown below

$$\ell_v = \left(\mathbb{E}_{\mathbb{Q}}\left[v\left(\mathbf{x}_{t+\Delta t}, t + \Delta t\right)\right] - v\left(\mathbf{x}_t, t\right) + \ell\left(\mathbf{x}_t, \mathbf{u}_t^*, t\right) \Delta t\right)^2 \tag{5.9}$$

We then define a mathematical program that learns an approximate value function over a horizon by minimising penalty (5.9). To do so we define the problem over a time horizon T with initial conditions $\mathbf{x}_0$ sampled from a compact set $\mathbf{X}_0 \in \mathbb{R}^{K \times n}$. The value function is parameterized by a neural network $v(\mathbf{x}_{k,t_n}, t_n; \theta)$. We formulate the learning problem as:

$$\begin{aligned} &\min_{\theta} \frac{1}{K} \sum_{k=0}^{K-1} \sum_{i=0}^{I-1} \ell_v \left(\mathbf{x}_{k,t_i}, \mathbf{u}_{k,t_i}, \mathbf{x}_{k,t_{i+1}}, t_i; \theta \right) \\ &\text{s.t:} \\ &\mathrm{d}\mathbf{x}_{k,t} = \left(\mathbf{f}\left(\mathbf{x}_{k,t}, t\right) + \mathbf{G}\left(\mathbf{x}_{k,t}, t\right) \mathbf{u}_{k,t} \right) \mathrm{d}t + \mathbf{B}\left(\mathbf{x}_{k,t}, t\right) \mathrm{d}\mathbf{w}(t), \\ &\mathbf{u}^*(\mathbf{x}_{k,t}, t) = -\frac{1}{2} \mathbf{R}^{-1} \mathbf{G}\left(\mathbf{x}_{k,t}, t\right)^\top \nabla_{\mathbf{x}_{k,t}} \tilde{v}\left(\mathbf{x}_{k,t}, t; \theta\right). \end{aligned} \tag{5.10}$$

Here, $I = \frac{T - t_0}{\Delta t}$ signifies the count of discrete intervals between t_0 and T. Each point in time t_i is defined as $t_i = i\Delta t + t_0$, while t_{i+1} is given by $t_{i+1} = (i+1)\Delta t + t_0$, denoting the i-th and $(i+1)$-th time steps, respectively. The terms $\mathbf{x}_{k_i}$ and $\mathbf{u}^*_{k_i}$ represent the state and the control input at the i-th timestep.

Algorithm 8: Neural SOC

Define hyper-parameters $\mathbf{R}, \mathbf{Q}, \mathbf{Q}_T, T, K, \Delta t, I = T/\Delta t$
Initialize value function $\tilde{v}(\mathbf{x}_{k,t_i}, t_i; \theta)$
System dynamics and control update rule:
  $\mathrm{dyn}^*(\mathbf{x}_{k,t_i}, \mathbf{u}^*_{k,t_i})$ using Equation 5.1
  $\mathbf{u}^*(\mathbf{x}_{k,t_i}, t_i) = -\mathbf{R}^{-1}\mathbf{G}(\mathbf{x}_{k,t_i}, t_i)^\top \nabla_{\mathbf{x}_{k,t_i}} \tilde{v}(\mathbf{x}_{k,t_i}, t_i; \theta)$
Define loss function $\ell_{\text{bell}}(\mathbf{x}_{k,t_i}, \mathbf{u}^*_{k,t_i}; \theta)$ using Equation (5.9)
while *epochs left* **do**
  Sample initial states $\mathbf{x}_0^{(K)} \sim U(\mathbf{x}_{\text{lower}}, \mathbf{x}_{\text{upper}})$
  $\mathbf{x}_{0:I}^{(K)} = \mathrm{sdeint}(\mathrm{dyn}^*, \mathbf{x}_0^{(K)}, T, \Delta t)$
  Update θ using gradient descent:
  $\theta = \theta + \alpha \nabla_\theta \ell_{\text{bell}}(\mathbf{x}_{0:I}^{(K)}, \mathbf{u}_{0:I}^{(K)})$
end

Backpropogation

Efficient schemes for computing the derivative of the loss function of program (5.2) have been explored in neural differential equations. In deterministic cases, recent works have utilised the Pontryagin minimum principle or the adjoint sensitivity method to derive efficient gradient estimators of the loss. However, adapting these approaches to stochastic differential equations (SDEs) proves challenging. This adaptation relies on the theory of Forward-Backward SDEs (FBSDEs). Yet, unlike in deterministic scenarios, the FBSDE framework lacks numerical efficiency

as it requires computing conditional expectations of the backward SDE Tzen and Raginsky (2019); Kidger et al. (2021). As a result, following Automatic Differentiation (AD) based approaches Giles and Glasserman (2006), we resort to backpropagating through the internal operations of the solver.

Here, we explain a brief mathematical overview of the method under Euler discretisation. We define $\mathbf{X}_{t_i} \in \mathbb{R}^{K \times n}$ to be a set of states at the time of size K at discrete time-mesh n where $0 < i < I$ and Δt is the discretisation time step.

Step 1 computes the state trajectory via Euler-Maruyama Platen and Bruti-Liberati (2010):

$$\begin{aligned} \mathbf{X}_{t_{i+1}} = \mathbf{X}_{t_i} + \left(\mathbf{f}\left(\mathbf{X}_{t_i}, t_i\right) + \mathbf{G}\left(\mathbf{X}_{t_i}, t_i\right) \mathbf{U}^*_{t_i}\right) \Delta t \\ + \sqrt{\Delta t} \mathbf{B}\left(\mathbf{X}_{t_i}, i\right) \mathbf{W}_{t_i} \end{aligned} \tag{5.11}$$

Equation 5.11 computes a set of trajectories $\mathbf{X}$ under the time-varying optimal policy given by the value function $-\mathbf{R}^{-1}\mathbf{G}(\mathbf{X}_{t_i}, t_i)^\top \nabla_{\mathbf{X}} v(\mathbf{X}_{t_i}, t_i; \theta)$ equivalent to $\mathbf{U}^*_{t_i}$.

Step 2, optimise the value function by computing and backpropagating through the stochastic Bellman backup using the trajectories $\mathbf{X}$

$$\begin{aligned} \ell_v = \left(\mathbb{E}_{\mathbb{Q}}\left[v\left(\mathbf{X}_{t_{i+1}}, t_{i+1}\right)\right] - v\left(\mathbf{X}_{t_i}, t_i\right) \right. \\ \left. + \ell\left(\mathbf{X}_{t_i}, \mathbf{U}^*_{t_i}, t_i\right) \Delta t\right)^2 \end{aligned} \tag{5.12}$$

The basis for the validity of backpropagating through equation (5.9) is explained in Glasserman (2004) and, at its core, is dependent on the assumption where $\nabla_{\mathbf{X}_{t_{i+1}}} \mathbb{E}_{\mathbb{Q}}\left[v(\mathbf{X}_{t_{i+1}}, t_{i+1}; \theta)\right] = \mathbb{E}_{\mathbb{Q}}\left[\nabla_{\mathbf{X}_{t_{i+1}}} v(\mathbf{X}_{t_{i+1}}, t_{i+1}; \theta)\right]$, the interchange of derivative and expectation.

5.4 Results

This section provides the results of our work on domains that test our hypothesis. In the introduction, we motivated our method through the lens of SOC. Our hypothesis was on three main grounds:

- Faster convergence from first-order derivatives.
- Smoothing of nonsmooth objective.
- Generalisation from function parameterisation.

To evaluate the effectiveness of first-order gradients, we compare them to 0th-order model-free RL algorithms, namely SAC and PPO. These experiments are performed on classic continuous control tasks.

To evaluate the effectiveness of smoothing, we first evaluate the effect of noise on a simple LQG problem. We then move to a higher-dimensional navigation

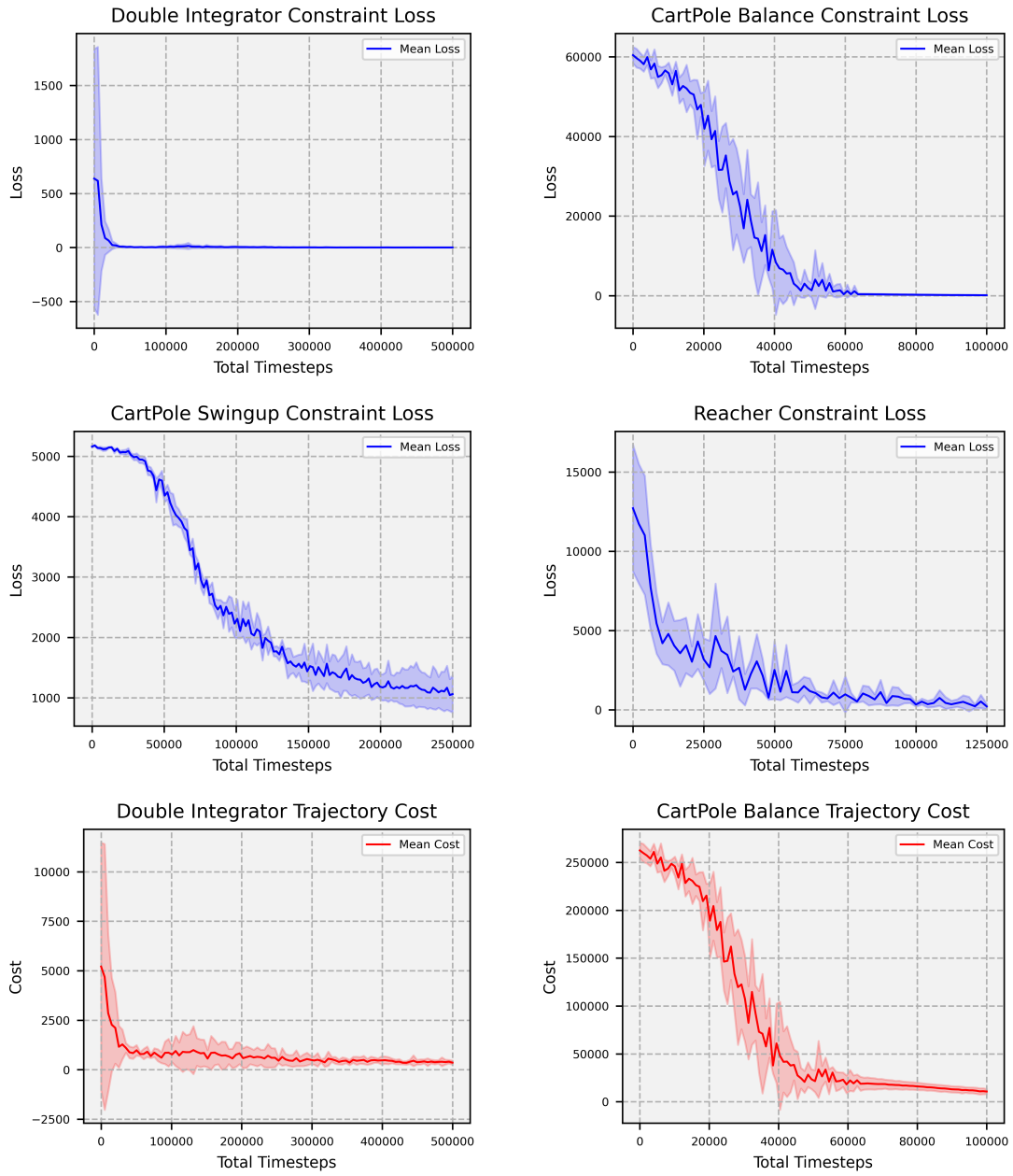


Figure 5.1: Top and middle rows: Constraint satisfaction loss and trajectory cost for value function constraints using our method.

problem with discontinuous dynamics and costs. We also evaluate the effects of smoothing on obstacle avoidance tasks on continuum robot in simulation with learnt dynamics and discontinuous costs.

Finally, to evaluate generalisation, we use the previously learnt value function for a trajectory tracking task on a real continuum robot. We base our evaluation on greedy sampling with respect to the value function using Model Predictive Path Integral Control(MPPI).

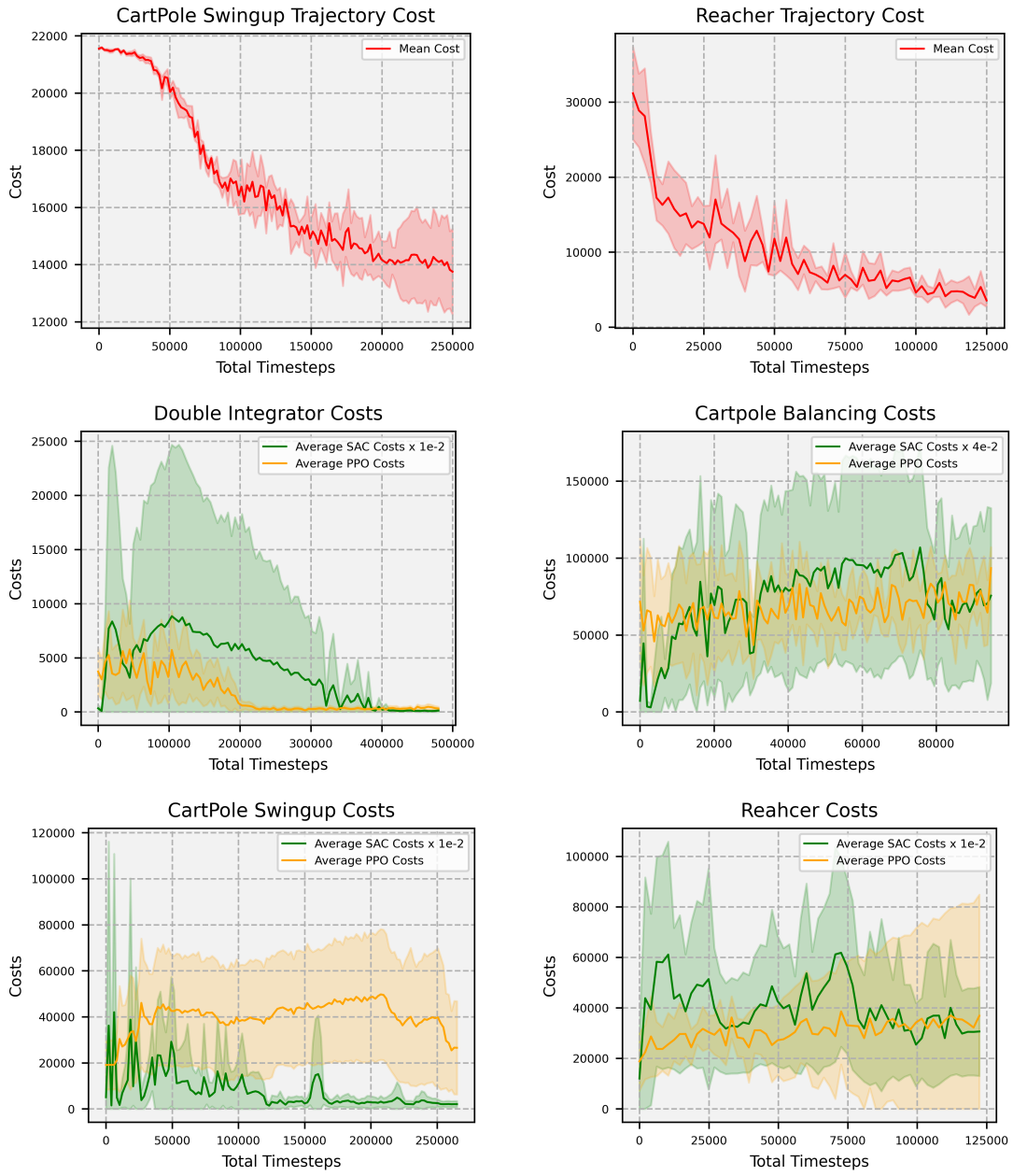


Figure 5.2: Middle and bottom rows: Trajectory cost and SAC/PPO trajectory cost for value function constraints, scaled for visualisation.

Environment	Constraint Loss	Trajectory Cost			Cost Improvement		Horizon
		Ours	PPO	SAC	PPO	SAC	
Reacher	208.50 ± 161.97	3560.60 ± 862.69	$15514.49, \pm 4465.52$	3831267.56 ± 471973.23	4.36	1076.02	170
Swing up	1060.00 ± 307.74	13749.96 ± 1504.31	33147.12 ± 16041.53	$255854.68, \pm 81662.83$	2.41	18.60	171
Balancing	108.06 ± 43.95	10615.60 ± 2712.72	93109.16 ± 15629.57	$2360774.63, \pm 1098978.99$	8.77	22.12	79
Double Integrator	0.43 ± 0.55	356.12 ± 77.63	$359.81, \pm 128.76$	$13311.58, \pm 7681.00$	1.01	3.68	400

Table 5.1: Training statistics and performance comparison against SAC and PPO

Task	$\mathbf{Q}$	$\mathbf{Q}_T$	$\mathbf{R}$
Reacher	$\text{diag}(1, 1, 0, 0)$	$\text{diag}(100, 100, 1, 1)$	0.25
Swingup	$\text{diag}(0, 0, 0, 0)$	$\text{diag}(80, 600, 0.8, 4.5)$	2
Balancing	$\text{diag}(0, 25, 0.5, 0.1)$	$\text{diag}(0, 25, 0.5, 0.1)$	1.1
Double Integrator	$\text{diag}(10, 0.1)$	$\text{diag}(10, 0.1)$	1

Table 5.2: Task parameters

5.4.1 Comparison to model-free RL

This section compares our method to SAC and PPO on simple linear and nonlinear control affine tasks. To keep the comparison fair, we keep the environment settings identical between methods. Therefore, the episode length, cost function dynamics, and the total number of timesteps are all identical. It is important to mention that since our algorithm requires differentiability, we implement our own dynamics. This causes the parameters of our environments to be entirely different from the classic continuous control tasks implemented by OpenAI Gym Brockman et al. (2016b). In addition to the difference in dynamics, we do not perform any early termination of episodes, dampen dynamics via high friction, or constrain the search space by limiting controls, strategies which are used in the OpenAI Gym environments. A more detailed explanation of typical strategies in such environments is shown in Layeghi et al. (2023). The choice of solver settings and parameters was made experimentally. For example, discretisation time step Δt was chosen to balance accuracy with speed and episode length T was chosen to be long enough to accomplish the task inside the horizon. To limit cost function engineering, we use simple parsimonious quadratic costs in the forms of $\mathbf{x}_t^\top \mathbf{Q} \mathbf{x}_t$ for running costs and $\mathbf{x}_T^\top \mathbf{Q}_T \mathbf{x}_T$ for terminal costs, with $\mathbf{Q}, \mathbf{Q}_T \in \mathbb{R}^{n \times n}$. Additionally, we use quadratic regularisation for control, $\mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t$ where $\mathbf{R} \in \mathbb{R}^{m \times m}$.

The results demonstrate that our approach significantly outperforms PPO and SAC, achieving best-case improvements by factors of up to 8 and 1076.02 and worst-case improvements of 1.01 and 3.68, respectively. The details of our results can be found in table 5.1 and figure **??**. Experiments are conducted across a range of initial conditions $X_0 \in \mathbb{R}^{K \times n}$. The total number of timesteps is calculated as $K \times \text{EPOCH} \times N$, with parameters consistent across different solvers. The tasks are governed by a quadratic objective, detailed in table 5.2. We employ Euler integration and, therefore, constant discretisation timestep $\Delta t = 0.01$, which is the same across all tasks and methods. For consistency, we maintain identical objectives across all solvers. The hyperparameters for SAC and PPO are adopted directly from their respective foundational papers, where they were demonstrated to perform well across a broad spectrum of continuous control tasksHaarnoja et al. (2018); Schulman et al. (2017).

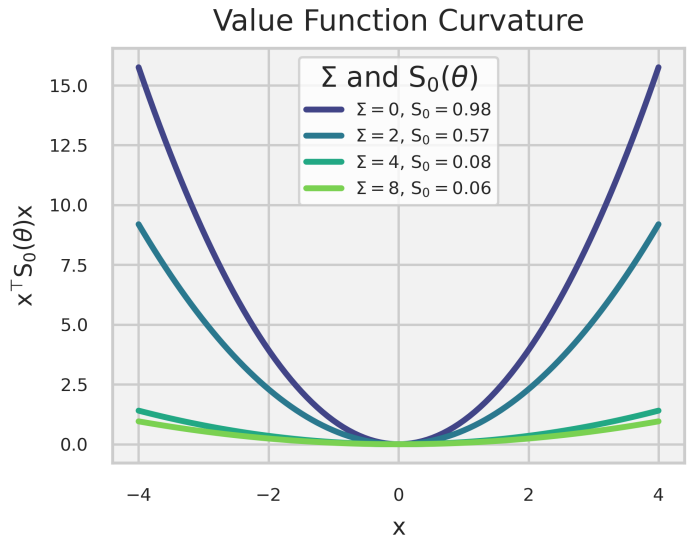


Figure 5.3: Impact of noise on value function curvature.

5.4.2 Effects of smoothing

In the introduction, we mentioned that 0th-order gradient estimates such as REINFORCE provide smoothing effects that help to deal with discontinuous objectives. We hypothesised that SOC allows us to achieve this effect while maintaining the efficiency of first-order gradients. In this section, we provide empirical results to support our hypothesis.

Linear Quadratic case

The Linear Quadratic (LQ) case describes a setting where dynamics are linear, and the cost function is quadratic in $\mathbf{x}$ and $\mathbf{u}$. Under this assumption, it is known that the value function is also quadratic where $v(\mathbf{x}_t, t) = \mathbf{x}_t^\top \mathbf{S}_t \mathbf{x}_t$ with $\mathbf{S}_t > 0$. As a result, in the LQ case under the dynamics $\dot{\mathbf{x}} = \mathbf{A}_t\mathbf{x}_t + \mathbf{G}_t\mathbf{x}_t$ and the control cost $\ell(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_t^\top \mathbf{Q}\mathbf{x}_t + \frac{1}{2}\mathbf{u}_t^\top \mathbf{R}\mathbf{u}_t$, the optimal control law becomes $2\mathbf{R}^{-1}\mathbf{G}_t^\top \mathbf{x}_t^\top \mathbf{S}_t$. Crucially, in this setting, learning the value function means we directly parametrise the curvature since $\nabla^2_{\mathbf{x}_t} v(\mathbf{x}_t, t) = \mathbf{S}_t(\theta)$.

Our test is performed on a single integrator with state $\mathbf{x} \in \mathbb{R}^2$ and control $\mathbf{u} \in \mathbb{R}^1$. We parameterise the value function with a single scalar where $v(\mathbf{x}_t, t; \theta) = \mathbf{x}_t^\top \mathbf{S}_t(\theta)\mathbf{x}_t$ where $\mathbf{S}_t(\theta) \in \mathbb{R}^1$.

Figure 5.3 shows the learnt curvature $\mathbf{S}_{t=0}(\theta)$ under different incremental variance $\mathbf{\Sigma}$ of the Wiener process. The results show that our method learns lower curvature value functions under higher noise. This is consistent with our interpretation of the curvature term in the continuous HJB case shown in equation (5.5)

Navigation task

This task evaluates our approach's effectiveness in dealing with discontinuous objectives in both dynamics and costs. We formulate our problem as a double integrator navigating amongst obstacles. The states of the robot and obstacles: $\mathbf{x} = [\mathbf{q}_{\text{rob}}, \mathbf{q}_{\text{obs}}, \dot{\mathbf{q}}_{\text{rob}}, \dot{\mathbf{q}}_{\text{obs}}]$ where $\mathbf{x} \in \mathbb{R}^{2+14}$ and the control $\mathbf{u} \in \mathbb{R}^2$ corresponding to accelerating in planar Cartesian coordinates. Our dynamics imply that contact leads to halting by nullifying the control authority matrix $\mathbf{C}(\mathbf{x}_t)$ based on the distance limit d to obstacles. This is presented in equation (5.13)

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}(\mathbf{x}_t) + \mathbf{B}(\mathbf{x}_t)\mathbf{u}_t \\ \mathbf{B} &= \begin{pmatrix} \mathbf{C}(\mathbf{x}_t) \\ \mathbf{O} \end{pmatrix} \\ \mathbf{C} &= \lambda \times I_{2\times 2} \in \mathbb{R}^{2\times 2} \\ \mathbf{O} &= 0_{14\times 2} \in \mathbb{R}^{14\times 2} \\ \lambda &= 1 - \mathbb{1}\left(\|\mathbf{q}_{\text{rob}} - \mathbf{q}_{\text{obs}}\| < d\right) \end{aligned} \tag{5.13}$$

In addition to discontinuous dynamics, we introduce discontinuous on non-differentiable cost function, with quadratic penalisation on the state ($\mathbf{x}_t^\top \mathbf{Q} \mathbf{x}_t$ and $\mathbf{x}_T^\top \mathbf{Q}_T \mathbf{x}_T$) and distance constraints on robot state, ($\|\mathbf{q}_{\text{rob}} - \mathbf{q}_{\text{obs}}\| > d$) penalised by $\max(\|\mathbf{q}_{\text{rob}} - \mathbf{q}_{\text{obs}}\| - d, 0)$. Our results show that learning over noisier dynamics reduces cost by avoiding contact with obstacles. In the best case, the process with incremental variance $\mathbf{\Sigma} = 8$ outperforms the deterministic case by approximately a factor of 5. This is quantitatively shown in figure 5.5 and quantitatively in figure 5.4.

5.4.3 Generalisation

Part of our motivation was inspired by optimising in function space, much like RL. This is because sufficiently good approximations of value functions should be generalisable in novel settings. We test this hypothesis in this segment through two experiments on a continuum robot. First, we learn the value function for trajectory tracking while avoiding randomly placed obstacles. We then test generalisation on two fronts: first, in simulation, we use the learnt value function as the terminal cost for a 1-step MPPI controller. And second, by evaluating the quality of trajectory performance on real hardware under the same MPPI controller. Here, we briefly describe the learnt model of our continuum robot. The continuum robot shown in figure 5.7 consists of a central flexible backbone with compression springs and three parallel tendons that can bend the backbone. The movement of the robot is facilitated by a screw joint for prismatic motion, along with antagonistic motion of tendons enabling bending and twisting along the length. As a result of the compression and the tendon-based structure, the differential kinematics of this robot are highly nonlinear and difficult to model

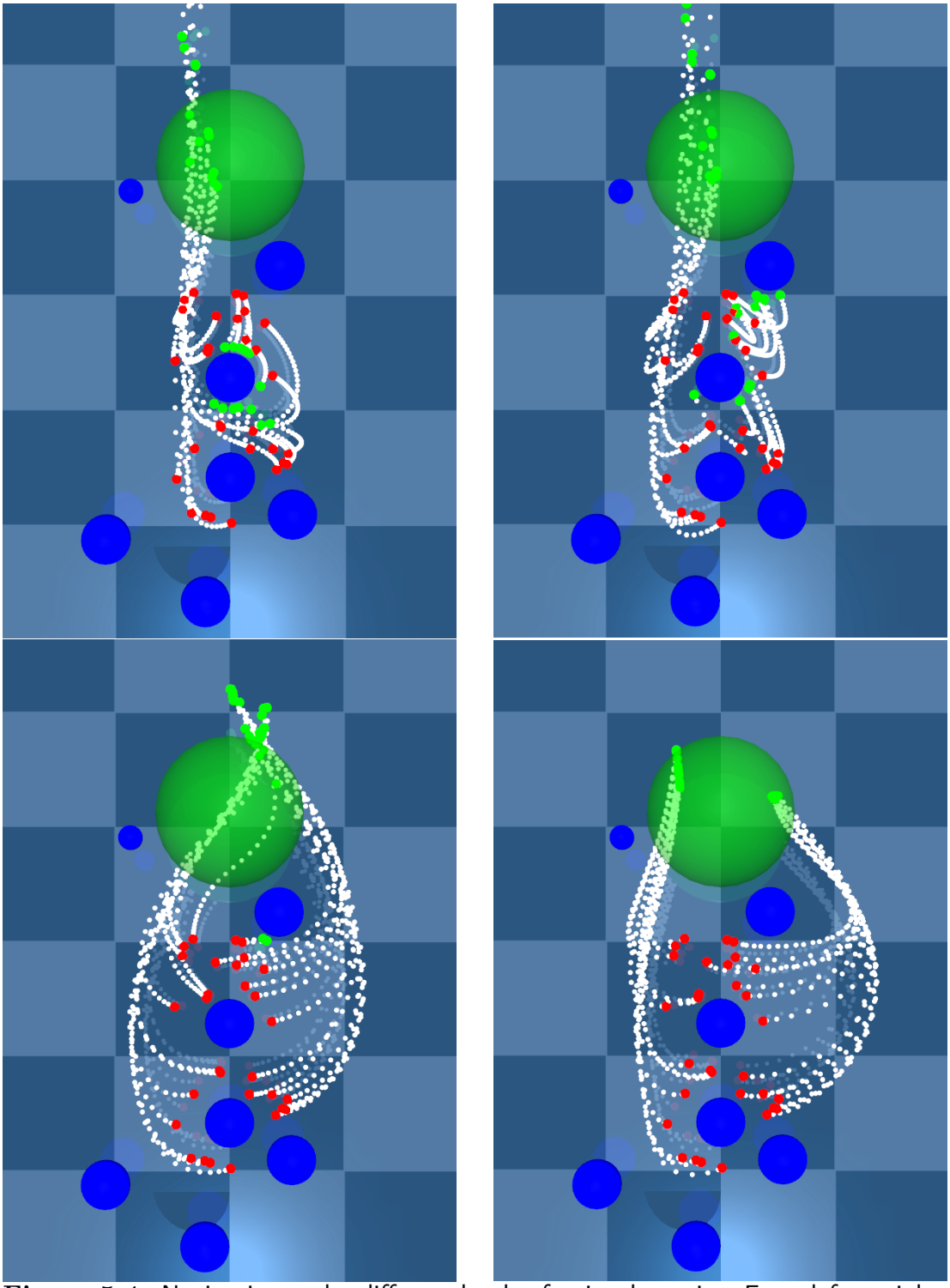


Figure 5.4: Navigation under different levels of noisy dynamics. From left to right, top to bottom: $\mathbf{\Sigma} = 0$, $\mathbf{\Sigma} = 2$, $\mathbf{\Sigma} = 4$, $\mathbf{\Sigma} = 8$. The large green area is the goal location, the green dots are the starting point of the trajectories, and the red dots are the final points in the trajectories.

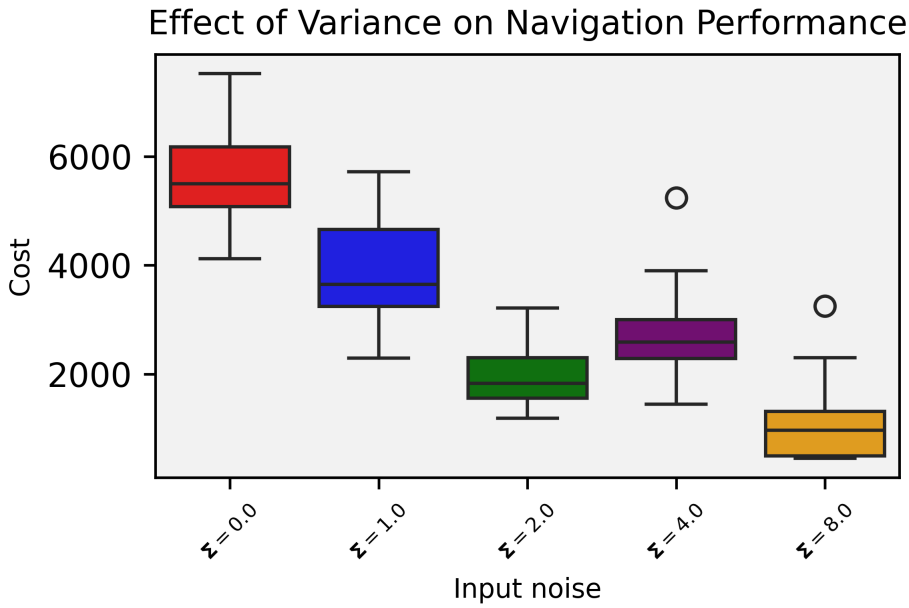


Figure 5.5: Cost vs noisy policy.

analytically. As a result, the differential kinematics of this robot is modelled via a neural network.

The robot's state consists of tendon length and the end-effector's position. and the full state of the dynamics with obstacles is defined as $\mathbf{x} = [\mathbf{q}_{\text{tip}}, \mathbf{q}_{\text{tend}}, \mathbf{q}_{\text{obs}}]$ where $\mathbf{x} \in \mathbb{R}^9$. The robot is velocity controlled where $\mathbf{u} = \dot{\mathbf{q}}_{\text{tend}}, \mathbf{u} \in \mathbb{R}^3$.The final differential kinematics of the robot is written as

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{B}(\mathbf{x}_t)\mathbf{u}_t \\ \mathbf{B} &= \begin{pmatrix} \mathbf{J}^\top(\mathbf{q}_{t_{\text{tend}}}; \Phi) \\ \mathbf{I} \\ \mathbf{O} \end{pmatrix} \\ \mathbf{I} &= \mathbb{I}_{3\times 3} \in \mathbb{R}^{3\times 3} \\ \mathbf{O} &= 0_{3\times 3} \in \mathbb{R}^{3\times 3} \end{aligned} \tag{5.14}$$

where $\mathbf{J}(\mathbf{q}_{t_{\text{tend}}}; \Phi)$ is the neural network representation of the jacobian of the robot.

Figure 5.6 illustrates the helix trajectory tracking of simulated dynamics. This is demonstrated under two conditions: a 1-step horizon MPPI with a learnt value function and MPPI over 1 and 15 steps without the value function. As shown in table 5.3, using the value function in the 1-step case results in 5.8 times better average performance. Additionally, the 15-step horizon case without the value function still underperforms by approximately a factor of 2 at a 15 times longer planning horizon.

Finally, as part of our generalisation experiments, we leverage the same value function and the 1-step MPPI controller on a trajectory tracking task on real hardware. The MPPI controller achieved a trajectory tracking error of 0.0018m on the helix shown in figure 5.7.

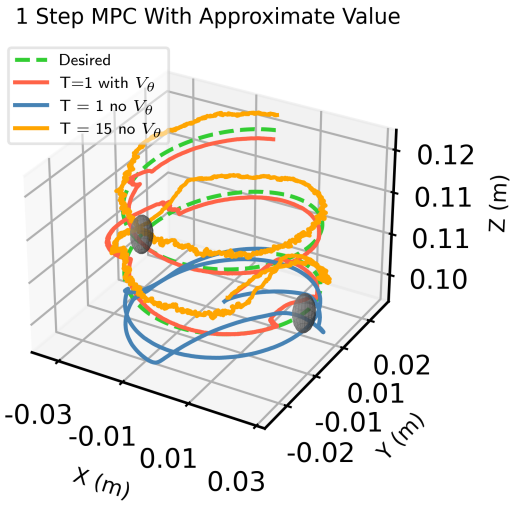


Figure 5.6: Effect of value function on trajectory tracking.

Configuration	With value function v_θ	Without value function v_θ	
Horizon (T)	T=1	T=1	T=15
MSE (m)	0.0023	0.0134	0.0048

Table 5.3: Tracking MSE error

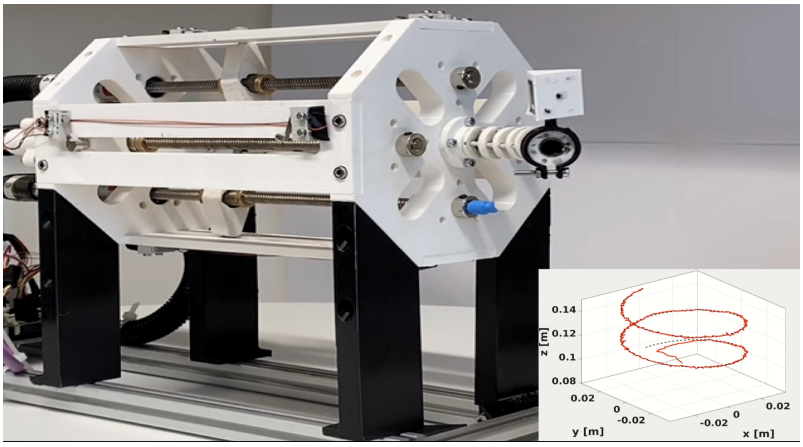


Figure 5.7: Real hardware experiment setup and trajectory tracking.

5.5 Discussion

In this section, we will expand on the results and their implications.

5.5.1 Convergence

Our algorithm leveraged differentiable dynamics, first-order gradients and analytically derived policies that encode inertial/differential effects. Our comparison to model-free RL confirms that gradients enjoy an improved convergence rate compared to model-free cases. However, our results are obtained on continuous dynamics where biased first-order gradients are not pathological.

Our method discretises the time axis to obtain a penalty on the rate of change of the value function. The choice of the discretisation timestep is crucial to the accuracy of this approximation, as in the limits, we recover the true constraint. However, finer approximations increase computation costs. This leads to one of the general challenges of our approach that is, as shown in 5.1, we are not able to completely minimise the penalty in 5.12, leading to residual constraint loss. The broader impact of this margin will be present in the policy since small approximation errors in the value function may lead to a disproportionate loss in task performance. Finally, the task cost defines the penalty on the rate of change of the value function. Poor selection of this metric can cause invalid value functions. Therefore, credit assignment remains crucial to final task performance.

We make a final remark on the performance of SAC and PPO. Classical control problems have been widely studied in model-free RL. However, further inspection has shown that such studies have been conducted in strict settings. For example, the classic CartPole-v1 has discrete action space with region operation between $\pm 12°$. The MuJoCo InvertedPendulum-v4 environment uses a continuous action space with an even smaller state space for learning at $11.45°$. These termination conditions avoid nonlinear regimes and simplify the algorithm's learning process. In addition to termination conditions, our reacher model also differs from the one OpenAI used in the Reacher-v4 environment. The Reacher-v4 environment uses very high values of armature coefficients, which nullify the nonlinearities and coriolis effects between joints, once again simplifying the problem.

5.5.2 Smoothing

In this section, we assess the impact of noise on smoothing the value function through two experimental setups. According to equations (5.7) and (5.3), as well as Ito's theorem, a notable difference from deterministic cases is the inclusion of the second-order derivative or the curvature of the value function with respect to the state $\mathbf{x}$. In our case, where we aim to learn the value function under a constant variance by minimising the program (5.2), the optimiser is encouraged to choose lower curvature values under noisier dynamics. In other words, the higher the incremental variance of the wiener process, the lower or smoother the curvature of the approximate value function. The direct empirical evidence for this interpretation is shown in the LQG experiment, where we observe an inverse relationship between variance $\boldsymbol{\Sigma}$ and $\nabla^2_{\mathbf{x}} v(\mathbf{x}, t)$.

The effect of smoothing is also observed in the navigation experiment. In this

context, smoothing enhances both robustness and exploration. The experimental results show that values learnt under larger $\mathbf{\Sigma}$ result in lower trajectory costs. This can be attributed to larger variance inducing broader representations of obstacles in the value space, pushing trajectories away from obstacles and increasing their margins. As a result, value functions learnt under noisier dynamics are more robust. Figure 5.4 illustrates this effect. This phenomenon also somewhat supports exploration, as smoothing out many local minima and merging them into fewer, broader minima results in paths that systematically avoid obstacles.

5.5.3 Generalisation

The final part of our empirical analysis concentrates on generalisation. We aimed to develop an OC-theoretic method using value function approximation to sidestep the need for re-optimisation typically required by trajectory optimisers. Our evaluation of this approximation's generalisation focused on its adaptability to new trajectories and its effectiveness under real-world dynamics. We demonstrated this by training the value function on point-to-point tasks with randomly placed obstacles and testing it on an unseen helical trajectory disrupted by random obstacles. The results confirmed that our method effectively learns a generalisable encoding of the cost-to-go for new trajectories. Further, we assessed the value function's performance on actual dynamics, where our value function based on approximate learnt dynamics achieved low tracking error on real hardware.

5.6 Conclusion

We aimed to introduce an OC-theoretic approach that achieves the benefits of RL, such as generalisation and smoothing of discontinuous objectives while maintaining fast convergence of trajectory optimisers. We formulated the problem through the lens of stochastic OC and derived a value learning approach based on the discredited Bellman backup and differentiable dynamics. We showed that the Ito-based differential representation of this backup provides a clear interpretation of the effect of noise on smoothing by regularising curvature. Compared to model-free RL, our experiments showed that we could greatly outperform in task cost at a much lower number of timesteps. We explained this outperformance through the use of first-order gradients and analytical policies. We also explored the effects of noise on smoothing, first, on a simple LQG problem and then on a navigation task with discontinuous dynamics and costs. Our results demonstrated the impact of noise on smoothing and its connection to robustness and exploration. We demonstrated that noisier dynamics lead to better-performing value function representations. Finally, we evaluated the generalisation of our value function approximation for reaching and avoiding tasks on a continuum robot with approximate model and discontinuous costs. In our experiments, we used an MPPI controller to greedily plan using the value function learnt on point-to-

point tasks. Our results showed that the value function could generalise to novel trajectories in simulation and to real hardware.

We believe our approach effectively combines the strengths of both RL and trajectory optimisers. Our results show faster convergence and the capability to manage non-differentiable objectives in simple yet challenging tasks that evaluate our contributions. As more advanced, extensible, differentiable simulators become available, we plan to expand our research to assess our method's performance on high-dimensional, contact-rich tasks.

Chapter 6

Conclusion and future work

Recent progress in technology has been converging towards the notion of autonomy. With the advent of autonomous vehicles and large language model agents, it is apparent that autonomous systems will likely shape the future of our world. Breakthrough advances inspire new applications, and in this thesis, we focused on RL as such inspiration for controlling dynamical systems like robots. We argued that optimal control-theoretic approaches should not be discarded; in fact, we should aim to combine the merits of both methods. The recent breakthrough of RL showed us that the paradigm can solve grand challenges like the game of Go, a task with a discrete but large state space and a concrete yet sparse objective definition: winning the game. In essence, RL can remove the objective design by smoothing sparse objectives and solving for an approximate global optimal policy. A direct transfer of this approach to continuous systems is possible but costly. In section 13, we analysed the causal factor of this phenomenon: Policy Gradients. Our analysis revealed that although RL enables such flexibility, we pay for it by giving up fast and stable convergence due to the high variance nature of policy gradients. This variance becomes a separate burden of hyperparameter tuning on the user. On the other hand, OC-theoretic approaches can provide stable, fast converging solutions, albeit given a smooth differentiable objective function. Our goal in this thesis was to formalise methods capable of providing flexibility in objective definition to users while efficiently learning approximate global policies.

Our primary components to achieving this goal can be divided into three core ideas:

- Leverage differentiability and the derivative operation for efficiency. We have good approximations for dynamics models, and in nearly all simulators, the derivative operation is valid even if the dynamics are piece-wise differentiable (e.g. contact dynamics)

- Parametrise function spaces instead of local trajectories. Function approximation enables generalisation. Our choices for this approximation of optimal decision can be with respect to the policy space and/or the value space. In

this thesis, we leverage the value function as the optimal policy, which, for many interesting dynamical systems, is the gradient of the value function.

- Use noise, specifically the stochastic formulation of the HJB equation, to handle non-differentiable or discontinuous areas of the objective function. Stochasticity has a direct impact on the approximation of our function. Essentially, the noisier our process, the smoother or lower the curvature of our approximation. Therefore, we can use noise to regularise the sharpness of our learnt objective landscape and smooth the approximations of the discontinuous regions. In turn, stochasticity provides us with a natural interpretation of robustness as a consequence of smoothing.

In line with the above ideas, our *first contribution* introduced a local method capable of combining the flexibility of cost function design from sampling with the efficiency of derivative-based methods, namely iLQR. Our approach was able to complete manipulation and navigation tasks with discontinuous dynamics and sparse costs. Intuitively, our method could sample when there is no derivative information and follow optimised trajectories when derivatives arise. Empirically, we outperformed the purely derivative-based method iLQR and the sampling-based method MPPI.

Our *second contribution* focused on generalisation while maintaining efficiency. We introduced a method based on the deterministic HJB equation and leveraged differentiable simulation. With this method, we were able to define mathematical programs that could learn an approximate optimal value and Lyapunov functions. Our results showed that model-free RL methods suffer from non-convergence and require environment engineering. In contrast, our approach was able to converge with simple but parsimonious cost functions. Our results showed that in the best case, we are able to outperform widely used RL algorithms SAC and PPO in task cost by 4 and 2 orders of magnitude.

Our *Third and final contribution* brought together the benefits of function approximation, differentiability and smoothing from stochasticity in one framework. We leverage the stochastic HJB equation and derive an instantaneous penalty to learn a temporal and spatial approximation of the optimal value function. Our results show that similar to the previous contribution under continuous dynamics, we are able to outperform SAC and PPO by factors up to 1076.02 and 8, respectively. Additionally, we explore the effects of process noise and provide empirical results that support our smoothing hypothesis. Under a noisier process, our method is able to outperform the deterministic formulation in tasks with discontinuous objectives and dynamics. This smoothing also improves robustness in navigation tasks where contacts are costly. Finally, in line with our goal, we also show that encoding this value function approximation also allows us to reduce the search horizon of our local methods in our tests, specifically by a factor of 15.

In the future, we plan to test the machinery described in this thesis in much larger environments and assess its applicability to complex real-world settings.

The key to achieving this goal is having access to massively parallel, extensible, and differentiable simulators. Recent progress in parallel simulation has sparked increasing interest in differentiability. These simulators are still in the early stages of development, presenting an ideal opportunity to advance and work towards our goal. By using the methods outlined in this thesis, our aim is to make significant advancements in testing and refining our proposals across a wider range of tasks in the future.

References

Bernardo Aceituno-Cabezas, Carlos Mastalli, Hongkai Dai, Michele Focchi, Andreea Radulescu, Darwin G Caldwell, José Cappelletto, Juan C Grieco, Gerardo Fernández-López, and Claudio Semini. Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization. *IEEE Robotics and Automation Letters*, 3(3):2531–2538, 2017.

Samuel Ainsworth, Kendall Lowrey, John Thickstun, Zaid Harchaoui, and Siddhartha Srinivasa. Faster policy learning with continuous-time gradients. In *Learning for Dynamics and Control*, pages 1054–1067. PMLR, 2021.

Brandon Amos, Lei Xu, and J. Zico Kolter. Input convex neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 146–155. PMLR, 2017.

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.

Francis Bach. The many faces of integration by parts – ii: Randomized smoothing and score functions. https://francisbach.com/integration-by-parts-randomized-smoothing-score-functions/, September 7 2020. Accessed: 2024-05-27.

Somil Bansal and Claire J Tomlin. Deepreach: A deep learning approach to high-dimensional reachability. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1817–1824. IEEE, 2021.

Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

Albert S Berahas, Liyuan Cao, Krzysztof Choromanski, and Katya Scheinberg. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *Foundations of Computational Mathematics*, 22(2):507–560, 2022.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.

Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.

Dimitri Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

Steven Borowiec. Google's alphago seals 4-1 victory over grandmaster lee sedol. *The Guardian*, March 2016. URL https://www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016a.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016b.

Jan Carius, René Ranftl, Farbod Farshidian, and Marco Hutter. Constrained stochastic optimal control with learned importance sampling: A path integral approach. *The International Journal of Robotics Research*, 41(2):189–209, 2022.

Shawn M. Carter. Boston dynamics' backflipping robot shows off new parkour routine. *New York Post*, aug 2021. URL https://nypost.com/2021/08/18/boston-dynamics-backflipping-robot-shows-off-new-parkour-routine/. Accessed: 2024-05-13.

Oguzhan Cebe, Carlo Tiseo, Guiyang Xin, Hsiu-chin Lin, Joshua Smith, and Michael Mistry. Online dynamic trajectory optimization and control for a quadruped robot. *arXiv preprint arXiv:2008.12687*, 2020.

Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural lyapunov control. *Advances in neural information processing systems*, 32, 2019.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.

Charles Dawson, Zengyi Qin, Sicun Gao, and Chuchu Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. In *Conference on Robot Learning*, pages 1724–1735. PMLR, 2022.

Selim Engin and Volkan Isler. Neural optimal control using learned system dynamics. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 953–960. IEEE, 2023.

Ioannis Exarchos and Evangelos A Theodorou. Stochastic optimal control via forward and backward stochastic differential equations and importance sampling. *Automatica*, 87:159–165, 2018.

Wendell H Fleming and Halil Mete Soner. *Controlled Markov processes and viscosity solutions*, volume 25. Springer Science & Business Media, 2006.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

Mike Giles and Paul Glasserman. Smoking adjoints: Fast monte carlo greeks. *Risk*, 19(1):88–92, 2006.

Paul Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer, New York, NY, USA, 2004.

Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1954.

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.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

David H Jacobson. New second-order and first-order algorithms for determining optimal control: A differential dynamic programming approach. *Journal of Optimization Theory and Applications*, 2:411–440, 1968.

Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. Pontryagin differentiable programming: An end-to-end learning and control framework. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7979–7992. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/5a7b238ba0f6502e5d6be14424b20ded-Paper.pdf.

Hilbert J Kappen. Linear theory for control of nonlinear stochastic systems. *Physical review letters*, 95(20):200201, 2005.

Hilbert J Kappen, Vicenç Gómez, and Manfred Opper. Optimal control as a graphical model inference problem. *Machine learning*, 87(2):159–182, 2012.

Benjamin Katz, Jared Di Carlo, and Sangbae Kim. Mini cheetah: A platform for pushing the limits of dynamic quadruped control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6295–6301, 2019. doi: 10.1109/ICRA.2019.8793865.

Patrick Kidger, James Foster, Xuechen Chen Li, and Terry Lyons. Efficient and accurate gradients for neural sdes. *Advances in Neural Information Processing Systems*, 34:18747–18761, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous robots*, 40:429–455, 2016.

Vikash Kumar, Emanuel Todorov, and Sergey Levine. Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383. IEEE, 2016.

Daniel Layeghi, Steve Tonneau, and Michael Mistry. Optimal control via combined inference and numerical optimization. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3429–3435. IEEE, 2022.

Daniel Layeghi, Steve Tonneau, and Michael Mistry. Neural lyapunov and optimal control. *arXiv preprint arXiv:2305.15244*, 2023.

Quentin Le Lidec, Fabian Schramm, Louis Montaut, Cordelia Schmid, Ivan Laptev, and Justin Carpentier. Leveraging randomized smoothing for optimal control of nonsmooth dynamical systems. *Nonlinear Analysis: Hybrid Systems*, 52:101468, 2024.

He Li, Robert J. Frei, and Patrick M. Wensing. Model hierarchy predictive control of robotic systems. *IEEE Robotics and Automation Letters*, 6(2):3373–3380, 2021. doi: 10.1109/LRA.2021.3061322.

Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction*. Princeton university press, 2011.

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

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.

Michael Lutter, Shie Mannor, Jan Peters, Dieter Fox, and Animesh Garg. Robust value iteration for continuous control tasks. *arXiv preprint arXiv:2105.12189*, 2021a.

Michael Lutter, Shie Mannor, Jan Peters, Dieter Fox, and Animesh Garg. Value iteration in continuous actions, states and time. *arXiv preprint arXiv:2105.04682*, 2021b.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Nicolas Mansard, Andrea DelPrete, Mathieu Geisert, Steve Tonneau, and Olivier Stasse. Using a memory of motion to efficiently warm-start a nonlinear predictive controller. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2986–2993. IEEE, 2018.

Luke Metz, C Daniel Freeman, Samuel S Schoenholz, and Tal Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):1–8, 2012.

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V Todorov. Interactive control of diverse complex characters with neural networks. *Advances in neural information processing systems*, 28, 2015.

Tao Pang, HJ Terry Suh, Lujie Yang, and Russ Tedrake. Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models. *IEEE Transactions on Robotics*, 2023.

Diego Pardo, Michael Neunert, Alexander W Winkler, Ruben Grandia, and Jonas Buchli. Hybrid direct collocation and control in the constraint-consistent subspace for dynamic legged robot locomotion. In *Robotics: Science and Systems*, 2017.

Marcus Pereira, Ziyi Wang, Ioannis Exarchos, and Evangelos A Theodorou. Learning deep stochastic optimal control policies using forward-backward sdes. *arXiv preprint arXiv:1902.03986*, 2019.

Eckhard Platen and Nicola Bruti-Liberati. *Numerical solution of stochastic differential equations with jumps in finance*, volume 64. Springer Science & Business Media, 2010.

Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019.

Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Joose Rajamäki, Kourosh Naderi, Ville Kyrki, and Perttu Hämäläinen. Sampled differential dynamic programming. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1402–1409. IEEE, 2016.

Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2: 253–279, 2019.

Ilya Orson Sandoval, Panagiotis Petsagkourakis, and Ehecatl Antonio del Rio-Chanona. Neural odes as feedback policies for nonlinear optimal control. *arXiv preprint arXiv:2210.11245*, 2022.

Simo Särkkä and Arno Solin. *Applied stochastic differential equations*, volume 10. Cambridge University Press, 2019.

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.

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

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Robert F Stengel. *Stochastic optimal control: theory and application*. John Wiley & Sons, Inc., 1986.

Hyung Ju Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? In *International Conference on Machine Learning*, pages 20668–20696. PMLR, 2022.

Rich Sutton. The bitter lesson, 2019. URL https://www.cs.utexas.edu/~eunsol/courses/data/bitter_lesson.pdf. Accessed: 2024-05-27.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Yuval Tassa and Tom Erez. Least squares solutions of the hjb equation with neural network value-function approximators. *IEEE transactions on neural networks*, 18(4):1031–1041, 2007.

Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Russ Tedrake. *Underactuated Robotics*. 2023. URL https://underactuated.csail.mit.edu.

Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *The Journal of Machine Learning Research*, 11:3137–3181, 2010.

Evangelos A Theodorou. Nonlinear stochastic control and information theoretic dualities: Connections, interdependencies and thermodynamic interpretations. *Entropy*, 17(5):3352–3375, 2015.

Evangelos A Theodorou and Emanuel Todorov. Relative entropy and free energy dualities: Connections to path integral and kl control. In *2012 ieee 51st ieee conference on decision and control (cdc)*, pages 1466–1473. IEEE, 2012.

E. Todorov and W. Li. Optimal control methods suitable for biomechanical systems. In *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE Cat. No.03CH37439)*, volume 2, pages 1758–1761 Vol.2, 2003. doi: 10.1109/IEMBS.2003.1279748.

Emanuel Todorov. Linearly-solvable markov decision problems. In *Advances in neural information processing systems*, pages 1369–1376, 2007.

Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE, 2005.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

Steve Tonneau, Daeun Song, Pierre Fernbach, Nicolas Mansard, Michel Taïx, and Andrea Del Prete. Sl1m: Sparse l1-norm minimization for contact planning on uneven terrain. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6604–6610. IEEE, 2020.

Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual international conference on machine learning*, pages 1049–1056, 2009.

Marc Toussaint, Jung-Su Ha, and Danny Driess. Describing physics for physical reasoning: Force-based sequential manipulation planning. *IEEE Robotics and Automation Letters*, 5(4):6209–6216, 2020.

Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019.

Julian Viereck, Avadesh Meduri, and Ludovic Righetti. Valuenetqp: Learned one-step optimal control for legged locomotion. In Roya Firoozi, Negar Mehr, Esen Yel, Rika Antonova, Jeannette Bohg, Mac Schwager, and Mykel Kochenderfer, editors, *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, volume 168 of *Proceedings of Machine Learning Research*, pages 931–942. PMLR, 23–24 Jun 2022. URL https://proceedings.mlr.press/v168/viereck22a.html.

Jiayi Wang, Teguh Santoso Lembono, Sanghyun Kim, Sylvain Calinon, Sethu Vijayakumar, and Steve Tonneau. Learning to guide online multi-contact receding horizon planning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12942–12949, 2022. doi: 10.1109/IROS47612.2022.9981234.

Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721. IEEE, 2017.

Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Information-theoretic model predictive control: Theory and applications to autonomous driving. *IEEE Transactions on Robotics*, 34(6): 1603–1622, 2018.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Alexander W Winkler, C Dario Bellicoso, Marco Hutter, and Jonas Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018.

Wei Xiao, Ramin Hasani, Xiao Li, and Daniela Rus. BarrierNet: A Safety-Guaranteed Layer for Neural Networks. *arXiv e-prints*, art. arXiv:2111.11277, November 2021. doi: 10.48550/arXiv.2111.11277.

Wei Xiao, Tsun-Hsuan Wang, Ramin Hasani, Makram Chahine, Alexander Amini, Xiao Li, and Daniela Rus. Barriernet: Differentiable control barrier functions

for learning of safe robot control. *IEEE Transactions on Robotics*, pages 1–19, 2023. doi: 10.1109/TRO.2023.3249564.

Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. *arXiv preprint arXiv:2204.07137*, 2022.

Xuanxi Zhang, Jihao Long, Wei Hu, Weinan E, and Jiequn Han. Initial value problem enhanced sampling for closed-loop optimal control design with deep neural networks, 2023. URL https://openreview.net/forum?id=oXM5kdnAUNq.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryxmb1rKDS.