A supervised formulation of Reinforcement Learning - with SuperLinear Convergence

Amit Parag

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Institut National des Sciences Appliquées de Toulouse

---

Présentée et soutenue par

Amit PARAG

Le 20 avril 2023

Une formulation supervisée de l'apprentissage par renforcement avec convergence superlinéaire

---

Ecole doctorale : **SYSTEMES**

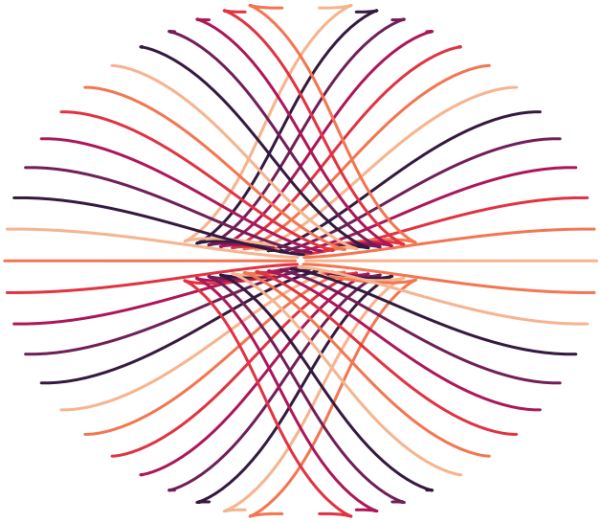
Spécialité : **Robotique et Informatique**

Unité de recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
Nicolas MANSARD

Jury

M. Manuel LOPES, Rapporteur
M. Vincent PADOIS, Rapporteur
M. Olivier STASSE, Examinateur
Mme Georgia CHALVATZAKI, Examinatrice
M. Olivier SIGAUD, Examinateur
M. Nicolas MANSARD, Directeur de thèse



A Supervised Formulation of Reinforcement Learning - toward Superlinear Convergence Properties

---

Amit Parag

May 24, 2023

Version: 1.0

Doctoral Thesis

A Supervised Formulation of Reinforcement Learning - toward Superlinear Convergence Properties

Amit Parag

Reviewer Vincent Padois
Inria Bordeaux Sud-Ouest

Reviewer Manuel Lopes
Instituto Superior Tecnico

Examiner Olivier Stasse
Laboratoire d'analyse et d'architecture des systèmes
LAAS-CNRS

Examiner Georgia Chalvatzaki
Intelligent Autonomous Systems
Technische Universitaet Darmstadt

Examiner Olivier Sigaud
Institut des systèmes intelligents et de robotique
Sorbonne Universite

Advisor Nicolas Mansard
Laboratoire d'analyse et d'architecture des systèmes
LAAS-CNRS

Amit Parag

A Supervised Formulation of Reinforcement Learning - toward Superlinear Convergence Properties

Doctoral Thesis, May 24, 2023

Reviewers: Vincent Padois and Manuel Lopes

Examiners: Olivier Stasse Georgia Chalvatzaki and Olivier Sigaud

Supervisors: Nicolas Mansard

Laboratoire d'analyse et d'architecture des systèmes

Gepetto Team

Avenue du Colonel Roche

31400 and Toulouse

Abstract

Deep reinforcement learning uses simulators as abstract oracles to interact with the environment. In continuous domains of multi-body robotic systems, differentiable simulators have recently been proposed, still, they are yet underutilized, even though we have the knowledge to make them produce richer information. This problem when juxtaposed with the usually high computational cost of exploration-exploitation in high dimensional state space can quickly render reinforcement learning algorithms less effective. In this thesis, we propose to combine learning and simulator-based optimization such that the quality of both increases while the need to exhaustively search the state space decreases. We propose to learn value function and state, and control trajectories through locally optimal runs of a trajectory optimizer. The learned value function, along with estimates of optimal state and control policies, is subsequently used in the trajectory optimizer: the value function estimate serves as a proxy for shortening the preview horizon, while the state and control approximations serve as a guide in policy search for our trajectory optimizer. The proposed approach demonstrates a better symbiotic relation, with superlinear convergence, between learning and simulators, that we need for end-to-end learning of complex poly articulated systems.

Résumé

L'apprentissage profond par renforcement utilise des simulateurs comme oracles abstraits pour interagir avec l'environnement. Dans les domaines continus des systèmes robotiques multi-corps, des simulateurs différentiables ont récemment été proposés mais sont encore sous-utilisés, même si nous avons les connaissances

nécessaires pour leur faire produire des informations plus riches. Ce problème, lorsqu'il est juxtaposé au coût de calcul élevé de l'exploration-exploitation dans un espace d'état de haute dimension, peut rapidement rendre les algorithmes d'apprentissage par renforcement impraticables. Dans cette these, nous proposons de combiner l'apprentissage et les simulateurs de sorte que la qualité des deux augmente, tandis que la nécessité d'explorer exhaustivement l'espace d'état diminue. Nous proposons d'apprendre la fonction de valeur, l'état et les trajectoires d'etat et de contrôle à travers les exécutions localement optimales de l'optimiseur de trajectoire. La fonction d'valeur apprise, ainsi qu'une estimation des politiques optimales d'état et de contrôle, est ensuite utilisée dans l'optimiseur de trajectoire l'estimation de la fonction d'valeur sert de proxy pour raccourcir l'horizon de prévision, tandis que les approximations d'état et de contrôle servent de guide dans la recherche de politiques pour notre optimiseur de trajectoire. L'approche proposée démontre une meilleure relation symbiotique, avec une convergence super linéaire, entre l'apprentissage et les simulateurs, dont nous avons besoin pour l'apprentissage de bout en bout de systèmes polyarticulés complexes.

Acknowledgement

Hogwarts, Hogwarts
Hoggy Warty Hogwarts
Teach us something please
Whether we be old and bald
Or young with scabby knees
Our heads could do with filling
With some interesting stuff
For now they're bare and full of air
Dead flies and bits of fluff
So teach us things worth knowing
Bring back what we've forgot
Just do your best, we'll do the rest
And learn until our brains all rot

Contents

1 *It starts with ...* 1

1.1 Motivation . . . 2

1.2 Contributions . . . 3

1.3 Thesis Structure . . . 5

1.3.1 Associated Publications . . . 6

2 Optimal Control and Reinforcement Learning - a discussion of motion generation for robots 9

2.1 Notations . . . 10

2.2 Generalized Optimal Control - Problem Formulation . . . 11

2.2.1 Solutions . . . 13

2.3 Dynamic Programming and Markov Decision Process . . . 14

2.3.1 An overview of Reinforcement Learning in robotics . . . 16

2.4 Direct Methods - Transcription and Resolution . . . 19

2.4.1 Transcription . . . 19

2.4.2 Trajectory Optimization - A literature review . . . 22

2.4.3 Model Predictive Control . . . 23

2.5 Hybrid Methods . . . 25

2.6 Discussion . . . 27

I Theory 29

3 Differential Policy Value Programming - ∂PVP 31

3.1 Differential Dynamic Programming . . . 32

3.1.1 Quadratic Approximation . . . 34

3.1.2 Line Search . . . 36
3.1.3 Complexity and Regularization . . . 36
3.1.4 Discussion . . . 37
3.2 Differential Value Programming - *DVP* . . . 39
3.2.1 Algorithmic Principles . . . 39
3.2.2 Algorithm . . . 41
3.2.3 Architecture of V_α . . . 43
3.3 DVP with Sobolev Descent - ΔDVP . . . 44
3.3.1 Sobolev Regression . . . 45
3.3.2 Sampling the State Space . . . 47
3.3.3 Conclusion . . . 52
3.4 Differential Policy Value Programming - ∂PVP . . . 53
3.4.1 Design of V_α, X_β, U_γ . . . 54
3.4.2 Discussion . . . 56
3.5 Conclusion . . . 58

II Experimental Evaluations **59**

4 Experimental Setup **61**
4.1 Unicycle . . . 61
4.2 Cart-Pole . . . 63
4.3 Inverted Pendulum . . . 64
4.4 7 *dof* Manipulator Arm . . . 65
4.5 Discussion . . . 66

5 ∂PVP$_v$ - Estimating global value function **67**
5.1 Problem Statement and Experimental Setup . . . 68
5.2 Estimates of Value Function - Classic Control . . . 69
5.2.1 Overall Convergence . . . 69
5.2.2 Influence of horizon length . . . 72
5.2.3 Robust Convergence . . . 74
5.2.4 Convergence Issues: Singularities . . . 75

5.2.5 Importance of Sobolev Loss 78
5.2.6 Conclusion 81
5.3 Comparison with PPO 82
5.4 Application to the 7 *dof* Manipulator Arm 85
5.4.1 Results 85
5.5 Discussion and Conclusion 87

6 ∂PVP$_{x,u}$ - Learning Warmstarts **93**
6.1 Problem statement and Experimental Setup 94
6.1.1 Introduction 94
6.1.2 Experimental objectives and setup outline 95
6.2 Results 96
6.3 Discussion and Conclusion 97

7 *Conclusion* **105**

8 Appendix **109**
8.1 Q-Function 109
8.2 The Loco3D project : *Crocoddyl* - *C*ontact *Ro*bot *Co*ntrol by *D*ifferential *Dy*namic Programming *L*ibrary 110
8.2.1 Features of Crocoddyl 111

Bibliography **115**

Declaration **135**

1 It starts with ...

> *I just sit at a typewriter and curse a bit.*
>
> **— P G Wodehouse**

The theory of Reinforcement Learning (RL) provides a normative method of animal behavior based on psychological and neuro-scientific learning mechanisms [SB18; Wat89; Bar97]. These methods then define how an agent can optimize and achieve optimal control of its environment. The resulting control is optimal with respect to an objective function - the objective function is either written positively as a reward to maximize or negatively as a cost to minimize and is usually combined with constraints on state and control that characterize behavior. The key characteristics of RL algorithms - online adaptability, self-learning of features, and sequential decision-making under uncertainties - often underpin their successful deployment in robotics. RL has been successfully used to rotate valves with multi-fingered robotic hands [Zhu+19], construct dynamic locomotion policies for legged robots [Xie+19], and learn vision-based dynamic manipulation skills with scalable self-supervised vision-based RL frameworks [Kal+18].

However, the brittleness of the RL agent restricts its applicability to domains where useful features can be handcrafted or to domains with fully observed, low-dimensional state space. In continuous high dimensional state space of robotics, the RL agent may either altogether fail to converge to the optimal solution due to *curse of dimensionality* [MA00] or requires clever sampling solutions [Str00; KKR19]. The overall goal is to learn the solution of an optimal control problem and in the general landscape of robot learning this can quickly lead to extensive *trials and errors*. In this thesis, we propose a radically different formulation of RL for robotics. Our objective

is to define a basis on which new algorithms more closely related to numerical optimization methods can be formulated with guarantees of convergence and accuracy that are necessary for tasks that involve learning complex poly-articulated behavior. This establishes a broad road map for our research. Consequently, we explore the various aspects of building a framework that binds trajectory optimization and learning - from the architectural design of neural networks, quantities that can be learned, theoretical guarantees on learning, and details of practical implementation of the subsequent framework.

1.1 Motivation

Using RL requires an exhaustive exploration of the environment and in high-dimensional robotic systems computing large enough datasets can quickly become infeasible. Consequently providing guarantees of reliability and accuracy can become challenging. A key feature of our proposition is to take advantage of the capabilities of Trajectory Optimization (TO) solvers. Trajectory Optimization methods [WK88] also solve an optimal control problem, although they typically do not seek an explicit representation of the optimal policy but rather for an optimal trajectory. While a trajectory is a far less expressive object than a policy - in particular, being only able to lead to open-loop behaviors if not systematically reevaluated - it can also often be represented in lower dimensions than a policy. This is easy to see - typically a few polynomials can sufficiently encode information about the trajectory while deep neural networks are now the standard representations for policy.

Consequently, trajectory optimizers are able to provide local optimum with a few trials and errors and can reach arbitrary levels of convergence and accuracy. These are the two criteria not met by RL algorithms. In addition, trajectory optimizers can also handle explicit constraints on the robot state which can then be used to certify the behavior of physical systems in real conditions. A key element to acquiring these properties is the use of derivatives of the objective function within the evolution model of the system: the derivatives can be evaluated efficiently through

differentiable simulators. Yet close-loop behaviors then also imply permanent re-optimization of the robot trajectory given the observed situation and consequently, the solver may also fall in local minimum with disastrous consequences.

Overall we observe that the limiting factors of both RL and TO hamper efforts to develop a robust reliable framework for predictive control. This forms the core issue we tackle in this thesis *learning complex poly articulated behavior* -

1. *with few demonstrations or trials inside a differentiable simulator*
2. *with high accuracy of predictions*

The main question is how to fully exploit the knowledge about the system to learn in order to improve the efficiency of our algorithm. In this thesis, we try to answer this question by searching for a new algorithm that can exploit both RL and TO for better performance.

1.2 Contributions

To achieve our stated objectives, we combine TO with a reinforced learning loop. This is accomplished by setting an iterative loop in the backdrop of the recursivity of *Bellman's optimality principle* [Bel66] such that learning depends on the data provided by TO, while the efficient computation of optimal trajectories over a preview horizon by TO depends on accurate learning. In an abstract way, we can think of this as a synergistic coupling between learning and trajectory optimization where the coupling is explicitly formulated to improve trajectory optimization through learning and vice-versa. We do not try to improve one at the cost of other.

Intuitively, we can reason that since the value function defines a partial ordering over policies, it can then be said that some policy is better than other policies if its expected return is greater than the expected return of other policies (from the same state) - the expected return is, of course, encapsulated in value functions. Therefore,

if the optimal value function can be estimated, it becomes relatively easy to estimate the corresponding optimal policy - if the optimal value function at some state is known, then the corresponding actions after 1 step search will then be optimal. Therefore optimal actions from some state can be selected without knowing all possible future behaviour, if the optimal value function at that state is known. This is the guarantee provided by Bellman´s optimality equations.

Furthermore, we explicitly focus on learning with high accuracy and with reduced rollouts. We use trajectory optimization to give us state-value pairs and state-control trajectories which we learn in a supervised phase using three neural networks representing the value function, policy function, and the predicted optimal trajectory. The estimates of value function are subsequently used inside TO at the terminal position, while the approximation of the optimal policy serves as a guide for TO.

The road-map we set for ourselves is as follows:

- Estimate global value function given a time-dependent estimate of value function.
- Learn an accurate representation of state-control trajectories that can be used to warmstart.
- Establish optimality criteria on learning.
- Develop means to use additional information provided by the TO during the learning phase.

While the algorithm presented in this thesis can be used with any TO, we used Differential Dynamic Programming (DDP) [May73; XLH17], a particular class of trajectory optimization. The choice of using DDP is primarily due to its ability to compute 1^{st} and 2^{nd} order derivatives of the value function. We explicitly use gradients of the value function during training and we will show that that enables us to learn with high accuracy and in fewer trials.

1.3 Thesis Structure

This thesis is organized into two parts. In part 1 corresponding to **Chapters 2-4** we present the theoretical aspects of our work. **Chapters 2** focuses on the state-of-art and background, while in **Chapter 3** we present our algorithm.

In the next part, from **Chapters 4-6** we present the experimental evaluations of our work. We summarize and conclude in **Chapter 7**.

Chapter 2

In this chapter, we first establish notations and foundations that we will use throughout this thesis. We then present an overview of optimal control and Reinforcement Learning. We also discuss hybrid methods that combine learning and trajectory optimization.

Chapter 3

In this chapter, we first describe the general optimal control problem setup. Then we follow it up with a brief description of the Differential Dynamic Programming algorithm.

We then describe the algorithms that we developed during the course of this thesis - DVP, ΔPVP, ∂PVP. Then we define a form of constrained learning - Sobolev Regression - that we use to learn value function. We give a general introduction to using $1st$ order regression methods using *target* gradients.

This chapter marks the end of the theoretical section of our work.

Chapter 4

This is a short standalone chapter where we describe the robots we used to evaluate our algorithm.

We tested our algorithm in two steps during our thesis. Step 1 consisted in learning only the global value function, and the second step involved learning state-control

trajectories along with the global value function. Therefore, we split our results into two parts with each part presented in the next two chapters respectively.

Chapter 5

In this chapter, we will present the results of learning value function. We will discuss the issues of singularities and empirically demonstrate the robustness and generalization capabilities of our algorithm.

Chapter 6

This chapter will present our results on learning a good initialization of the TO solver. We will compare the quality of the predictions of our algorithm and show that it can achieve super-linear convergence in the number of attempts required to compute and refine a (locally) optimal trajectory.

Chapter 7

Perspectives and Conclusion

1.3.1 Associated Publications

Parts of the thesis especially **Chapters 4-7** feature in the following publications.

- **Amit Parag**, Nicolas Mansard. A Supervised Formulation of Reinforcement Learning: with super linear convergence properties. 2022 - Under Review at IROS 2023. ⟨hal-03674092v2⟩

- **Amit Parag**, Sébastien Kleff, Léo Saci, Nicolas Mansard, Olivier Stasse. Value learning from trajectory optimization and Sobolev descent: A step toward reinforcement learning with superlinear convergence properties. International Conference on Robotics and Automation (ICRA 2022), May 2022, Philadelphia, United States.⟨hal-03356261v2⟩

- Rohan Budhiraja, **Amit Parag**, Ewen Dantec, Justin Carpentier, Carlos Mastalli, et al.. Crocoddyl: Fast computation, Efficient solvers,Receding horizon and

Learning. Journées Nationales de la Robotique Humanoïde, May 2020, Paris, France. ⟨hal-02898916⟩

2

Optimal Control and Reinforcement Learning - a discussion of motion generation for robots

> *Come on now, I hear you're feeling down*
> *Well I can ease your pain*
> *Get you on your feet again*
> *Relax*
> *I'll need some information first*
> *Just the basic facts*
> *Can you show me where it hurts?*
>
> **— Pink Floyd**
> Comfortably Numb

Contents

2.1 Notations 10
2.2 Generalized Optimal Control - Problem Formulation 11
2.2.1 Solutions 13
2.3 Dynamic Programming and Markov Decision Process 14
2.3.1 An overview of Reinforcement Learning in robotics 16

2.4 Direct Methods - Transcription and Resolution 19
2.4.1 Transcription 19
2.4.2 Trajectory Optimization - A literature review 22
2.4.3 Model Predictive Control 23
2.5 Hybrid Methods 25
2.6 Discussion 27

2.1 Notations

The work presented in this thesis lies at the intersection of Reinforcement Learning and Trajectory Optimization. It is then necessary to define a common vocabulary that can be used to elucidate ideas with enough clarity.

We will first concretely define certain terms that we use throughout this thesis.

- *Robot* - A machine capable of carrying out a complex series of actions automatically either autonomously or semi-autonomously. Alternatively, in RL literature, a robot can be thought of as an *agent* in an environment. We will use the words robot and agent interchangeably.

- *Environment* - The world where the agent/robot lives. In Optimal Control formulation, the environment is typically referred to as *system*. The environment can either be :

    1. Fully observable - The agent can determine the state of the environment at all times.

    2. Partially observable - The agent is partially aware of the state of the environment.

  We refer to the model of the environment as Ω.

- *State Space* - The state space is the space of possible values that the agent in the environment can take. The agent assumes a state within the environment and can traverse the environment by changing its state. The state space can either be discrete or continuous. We refer to the state space as $\mathcal{X}$.

- *Control Space* - The control space is a set of controls that are permissible for the agent in a given environment. In RL literature, this is referred to as the *Action Space*. The control space can also be either discrete or continuous. The control space is referenced with $\mathcal{U}$.

- *State Transition Function* - The state transition function governs the evolution of the system. We will refer to the state transition function as a function f that takes the current state and action x, u at the current timestep t and returns the next state x_+ of the agent in the environment, i.e $x_+ = f(x, u, t, \Omega)$ where $x, x_+ \in \mathcal{X}$ and $u \in \mathcal{U}$. The optimal control problem can be defined with a stochastic evolution, however, in this thesis, we concern ourselves with **deterministic** models - in a deterministic setting the probability of the next state in the state space reached by the robot is either 1 or 0. In OCP literature the state transition function, f, is often called the system dynamics and is defined by numerical integration schemes such as Euler or Runge-Kutta methods through an initial value problem formulated from a mechanical study of the physical state [Car+19; Fea14]. In a more general term, f is a simulator of the agent behavior.

We will use these basic notations to establish further definitions.

In the next Section, we give a formal description of Optimal Control.

2.2 Generalized Optimal Control - Problem Formulation

In its generalized formulation, optimal control problems involve the minimization of some cost functions. The resulting optimization problem can be written as

$$\underset{\underline{x},\,\underline{u}}{\text{minimize}} \quad L(\underline{x},\underline{u},\Omega) \tag{2.1a}$$

subject to

$$x_0 = \hat{x}, \tag{2.1b}$$

$$x_{t+1} = f(x_t, u_t) \forall t, \tag{2.1c}$$

$$x_t \in \mathcal{X}, \tag{2.1d}$$

$$u_t \in \mathcal{U}, \tag{2.1e}$$

$$g_t(x_t, u_t) \geq 0, \tag{2.1f}$$

$$g_T(x_T) \geq 0 \tag{2.1g}$$

In the continuous formulation, the state and control variables $\underline{x}, \underline{u}$ are infinite-dimensional vectors in the corresponding state and control space. The starting state of the system is x_0 and $\underline{x} : t \in [0, T] \rightarrow x_t$ and $\underline{u} : t \in [0, T] \rightarrow u_t$ and f is the state transition function.

We will drop Ω from hereon.

Cost The cost function in (2.1a) is typically defined as an integral over some efficiency criteria and is usually written as :

$$L(\underline{x},\underline{u}) = \int_0^T \ell(x_t, u_t)dt + \ell_T(x_T) \tag{2.2}$$

where $\ell(.)$ is the integral cost (Lagragian term) and $\ell_T(.)$ is the terminal cost (Mayer term). This particular form is mostly used in practice and will lead to the resulting sequential decision problem. The shape of the cost term can be different however classical algorithms demand the Markovian property to function [Ber12].

Initial Constraint The initial state of the system in (2.1b) is assumed to be given : $x_0 = \hat{x}$.

Dynamics Constraints The constraints in (2.1c) impose limitations over the dynamical evolution of the system for all time.

State and Control Constraints Additionally, constraints on state and control in (2.1f) and (2.1g) can be put on the system for instance, to prevent slippage if contact sequences are being considered. In general, any other equality or inequality constraint can be formulated, in particular on the terminal state. We will barely consider that in this thesis.

2.2.1 Solutions

The generalized optimal control problem is a *functional optimization problem* where the inputs themselves are functions and the resolution of the problem lies in determining that particular input function and trajectory that optimizes some cost functional.

To further elucidate, the problem in (2.1) is an optimality decision problem, it is not a (static) optimization program - often called Non-Linear Programming (NLP) [Ber97; BSS13] - as it involves an infinite number of variables under an infinite number of constraints. The optimality conditions are formally described by a set of partial differential equations (PDEs) - the Hamilton-Jacobi-Bellman equations (HJB) [Pen92]. When possible directly integrating the HJB PDEs gives the complete solution of OCP in (2.1), yet it is rarely feasible in practice.

There are two methods for solving the optimal control problem in practice:

1. Indirect Methods - The OCP is rewritten under some kind of differential equation whose complexity can be numerically handled, typically using more convenient optimality criteria such as Pontyagrin Maximum Principle [Kop62].

2. Direct Methods - Transcribe the problem into a static optimization problem by discretizing $\underline{x}$ and/or $\underline{u}$ to find the minimum of the objective cost function.

Both RL and TO can be classified as Direct Methods - TO considering polynomials/discretization of $\underline{x}, \underline{u}$ and RL considers a neural network as an approximation of the optimal policy π. To better understand the connection with RL, we will first reformulate (2.1) into a deterministic MDP.

2.3 Dynamic Programming and Markov Decision Process

The overall optimality problem in (2.1) can also be solved via *Dynamic Programming*. Based on Hamilton-Jacobi-Bellman equations [Nai02], dynamic programming uses *value function* to model the interactions of the robot with the environment and proceeds to solve the overall optimal control problem by breaking it down into simpler subproblems in a recursive manner. Note that Dynamic Programming uses recursion to solve problems by breaking it into smaller sub-problems. It can be used to not only solve Markov Decision Problems but problems in general where it is possible to nest sub-problems recursively inside larger problems - this implies that there is relation between the value of the sub-problems and the value of the overall problem [CL01].

Reinforcement Learning aims to find the solution to a subset of optimal control processes solved via dynamic programming that can also be described as a Markov Decision Process.

Formally, a (deterministic) Markov Decision Process is defined as a tuple of $(\mathcal{X}, \mathcal{U}, f, l, \gamma)$

- f is the state transition function and next state x_+ reached by the robot under the application of control u in state x where $x_+, x \in \mathcal{X}$ and $u \in \mathcal{U}$.
- ℓ is the feedback obtained by the robot at time t for moving from state x to x_+. In RL this is canonically called the *reinforcement*, r, obtained on

transitioning to x_+ from x under u. Alternately the reinforcement can be written as $r(x_+|x,u) = -\ell(f(x,u),u)$.

- γ is the discount factor. $\gamma \in (0,1)$.

A policy is therefore a decision rule π that maps states to controls :

$$\pi : x \in \mathcal{X} \rightarrow u = \pi(x) \in \mathcal{U} \tag{2.3}$$

As discussed before, in this thesis we consider a deterministic formulation. π can also be considered stochastic which is convenient to handle multi-modalities however the proposed algorithms cannot yet handle stochasticity.

Informally, the goal of an RL agent is to search for a policy that optimizes the discounted sum of rewards over a horizon. The horizon itself can be :

- *Finite time horizon* T: The problem is characterized by a deadline T and the RL agent only focuses on the sum of rewards up to that time.
- *Infinite time horizon*: The problem never terminates but the agent will either reach a termination state or rewards closer in time will receive higher importance or the agent will seek to maximize the average of rewards.

We consider the case of the finite-time horizon to establish two further definitions.

State-Value function In the finite-horizon case, the state value function is written as :

$$V(x,t) = \sum_{s=t}^{T-1} \ell(x_s, \pi_x(x_s)) + \ell_T(x_T) | x_t = x; \pi \tag{2.4}$$

where ℓ_T is the reward function for the final/terminal state at the end of the horizon. From hereon, we will denote the state value function as the value function - we can

also write the value function as a function of π, as is done in RL literature, however for the remainder of this thesis, we will keep this notation.

Note that this value function is time-dependent. We will later use this to show an equivalence between dynamic programming and trajectory optimization. We would also note that a related quantity called that state-control value function or quality function - Q function - can also be defined, but is not necessary for the ideas in this thesis. We show the Q function in the Appendix 8.1

Optimal Value Function and Optimal Policy The solution to MDP is an optimal policy π^* that satisfies :

$$\pi^* \in \text{argmax}_{\pi \in \Pi} V \tag{2.5}$$

in all states $x \in \mathcal{X}$, where Π is some policy of interest. The corresponding value function is then the optimal value function: $V^* = V$. Furthermore, there is always atleast one policy that satisfies the above optimality criteria. We denote π^* as the collection of all policies that maximize the the value function. Alternatively, we can think of an optimal policy as any policy that is *greedy* with respect to V^*.

Note that MDPs are usually formulated as a maximization problem while OCP is formulated as a minimization problem however maximization of rewards is identical to the minimization of cost. The difference is largely anecdotal. We will use the minimization form in this thesis.

2.3.1 An overview of Reinforcement Learning in robotics

Reinforcement Learning learns forecasts of behavior through trial and error. RL algorithms can be generally divided into three classes [SB18; KT03]: actor-only, critic-only, and actor-critic methods, where actor and critic are references to policy and value function.

- Actor-only methods such as SRV [Gul90] and REINFORCE [Wil92] optimize over a parameterized family of policies through policy gradient methods, however, this can lead to slow learning. Actor-only methods provide two primary advantages - strong convergence properties due to gradient descent methods and the ability to generate actions in complete continuous action spaces. A much broader discussion of methods used to estimate gradients are given in [PS08; BB01].

- Critic-only methods such as Q-learning [WD92] or SARSA [RN94] learn the Q function without any explicit function for the policy. These methods tend to show a lower variance in estimates of expected return [Sut88]. In turn, the policy is then derived by selecting *greedy* actions - actions for which the expected reward is maximum [Sch02]. This results in Critic-only methods being computationally intensive - if the action space is continuous then one needs to resort to some optimization procedure in every state to find the action that maximizes value.

- Actor-Critic methods [Gro+12; Aru+17] combine the advantages of actor-only and critic-only methods approaches. The parameterized actor brings the advantage of computing continuous actions while the critic supplies the actor with low-variance knowledge of the performance. However, these methods often require a large number of samples [Tan+18] followed by considerable tuning. To mitigate these problems, in [Haa+19] an extension to soft actor-critic approach [Haa+18] is developed to counter over-sensitivity to hyper-parameters [Hen+18]. The quality of predictions of the actor itself was examined in [HFH20] with the conclusion that the actor learns better when learning to act optimally over a horizon rather than learning the next optimal state. On the whole, however, actor-critic methods usually show better convergence properties than critic-only methods [KT03].

At the core of RL lies stochastic gradient methods that when used in other deep learning methods exhibit different behavior than in RL. For instance, Proximal Policy Optimization developed by [Sch+17] uses ADAM [KB15], while Advantage

Actor-Critic in [Sch+17] relies on RMSProp [HSS12]. It is unclear why different optimizers exhibit different behaviors in RL and whether the theoretical properties of these optimizers in supervised regression settings generalize to Deep RL algorithms [HRP18]. A possible way to study this problem, borne by the ideas in this thesis, would be to replace gradient descent with Sobolev descent [Par+22]. This presents a very interesting avenue of research.

The immediate concern of this thesis is to use RL effectively in robotics. RL has been successfully used in robotics from dexterous manipulation [MAW07; AG12], trajectory and route tracking [NZZ19; Ota+19; Xia+19] , navigation [Kim+20; Cad+16] and path planning [Gar+09; Ten+09]. In [Kat+16] a pure vision-based RL method was proposed for highly sensitive movements for a pneumatic five-fingered arm rotating an object. Hindsight Experience Replay [And+17] was also used for sample efficient learning on three different tasks - pushing, sliding, and pick-and-place. Similarly, RL algorithms were successfully used for manipulation in cluttered environments [BMK19; Joh+19]. In [FLA16] RL was improved to tackle the problem of trajectory tracking for autonomous vehicles underwater. Similarly, in [Wei+18] Deep Deterministic Policy Gradient [Lil+15] was trained to track the optimal route while [Lon+18] presented an a safe and efficient collision avoidance policy while route tracking. Reducing the probability of collision remains an important task in RL. In [Wan+18] a hybrid RL model is presented that solves a real-world vision language navigation task.

RL in robotics is thus characterized by wide-ranging datasets and high sampling complexity and a primary challenge is then to learn with fewer demonstrations and trials. A possible way to ameliorate this problem is through the use of underlying geometric structures of the robotic datasets, for instance recasting inference in Riemannian or Grassman manifolds as shown in [Cal20] can provide a principled way of using data in robotics. Similarly, decomposing complex robotic movements into a series of simpler *movement primitives* and formulating the learning process as an information fusion problem to generate complex movement was explored in [PSC22].

Either way learning in robotics is characterized by two key issues - sampling in high dimensional spaces [Dul+21] and hypersensitivity to parameters (although hypersensitivity is a far more pervasive problem). This is the primary motivating factor behind this thesis. These difficulties are studied and analyzed in [Mah+18a; Mah+18b] over 450 independent experiments which took over 950 hours of robot usage with the conclusion that learning performance can be highly sensitive to different elements of the task setup such as the control space, the control cycle time (defined as the time between two subsequent application of controls), and system delays. A possible way to counter the over-sensitivity problem [HRP18] is through Maximum Entropy Deep Reinforcement Learning [Haa+17; Haa18]. MEDRL provides a basis for constructing hierarchical strategies through probabilistic reasoning that can eliminate the need for extensive tuning.

Conclusion When compared with TO, RL is less prone to getting stuck in local minima and is much faster at run-time than TO. However, its sample efficiency, bloated convergence time, and accuracy of predicted policies with respect to HJB optimality of substructures [Bel66] criteria stand in contrast to that of TO. Whereas TO strongly integrates the simulator with numerical optimization, RL tends to idealistically decouple the algorithm from the simulator by considering it as an abstract oracle. This presents a case to more tightly couple the simulator and the learning algorithm.

2.4 Direct Methods - Transcription and Resolution

2.4.1 Transcription

To solve the optimality problem described in 2.1 with finite resources, we first need to transcribe it as an NLP such that a static discretized optimal control problem can be formulated. There are two classes of transcription methods used to convert the optimality problem into a general constrained optimization problem - *shooting* and

simultaneous [Kel17]. Either of these two methods first transforms the continuous problem in Section 2.2 into a non-linear programming problem [Bet10]. The primary difference between shooting and simultaneous methods lies in their implementation of constraints on system dynamics - shooting methods use simulations to explicitly enforce the dynamics of the system while simultaneous methods enforce dynamics at a series of points along the trajectory. Once the problem has been transcribed various solvers such as SNOPT [GMS05], IPOPT [WB06], FMINCON [Too+93] or ACADO [HFD11] can be used to compute the solution.

Shooting Methods Shooting methods fall into two classes - single-shooting [Ger03] and multiple-shooting [Die+06]. Single-shooting methods, also known as Initial Value Approach [Isl+15], the dynamical system is solved by a numerical integrator. Typically, it involves some function such zero-order-hold, piecewise linear, piecewise cubic, or orthogonal polynomials to approximate the control trajectory [Rös19].

Multiple-shooting methods [BP84] such as Differential Dynamic Programming [May73] work by breaking up a trajectory into a series of segments and using single shooting to compute the solution for each segment as shown in Figure 2.1. As these segments get shorter, the relationship between decision variables and the objective function to be minimized becomes more linear. Usually, in multiple-shooting, the end of a segment will not necessarily match the beginning of the next segment. This difference is known as defect or gap [BP84] and is added to the constraint function such that gaps decrease.

Simultaneous Methods Simultaneous methods represent state trajectory using decision variables [CB89]. The solution is then computed in a way such that the constraints on dynamics are satisfied only as special points along the trajectory. Two commonly used simultaneous methods are direct transcription [Vas10] and direct collocation [Ged11]. To understand the differences between these methods, we first need to note that the dynamics constraints can be represented in either *derivative* or integral form. The derivative method of representing constraints requires that

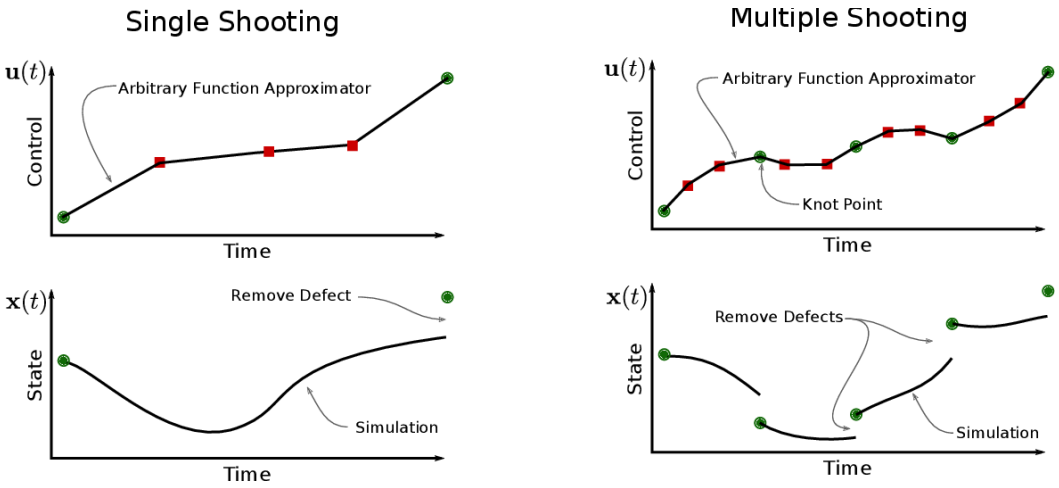


Fig. 2.1.: Single vs Multiple-Shooting [Kel15]. Note that multiple-shooting consists of a series of single shooting methods but with defect/gap constraints added to make the resulting trajectory continuous.

the derivative of state with respect to time must be equal to the system dynamics: $\dot{x} = f(x, u)$. The integral method requires state trajectory to match the integral of the dynamics with respect to time: $x = \int f(x, u)$.

Direct transcription uses the integral form of the dynamics constraint and the control and state trajectories are then represented through piecewise-constant and piecewise-linear functions. Direct collocation methods are slightly different in that they represent input as a piecewise-linear function of time and the state trajectory is piecewise-cubic. The value of the state and control at each knot/node point along a trajectory is then modeled as decision variables. Orthogonal collocation is another simultaneous method that uses orthogonal polynomials to approximate state and control function. These methods tend to be fast and lead to accurate numerical interpolation, differentiation, and integration of the polynomial [BT04]. These methods are very popular in building quick prototypes since they provide great flexibility. Yet they tend to be very sensitive to hyperparameters.

2.4.2 Trajectory Optimization - A literature review

Once the problem has been transcribed, TO solves the optimal control problem while allowing for the full exploitation of the system dynamics within a prediction horizon. In a discretized transcription, $L(X, U)$ can be written as an infinite sum of running cost, $\ell(x, u)$:

$$L(X, U) = \sum_{k=0}^{+\infty} \ell(x_k, u_k) \tag{2.6}$$

where $\ell(x, u) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$.

Given some initial condition x_0 and some input control trajectory u_t over a finite time interval T, trajectory optimization methods [RMM94; AHM22] computes the long-term but finite-horizon cost of executing that trajectory using standard additive-cost optimal control objective. For finite time horizon problems of length T, the $L(X, U)$ is split in two parts:

$$L(X, U) = \sum_{k=0}^{T-1} \ell(x_k, u_k) + \ell_T(x_T) \tag{2.7}$$

where $\ell_T(x_T)$ is the cost at the terminal state. We denote X^*, U^* as the optimal solution to this minimization problem over a finite time horizon T and from the initial starting state x_0.

Equivalence with MDP This minimization problem is exactly equivalent to the maximization of the accumulated sum of rewards problem over a time discrete finite-horizon, albeit a discount factor. The additive cost function can then be thought of as a negative cumulative reward or negative reinforcement over a horizon. Under these conditions, the search for an optimal policy in RL can be said to be equivalent to the search for an optimal control trajectory in TO [KZG22; Ber12; Ber97].

2.4.3 Model Predictive Control

The solution computed by TO is a fixed trajectory that cannot be directly applied in open-loop due to the problem of robustness when modeling any relevant robotic dynamical system. A controller is then hand-tuned to rigidly track if the accuracy is acceptable in the operation context. Yet the optimal policy can be evaluated by systematic re-evaluation of a new trajectory for each new measured state, $\hat{x}$. Model Predictive Control methods use trajectory optimization as feedback policy. MPC methods execute the following steps sequentially :

1. Measure the current state $\hat{x}$
2. Optimize trajectory from that current state.
3. Execute the first action, u_0, from that optimized trajectory.
4. Allow dynamics to evolve for one step and repeat.

A standard approach is to use trajectory optimization for optimization over a horizon longer than T through *receding horizon* MPC. Since the trajectory optimization problem is formulated over a finite-horizon, then it is typical to continue solving for T step horizon problem at each evaluation of the controller.

While MPC enables recovery to strong disturbances, TO is commonly used to plan complex open-loop trajectories [MZP22] since the problem can be too complex to be solved by MPC in real-time. The effectiveness of model-based techniques can be seen in their successful application over a variety of complex tasks, including locomotion and manipulation [Di +18; Kui+16; Koe+15; Bel+21; Neu+18].

The level of autonomy provided by these methods offers a viable solution that can solve large-scale optimal control problems over a receding finite-horizon while dealing efficiently with constraints, non-linearity, uncertainties, and unforeseen perturbations [Hub+11]. For instance, in [WFK22; Sma+22], we see the successful deployment of a fast MPC based on Quadratic Programming (QP) formulation for the task of locomotion in simulated bipedal systems. Similarly, TO is used to

compute trajectories for dexterous manipulation and whole body locomotion of the REEM-C robots in [LRM22; AHM22] respectively. To account for uncertainties under contact, the authors of [DZZ21] propose *chance complementarity constraints* that changes stochastic constraints into deterministic constraints while in [LBS22], the authors show a robust framework that can tackle the issue of slippage in unknown environments.

Despite their wide use, several challenges need to be addressed :

1. Non-linearities in the robot model can force TO in local minima [BBV04]. This automatically leads to suboptimal solutions. Escaping from these local minima can be done through Entropy-based DDP [SWT22] or RRT [NKH16].

2. A problem which we also tackle in this thesis is to reduce the number of iterations needed by the solver to converge to a (locally) optimal solution. This is usually done by either reformulating the OCP as a convex optimization problem or providing the solver with a good guess. Warmstarting the solver typically involves either learning offline some representations of the optimal behavior via Reinforcement Learning (RL) or supervised regression [Man+18; Lem+20; Lid+22a], or re-using the solution computed in the previous control cycle as warmstart [DBS05].

3. The computational cost is also a function of the complexity of the task which scales with the cube of the dimensionality of the system, with the result that MPC has mostly been applied to systems with fewer degrees of freedom [Neu+16; GM16; HD10].

   Recent works such as [Hut+17; Dan+21] have tried to target more complex robots such as quadrupeds or bipeds by focusing on a more accurate modeling of system dynamics and a more advanced (stronger, more robust, faster) simulation of poly-articulated behavior [TET12; HLH18]. The computational efficiency also scales with the preview horizon of the OCP. This is easy to see - the computational cost of computing a locally optimal trajectory over a

larger preview horizon is much higher than the cost incurred in computing trajectories over a short preview horizon.

4. Particular care must be put on an efficient implementation of rigid body dynamics, for instance for complex systems more efficient numerical optimal controls such as [Car+19; Kim+19] that can work at higher frequencies are needed.

2.5 Hybrid Methods

Hybrid solutions lie at the intersection of learning and TO. Combinations of these can be divided broadly into two categories depending on their goals.

Learning some *function* to enhance the performance of TO In this approach, the primary goal is to boost the performance of TO either by speeding up the computation time or by improving the quality of computed solutions. Either way, this usually involves learning a function *offline* through a *library* and subsequently using it *online* inside TO: the library of representations itself is generated through simulations or through offline solving of OCP.

The learned function can represent the dynamics of the system to alleviate the problem of designing controllers for tasks with complex non-linear dynamics [LKS15] or a cost function to allow for re-planning with hindsight [Tam+17] or value function either to improve the quality in sampling-based planning [Bha+14] or reducing planning horizons [Low+19].

Much closer to our work is [Zho+13], where the authors propose to learn the global value function through Gaussian Mixture Models (GMM) and Nearest Neighbor through (locally) optimal runs of TO. The learned value function is then used inside TO as a proxy for terminal cost. This is also the primary reason why we chose to learn the global time-independent value function - estimates of the global value

function, V^* when used as a proxy for terminal cost can turn short horizon problem into *infinite-horizon* optimal control problem [CA98; HL02; ETT11] with theoretical guarantees of optimal behavior as long as the estimated value function accurately reflects global value function. Similarly, in [VMR22; DKT19], the authors learn the value function for one step model predictive controller.

Learning a library of trajectories to provide initial guesses to warmstart predictive controllers has been extensively explored in works such as [Man+18; Lem+20; MT14; Lid+22a]. The critical importance of warmstarts is easy to see - in general, predictive control is dependent on the resolution of a large nonlinear optimization problem at each control cycle, therefore the absence of a good initial guess can slow down convergence or fail to avoid poor local optima. However, providing a good warmstart implies learning in continuous domains which is usually challenging - the RL training time may be excessively long and its convergence is strongly dependent on the exploration strategy.

Leverage TO to speed up the training of RL. To speed up training time, the authors of [Zha+19] developed an exploration strategy that made the RL agent effectively recognize "good" trajectories by asynchronous episodic control. Similarly [LK13] and its successor [LK14] proposed to bias the exploration strategy in RL toward low-cost regions by using DDP to guide policy search (GPS) to avoid the problem of falling in poor local optima during the search for complex policies with hundreds of parameters.

While GPS has spawned numerous variants such as [Bue+18; ML16; Yah+17; Men+19; Tag+22; Lev+16] to name a few, guiding policy search was recast as a teacher-student problem where the OCP solver acts as a teacher [CFH20; Kah+17]. The idea was further refined in [Zha+16] where the offline trajectory optimization phase was replaced with MPC. A similar variant of GPS uses path integral to optimize the individual instance trajectories and later combines them into a single policy [Che+17]. In [Vie+18] guided policy search was used to learn contact-rich dynamics for underactuated systems along locally optimal trajectories in a sample-efficient

manner and was tested on real robots. In [MT14], the authors combine a trajectory optimizer with a policy learning phase through the Alternative Direction Method of Multipliers.

The key limiting factor here is the use of canonical methods of RL and its dependency on the quality of guiding samples. This is where our method differs from previous works on learning a good initialization. We use supervised regression to learn guiding samples provided by DDP while the value function approximator refines the quality of trajectories computed by DDP.

2.6 Discussion

In the preceding sections, we showed the equivalence of discrete-time optimal control problem with shooting transcription and Markov Decision Process in the deterministic case. Under this equivalence, we can establish a notion of *complementarity* between RL and TO - search for an optimal policy can be thought of as a search of optimal trajectory.

Then, we discussed methods that explicitly solve discrete-time OCP through TO-based methods while solutions to MDP were discussed under RL approaches. The primary advantages of RL: speed of inference, and the ability to approximate complex policies: are negated by slow convergence and over-sensitivity to hyper-parameters leading to brittleness and sample inefficiency. The sample inefficiency in RL also comes from using differentiable simulators as abstract oracles and failing to fully exploit the available knowledge of the system, whereas TO combines optimization algorithms with simulators.

This is the central theme of this thesis - *exploit the knowledge of the system as much as possible* - for an end-to-end learning framework. We think this is a good direction to explore. To that end, we combine TO in a reinforced learning loop with the aim of improving the quality of TO while improving the accuracy of predictions. The work we present in this thesis is much more aligned with the *hybrid* solutions we discuss

in Section 2.5. Our proposed method learns value functions and warmstarts for model predictive control which gets improved iteratively. We learn value functions primarily because of its importance.

In a Markov Decision Process value functions are unique fixed points of *Bellman operators* and govern interactions of RL agent with its environment [Kam+12]. This is where the algorithmic implementations of RL fail to realize its mathematical foundations: an extensive analysis in [Ily+20] showed that value function estimates never match the true value function and only marginally guide the search for the policy. The mathematical foundation is the maximization of some *stochastic objective* function based on the governing dynamics of the system and RL usually proceeds by estimating the 0^{th} order gradient of that objective [SB18; Sch+17]. This is also where our implementation of the learning process slightly differs from the canonical methods used in robot learning or learning in general. We explicitly use 1^{st} order Sobolev regression [Cza+17] to learn the value function [Par+22]. Learning in Sobolev Spaces has the added benefit of imparting more interpretability to the hidden layers and minimizes the problem of local minima[1].

[1]The problem of local minima is closely related to information bottleneck in DNNs studied in [TPB01]. The theory of information bottleneck suggests that the hidden layers in DNNs trade-off between keeping enough information about the input variables for the prediction of the learned function and a concise representation of the learned function itself [ST17; TZ15]. This is where using higher order derivatives of the target function in the learning process can be quite useful.

Part I

Theory

3 Differential Policy Value Programming - ∂PVP

> *I am so clever that sometimes I don't understand a single word of what I am saying..*
>
> **— Oscar Wilde**
>
> The Happy Prince and Other Tales

Contents

3.1 Differential Dynamic Programming 32
3.1.1 Quadratic Approximation 34
3.1.2 Line Search 36
3.1.3 Complexity and Regularization 36
3.1.4 Discussion 37
3.2 Differential Value Programming - *DVP* 39
3.2.1 Algorithmic Principles 39
3.2.2 Algorithm 41
3.2.3 Architecture of V_α 43
3.3 DVP with Sobolev Descent - ΔDVP 44
3.3.1 Sobolev Regression 45
3.3.2 Sampling the State Space 47
3.3.3 Conclusion 52

3.4 Differential Policy Value Programming - ∂PVP 53
3.4.1 Design of V_α, X_β, U_γ 54
3.4.2 Discussion 56
3.5 Conclusion 58

3.1 Differential Dynamic Programming

Differential Dynamic Programming refers to a general class of dynamic programming algorithms that iteratively solve finite-horizon discrete-time optimal control problems described in (3.1) by using locally quadratic models of cost and dynamics [May73; LS92] and show quadratic convergence. Current research has shown that DDP can effectively compute solutions in high dimensional state spaces by successfully producing trajectories for UAM vehicles [TMT14].

There are three canonical ways to view the DDP algorithm :

- As a 2-pass algorithm
  - A backward pass to backpropagate the value derivatives and a forward pass to rollout the dynamics
- As iterative linear quadratic regulator - iLQR
  - Dynamic Programming applied in the tangent (differential space). Classic DDP requires second-order derivatives of the dynamics, which are usually the most expensive part of the computation. If only the first-order terms are kept, one obtains a Gauss-Newton approximation known as iterative Linear-Quadratic Regulator - iLQR [Cho+75].
- As sparse sequential quadratic programming - Sparse SQP.
  - Optimal way of using sparsity in quadratic programming by solving a sequence of optimization subproblems, each of which optimizes a

quadratic model of the objective function subject to a linearization of the constraints to induce sparsity [Bet93].

DDP is a second-order shooting method that can admit quadratic convergence under mild assumptions for any system with smooth dynamics. It has also been shown to have convergence properties similar or better than Newton's methods performed in the entire control sequence [LS92].

Recall that under shooting transcription, the optimal control objective is exactly equivalent to a Markov Decision Process. The objective function to be minimized is :

$$L(X, U) = \sum_{k=0}^{T-1} \ell(x_k, u_k) + \ell_T(x_T) \tag{3.1}$$

where $\ell_T(x_T)$ is the cost at the terminal state and X^*, U^* are the optimal solution pairs to this minimization problem over a finite time horizon T and from the initial starting state x_0.

DDP takes advantage of the recursivity of Bellman's Optimality Principle [Bel66] by adding the condition, $V_T(x_T) = \ell_T(x_T)$, where $V_T(x_T)$ is the value function at the terminal step x_T. In each iteration, it numerically solves the optimal control problem described above by performing a backward and a forward pass on the current estimate of the state-control trajectories (X, U) : a backward phase to estimate the value function as quadratic fit along the current candidate trajectory, a forward phase to refine the candidate trajectory based on the value function.

To construct a quadratic fit of the value function, DDP measures the deviations from the current candidate trajectory through Taylor's expansion [XLH17], discarding terms beyond second-order. It then returns a quadratic approximation of the cost-to-go and the hessian, gradient at every step along the preview horizon. In the next few sections, we quickly formulate the quadratic approximation and comment on the overall complexity of the DDP algorithm.

3.1.1 Quadratic Approximation

DDP searches locally for the optimal state and control sequences for the OCP in (3.10) through a *forward pass* or rollout phase and a *backward pass* to compute a local solution to (3.10) using a quadratic Taylor expansion. To obtain a quadratic approximation of the value function, we need to first define a Q function that can measure perturbations of (3.10).

Let $Q(\delta x, \delta u)$ be the measure of deviation of $l(x_i, u_i) + V_{i+1}(f(x_i, u_i))$ around the current candidate state-control trajectories x_i, u_i. The Q function is a scalar function taking vector inputs and expresses the change in cost that results from perturbing a point in the nominal trajectory. The goal of the DDP algorithm is to find perturbations that minimize the Q function.

Therefore,

$$\begin{aligned} Q(\delta x, \delta u) = l(x_i + \delta x, u_i + \delta u) - \ell(x_i, u_i) \\ + V_{i+1}(f(x_i + \delta x, u_i + \delta u)) - V_{i+1}(f(x_i, u_i)) \end{aligned} \tag{3.2}$$

With a quadratic approximation and dropping terms beyond the 2^{nd} order, DDP rewrites $Q(\delta x, \delta u)$ as :

$$Q(\delta x, \delta u) \approx \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 0 & Q_x^T & Q_u^T \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix} \tag{3.3}$$

where the expansion coefficients are :

$$
\begin{aligned}
Q_x &= \ell_x + f_x^T V_x' \\
Q_u &= \ell_u + f_u^T V_x' \\
Q_{xx} &= \ell_{xx} + f_x^T V_{xx}' f_x + V_x' f_{xx} \\
Q_{uu} &= \ell_{uu} + f_u^T V_{xx}' f_u + V_x' f_{uu} \\
Q_{ux} &= \ell_{ux} + f_u^T V_{xx}' f_x + V_x' f_{ux}
\end{aligned}
\tag{3.4}
$$

The subscripts $\{\}_x, \{\}_u, \{\}_{xx}, \{\}_{uu}, \{\}_{ux}$ are the $1^{st}, 2^{nd}$ order derivatives with respect to the state and control variables. The primes denote the values at the next time step.

Minimizing the quadratic approximation in (3.3) with respect to δu, we have :

$$
\delta u^* = arg \min_{\delta u} Q(\delta x, \delta u) = k + K \delta x \tag{3.5}
$$

where $k = -Q_{uu}^{-1} Q_u$ and $K = -Q_{uu}^{-1} Q_{ux}$ are the feed forward and feedback terms. The corresponding recursive updates to the 1^{st} and 2^{nd} derivatives of the value function, which we denote by $V_x(i), V_{xx}(i)$ are done as follows :

$$
\begin{aligned}
V_x(i) &= Q_x + K^T Q_{uu} k + K^T Q_u + Q_{ux}^T k \\
V_{xx}(i) &= Q_{xx} + K^T Q_{uu} K + K^T Q_u + K^T Q_{ux} + Q_{ux}^T K
\end{aligned}
\tag{3.6}
$$

At the end of this backward pass, we now know the quadratic approximation of the value function along the horizon.

3.1.2 Line Search

Once the backward pass is completed, the proposed locally-linear policy is evaluated with a forward pass by integrating the dynamics along the computed feed-forward and feedback terms k, K :

$$\begin{aligned} \hat{u}_i &= u_i + \alpha k_i + K_i(\hat{x}_i - x_i) \\ \hat{x}_{i+1} &= f(\hat{x}_i, \hat{u}_i) \end{aligned} \tag{3.7}$$

where $x_1 = x_1$ and $\{\hat{x}_i, \hat{u}_i\}$ is the new candidate state-control pair. α is a backtracking search parameter, set to 1 and iteratively reduced. Backtracking line search is a common technique employed to ensure that the generated trajectory converges to a local minimum, i.e. at every iteration of DDP we obtain a trajectory with a lower total cost.

This method works by introducing a step-size parameter α that is applied to the control policy. The purpose of α is to regularize the trajectory and ensure that the new trajectory is of lower cost than the previous one. Note that if $\alpha = 0$, the state and control trajectories are not modified. This backward-forward process is repeated until convergence to the (locally) optimal trajectory.

3.1.3 Complexity and Regularization

DDP uses the value function V along with its derivatives V_x, V_{xx} to iteratively invert the block diagonal components. The inverted Hessian also characterizes the direction during Newton's descent. The line search then computes the next candidate trajectories for the optimal solution based on the descent direction. While typically, this is done by approximating the dynamics $x_{t+1} = f(x_t, u_t)$ as linear, DDP performs the forward pass on the exact non-linear dynamics and not its linearized version. This is done to ensure the feasibility of the corresponding solutions, unlike

the classical Newton step which tends to produce discontinuities in the solution. Under a linearized approximation of dynamics, the DDP solution is identical to the Newton Step. Although DDP searches in the space of control trajectories $\mathcal{U} \in \mathcal{R}^{m \times N}$, it solves the m dimensional problem N times. This difference is even more apparent when considering N Hessians of size $m \times m$ rather than a larger $Nm \times Nm$ matrix. This immediately shows that the factorization complexities are of the order $\mathcal{O}(Nm^3)$ and $\mathcal{O}(N^3m^3)$.

Furthermore, in order to guarantee a descent direction, additional regularization is used when the hessian loses positive definiteness, for instance by adding a Tikhonov regularization term [GHO99] in (3.5). Additionally when the cost terms are least squares residuals, then the Hessians can be approximated through the square of the Jacobian, i.e $\ell_{xx} \approx r_x^T r$, where r is the residual that models the cost. We also use this idea to design the architecture of our neural networks. This approximation corresponds to the Gauss-Newton variation and is referred to as iLQR. The regularization parameter and α are adapted online following a Levenberg-Marquardt heuristic [Mor78].

3.1.4 Discussion

DDP is an iterative improvement scheme that finds a locally optimal trajectory from a fixed starting point. In every iteration, a quadratic approximation of the time-dependent value function is constructed over some horizon of length T. By iteratively moving toward the minima of the quadratic approximations, the trajectory is progressively improved toward a local optimum with superlinear convergence.

The DDP algorithm is relatively cheap and simple to implement and also takes advantage of the sparsity pattern of the problem. Additionally, it also provides, along with the solution, a linear feedback term that can be used to correct the control sequence when the observed trajectory deviates from the optimal one. In particular, this allows the solution to be robust to some amount of external noise.

However, DDP based methods provide limited convergence guarantees (no globalization strategies) or require a significant amount of iterations of the DDP algorithm to converge to a feasible solution. This forms a major limiting factor in the deployment of DDP for online Model Predictive Control for instance. The reader is also invited to consult [CM+19] for a tutorial on the DDP solver.

DDP exploits the 1^{st} and 2^{nd} order estimates of the value function. This then allows access to the superlinear convergence rate if we are able to provide derivatives of ℓ and f along the preview horizon. In practice, DDP computes time-dependent estimates of the value function along a finite preview horizon. The time dependence of the value function makes DDP unsuitable for *infinite-horizon* problems and leads to significant performance degradation.

This limitation was examined and an alternate scheme to estimate the global time-independent value function was proposed in [Low+19] through a *plan online and learn offline* - POLO - framework. POLO tightly couples local trajectory optimization with global value function learning to overcome the drop in performance when using an approximate value function. Computing the optimal value function exactly is not tractable except in a few cases such as LQR [SR98]. The POLO framework overcome intractability by using the popular fitted-value iteration [Lut+21] to approximate the global value function through locally optimal runs of a trajectory optimizer. The trajectory optimization computes the solution over some predefined horizon of length T which then generates the targets for fitting the value approximation.

We build upon this idea to estimate the global value function and use it inside DDP to formulate an *infinite-horizon* problem by replacing the terminal cost, ℓ_T, with the global value function, V^*, that remains solvable with finite resources. Furthermore, we also learn estimates of state and control trajectories which we then use to provide a good initialization to DDP.

This brings us to the three main contributions of this thesis -

- *DVP*

  - Estimating value function at ∞ horizon.
  - Supervised classical regression
- ΔDVP
  - Estimating value function at ∞ horizon.
  - Supervised Sobolev descent: using gradients in training
- ∂PVP
  - Estimating value function at ∞ horizon.
  - Supervised Sobolev descent - using gradients in training.
  - Learning state-control trajectories for warmstarting.

3.2 Differential Value Programming - *DVP*

To estimate the value function at an infinite-horizon such that the finite-horizon problem can be turned into an infinite-horizon MPC, we first need to formulate and subsequently exploit, the algorithmic principles of Bellman's Optimality conditions to build *DVP*.

3.2.1 Algorithmic Principles

Bellman's principle of optimality [Bel66; Bel54; Dre02] divides a sequential decision process into a series of smaller subproblems. Formally it can be stated as follows :

The Principle of Optimality

An optimal policy has the property that whatever the initial state and the initial decision is, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

The optimality principle allows us two insights.

The first insight is that we do not need to consider the past when optimizing the OCP from any given state. This allows us to define a cost-to-go or the net objective cost from some given state x_i as :

$$J_i(x_i, u_{i:T-1}) = \sum_{t=i}^{T-1} \ell(x_t, u_t) + \ell_T(x_T) \tag{3.8}$$

where $u_{i:T-1}$ are the decisions taken from x_i to arrive at the terminal state x_T.

This is due to a forward simulation of the system dynamics. The optimality principle can be restated as - the optimal trajectory from some x_i only depends on finding the optimal control sequence $u_{i:T-1}$. In turn, this allows us to define the value function V_i as the minimum cost-to-go from the state x_i :

$$V_i(x_i) = \min_{u:i:T-1} J_i(x_i, u_{i:T-1}) \tag{3.9}$$

The second insight lies in the optimality of substructures: if we consider the optimal trajectory from some state x_0 to x_T, then any sequence of sub-trajectory $x_i, ..., x_T$ of this optimal trajectory is also optimal.

The optimality of sub-structures along with the value function, Equation 3.9, allows us to make recursive decisions of the form :

$$V_i(x_i) = \min_{u_i}[\ell(x_i, u_i) + V_{i+1}(f(x_i, u_i))] \tag{3.10}$$

The recursivity provided by Bellman's optimality principle associates a value function V_i to each feasible state x_i for the optimization problem. Therefore V_0 is the solution for the OCP at state x_0. This property of ***recursive optimality*** is important. To estimate the global value function, we use an approximation of the value function as a proxy to represent the truncated horizon end, $\ell_T(x_T)$. This implies that we should be able to evaluate an approximation of the value function and its first and second order derivatives at the end of the horizon. The solver will then return a refined approximation of the value function at the beginning of the horizon, and its derivatives.

3.2.2 Algorithm

Cost-to-go learning The first iteration of *DVP* simply generates a batch of optimal trajectories of horizon length T . We then learn the value function by supervised learning. The result of this first iteration is a neural network approximating the cost-to-go for a horizon of T, denoted by $V_\alpha^{i=0}$. The superscript of V_α^i denotes the V^α at iteration i.

Iterative value learning *DVP* then proceeds by iteratively building upon its estimates of value functions. In the subsequent iterations, we replace the terminal cost with the approximated value predicted by the neural network. So at the end of every iteration, (3.10) is changed to:

$$V^i(x) = \min_u \sum_{k=0}^{T-1} l(x_k, u_k) + V_\alpha^{i-1}(x_T) \tag{3.11}$$

where i is the iteration number, $i \geq 2$, V_α^{i-1} is the value function approximated in the previous iteration. Should each iteration result in a perfect training, the i^{th} iteration

would lead to the approximation of the cost-to-go for an horizon of $(i+1) \times T$ which would tend toward the global V^* as i increases. The algorithm is summarized in Algorithm 1.

Algorithm 1: *DVP*

Algorithm parameters: horizon length T, iterations i, sample size s;
Initialize V_α;
Initialize DDP ;
foreach i **do**
    Sample s trajectories from DDP ;
    Train V_α through (3.15a) ;
    Update DDP terminal cost $\ell_T \leftarrow V_\alpha$
end foreach

Estimating the global value function and providing guarantees on its convergence to the infinite-horizon effectively removes time dependence from the value function. This induces an invariance that can then be mathematically shown to force states at the end of the finite prediction horizon to be in some neighborhood of an invariant terminal region.

Another important point to note would be that in MDP formulation, value functions are the interactions between the agent and an external environment and are therefore unique fixed points of their corresponding Bellman operators. A fundamental property of the Bellman operator is that it is a contraction in the value function space in the ∞-norm [Put94]. Therefore, starting from any bounded initial function, with repeated applications of the operator, the value function converges to the time-independent value function. This invariance effectively turns the prediction horizon from finite to infinite which is solvable with finite resources as long as V_α can provide *good enough* estimates of value function at infinite-horizon. The guarantees on stability and convergence to the infinite horizon then involves proving that the Jacobian linearization of the system at the terminal region is stabilizable. While the full mathematical proof is beyond the scope of this thesis, the reader is invited to consult [CA98].

The implementation of our algorithm merits discussion on three primary aspects - architecture of V_α, sampling strategy, and choice of regression method. We discuss the choice of regression method in Section 3.3.

3.2.3 Architecture of V_α

DDP is a second-order algorithm. It computes the 2^{nd} and 1^{st} order derivatives of the value function in the backward pass and therefore requires $V_\alpha^{'}$ and $V_\alpha^{''}$ in every iteration of our algorithm. We had initially designed V_α as a feed-forward network with one output that models the value function. However, double differentiating V_α did not prove to be feasible. To overcome this, we initially reduced the size of V_α - fewer hidden layers and fewer units - however the lack of depth in the network prevented us from obtaining a fair representation of the value function. Computing the hessian of a feed-forward network with one output proves to be computationally slow, especially for use in MPC and learning depends on the depth of the hidden layers, we use Gauss-Newton approximation to design V_α as the squared sum of residuals :

$$V(x|\alpha) = R(x|\alpha)^2 \tag{3.12}$$

This immediately allows us to write the 1^{st} and 2^{nd} order derivatives as :

$$V'(x|\alpha) = 2R'(x|\alpha)^T R(x|\alpha) \tag{3.13}$$

$$V''(x|\alpha) \approx 2R'(x|\alpha)^T R'(x|\alpha) \tag{3.14}$$

We implement this with a feed-forward network with 3 hidden layers of 64 units and hyperbolic tangent activation applied to every layer. The final layer outputs a three-vector residual. Modeling the value function as the output of Gauss-Newton approximation seemingly imparts a more physical interpretation to the hidden layers as compared to a simple feed-forward network. It also provides us with the added

benefit of not having to resort to the time consuming automatic differentiation for computing the Hessian of V_α, since now it can be estimated through (3.14).

3.3 DVP with Sobolev Descent - $\triangle$DVP

The *DVP* algorithm, in Algorithm 1, we initially developed was tested with classical regression. However, classical regression failed to improve the gradients of V_α with respect to the input that DDP demands. In the majority of applications of deep neural networks, classical regression usually consists in receiving a dataset of input-output pairs from a ground truth function and computing a loss to encourage the network to generate the same output as the ground truth function for some given input. In the traditional supervised setting, many of these ground truth functions may have an unknown analytic form. However in many other scenarios we do know the analytic form or are able to compute the ground truth gradients (or higher order derivatives) or the gradients are simply observables as is the case in [Rus+15; HVD+15; Jad+17; HMD15].

While the function approximation theoretically guarantees that neural networks can learn arbitrarily well, the accuracy and generalization capabilities yet depend on the quality (exactness and density) of the training dataset. This is where Sobolev learning differs from classical regression. It closely follows the work in [Hor91] that examined the approximation capabilities of multi-layered feed-forward networks and proved the universal approximation theorems for neural networks in Sobolev spaces - a Sobolev space is a metric space that measures the closeness or the distances between functions in terms of their differences in values and differences in the values of their derivatives.

More formally, a Sobolev Space is a vector space of functions equipped with a norm that is a combination of L^p norms of the functions together with its derivatives up to a given order [AF03]. An interesting result provided in [Hor91] showed that a feed-forward network with sigmoid activation can approximate both the value of

the target function and the derivatives of the function arbitrarily well. This is the key insight of Sobolev training - in the training phase, the neural network is not only trained on the output of the function but also the derivatives of the function.

Incorporating derivative information in function approximation to make more physics-informed neural networks has been previously explored in numerous contexts. In [WAP17] Bayesian optimization is assimilated with information about the gradient and Hessian to improve the predictive power of Gaussian Processes. The derivatives of the approximators (with respect to inputs) have also been used to either penalize model complexity [Rif+11] for effective knowledge discovery or to encode invariances by making symmetry aware neural networks [Sim+91] or to provide additional learning in attention distillation for Convolutional Neural Networks [ZK16]. Somewhat closer to our approach is the use of Sobolev training in Reinforcement Learning to match the derivatives of the critic with the target derivatives using small sigmoid-based architectures [TE07; FAP12; FA12; Wer92].

Sobolev learning has been shown to lead to better generalization and imparts more interpretability to neural networks albeit at a higher computation cost, as it constrains training by forcing neural networks to fit a target slope [MT92]. However, encoding the target derivative information in neural networks has been shown, empirically, to increase robustness against noise, as proven in, [Mas93] and mitigates the problem of increased computation cost by being more data efficient [LO97].

3.3.1 Sobolev Regression

Let f be a learnable function. For training points x_i, we assume that we have access to the output values $f(x_i)$ and the $j-$th order derivatives of f with respect to x_i. The training dataset, therefore, consists of $(K+2)$ tuples :

$$\{(x_i, f(x_i), D_x^1 f(x_i), ..., D_x^K f(x_i)\}_{i=1}^N.$$

Let m be some neural network parameterized by θ. The loss function in this case is then composed of two terms:

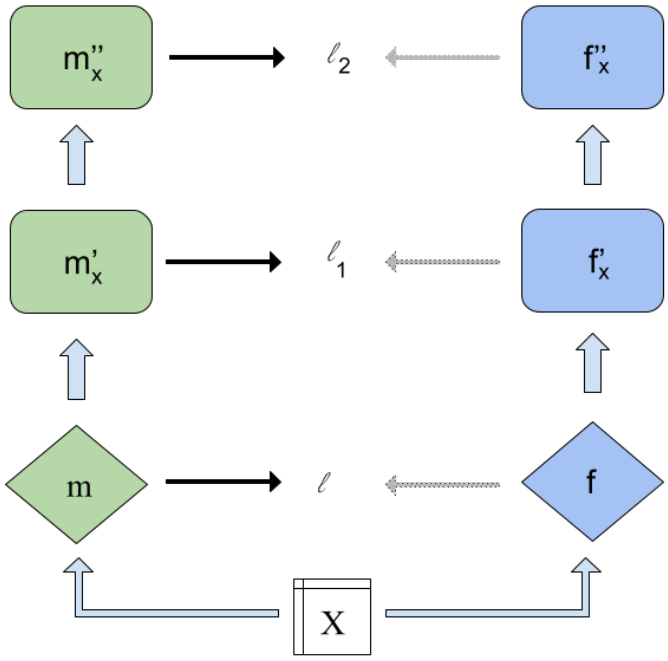


Fig. 3.1.: Illustration of Sobolev training of order 2. Diamond nodes m and f indicate parameterized functions where m is trained to approximate f. Green nodes receive supervision and the losses l_2, l_1, l are backpropagated to train m.

$$l_f = \sum_{i=1}^{N} \lambda(m(x_i|\theta), f(x_i)) \tag{3.15a}$$

$$l_d = \sum_{i=1}^{N} \sum_{j=1}^{K} \lambda_j(D_x^j m(x_i|\theta), D_x^j f(x_i)) \tag{3.15b}$$

l_f in (3.15a) is identical to the traditional loss used in classical regression that penalizes the difference between the output of the target function $f(x_i)$ and the output of the neural network $m(x_i|\theta)$ with some norm λ. The second loss function, l_d in (3.15b), is the Sobolev loss that measures the errors of the $j-$th order derivatives and constraints m to encode information about the derivatives of the target function in its own derivatives. Figure 3.1 shows compute graph for back-propagation during Sobolev training of order 2.

This led to our second attempt where we modified our initial algorithm to include Sobolev training, since we already have access to the derivatives of the value function - both 1^{st} and 2^{nd}. Our new modified algorithm can then be written as

Algorithm 2: ΔDVP

Algorithm parameters: horizon length T, iterations i, sample size s;
Initialize V_α;
Initialize DDP ;
foreach i **do**
    Sample s trajectories from DDP ;
    Train V_α through (3.15b) ;
    Update DDP terminal cost $\ell_T \leftarrow V_\alpha$
end foreach

Notice that the training part now includes derivatives through 3.15b. In theory, we can include the 2^{nd} order derivatives too in training, however, this largely depends on the time-computation-accuracy requirement trade-off. In practice, we do not use the Hessian in training.

3.3.2 Sampling the State Space

The classic control systems we use to establish the basic properties of our algorithm hardly require any clever sampling strategies. For computationally inexpensive systems such as Unicycle and Cart-Pole it is easy to compute large datasets by uniformly sampling for s initial configurations from the state space and keeping only the nodes estimated at horizon T.

The problem with this approach lies in the observation that V_α is only used to provide the cost-to-go (along with derivatives) at the terminal state. The terminal state reached by the manipulator is invariably near the target: computing a dataset sampled uniformly from the state space, is wasteful if the trained approximator is only used at the terminal position. On the other hand, the obvious solution of sampling for starting configurations that are near the target position more often than not leads to over-fitting.

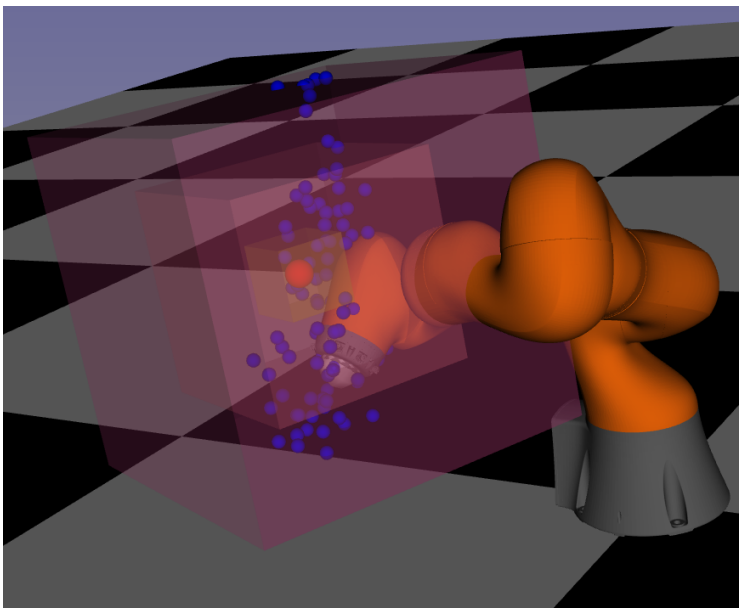


Fig. 3.2.: Adaptive sampling for the Manipulator Arm for the pose reaching task.

The second restriction in sampling is related to the dimensionality of the state space of the system under consideration. With an increase in dimensionality, it becomes infeasible to compute huge datasets. This problem when compounded by the corresponding training time, even though the training loop is supervised, can lead to bloated training time.

A possible way to ameliorate this problem is through *adaptive sampling*. In Figure 3.2, we show adaptive sampling for the End Effector pose-reaching task. The state space is 14 dimensional and the goal is to reach the target from various initial configurations sampled from the state space.

Through adaptive sampling we define 3 bounded boxes, near the static target position, shown in red, to constrain the possible terminal state reached by DDP (EE position and velocity) to be in the vicinity of the target. For each possible target position, shown in blue, we apply inverse kinematics to establish the corresponding joint space samples. This subsequently allows us to define initial conditions for our OCP. The dataset so generated is extremely accurate. The primary problem with this approach lies in its use of inverse kinematics to generate samples, in every training iteration.

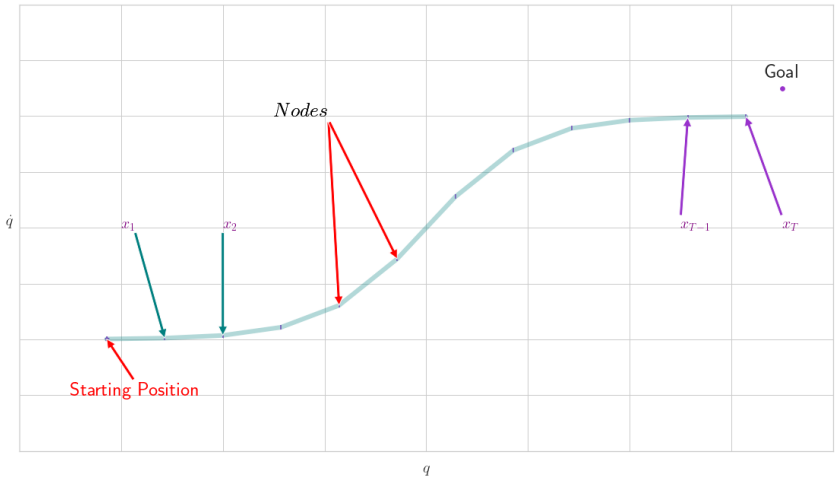


Fig. 3.3.: Nodes computed along a trajectory by DDP. The curve is the state-trajectory computed by DDP from a starting position to a goal. Each node on the curve is a state.

Another aspect of the sampling lies in the utilization of the state trajectories computed by DDP. DDP provides us with its estimation of time-dependent value function at each node/state for every state trajectory. We use the notation of *node* to denote any state in the state trajectory. In the Optimal Control literature, nodes are used interchangeably with knots points of a state trajectory.

Let us consider some optimal control task where the state-trajectory computed by DDP looks like Figure 3.3. It is easy to see that each state trajectory gives us T state-value pairs. Of this, we generally keep the node with the highest preview horizon and discard the rest - since DDP computes the time-dependent value function for every node in the state trajectory the most reliable estimate is the node with the highest preview horizon. This effectively implies that we can use the time dependence of the value functions provided by DDP to establish a notion of the quality of the dataset.

For example, consider a state trajectory such as shown in Figure 3.3. Each node along the trajectory would then be written as $x_0, x_1, ..., x_T$, where x_0 and x_T are the initial starting state and the terminal position reached. DDP computes for node in this state trajectory three corresponding quantities - the value function from the

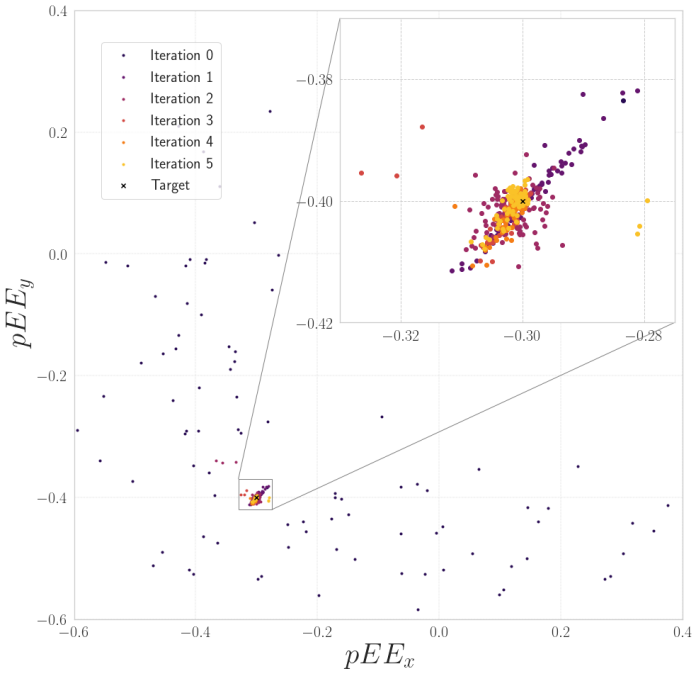


Fig. 3.4.: Illustration of refine sub-sampling approach to generate a dataset for the Manipulator Pose reaching task, shown here in End-Effector Space.

state, the gradient of the value function and the hessian of the value function - V, V', V''. This value function is, of course, a time dependent quantity.

Therefore, for x_0, DDP would compute $v(x_0|T), v'(x_0|T), v"(x_0|T)$. For x_1, DDP computes $v(x_0|T-1), v'(x_0|T-1), v"(x_0|T-1)$ and so on for each x_i in the trajectory. For the terminal state, x_T, the value function (and its gradient and hessian) would be estimated at a planning horizon of 1 : $v(x_0|1), v'(x_0|1), v"(x_0|1)$.

Therefore, the state with the highest fidelity is x_0 and its corresponding $\{v_{x_0}, v'_{x_0}, v''_{x_0}\}$ since it is estimated over a planning horizon of T. The nodes nearer the target would be of lesser quality since they are estimated by DDP over shorter and shorter preview horizons. The terminal state, x_T, with $\{v_{x_T}, v'_{x_T}, v''_{x_T}\}$ would be of the poorest quality since the corresponding preview horizon from x_T is 1.

Effectively, for one run of DDP with a preview horizon T, we get T observations and T target triplets in decreasing order of precision. Therefore the subset of this instance of data with the highest fidelity would yield 1 target triplets. For s runs of DDP, the purest dataset would be of the order of $s \times 1$.

To gauge the extent to which nodes with smaller preview horizons while not relying on adaptive sampling, we initially sample uniformly for s locally optimal trajectories. Of this, we sub sample the first k nodes, $\{x_0, x_1, ..., x_k\}$ from every state sequence for our training datasets. Thus the size of the training dataset is $s \times k$. The terminal node/state, x_T, from every state sequence is used in subsequent iterations as the initial starting configuration to sample for the new dataset. Figure 3.4 shows the distribution of the training datasets across the iterations in the End- Effector cartesian space for the manipulator pose estimation task. As we see, in higher iterations the data tends to become more concentrated around the target.

3.3.3 Conclusion

Both Algorithm 1 and 2 focus on learning the value function - the first algorithm with classical regression and the second with Sobolev regression. The learned value function was then used *in lieu* of terminal cost.

An interesting point to note here would be using V_α to provide the *cost-to-go* at every node along the state trajectory: effectively providing optimal value function estimates at every state and not just the terminal position. We did not do this for two reasons :

- DDP requires also requires 1^{st} and 2^{nd} order derivatives of the value function. Therefore, using V_α for next-step optimal control would also involve differentiating the neural network at every step. This would immediately increase the computation time.

  Consider some (locally) optimal trajectory computed over a preview horizon of length T. Assuming that the solver took N iterations/roll-outs to compute and refine its estimates of that trajectory, then it is simple to see that if V_α were substituted for *cost-to-go* at every step it would require differentiating the neural network at least $T \times N \times 2$ times.

- V_α is ultimately function approximation, therefore even small errors in estimates of value function could lead to divergence later on. Therefore, the safest way to incur as little divergence as possible seemed to be using V_α only at the terminal position. To quote Aristotle - The least initial deviation from the truth is multiplied later a thousandfold.

This leads us to the final contribution of the work presented in this thesis.

3.4 Differential Policy Value Programming - ∂PVP

As noted before, in the first iteration of ΔDVP we simply compute a batch of optimal trajectories of horizon length T. We use this to learn the global value function. Additionally, we can use this offline database to learn state and control trajectories which are then used to warmstart (i.e provide good initial guesses) DDP in the forward pass phase. This allows DDP in avoiding poor local optima while speeding up the convergence and also improves its performance in real-time in MPC.

Learning state-control trajectories along with global value function was the aim of our work. Our original goal was to bind TO and learning in a synergistic coupling through a reinforced loop and to accomplish that we build our algorithm in 3 steps over the course of numerous experiments. Algorithm 2 augments Algorithm 1 by adding a Sobolev loss term thereby taking advantage of additional information provided by the TO.

This brings us to our final contribution which we call ∂PVP. The algorithmic implementation is shown in Algorithm 5, where we also learn state-control trajectories, along with the value function. This synergistic coupling is reinforced in that the performance of V_α, X_β, U_γ depends on the quality of data provided by DDP and the quality of computations of DDP depends on accurate predictions from V_α, X_β, U_γ. By setting off an iterative loop, we force TO and learning to depend on each other. The purpose of utilizing Bellman's optimality principle is to ensure that the quality of both TO and learning remains guided by optimality.

The parameters of Algorithm 5 show that the empirical convergence of ∂PVP is a function of T, s and i - given some initial planning horizon T, the neural network V_α should approximate the value function over a horizon of $T \times i$. Therefore, as we iteration, i.e as i increases, V_α should, at least empirically, asymtote to the value function at infinite horizon.

Note that we can warmstart DDP in every iteration $i > 1$, however, this depends on the quality of learning and the capability to generalize which has to be established

Algorithm 3: ∂PVP

Algorithm parameters: horizon length T, iterations i, sample size s;
Initialize V_α, X_β, U_γ;
Initialize DDP ;
foreach i **do**
    Sample s trajectories from DDP ;
    Train V_α through (3.15b) ;
    Train X_β through (3.15a) ;
    Train U_γ through (3.15a) ;
    Update DDP terminal cost $\ell_T \leftarrow V_\alpha$;
    Warmstart DDP through X_β, U_γ
end foreach

empirically. The iterations i are equivalent to the concept of episodes in RL. We will use iterations and episodes interchangeably.

Sampling - The sampling strategies we employed in Algorithm 1 and 2 cannot be used with the complete algorithm in 5 since we now have to accommodate for learning state-control trajectories too. So we uniformly sample for s points in the state space as starting positions for DDP to compute an offline database for training.

3.4.1 Design of V_α, X_β, U_γ

The primary difference between ΔDVP and ∂PVP is that in ∂PVP we are also learning warmstarts along with the value function. This brings us to the question *how to design* V_α, X_β, U_γ such that learning is computationally efficient. We experimented with numerous designs of V_α, X_β, U_γ some of which we now describe.

Design 1 Implement V_α, X_β, U_γ as the outcome of a three-headed feed-forward network with common hidden layers. This was specifically done to test the trade-off between the training time of a multi-headed network and the theoretical advantages of enabling the multiple heads to benefit from the rich information encoded in

the common hidden layers. However, this resulted in a bloated computation time of parameter updates during the training phase. The practical benefits of shorter training time far outweighed the theoretical advantages of common hidden layers.

Design 2 Connect V_α, X_β, U_γ sequentially such that the output of the previous network is the input of the next network. In this design, V_α models the relation $x \rightarrow v$, X_β learns the state trajectory from with input as v through $v \rightarrow XS$ and U_γ infers the control sequence from the trajectory predicted by X_β through $XS \rightarrow US$.

Design 3 As actor-critic *esque* setup between V_α and X_β, U_γ, where the output from X_β goes back to V_α and V_α predicts the value function associated with each node in every state trajectory. We could not find an efficient way to enforce this architecture - the corresponding loss computed by DDP could not be efficiently packed into a tensor architecture and back-propagated since gradient descent has now to account for the parameters, α, β of both V_α and X_β.

Design 4 Connect X_β, U_γ either through common hidden layers or through sequential connection either $X_\beta \rightarrow U_\gamma$ or through $U_\gamma \rightarrow X_\beta$. V_α is learned separately.

However, none of these experiments led to any conclusive results and more often than not led to severe performance degradation. Taking advantage of the rich information in common hidden layers proved too difficult. Similarly, we thought that by inducing any relation between V_α, X_β, U_γ would lead to better performance. However, it frequently led to either vanishing or exploding gradients problem [Hoc98] which is a common problem in time series data since modeling the state trajectory and control trajectory can also be thought of as learning a time series data [Wen+20]. Inspired by the design of Long Term Short Memory (LTSM) architectures [Yu+19], we further explored adding specific time connections. This again proved to be extremely challenging as it required providing guarantees on the derivatives of LTSM. This has been a pervasive problem - designing complex architectures is easy

however differentiating them (output of those architectures with respect to input) always led to nonsensical gradients.

We obtained the best results by learning the three different quantities - value function, state trajectory, and control trajectory - on three different feed-forward networks V_α, X_β, U_γ trained individually. X_β and U_γ are implemented with simple deep feed-forward networks with 6 hidden layers with ReLU, ELU, Tanh alternately applied while V_α was implemented with a residual network with 3 outputs and Tanh activation as described in Section 3.2.3.

Note that DDP also provides us with derivatives of the control trajectory that can be used for Sobolev regression during training of U_γ. However, this places a huge computational burden on the automatic differentiation engine as we now have to calculate the Jacobian of the output of U_γ with respect to the input. We were unable to find an efficient way to differentiate U_γ dues to *curse of dimensionality*. To avoid an increase in training time, we choose classical regression in (3.15a) for the training of X_β and U_γ.

3.4.2 Discussion

The formulation of ∂PVP was devised to combine the benefits of function approximation and trajectory optimization. This is a major research avenue and is being actively pursued culminating in works such as [Zho+13; Lid+22a; Tou09; Dub+20]. In [Man+18], the authors combine a local trajectory optimization with a sampling-based motion planner to learn better control policy. Their cyclic approach is quite similar to our proposed algorithm in that the learned control policy is used to warmstart the MPC to generate better sample trajectories. Similarly, in [MT14] the authors leverage the high-fidelity solutions obtained by trajectory optimization to speed up the training of neural network controllers.

The two learning problems are coupled using the Alternating Direction Method of Multipliers (ADMM). This coupling enables the trajectory optimizer to act as a

teacher, gradually guiding the network toward better solutions. These two ideas form a core inspiration for our work. We use this as building blocks to formulate ∂PVP such that the performance of the optimal control solver can be enhanced by initializing the search, i.e warmstarts, and placing a guiding term in the cost function through an approximation of the global value function.

The importance of obtaining a fair representation of the global value function is primarily based on the view that RL-based approximations of the value function in robotics often suffer from the curse of dimensionality, low accuracy, and low efficiency. Additionally, using the value function as a final cost for the MPC computation is guaranteed to produce the optimal behavior as long as the trajectory terminates in an area where the terminal cost accurately reflects the value function [Zho+13] since it effectively turns the problem into *infinite-horizon* MPC [CA98; HL02].

Given a planning horizon, Bellman's equation yields a time-dependent value function, defined recursively as the optimal cost-to-go. This time dependence, however, implies that the planning horizon of states at the tail end of the trajectory is extremely short, which in turn may cause myopic behavior in these states. A fine-tuned terminal cost function, ℓ_T, mitigates this problem by effectively informing the controller about all events that lie beyond its planning horizon. However, designing a handcrafted ℓ_T is both problematic and difficult. On the other hand, replacing ℓ_T with an approximation of V^* leads to a *quasi-infinite* prediction horizon that guarantees convergence as was shown in [CA98]. The guarantee of asymptotic stability is a huge improvement upon moving horizon methods [PY93] or receding horizon approaches [MM90] or methods that place a terminal inequality constraint [MM93] such that the states are on the boundary of a terminal region at the end of a variable prediction horizon.

We note that while convergence guarantees can be made, the number of iterations required by ∂PVP to converge depends on primarily the initial planning horizon. It is easy to see that the length of the planning horizon corresponds to steps taken to reach infinite-horizon - the bigger the step the quicker you converge to a steady state. Once the terminal cost function has been replaced by the steady-state value

function, the TO should dramatically improve its performance especially when it is warmstarted. This is what we will empirically test and demonstrate in the remainder of this thesis.

3.5 Conclusion

This marks the end of the theoretical section of the thesis. To summarize the important points of our algorithm ∂PVP :

- ∂PVP learns the global value function and warmstarts in an iterative manner by constantly replacing the terminal cost with the latest approximation of the global value function.
- Three different neural networks are used to represent the value function, state, and control trajectory from some given initial state - V_α, X_β, U_γ.
- V_α is a residual network that maps value functions as a squared sum of residuals and is trained through Sobolev regression.
- X_β, U_γ are trained through classical regression.

In the next chapters, we show the results of the empirical evaluation of ∂PVP. For better organization we show the empirical evaluations of ∂PVP in two parts - ∂PVP$_v$ for results regarding value function and ∂PVP$_{x,u}$ for results regarding learning warmstarts. In Chapter 5 we show ∂PVP$_v$ and in Chapter 6 results of ∂PVP$_{x,u}$ are presented.

Part II

Experimental Evaluations

Experimental Setup

4

> *Billions of bilious blue blistering barnacles in a thundering typhoon!.*
>
> **— Captain Haddock**
> Tintin

Contents

4.1 Unicycle 61
4.2 Cart-Pole 63
4.3 Inverted Pendulum 64
4.4 7 *dof* Manipulator Arm 65
4.5 Discussion 66

This is a short standalone chapter where we describe the robot models we use to empirically evaluate ∂PVP - 3 classic control systems and a manipulator arm. We then discuss the organization of the next chapters.

4.1 Unicycle

The Unicycle in Fig 4.1 features a kinematic model of evolving on the $2D$ horizontal plane either driving forward or turning on the spot. Denoting the configuration vector $q = (x, y, \theta)$ of dimension $n = 3$, the Unicycle model reads:

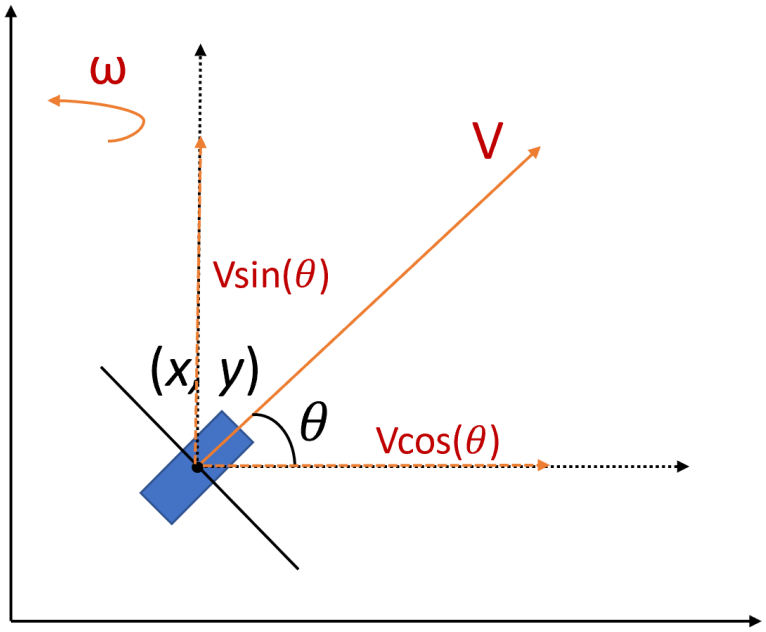


Fig. 4.1.: The Unicycle problem is formulated in $2D$.

$$\dot{x} = v\cos(\theta) \tag{4.1}$$

$$\dot{y} = v\sin(\theta) \tag{4.2}$$

$$\dot{\theta} = \omega \tag{4.3}$$

where the control $u = (v, \omega)$ includes the unconstrained longitudinal and angular velocities. The task is to reach the goal position $q = (0, 0, 0)$ while minimizing the residual sum of errors:

$$L = w_1||q||^2 + w_2||u||^2 \tag{4.4}$$

w_1, w_2 represent the weights on q and u

The Unicycle system is inherently subject to non-holonomic constraints. This non-holonomy leads to instabilities in learning which we shall discuss later.

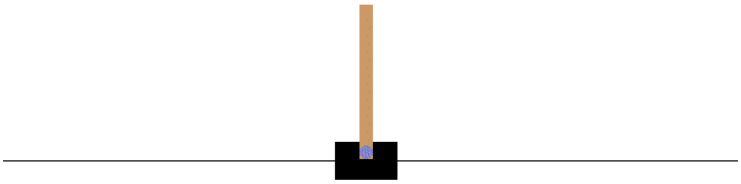


Fig. 4.2.: Cart-Pole - apply control forces such that the pole reaches the vertical position.

4.2 Cart-Pole

A Cart-Pole[1] is a classical dynamical system where an underactuated pole is attached on top of a 1D actuated cart. The task is to balance the pole around its unstable equilibrium (upper position) by controlling the horizontal forces acting on the cart [Flo07] as shown in Fig 4.2.

The velocity that is reduced or increased by the applied force is not fixed and it depends on the angle the pole is pointing. The center of gravity of the pole varies the amount of energy needed to move the cart underneath it. The cost function to be minimized is:

$$L = w_1||x||^2 + w_2||u||^2 \tag{4.5}$$

where $x = (q, \dot{q})$ is the configuration and the control u is force exerted on the cart and w_1, w_2 are the corresponding weights. The configuration space is 4 dimensional and describes the cart position and its velocity and the position and angular velocities of the pole.

[1] We use the Open Ai gym implementation of the dynamical model.

Fig. 4.3.: Pendulum. The optimal control task is to reach an unstable equilibrium position. The inverted pendulum is fixed at one joint.

4.3 Inverted Pendulum

The inverted pendulum[1] swing-up problem in Fig 4.3 consists in bringing the pendulum from a random position to its upper equilibrium and maintaining it upright. The cost function we use is similar to the Cart-Pole cost function, with u representing the torque applied about the pendulum's rotation axis and w_1, w_2 the weights :

$$L = w_1||x||^2 + w_2||u||^2 \tag{4.6}$$

where $x = (q, \dot{q})$ is the configuration and the control u is joint torque exerted on the cart. The configuration space is 2 dimensional and describes the angle and angular velocity of the pendulum. The joint torque u is not explicitly limited in our case, although such constraints can be, in principle, enforced.

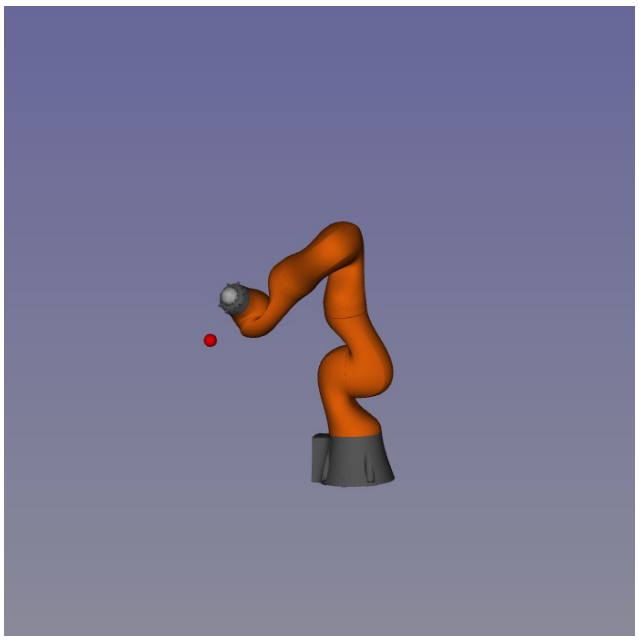


Fig. 4.4.: The 7 *dof* manipulator arm. The goal is to reach the red target.

4.4 7 *dof* Manipulator Arm

The manipulator arm introduced by KUKA is based on the developments of the German Space Center (DLR) in targeting direct machine-human interactions. The manipulator arm has 7 joints with corresponding control units and consequently allows 1 redundant degree of freedom (6+1 in total). Therefore, the configuration space is 14 dimensional with $x = (q, \dot{q})$

We formulate the optimal control problem as a static End Effector (EE) pose reaching OCP task from an initial q_0. We use a quadratic cost on translation and state limits. Additionally, we regularize the state and torque controls :

$$\min_{x,u} \sum w_1||q - q_0||^2 + w_2||\dot{q}||^2 + w_3||u||^2 + w_4||p(q) - p^*||^2 \tag{4.7}$$

where q, q_0 are the joint position and the initial joint position respectively, u is the torque control term and $p(q), p^*$ denote the end-effector position and the desired end-effector position respectively. Note that w_1, w_2, w_3, w_4 are weights.

4.5 Discussion

In this chapter, we presented the robot models that we will use to empirically evaluate our algorithm in the next two chapters. The classic control systems we use were decided based on their simplicity, however, non-holonomy and instability led to us to a much greater understanding of certain features regarding practical implementation. In Chapter 5 we discuss the results obtained in learning value function while in Chapter 6 we show the results obtained in learning warmstarts.

5

∂PVP$_v$ - Estimating global value function

> *To see a World in a Grain of Sand*
> *And a Heaven in a Wild Flower*
> *Hold Infinity in the palm of your hand*
> *And Eternity in an hour*
>
> **— William Blake**
> Auguries of Innocence

Contents

5.1 Problem Statement and Experimental Setup 68
5.2 Estimates of Value Function - Classic Control 69
5.2.1 Overall Convergence 69
5.2.2 Influence of horizon length 72
5.2.3 Robust Convergence 74
5.2.4 Convergence Issues: Singularities 75
5.2.5 Importance of Sobolev Loss 78
5.2.6 Conclusion 81
5.3 Comparison with PPO 82
5.4 Application to the 7 *dof* Manipulator Arm 85
5.4.1 Results 85
5.5 Discussion and Conclusion 87

In this Chapter, we show the results of learning the value function. We will discuss the convergence of our algorithm ∂PVP and the impact of Sobolev regression. In Section 5.1 we summarize the problem statement and establish the experimental setup. In Sections 5.2 we illustrate our results on classic control problems. Finally, in Section 5.4 we show the application of our algorithm on the more demanding 7 *dof* manipulator, which will be the topic of the next chapter.

5.1 Problem Statement and Experimental Setup

Recall that our primary algorithm for learning the value function is a composition of two algorithms - one to learn the global value function and the other to obtain fair estimates of warmstarts. The algorithmic formulation of ∂PVP that learns the global value function is ∂PVP_v was shown in Algorithm 2. We also recall it here :

Algorithm 4: ∂PVP_v

Algorithm parameters: horizon length T, iterations i, sample size s;
Initialize V_α;
Initialize DDP ;
foreach i **do**
    Sample s locally optimal trajectories using DDP ;
    Train V_α through (3.15b) ;
    Update DDP terminal cost $\ell_T \leftarrow V_\alpha$
end foreach

The two primary hyperparameters are the size of the dataset - s -, and the initial planning horizon - T. For the classic control system, we typically set $s = 100$ in each iteration or episode i of ∂PVP_v. The initial preview horizon, T, was set to 60 timesteps.

We also show the results of different preview horizons in Section 5.2.2. For the classic control systems, the starting positions (corresponding to x_0 in ddp problem in Crocoddyl shown in 8.2.1), were randomly sampled from their state space. For the 3 classic control systems, we computed a validation dataset V^* by sampling for

locally long optimal trajectories. We use this dataset V^* to establish a *ground truth* reference.

For the manipulator arm, randomly sampling for starting positions from its space space would lead to infeasible configurations. For that reason, we use Inverse Kinematics [KB06] and Adaptive Sampling [Buc88] at the beginning of ∂PVP$_v$ and subsequently use refined subsampling as mentioned in Chapter 3.3.2 to sample for starting configurations for the pose reaching task. The experiments performed in this Chapter did not involve learning warmstarts.

5.2 Estimates of Value Function - Classic Control

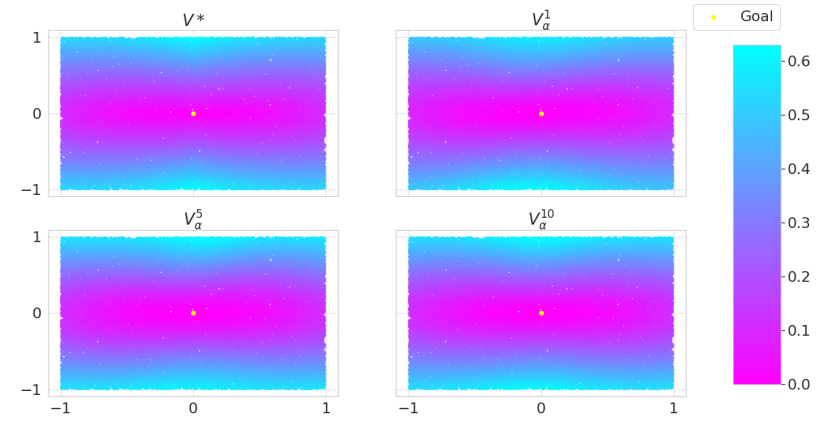
5.2.1 Overall Convergence

Figure 5.1 and Figure 5.10 respectively illustrate the value function learned by our algorithm and the corresponding mean squared error with respect to the ground truth value function established by V^*. Convergence to 10^{-5} is obtained for Pendulum and Cart-Pole systems, and convergence to 10^{-3} is obtained for the Unicycle (which would eventually reach the same accuracy with more iterations).

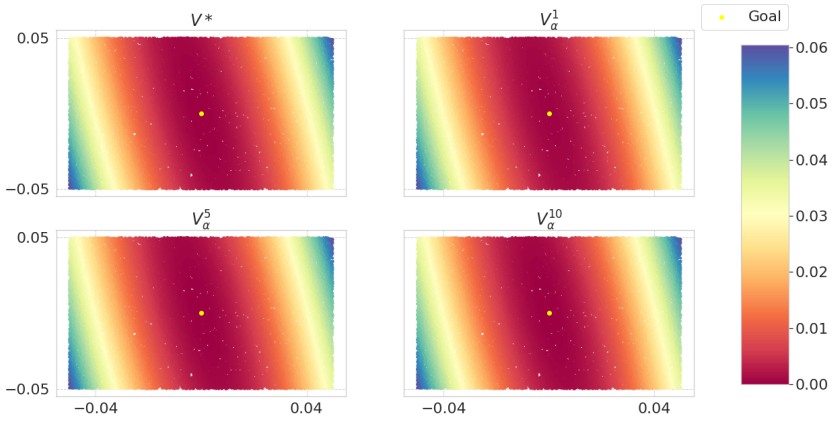
For the Pendulum case, just 1 iteration is sufficient for ∂PVP$_v$ to achieve convergence. For Cart-Pole, ∂PVP$_v$ takes a few more iterations to converge to a good enough approximation of the global value function. For systems with regions of local minima like unicycle, achieving convergence requires relatively more iterations.

Figure 5.2 quantifies the convergence through validation loss at the end of every iteration. As noted before, we also used the idea of *Bellman Residuals* [Bel66] to establish a similarity measure between two successive approximations - $i, i+1$ - of the value function through V_α :

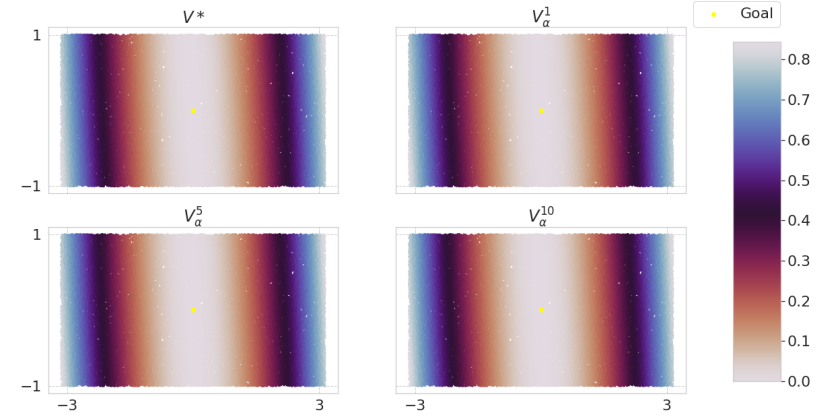
$$\ell_i = ||V_\alpha^{i+1} - V_\alpha^i||^2 \tag{5.1}$$



(a) Unicycle.



(b) Cart-Pole



(c) Pendulum

Fig. 5.1.: Value Functions learned by V_α after 1, 5 and 10 iterations. V^* is the validation dataset computed by sampling for locally long trajectories. The x, y axis represents q_1, q_2 of the state space of the robot. The dot in the center is the goal position.

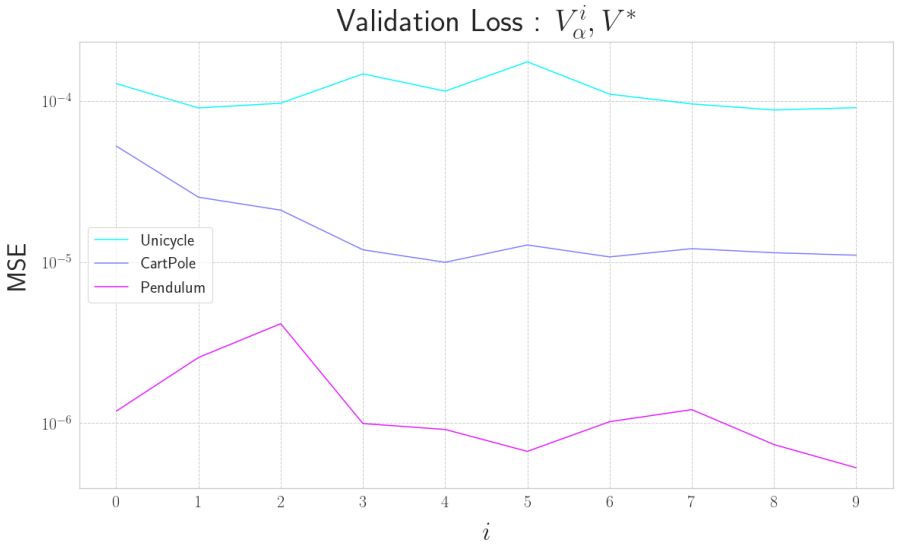


Fig. 5.2.: Validation Losses for Unicycle, Cart-Pole, Pendulum

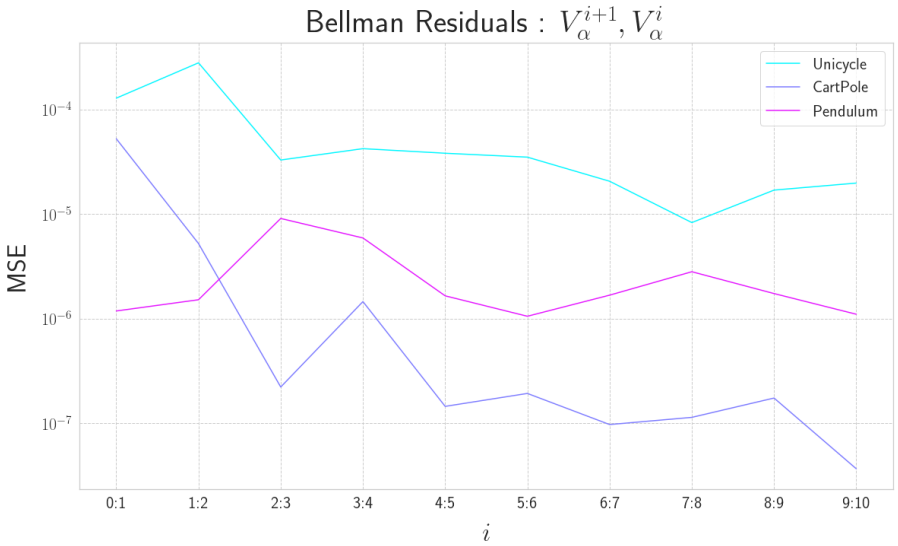


Fig. 5.3.: Bellman Residuals for Unicycle, Cart-Pole, Pendulum

where V_α^{i+1} and V_α^i are the value function approximated by the neural network, V_α, at iterations $i+1$ and i.

This criteria helps us establish the convergence of ∂PVP$_v$. In Figure 5.3 it is easy to see that as the value function estimates come closer to the optimal value function, the difference in successive estimates decreases. Residuals between two successive value functions can be a good indication of prediction. Upon or near convergence, the higher iterations of ∂PVP$_v$ should not show much difference between their behaviors. This residual quantifies the progress of the algorithm to an optimal estimate of V^* and should be used as stopping criteria to end the ∂PVP$_v$ loop. We see that the algorithm easily reaches an accuracy of $1e-6$ which is unusual for a typical RL algorithm, as discussed more comparatively in Section 5.3.

5.2.2 Influence of horizon length

In this Section, we discuss two results regarding the impact of the initial horizon length T. While convergence is readily achieved when T is higher as seen through Figures 5.4, 5.5 and 5.6 it can become increasingly difficult to achieve full convergence when ∂PVP$_v$ is started at lower values of T. We suspect that at lower T, the training data shown to the neural network is so corrupted that ∂PVP$_v$ finds it difficult to estimate V^*. Short horizons lead to important differences between the cost-to-go and the value, hence to a poor approximation of the value in early iterations of ∂PVP$_v$. This is illustrated in Figure 5.7, where the truncation to a short horizon leads to trajectories far from the optimum. Once the value is properly estimated, the bundle of trajectories converges closer to the optimum (on the unicycle, the convergence is not perfect despite an accurate convergence to the value, due to non-holonomy). The trajectories, shown in Figure 5.7, generated by ∂PVP$_v$ when trained under moderate to high horizons are far better than those computed by the solver alone.

Consequently, ∂PVP$_v$ converges faster when T increases. For the considered system, the typical duration of an episode (until system steady state) is 150 seconds, and

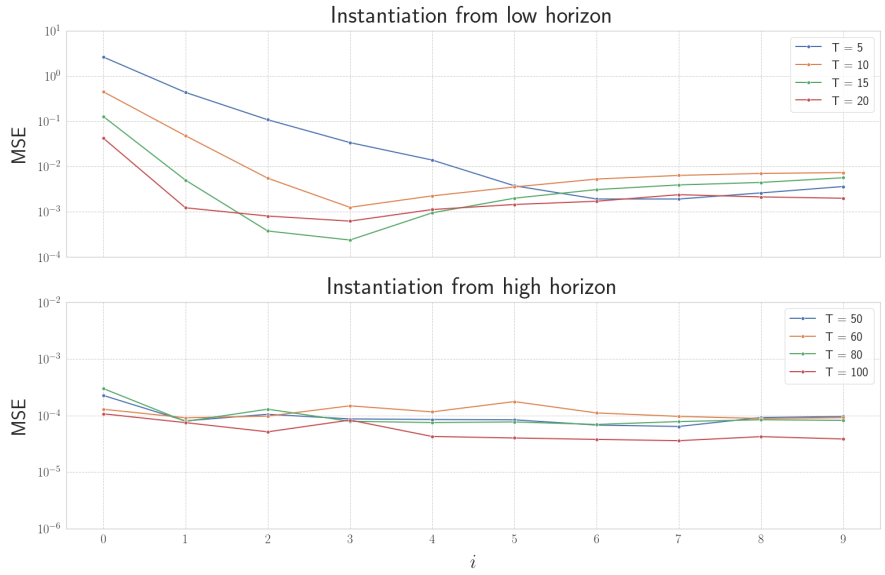


Fig. 5.4.: Impact of different initial preview horizons T on ∂PVP_v for Unicycle. Evolution of MSE between V_α^i and V^* under different runs of T .

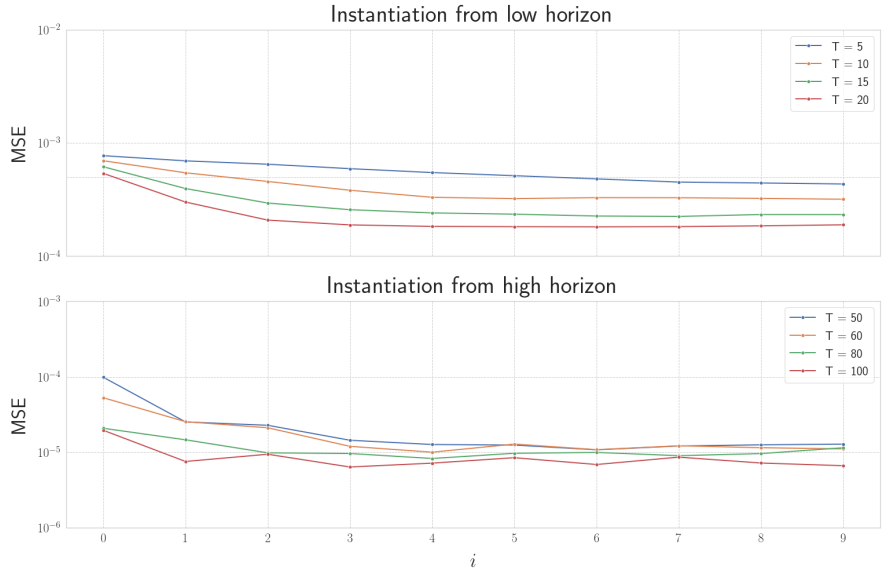


Fig. 5.5.: Impact of different initial preview horizons T on ∂PVP_v for Cart-Pole. Evolution of MSE between V_α^i and V^* under different runs of T

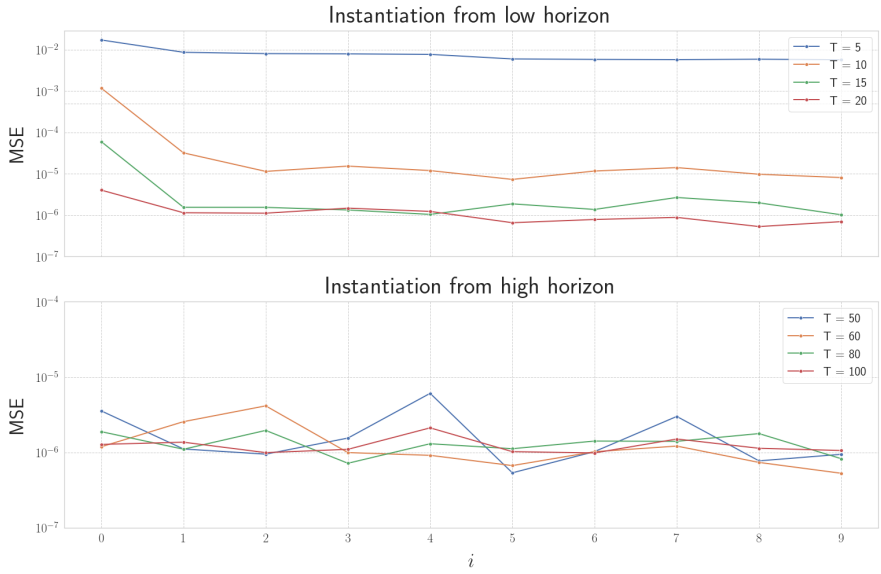


Fig. 5.6.: Impact of different initial preview horizons T on ∂PVP$_v$ for Pendulum. Evolution of MSE between V_α^i and V^* under different runs of T.

∂PVP$_v$ shows proper convergences for $T \geq 40$. For smaller T, the convergence is slower or even fails to reach a global optimum.

5.2.3 Robust Convergence

We empirically establish the stability and robustness of our algorithm by forcing ∂PVP$_v$ to learn a corrupted dataset at the first iteration by adding a predefined perturbation to the optimal data sampled using DDP. We find that ∂PVP$_v$ requires only a few iterations to converge back to the ground truth. Figure 5.8a, Figure 5.8b and Figure 5.8c illustrates the convergence of ∂PVP$_v$ under various levels of initialization noise. Robustness against noise also augments the generalization capabilities of ∂PVP$_v$. This is what we observe in Figure 5.9 - multiple trajectories computed by ∂PVP$_v$ with V_α^{10} serving as terminal cost.

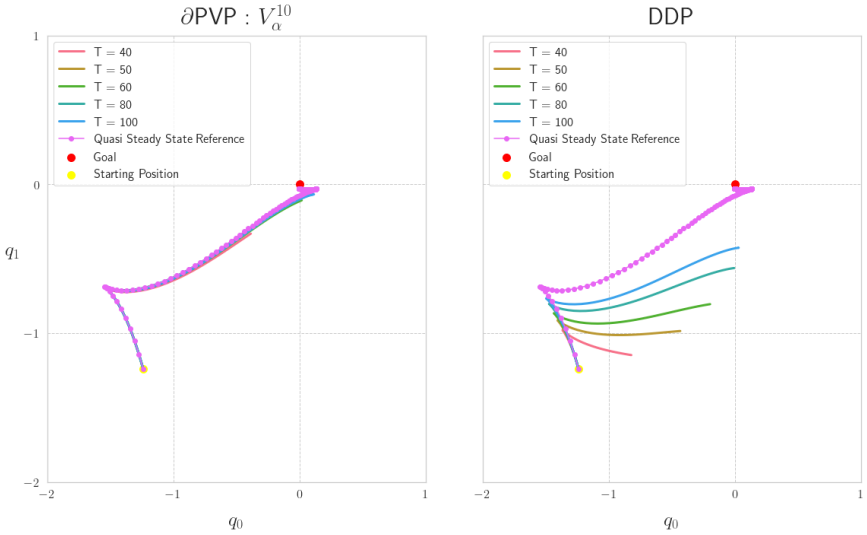
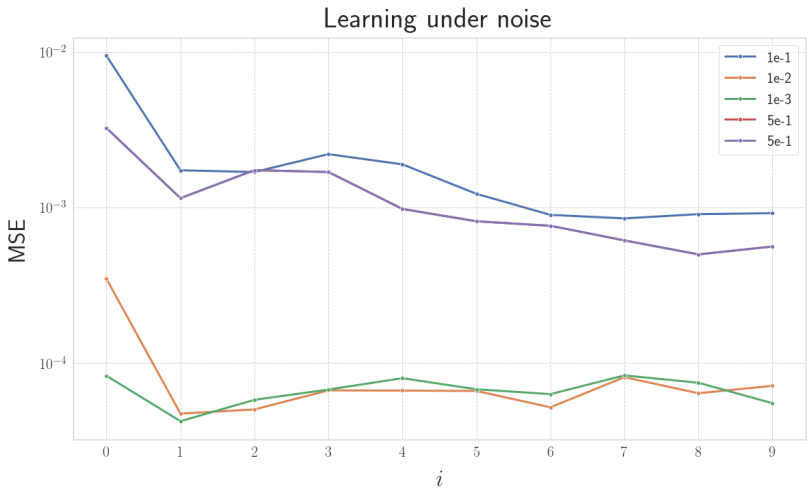


Fig. 5.7.: State trajectories computed by ∂PVP_v and DDP for different initial preview horizons T.

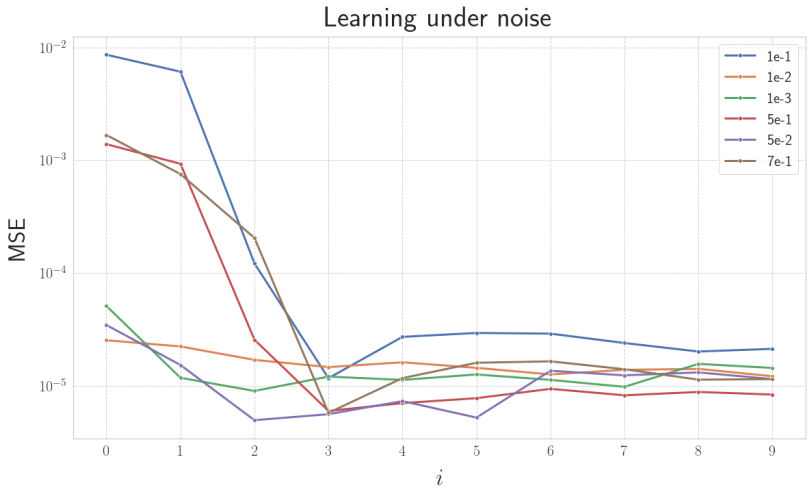
5.2.4 Convergence Issues: Singularities

The considered control systems show evident symmetries not expected by the algorithm and these symmetries come with singular points - i.e initial states from which multiple (typically symmetric) optimal trajectories exist. As we can see in Figures 5.10a and 5.10b, the algorithm quickly learns the overall topology of the value function across state space. As the number of iterations increases, the algorithm seems to refine the inherent symmetry in the topology. We observe that the iterative aspect of ∂PVP_v allows it to learn value function over long horizons quite well despite initialization in the short horizons. However, there seem to be regions in the configuration space difficult to handle. We suspect that the non-holonomic constraints in the Unicycle environment lead to singularities which in turn destabilizes learning. With more learning, the algorithm overcomes the presence of singularities. By the 10^{th} iteration, the algorithm had narrowed the location of singularities to be symmetrically distributed around the goal position.

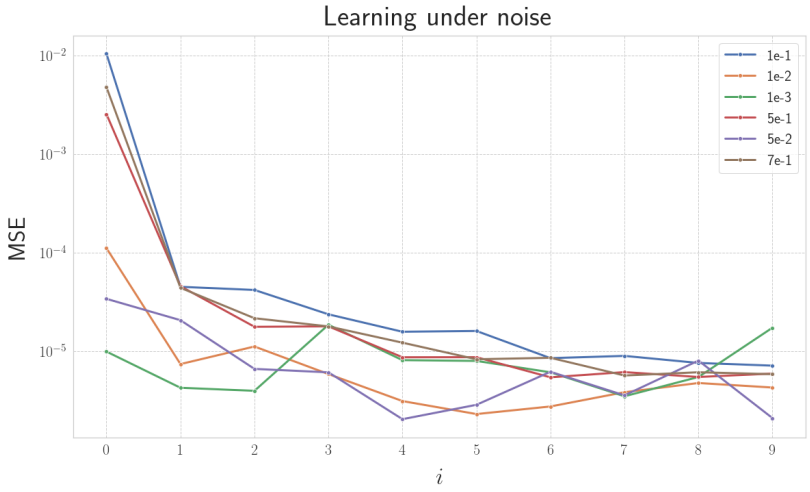
In our initial experiments, we understood singularities as special points in the state space from which, as noted above, multiple optimal trajectories are possible. This,



(a) ∂PVP$_v$ with noise initialization. Shown here for Unicycle



(b) ∂PVP$_v$ with noise initialization. Shown here for Cart-Pole



(c) ∂PVP$_v$ with noise initialization. Shown here for Pendulum

Fig. 5.8.: Illustration of the evolution of mean squared error between V_α for Unicycle, Cart-Pole, Pendulum under different noise initialization. The colors indicate the noise added to the initial dataset.

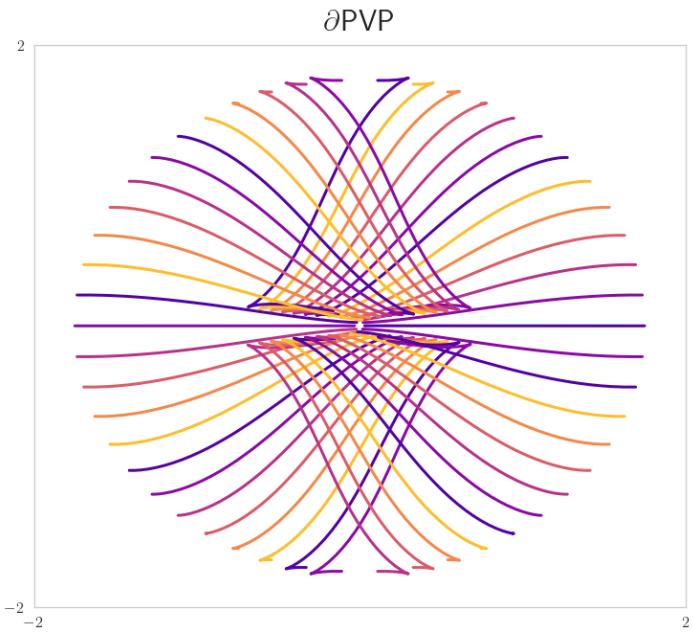


Fig. 5.9.: Unicycle trajectories computed by ∂PVP_v from multiple starting configurations. The goal is to reach the center/origin. The x, y axes denote q_1, q_2 of the state space for Unicycle. The colors of the trajectory do not imply any properties.

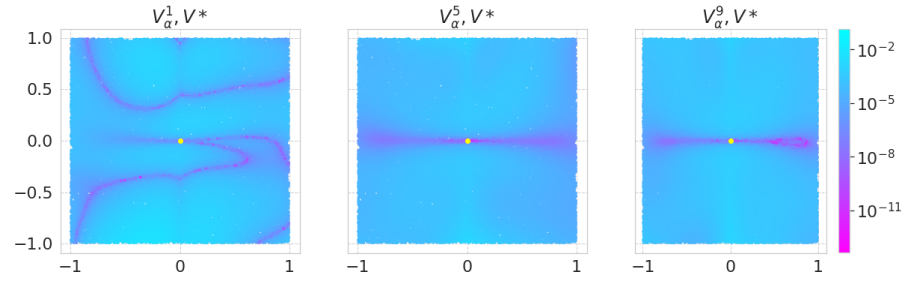
in turn, confuses the solver. However, our experiments showed that it not trivial to narrow down the origin of causes that destabilize learning.

We, therefore, use the term 'singularities' as a blanket word that covers all situations that can destabilize learning - from pathological behaviour of the solver, inaccuracy of hessians computed by the solver, to issues due to initialization in short horizon. This, of course, has the detrimental effect of not knowing precisely the nature or the cause of the problem, which, in turn, can be frustrating to resolve. This was one of the major problems we faced in estimating the global value function. Since the presence of these singularities seemed to destabilize learning, we modeled these points as outliers by tuning the cost weights in the corresponding optimal control problem such that the resulting value function computed by DDP was below 1. This *normalization* trick alleviated the problems associated with learning, however, it required extensive tuning.

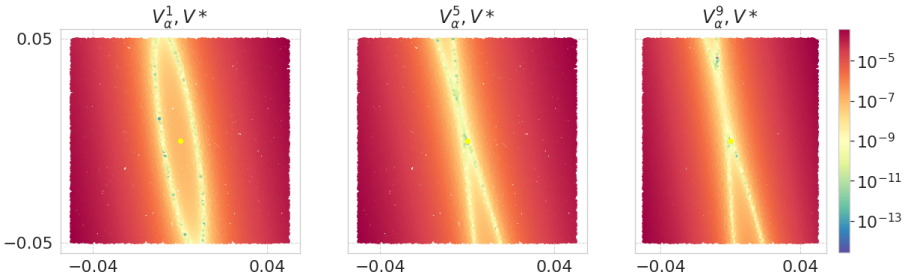
5.2.5 Importance of Sobolev Loss

Our experiments with Sobolev learning corroborate the generalization capabilities and confirm that Sobolev regression requires fewer training epochs than classical regression, see Figure 5.12. Sobolev training requires only 64 samples to achieve a higher accuracy than classical regression on the 0^{th} order output.

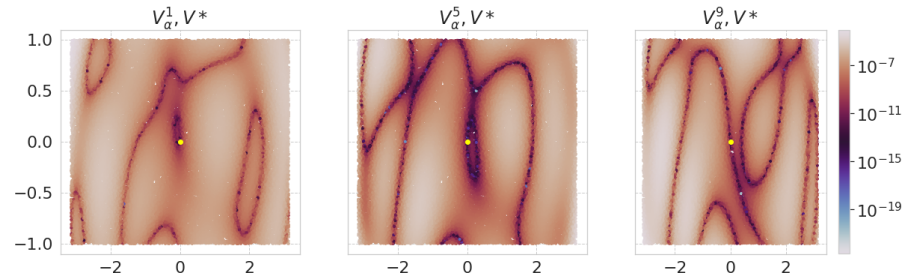
The effect of Sobolev training is also seen in the improvement of the quality of gradients, especially with our residual network. As mentioned in Chapter 3 for our experiments with value function we designed V_α as a 3 layered residual network with hyperbolic tangent as an activation function and 64 units in each hidden layers. The final residual layer contains 3 units. Empirically, we find that the advantage of modeling the value function as a squared residual lead to faster and more stable convergence during Sobolev training as shown in Figure 5.11. It seems that a simple feed-forward neural network, i.e a model with one output, is more unstable than V_α. The gradients of the residual network are also more accurate than those of feed-forward network as shown on Figure 5.13. Additionally, we benefit from being



(a) Unicycle



(b) Cart-Pole



(c) Pendulum

Fig. 5.10.: Illustration of evolution of mean squared error between V_α after 1, 5 and 10 iterations and V^* for Unicycle, Cart-Pole, Pendulum

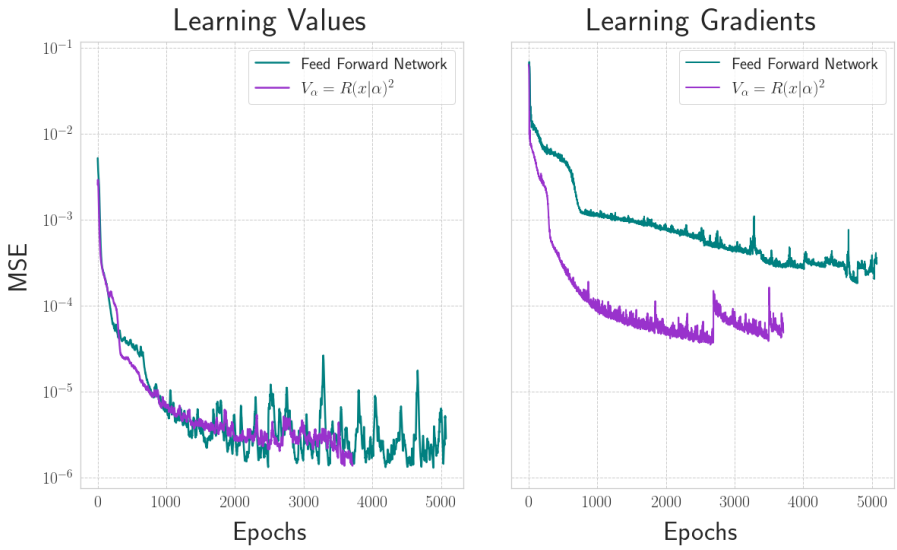


Fig. 5.11.: Sobolev Loss Curves

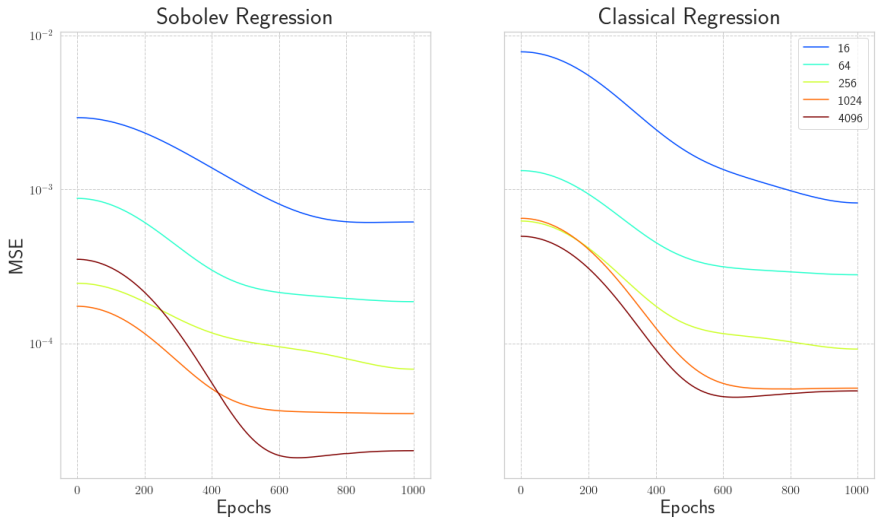
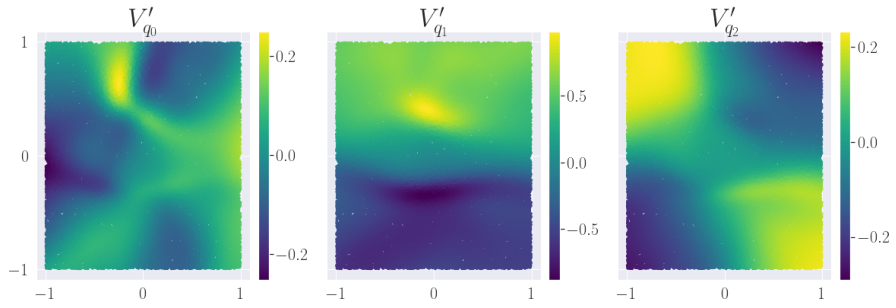
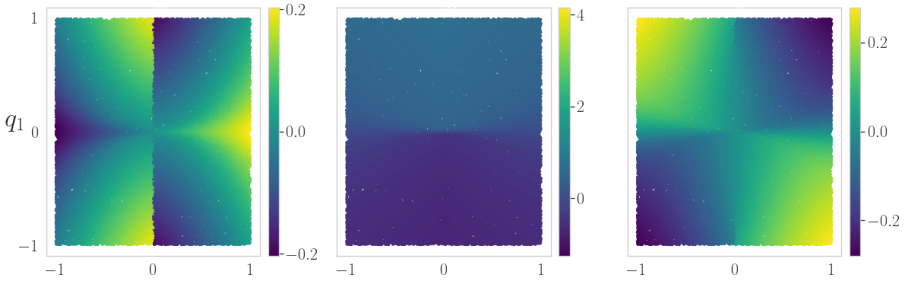


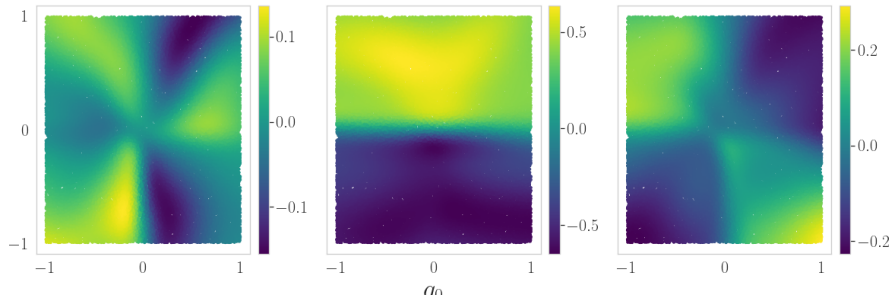
Fig. 5.12.: Comparison of 0^{th} loss curve during training - Sobolev and classical.



(a) Gradients of Feed Forward Network modeling value function with one output



(b) Gradients computed by DDP for locally long optimal trajectories - V'^*



(c) Gradients of V_α

Fig. 5.13.: Illustration of gradients of a simple feed-forward network, V'^* and V'_α

able to make Gauss approximation of the Hessian, since computing exact Hessians of the neural network can become prohibitively expensive when the number of hidden layers increases.

5.2.6 Conclusion

We have established the basic properties of our algorithm in the preceding sections. We have shown that ∂PVP_v converges to V_* (Section 5.2.1) with high accuracy even

with short rollouts (Section 5.2.2) wrong initialization (5.2.3) and singularities (Section 5.2.4. We also showed that this is in part also due to the proper use of derivatives in learning and in the trajectory optimization through a residual network (Section 5.2.5.

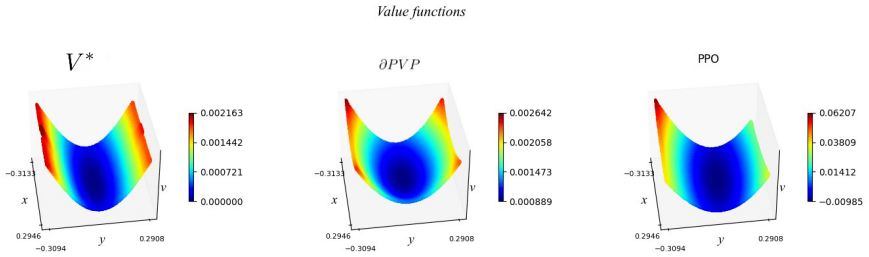
We will now compare our algorithm with a classical algorithm of RL - the solver PPO.

5.3 Comparison with PPO

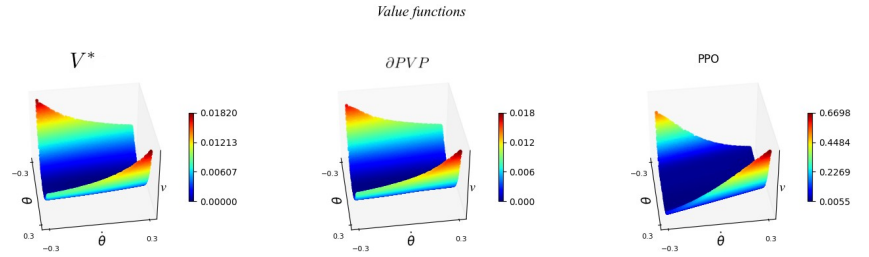
In Figure 5.14, shows the qualitative comparison of the value functions predicted for Unicycle, Pendulum, and Cart-Pole by ∂PVP_v and PPO against *ground truth* V^*. PPO properly captures the overall shape and the spread of the topology but overestimates it. This is to be expected since policy gradient methods often fail to accurately model value function as empirically established in [Ily+20].

From our experience, PPO was also more sensitive to small changes, either to the environment parameters (e.g seed, discount factor, learning rate) or algorithm hyperparameters which limited the experiments we have been able to carry out. From a practical perspective, using the DDP solver to act as the environment for the PPO agent is not straightforward and required excessive and extremely time-consuming tuning from adjusting the discount factor to the learning rate. While theoretically, PPO should converge with any discount factor, we observed a performance degradation for discount factors below 0.9 or above 0.99. This is not a limitation we observe with ∂PVP_v. Similarly, the training time for PPO is usually vastly greater than ∂PVP_v and depends drastically on hyperparameters. In our experiments, PPO took approximately $55, 37, 20$ minutes to converge while ∂PVP_v required $8, 5, 6$ minutes of training for Unicycle, Cart-Pole, and the Pendulum problems.

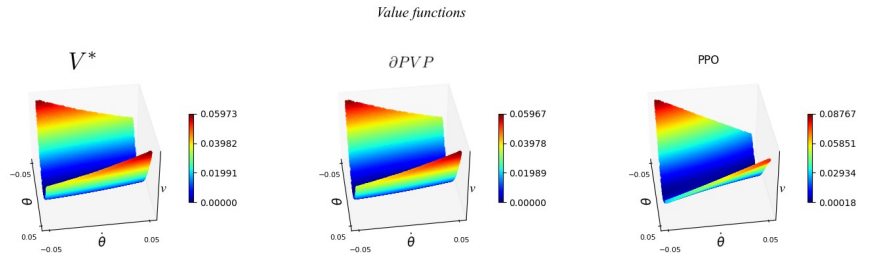
In conclusion, PPO does not manage to build but a coarse approximation of the value function. This inaccuracy in modeling the value function then has a negative impact on the accuracy of the corresponding policy as shown in Figure 5.15. On the other



(a) Unicycle

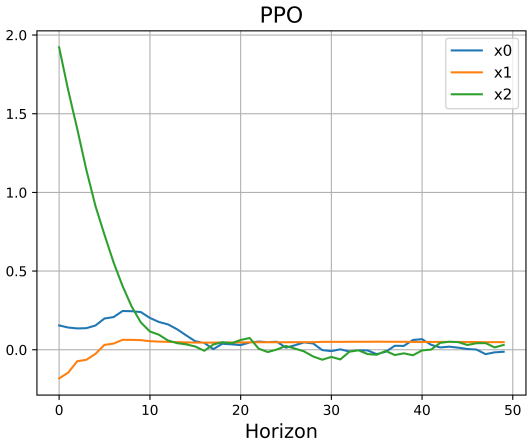


(b) Pendulum

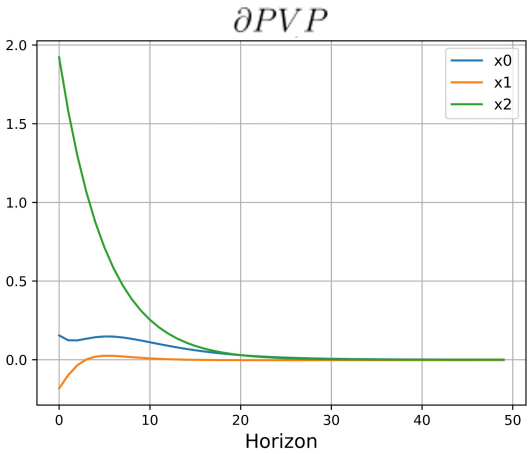


(c) Cart-Pole

Fig. 5.14.: Comparison of value functions between the solver, ∂PVP and PPO, shown here for Unicyce, Pendulum, Cart-Pole



(a) State Trajectories computed by PPO for the Unicycle goal reaching task.



(b) State Trajectories computed by ∂PVP$_v$ for the identical Unicycle goal reaching task.

Fig. 5.15.: Comparison of state trajectories computed by PPO and ∂PVP$_v$. $x0, x1, x2$ denote the 3 dimensions of the unicycle system.

hand, this allows us to better evaluate the accuracy of our algorithm which manages to quickly reach accuracy levels out of the scope of derivative-free RL solvers.

We will now show how ∂PVP$_v$ scales to higher dimensional system.

5.4 Application to the 7 *dof* Manipulator Arm

In this section, we show the scaling of our algorithm[1]. We consider a 7 *dof* manipulator, controlled in torque, where the robot state $x = (q, \dot{q})$ concatenates at the joint configuration and velocity and $u = Z_q$ are the joint torques. Torque control [Lam11] is seen as an important feature that allows robots to physically interact with their environments rather than in the usual controlled settings of factories and laboratories. However, it also raises important challenges, in particular instability, to levels that are not meant for position-controlled robots. We know that RL algorithms are typically strongly impacted by this instability and this is the reason why we chose to quantify the performance of our algorithm on the torque-controlled manipulator arm. The optimal control task and the manipulator itself is described in more detail in Chapter 4.4. The dynamics are computed using Pinocchio [Car+19] and policy trials are validated with Bullet [CB21].

5.4.1 Results

Computing a huge validation dataset that can be used *in lieu* of V^* is infeasible for the 7 *dof* manipulator arm. We have to rely on Bellman residuals to measure the convergence of ∂PVP_v for the EE pose-reaching task. In each iteration of the supervised Sobolev training phase, 100 locally optimal samples of horizon length 150 were drawn from the 14-dimensional configuration space. Figure 5.17 shows the difference in predictions between two successive iterations, i.e mean squared error between V_α^{i+1} and V_α^i. We see that the residual stabilizes by the 10^{th} iteration thus signifying that the algorithm has converged. We see the smoothness of the predicted value function, and its gradients, across the state space at convergence in Figure 5.18. The smoothness of the value function then allows the corresponding policy to behave smoothly, rather than develop *jerky* motions.

[1]The work presented here was done in collaboration with Sebastien Kleff

The primary feature of ∂PVP_v is that as iterations increase, the approximated value function asymptotes to the global time-independent value function. So, when used as a proxy for terminal cost functional, ∂PVP_v tends to drive the locally optimal solver toward the globally optimal solution. This immediately constrains the corresponding trajectories to satisfy the Hamilton-Jacobi-Bellman criteria of optimal sub-structures: sub-solutions of an indefinite horizon optimal control problem should also be optimal solutions to the corresponding definite horizon sub-problems. We can see this quite easily in Figure 5.16b for the EE trajectory computed by ∂PVP_v (in orange) and the ground truth EE trajectory computed by DDP at horizon 1000 (shown in blue). The EE trajectories for the truncated horizon problem are co-incident with the infinite horizon trajectory.

The trajectories computed with ∂PVP_v and DDP, for the truncated horizon, also maintain the recursive optimality and stability when used online in simulation. The ∂PVP_v terminal cost can also serve as a highly stable anchor that allows for quick re-planning online under external disturbances. Figure 5.19 shows the evolution of mean squared errors between EE trajectories computed by ∂PVP_v and DDP at infinite-horizon when external perturbations are injected in the system. We observe that ∂PVP_v recovers quickly when perturbed at intervals[2]. In Figure 5.16a we show the generalizability of our algorithm to compute optimal trajectories for different starting configurations. We approximate the terminal value function with the 10^{th} iteration of ∂PVP_v.

We discussed the two sampling strategies - Adaptive and refined sub-sampling - in Chapter 3.3.2. We experimented extensively to define a general sampling strategy that can be used in any system. One of our primary focuses was to reduce the overall training time and for that reason, we experimented with sub-sampling for more states from the state trajectory computed by DDP and creating a weighted dataset for learning. In our experiments with refined sub-sampling, we often encountered the problem of *overfitting* [Yin19]. However, this problem can be mitigated to some extent by utilizing the initial configurations generated in every iteration. In Figure

[2]The corresponding video is available at `https://peertube.laas.fr/w/16ocWJHhVwxNr4qYkjGNEX?start=3m39s`

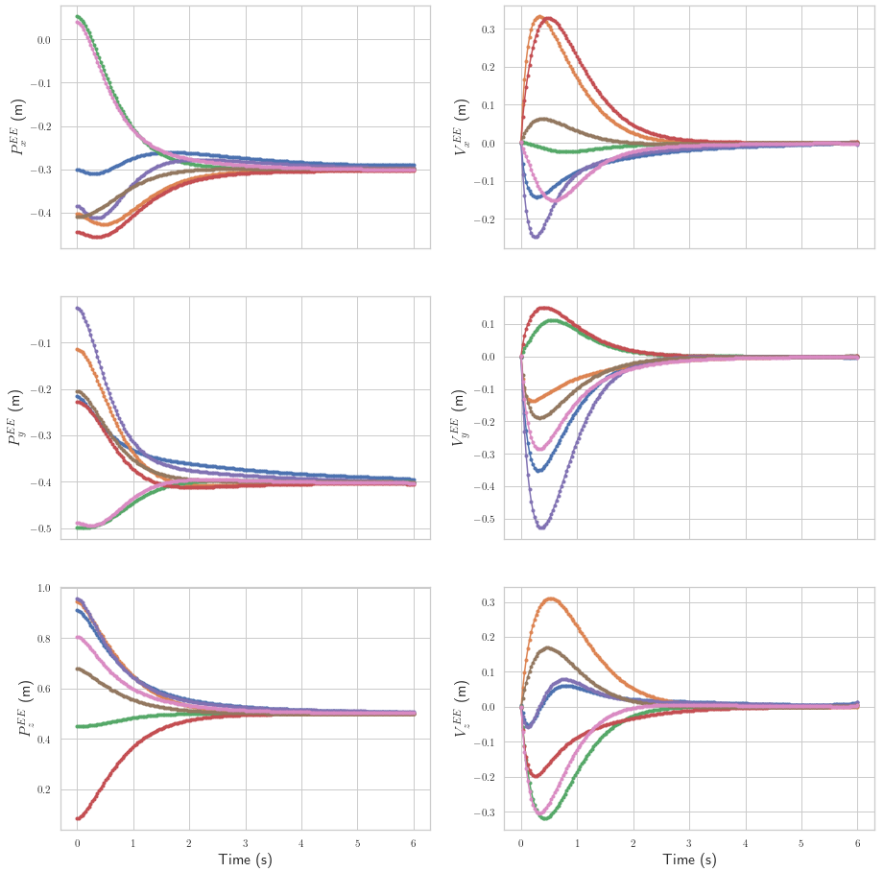
5.20, we see the trajectories computed as ∂PVP$_v$ iterates. In the initial iteration, the estimate provided by the neural network is far from V^* and leads to divergence in the corresponding state trajectory. The trajectories for $i < 6$ seem to diverge at the terminal position effectively implying that the learning converges on the 6^{th} iteration. The state trajectory computed by ∂PVP$_v$ at $i = 6$ is coincident with a locally long *ground truth* reference.

On the other hand, the data provided by adaptive sampling led to much better generalization, albeit at the expense of slightly higher computation costs incurred due to using Inverse Kinematics. The advantage provided by adaptive sampling is exactly the opposite of refined sub-sampling - refined sub-sampling concentrates the data around the target position as seen in Figure 3.4 whereas adaptive sampling, in Figure 3.2, provides a much more varied samples. For the results that we showed here, we used the more general Adaptive sampling approach, described in 3.3.2, to compute initial starting positions across the configuration space.

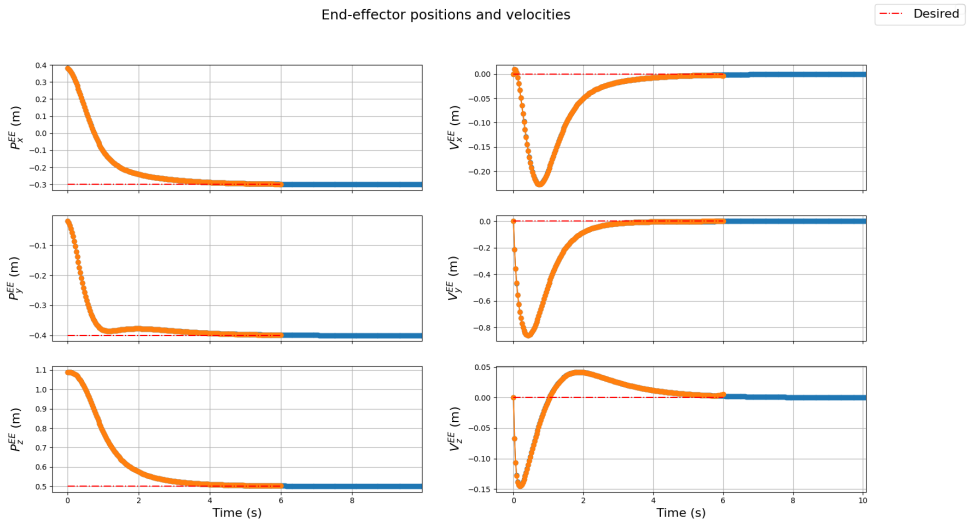
5.5 Discussion and Conclusion

In this Chapter, we showed key results regarding the foundational practical implementation of our algorithm. To summarize, the main features of ∂PVP$_v$ are :

- converges stably and quickly
- adjusts to noise during training
- uses gradients for better generalization
- faster and more data efficient than standard RL algorithms
- scales to difficult problems such as torque controlled 7 *dof* end-effector pose reaching task



(a) Trajectories computed by ∂PVP_v, shown here for the End-Effector.



(b) End-Effector trajectories computed by ∂PVP_v (orange) after 10 iterations and a locally long DDP (blue) trajectory.

Fig. 5.16.: Ilustration of End-Effector trajectories computed by ∂PVP_v and DDP.

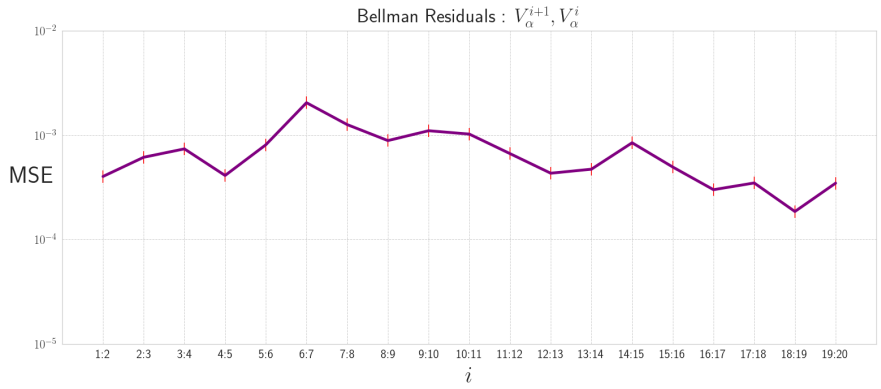
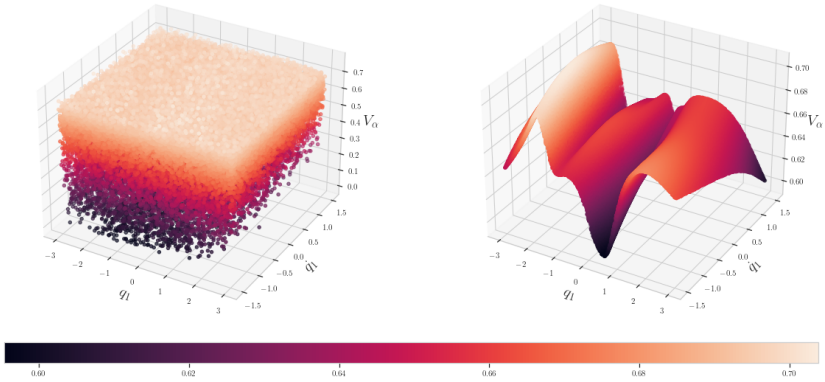


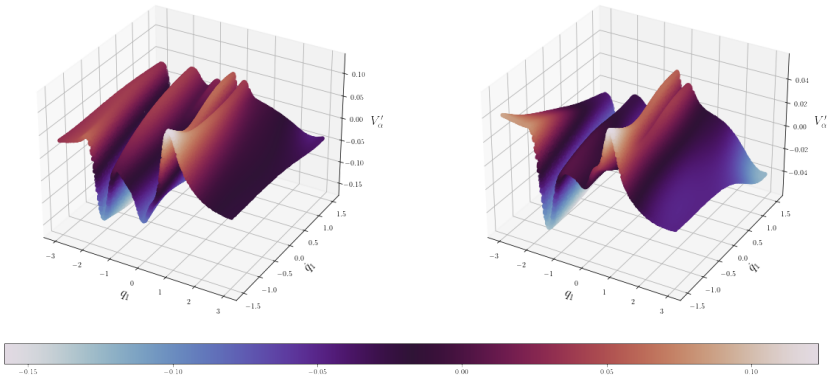
Fig. 5.17.: Bellman Residuals for manipulator arm

- provides stability when used online, even when perturbations are injected into the system
- estimates value function comparatively than PPO

The iterative supervised learning phase combined with Sobolev regression enforces the stability of predictions and produces consistent results. Learning in the backdrop of Bellman's optimality principle makes the convergence of learning less dependent on hyperparameters (e.g discount factor, learning rate). However, we also need derivatives and this prevents us yet from applying the algorithm to any problem. Interestingly enough, numerous subtleties in the implementation of the algorithm such as the design of the value function approximator or use of 2^{nd} order derivatives and singularities make the implementation quite complex. Similarly, formulating a general-purpose sampling strategy that can be used with any system remains quite challenging. The need for a general-purpose sampling scheme can be seen as a strategy to optimally explore the environment. In our formulation, this exploration is done by the Trajectory Optimizer that not only gives us the next state as is usual in classical RL algorithms but also the states along a preview horizon. Usually, the first state in a state trajectory is by far more reliable than other states and we use this first state in training while discarding the rest. This is one the core reasons behind the need for a proper sampling strategy - to make the algorithm more data efficient than it currently is.

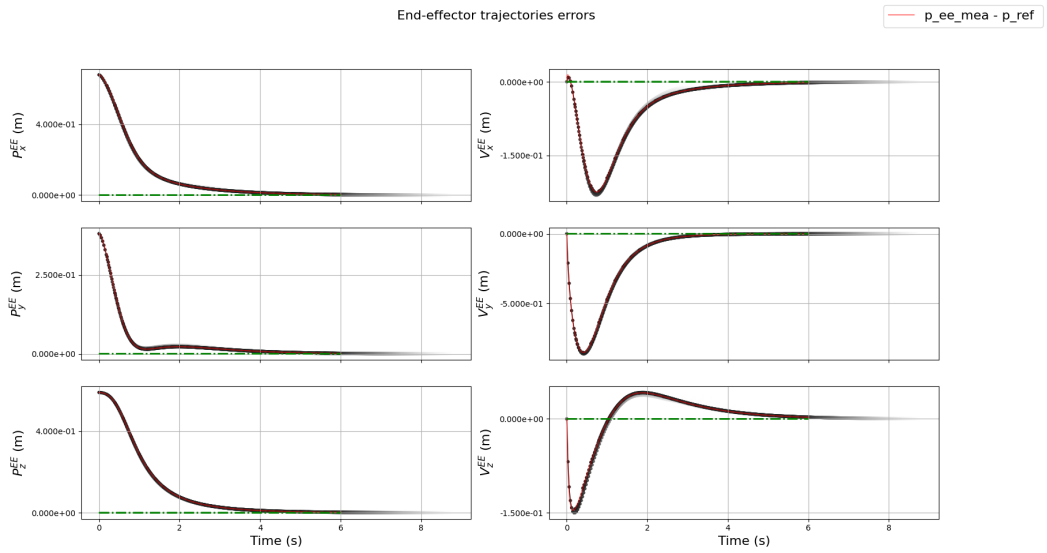


(a) Value functions predicted by $V_\alpha^{i=9}$. (left) Predicted value functions for random starting configurations in 14-dimensional state space, shown for $q_1, \dot{q}_1$. (right) Slice of value function with only $q_1, \dot{q}_1$ randomly sampled from the configuration space

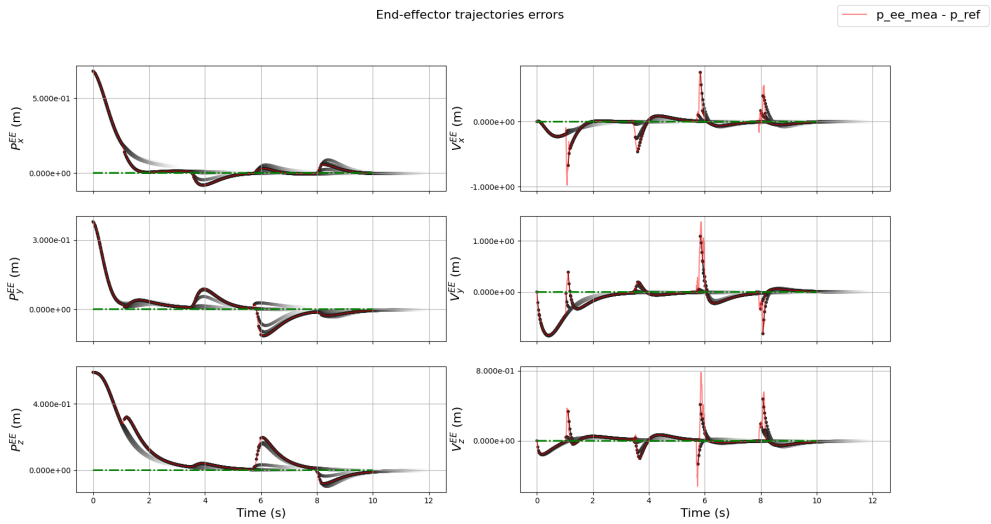


(b) Slice of $V_\theta^{'9}$ at q_1 (left) and $\dot{q}_1$ (right). $q_1, \dot{q}_1$ were randomly sampled for the configuration space.

Fig. 5.18.: Illustration of predictions, value, and gradients of value, for the manipulator arm.



(a) Model Predictive Control without perturbation



(b) Model Predictive Control with perturbations manually injected into the system

Fig. 5.19.: Illustration of MSE for manipulator arm in MPC.

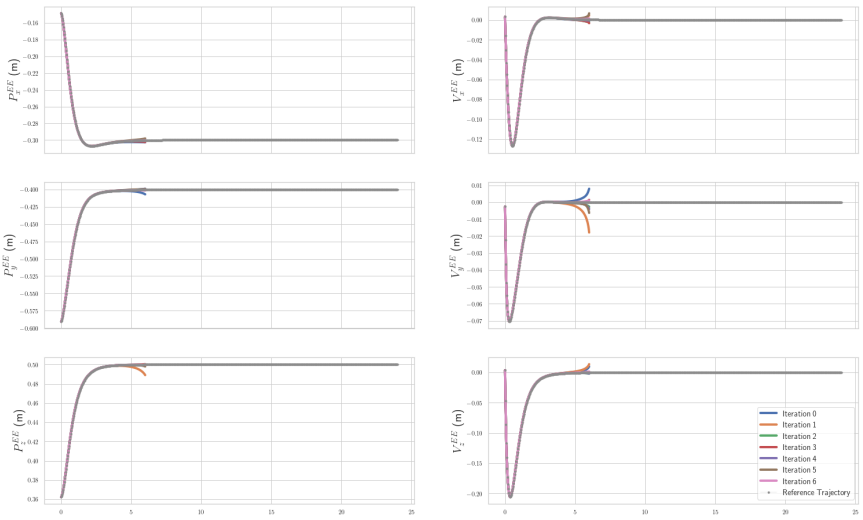


Fig. 5.20.: Multiple trajectories computed by ∂PVP_v with the refine sub-sample approach.

6

∂PVP$_{x,u}$ - Learning Warmstarts

> *A good speech isn't one where we can prove he's telling the truth. It's one in which nobody else can prove he's lying!*
>
> **— Sir Humphrey Appleby**
> Yes Minister

Contents

6.1 Problem statement and Experimental Setup 94
6.1.1 Introduction 94
6.1.2 Experimental objectives and setup outline 95
6.2 Results 96
6.3 Discussion and Conclusion 97

In this Chapter, we show the results of learning warmstarts. In Section 6.1 we summarize the problem statement and establish the experimental setup. In Sections 6.2 we show our results of learning state-control trajectories for the 3 classic control systems and the 7 *dof* Manipulator arm. We discuss the key role played by warmstarts before concluding in Section 6.3.

6.1 Problem statement and Experimental Setup

6.1.1 Introduction

In the previous Chapter, we used a value function approximator V_α to learn the time-dependent value function. We kept replacing the terminal cost with V_α and iterating with Sobolev regression to force the value function estimates toward infinite horizon.

In this chapter, we use two additional function approximators X_β, U_γ to learn the corresponding state-control trajectories, which we then use to warm-start DDP in subsequent iterations. We also learn the value function, but the results regarding the properties of V_α are identical to those shown in the previous Chapter.

Recall that the complete algorithm that we use is :

Algorithm 5: ∂PVP

Algorithm parameters: horizon length T, iterations i, sample size s;
Initialize V_α, X_β, U_γ;
Initialize DDP ;
foreach i **do**
  Sample s trajectories from DDP ;
  Train V_α through (3.15b) ;
  Train X_β through (3.15a) ;
  Train U_γ through (3.15a) ;
  Update DDP terminal cost $\ell_T \leftarrow V_\alpha$;
  Warmstart DDP through X_β, U_γ
end foreach

In this Chapter, we show the results regarding the training of X_β, U_γ, and the subsequent effect DDP has when warmstarted.

A few important points of interest in learning warmstarts are as follows :

1. X_β, U_γ are used to provide initial guesses to the trajectory optimizer. This immediately leads to a trade-off between the quality of predictions, training time, the predefined level of precision inside our choice of trajectory optimizer,

and the number of iterations eventually required by the trajectory optimizer to converge to a solution.

2. Architectural design leads to numerous trade-offs between ease of training, accuracy, and learning time.

3. The DDP solver provides us with Riccati Gains which can be used for Sobolev Regression for U_γ. However, this immediately leads to another trade-off between the dimensionality of U_γ and precision - it may become computationally infeasible to include Riccati Gains in the training loop since it would then require differentiating U_γ with respect to some state input x_0.

These points allow us to define the experimental objective and setup for ∂PVP.

6.1.2 Experimental objectives and setup outline

Our primary goal was to minimize the training time such that ∂PVP achieves superlinear convergence in the number of iterations taken by DDP to solve any problem when warmstarted. In our experiments, we observed that the precision of trajectories predicted by X_β, U_γ increases only slightly as ∂PVP iterates. This allows us flexibility in defining the number of training epochs in every iteration, which we need if the dimensionality of the problem under consideration increases. Therefore, we choose to learn and improve state-control trajectories only in the initial and final iterations of ∂PVP: the total number of iterations is dependent on the user. We found this training strategy, where estimations of value functions are refined at every iteration, whereas estimation of state-control trajectories is refined only at the start and end to be good enough for our purposes.

In our initial experiments, we chose to enforce connections between $V_\alpha, X_\beta, U_\gamma$. Yet, the efficiency also strongly depends on practical implementation choices, in particular, the network architecture. Empirical evaluations led us to believe that the simplest approach can lead to better results. So we decided to learn representations of the value function, state, and control trajectories on 3 separate neural networks.

To enforce a shorter training time, we discarded the use of second-order derivatives of the value function in the previous chapter. Experiments with Riccati Gains yielded precisely the identical consideration. To keep training time to a minimum, we precluded the use of first-order but high-dimensional derivatives of the control trajectory. Therefore, X_β, U_γ were trained using classical 0^{th} order regression.

In the next section, we compare the state-control trajectories predicted by X_β, U_γ at the end of ∂PVP with the corresponding ones from DDP.

6.2 Results

Our initial experiments were conducted on the **Unicycle** problem to establish benchmarks because of the simplicity (we thought !) of the corresponding optimal control problem. In Figure 6.1, we see the control and state trajectories predicted by U_γ, X_β for 4 different starting configuration. We observe that even though the quality of warm-start provided by U_γ, X_β is good enough, there is hardly any appreciable decrease in the number of roll-outs/iterations required by DDP when warmstarted. In Figure 6.2a we warmstart ∂PVP for 500 different starting configurations to compare the number of roll-outs needed by cold-started DDP. We hardly see any appreciable difference between the roll-outs required by ∂PVP and DDP.

The resolution of this discrepancy seems to be in the predefined precision factor in our DDP solver. Crocoddyl, by default, operates at $1e-9$ precision. If the precision factor is reset to $1e-6$, then we see in Figure 6.2b the number of roll-outs required by ∂PVP to be significantly lower[1]. This was the only unexpected result since the precision factor should not play that much of a role.

That is what we observe in Figure 6.3 for Cart-Pole balancing task, Figure 6.4 for the Inverted Pendulum problem and the End-Effector pose reaching task in Figure 6.5. The precision factor of $1e-9$ does not seem to play that much of a role and our

[1]Overall, I conclude dark wizards and dementors to involved with the unicycle problem

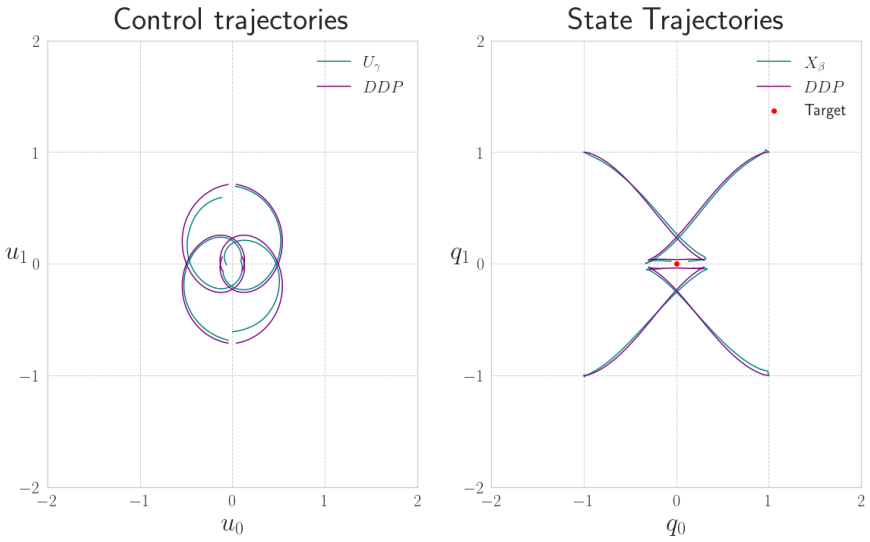


Fig. 6.1.: Warmstarts provided by U_γ, X_β for the goal-reaching task from different starting configurations in the Unicycle problem.

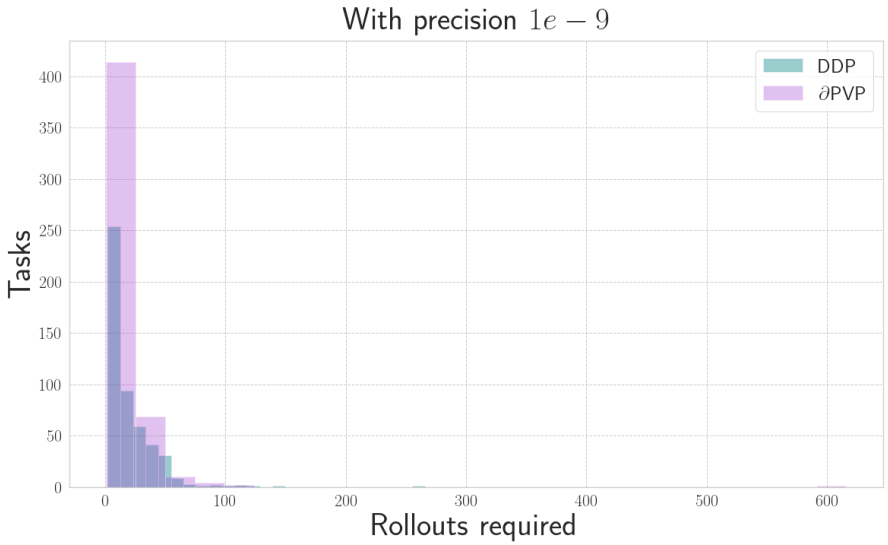
algorithm achieves super-linear convergence in the number of roll-outs required to solve an optimal control problem.

6.3 Discussion and Conclusion

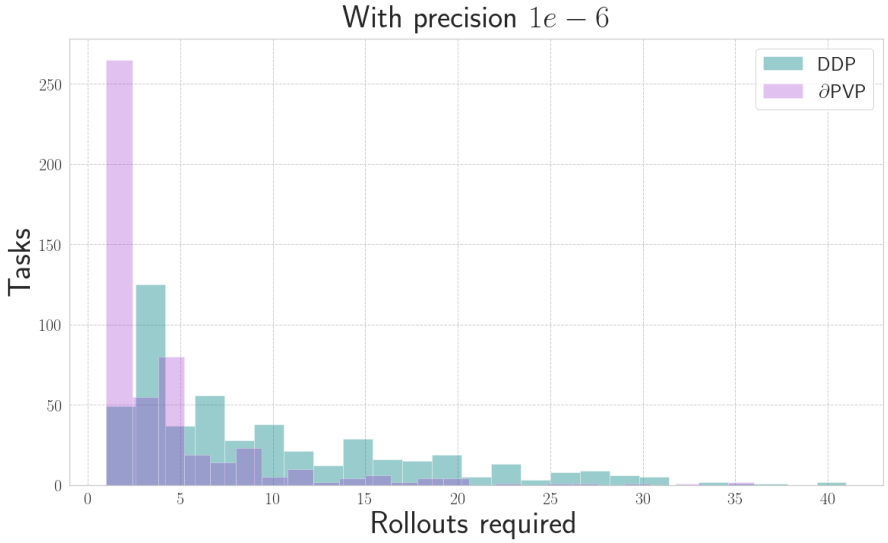
This chapter marks the end of the experimental evaluations of learning warmstarts. We showed the critical role of warmstarts in Trajectory Optimization and how it can lead to superlinear convergence in the number of attempts required by trajectory optimizer to converge to a solution. In particular, we studied

1. the role of learning state-control trajectories for warmstart in reducing the number of iterations required by DDP to converge.

2. the trade-offs between the accuracy of the learned representation of state-control trajectories and the effect it has in trajectory optimizer.

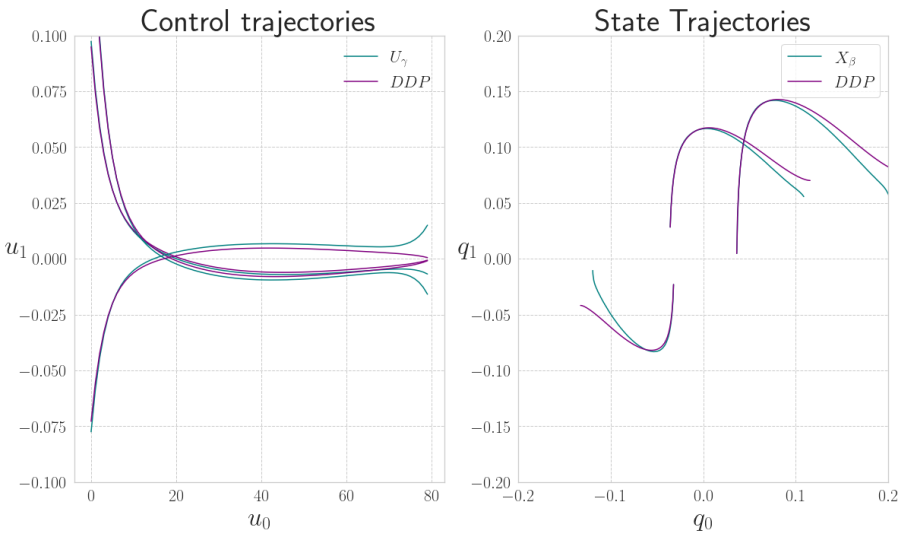


(a) ∂PVP vs DDP at precision of $1e-9$. The average number of rollouts across 500 tasks required by ∂PVP is 15.3 while DDP requires 19.812 rollouts to solve one task.

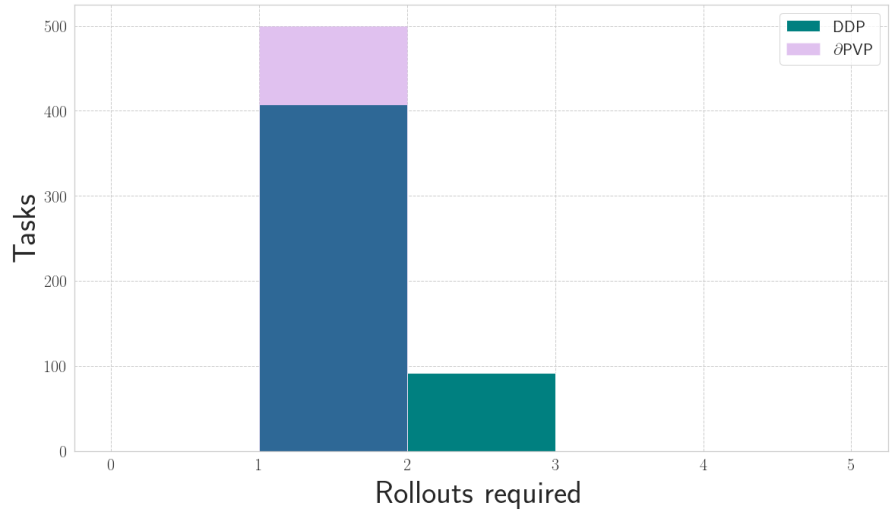


(b) ∂PVP vs DDP at $1e-6$. The average number of rollouts required across 500 tasks are 4.03 and 9.706 by ∂PVP and DDP respectively.

Fig. 6.2.: Illustration of roll-outs required for the unicycle task from 500 different starting configurations at precision factors of $1e-9$ and $1e-6$

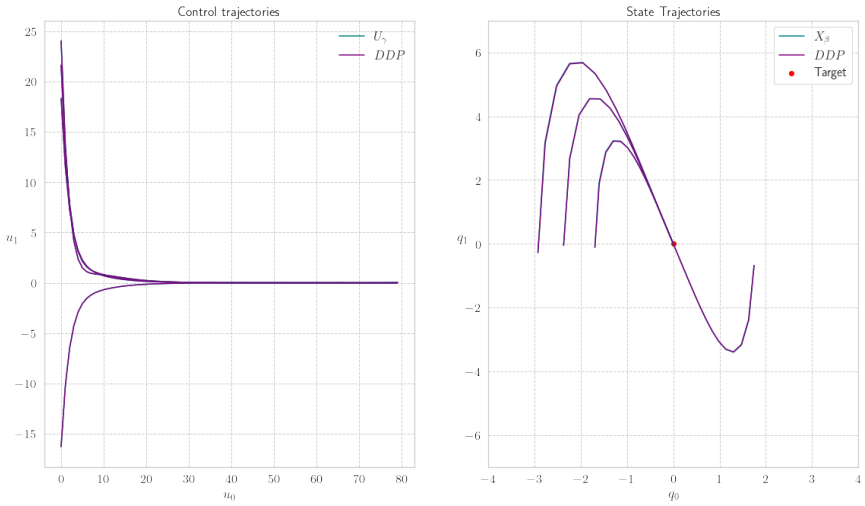


(a) Trajectories predicted by U_γ, X_β vs those of DDP

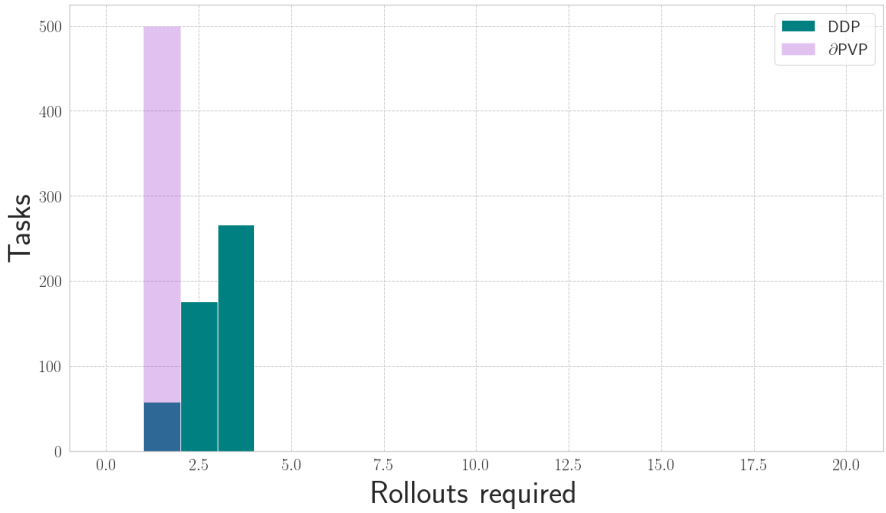


(b) Histogram of rollouts required by ∂PVP and DDP across 500 tasks

Fig. 6.3.: Cart-Pole : warmstart and roll-out - ∂PVP and DDP.

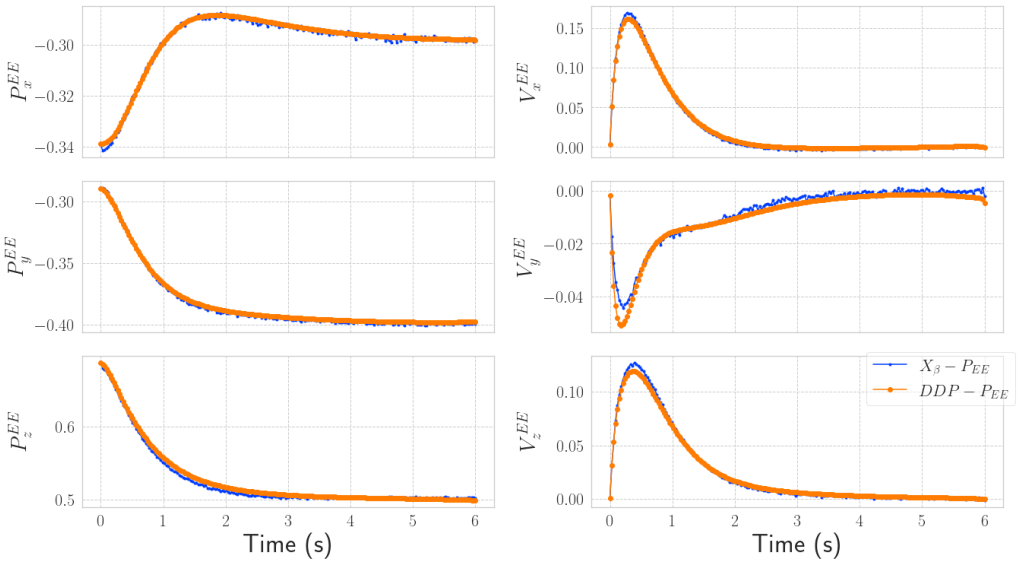


(a) Trajectories predicted by U_γ, X_β vs those of DDP

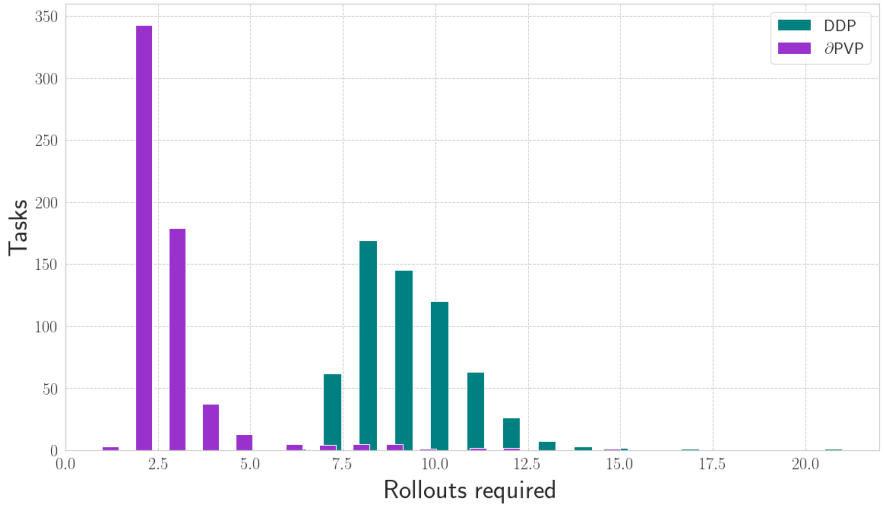


(b) Histogram of roll-outs required by ∂PVP and DDP across 500 tasks

Fig. 6.4.: Pendulum : warm-start and roll-out - ∂PVP and DDP.



(a) End Effector trajectories predicted by X_β vs DDP



(b) Histogram of roll-outs required by ∂PVP and DDP across 500 pose reaching tasks.

Fig. 6.5.: 7 *dof* Manipulator pose reaching task : warm-start and roll-out - ∂PVP and DDP. 200 were drawn from the 14 dimensional configuration space in each of the 20 iterations. This resulted in our TO computing 10347 rollouts overall. The training phase lasted 61 minutes.

3. the choice of the architectural design of ∂PVP given trade-offs between precision, computation time - of sampling for data and training in an iterative loop - and extraction of useful information from hidden layers.

Our original goal was to develop a new paradigm for RL algorithms that takes advantage of the rich data provided by simulation-based TO to prevent an exhaustive exploration of the state space. In the preceding Chapter, we first showed practical implementation and empirical evaluations of our proposed method to learn the global time-independent value function. We also showed that replacing terminal cost with the time-independent value function induces stability and tries to enforce optimality of the corresponding state-control trajectories computed by TO.

However, learning value function still does not lead to superlinear convergence - the trajectory optimizer still requires quite a few iterations to converge to a solution. This is where the second part of our algorithm is important - providing a good guess to the TO improves its efficiency. Overall, the nature of ∂PVP allows it to improve, in each iteration, the estimate of the global value function. This, in turn, improves the trajectory computed by TO. Subsequently, the refined and improved trajectories improve the learning of state-control trajectories by X_β, U_γ, and consequently as the predictions of X_β, U_γ become more accurate, the greater their role becomes in reducing the number of rollouts required by TO. This is the central aspect that we were trying to achieve with our algorithm - an iterative loop between learning and TO such that in each iteration, the learning (of value function, state-control trajectories) improve along with the quality of computations of TO.

While theoretically, at the end of the iterative loop, TO coupled with learning should allow for optimality and superlinear convergence since the HJB conditions should, in principle, force the predictions to optimality criteria, in practice, however, the implementation of ∂PVP is not so straightforward. Numerous experiments were needed to establish the degrees of efficiency of sampling approaches, design of neural networks, early stopping criteria during the learning phase along with extensive tuning of the robotic systems itself. The practical implementation of ∂PVP also guides our choice of the architectural design of X_β, U_γ and the associated increase

in training time precludes the use of Riccati Gains during training. Certain factors such as the stopping criteria in the DDP instance that we were using, also seemed to play a role, which we did not foresee in our initial experiments. This opens up a discussion on the accuracy required for real-world robotic systems, which we believe has to be established in an *ad-hoc* manner, depending on the system and the task in consideration - for instance, in grasping tasks far more accuracy is required than in reaching a target. On the whole, our contributions in the two preceding chapters provide a foundation for the further development of efficient RL-based approaches in robotics.

With the insights gained from our experiments, we can establish a road-map for the future of ∂PVP :

- Develop an efficient sampling approach for any n dimensional system.
- Incorporate the mathematical structure of TO in the design of the neural networks, for instance using time as an input state during learning.
- Explore the reliability of the data computed by TO for training in the initial iterations.
- Design a supervised training method that explicitly uses HJB equations and symmetries in loss functions.
- Engineer an automated identification protocol that can recognize singularities such that its impact during training is minimized.
- For faster deployment on a cluster and to remove the overhead, the $C++$ version of deep learning libraries should be directly used to close the gap between TO and learning.

We conclude in the following Chapter.

Conclusion

7

> *One last time.*
> *Relax, have a drink with me.*
> *One last time.*
> *Let's take a break tonight.*
> *And then we'll teach them how to say goodbye.*
> *To say goodbye.*
> *You and I.*
>
> **— G. Washington**
> in **Alexander Hamilton**

The primary aim of this thesis was to devise an end-to-end learning framework for robotics. To that end, we reformulated reinforcement learning as a combination of the iterative supervised learning phase, with emphasis on value functions and warmstarts - ∂PVP. This allowed for a reduction in trials needed to find an optimal solution. The iterative supervised learning phase enforced the stability of predictions through Sobolev regression while learning in the backdrop of recursive optimality further reduced the dependence on the hyperparameters.

Our goals have been to :

- Develop a learning approach such that a neural network V_α represents V^*.
- Learn representations of state-control trajectories through locally optimal runs of a Trajectory Optimizer.

We then used V_α to provide the *cost-to-go* at the terminal position. This effectively allowed us to formulate an infinite-horizon problem that can remain solvable with

finite resources. Using function approximation to learn state-control trajectories was much more straightforward. The iterative nature of our algorithm frequently led to trade-offs between precision and training time. For that reason, we discarded the use of derivatives of control and the second-order derivatives of the value function. V_α, X_β, U_γ were initially implemented as the outcome of a three-headed feed-forward network with common hidden layers. This was specifically done to test the trade-off between the training time of a multi-headed network and the theoretical advantages of enabling the multiple heads to benefit from the rich information encoded in the common hidden layers. However, this resulted in a bloated computation time for parameter updates during the training phase. The practical benefits of shorter training time far outweighed the theoretical advantages of common hidden layers. For our experiments, we decided to learn the three different quantities - value function, state trajectory, and control trajectory - on three different feed-forward networks.

There are a few points that should establish some perspective going forward.

Singularities The presence of singularities can very quickly destabilize learning and can lead to either *exploding gradients* or *vanishing gradients* during training. This has been one of the major peeves we faced in our experiments. Although there are no good ways to ameliorate this issue, we decided to model singularities as outliers in our dataset, which immediately led to another problem. We had to extensively hand-tune the weights of state and torque regularization terms in each of the classic control systems and the 7 *dof* arm such that the value function computed by DDP for the majority of initial starting configurations remained less than 1. We were in effect normalizing the dataset. In this case, singularities in the state space of any robot would have a corresponding value function (far) greater than 1. The problem then becomes the extensive hand-tuning required for any system. This has been particularly impactful on toy problems, especially with the unicycle, and we also get an indication that with more complex systems such as the 7 *dof* arm, there may be many more hidden problems.

Physics Informed Architectures of Neural networks The feasibility of deep learning algorithms is often the result of their generalization capabilities and sample efficiency. In continuous high-dimensional domains such as robotics where the underlying processes are known, the problem of learning an accurate representation while maintaining sample efficiency can become quickly intractable. Therefore the use of Group Equivariant Neural Networks (GENNs) [Ger+21]: a neural network is equivariant if the learned representation transforms under transformations of input in a linear predictable manner: should augment robot learning. The equivariance property itself is enforced by encoding symmetries and invariances in the architecture of the neural network and is useful for three reasons :

- Equivariance to a symmetry transformation leads to conservation laws which can be used to place additional constraints for accurate modeling, for instance, conservation of Hamiltonian in [GDY19]. Another idea would be to incorporate Noether's theorem since it can alleviate the problem of sequence prediction over long time horizons.
- Induced parameter sharing due to symmetries decreases the number of trainable parameters.
- Incorporating physical priors and inductive biases in learning automatically reduces data dependence making GENNs sample efficient.

In robotics we know the physical laws that drive motion and computing large datasets for learning is infeasible, therefore encoding inductive biases in learning should lead to sample efficiency.

Learning with Differentiable Simulators and Contact Phases One of the last things that we tried to do (and probably still doing) was to run ∂PVP with contact phases. This would have shown the scalability of our algorithm. However, this implies using a physics differentiable simulator such as NimblePhysics [DHO22] or DojoSim [How+22] inside the DDP instance we use. However, a recent comparison of the fidelity of gradients computed by these simulators [DHO22] hamper the

deployment of our algorithm for more complicated scenarios. On the other hand, in [Lid+22b], the authors proposed leveraging randomized smoothing to augment differentiable physics which can then be used to efficiently compute gradients in some neighbourhood. This is an avenue to be explored in the future.

Thank you.

8 Appendix

8.1 Q-Function

In discounted infinite horizon problems, for any policy π, the state-control value function $Q : \mathcal{X} \times \mathcal{U} \rightarrow R$ is defined as :

$$Q(x,u) = \mathbb{E}[\sum_{t>=0} \gamma^t r(x_t,u_t)|x_0 = x, u_0 = u, u_t = \pi] \tag{8.1}$$

$\forall t \geq 1$. Under application of policy π, the corresponding optimal Q function is :

$$Q^*(x,u) = \max_\pi Q(x,u) \tag{8.2}$$

V function and Q function are related in the following way.

$$Q(x,u) = r(x,u) + \gamma \sum_{x_+ \in \mathcal{X}} p(x_+|x,u)V(x_+) \tag{8.3}$$

$$V(x) = Q(x,\pi(x)) \tag{8.4}$$

$$Q^*(x,u) = r(x,u) + \gamma \sum p(x_+|x,u)V^*(x_+) \tag{8.5}$$

$$V^* = Q^*(x,\pi^*(X) = \max Q^*(x,u) \tag{8.6}$$

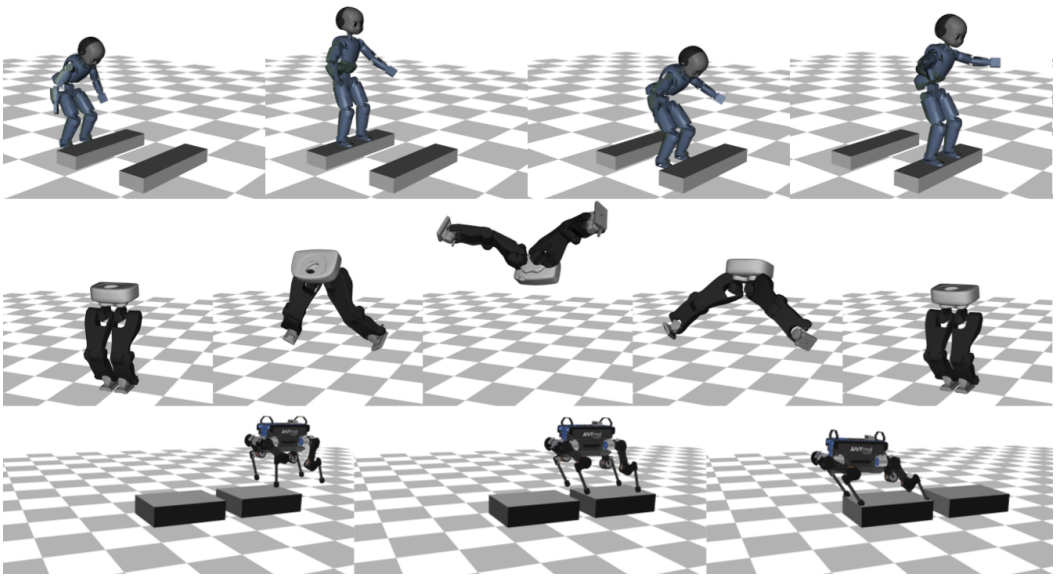


Fig. 8.1.: Whole Body contact sequence computed by Crocoddyl

8.2 The Loco3D project : *Crocoddyl* - *C*ontact *Ro*bot *Co*ntrol by *D*ifferential *Dy*namic Programming *L*ibrary

Locomotion in complex environment - Loco3D - is a suite of frameworks with the objective to plan, adapt and execute multi-contact sequence locomotion movements in an environment that allows for dynamic changes [Car+17]. The Loco3D project follows a modular approach to the task of complex motion generation. A short summary of the modules, along with its associated publications, is presented below.

- Contact Sequence Planner in [Ton+18; Fer+17].
- Centroidal Pattern generator introduced in [Car+16].
- Whole Body Motion Generator described in [Mas+20].
- Low-level Torque Control methods shown in [BL15]

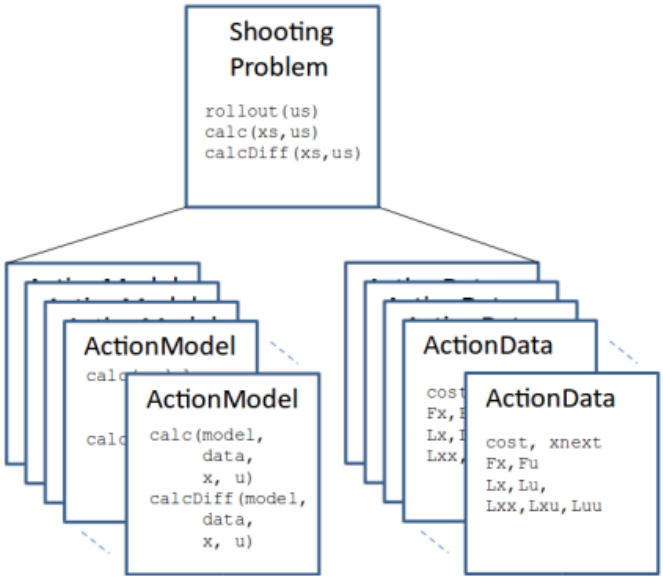


Fig. 8.2.: Assembling ActionModels with corresponding ActionDatas to form the global optimal control problem.

Crocoddyl is an open-source optimal control library[1] developed as part of the Loco3D project. The solvers are written in *C++* with *Python* bindings [Mas+20]. The solvers of Crocoddyl are based on a variant of the DDP algorithm proposed in [Bud+18]. For a deeper dive into Crocoddyl, the reader should consult [Bud19].

8.2.1 Features of Crocoddyl

The primary features of Crocoddyl are as follows:

- **Efficient Rigid Body Algorithms**: The central dynamics engine of Crocoddyl is Pinocchio [Car+19]. The choice of using Pinocchio is due to its fast and efficient implementations of Rigid Body Algorithms on Lie Algebra [Fea14]. In turn, this allows Crocoddyl to reduce its own computation time.

[1] `https://github.com/loco-3d/crocoddyl`

- **Analytical Derivatives**: Crocoddyl implements analytic derivatives as opposed to numerical finite differences methods to compute dynamical derivatives based on [CM18].

- **Transcription vs Resolution**: Crocoddyl handles problem resolution separately from problem transcription to allow for flexibility in the choice of solvers.

- **Multi Threading**: The computation time of Crocoddyl is further decreased by using multi-threading for computing derivatives.

- **Feasibility Prone DDP**: An important feature of Crocoddyl is its ability to handle infeasible guesses that can occur whenever there are gaps between nodes in the trajectory. The FDDP solver that we use for our own experiments can expand and improve upon its own search as opposed to the classic DDP solver. The full list of solvers available in Crocoddyl is shown in Table 8.2.1.

Contact-Constrained DDP Solver	[Bud+18]
KKT based Solver	see [Man19]
Feasibility Prone DDP Solver	see [Man19]
Control Limited Box DDP Solver	proposed in [TMT14]
Box KKT Solver	KKT based control-limited resolution of the full horizon

Crocoddyl decomposes the global problem through a series of **Action Models** with their corresponding problem-specific datas called **Action Datas**. A basic layout is shown in Figure 8.2. An Action Model contains an invariant description of the problem such as dynamics or costs or constraints. The Data classes store the problem-dependent quantities such as values and derivatives information.

Therefore, a stack of Action Models together describes the global problem. The global problem is then written as a Shooting Problem. Once a problem is defined inside the Shooting Problem class, different solvers in Table 8.2.1 can be used to find the solution. Therefore, the optimal control task in Crocoddyl is written as :

```
ddp_problem = ShootingProblem(x_0, [Running_Models], TerminalModel)
```

The list of Action Models inside the ShootingProblem class is collectively known as the **Running_Models** over a preview horizon while the **Terminal_Model** is also an Action Model but at the end of the preview horizon.

Bibliography

[AF03] Robert A Adams and John JF Fournier. *Sobolev spaces*. Elsevier, 2003 (cit. on p. 44).

[AG12] Sedigheh Ahmadzadeh and Mehdi Ghanavati. "Navigation of mobile robot using the PSO particle swarm optimization". In: *Journal of Academic and Applied Studies (JAAS)* 2.1 (2012), pp. 32–38 (cit. on p. 18).

[AHM22] Felix Aller, Monika Harant, and Katja Mombaur. "Optimization of Dynamic Sit-to-Stand Trajectories to Assess Whole Body Motion Performance of the Humanoid Robot REEM-C". In: *Frontiers in Robotics and AI* (2022), p. 155 (cit. on pp. 22, 24).

[And+17] Marcin Andrychowicz, Filip Wolski, Alex Ray, et al. "Hindsight experience replay". In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 18).

[Aru+17] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "Deep reinforcement learning: A brief survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38 (cit. on p. 17).

[Bar97] Andrew G Barto. "Reinforcement learning". In: *Neural systems for control*. Elsevier, 1997, pp. 7–30 (cit. on p. 1).

[BB01] Jonathan Baxter and Peter L Bartlett. "Infinite-horizon policy-gradient estimation". In: *Journal of Artificial Intelligence Research* 15 (2001), pp. 319–350 (cit. on p. 17).

[BSS13] Mokhtar S Bazaraa, Hanif D Sherali, and Chitharanjan M Shetty. *Nonlinear programming: theory and algorithms*. John Wiley & Sons, 2013 (cit. on p. 13).

[Bel+21] Italo Belli, M Parigi Polverini, Arturo Laurenzi, et al. "Optimization-Based Quadrupedal Hybrid Wheeled-Legged Locomotion". In: *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2021, pp. 41–46 (cit. on p. 23).

[Bel66] Richard Bellman. "Dynamic programming". In: *Science* 153.3731 (1966) (cit. on pp. 3, 19, 33, 39, 69).

[Bel54] Richard Bellman. "The theory of dynamic programming". In: *Bulletin of the American Mathematical Society* 60.6 (1954), pp. 503–515 (cit. on p. 39).

[BL15] Mehdi Benallegue and Florent Lamiraux. "Estimation and stabilization of humanoid flexibility deformation using only inertial measurement units and contact information". In: *International Journal of Humanoid Robotics* 12.03 (2015), p. 1550025 (cit. on p. 110).

[BT04] Jean-Paul Berrut and Lloyd N Trefethen. “Barycentric lagrange interpolation”. In: *SIAM review* 46.3 (2004), pp. 501–517 (cit. on p. 21).

[BMK19] Lars Berscheid, Pascal Meißner, and Torsten Kröger. “Robot learning of shifting objects for grasping in cluttered environments”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 612–618 (cit. on p. 18).

[Ber12] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*. Vol. 1. Athena scientific, 2012 (cit. on pp. 12, 22).

[Ber97] Dimitri P Bertsekas. “Nonlinear programming”. In: *Journal of the Operational Research Society* 48.3 (1997), pp. 334–334 (cit. on pp. 13, 22).

[Bet10] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010 (cit. on p. 20).

[Bet93] John T Betts. “Trajectory optimization using sparse sequential quadratic programming”. In: *Optimal Control*. Springer, 1993, pp. 115–128 (cit. on p. 33).

[Bha+14] Mukunda Bharatheesha, Wouter Caarls, Wouter Jan Wolfslag, and Martijn Wisse. “Distance metric approximation for state-space RRTs using supervised learning”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 252–257 (cit. on p. 25).

[BP84] Hans Georg Bock and Karl-Josef Plitt. “A multiple shooting algorithm for direct solution of optimal control problems”. In: *IFAC Proceedings Volumes* 17.2 (1984), pp. 1603–1608 (cit. on p. 20).

[BBV04] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004 (cit. on p. 24).

[Buc88] Christian G Bucher. “Adaptive sampling—an iterative fast Monte Carlo procedure”. In: *Structural safety* 5.2 (1988), pp. 119–126 (cit. on p. 69).

[Bud19] Rohan Budhiraja. “Multi-body Locomotion : Problem Structure and Efficient Resolution”. Theses. INSA de Toulouse, Nov. 2019 (cit. on p. 111).

[Bud+18] Rohan Budhiraja, Justin Carpentier, Carlos Mastalli, and Nicolas Mansard. “Differential dynamic programming for multi-phase rigid contact dynamics”. In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2018, pp. 1–9 (cit. on pp. 111, 112).

[Bue+18] Lars Buesing, Theophane Weber, Yori Zwols, et al. “Woulda, coulda, shoulda: Counterfactually-guided policy search”. In: *arXiv preprint arXiv:1811.06272* (2018) (cit. on p. 26).

[Cad+16] Cesar Cadena, Luca Carlone, Henry Carrillo, et al. “Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age”. In: *IEEE Transactions on robotics* 32.6 (2016), pp. 1309–1332 (cit. on p. 18).

[Cal20] Sylvain Calinon. “Gaussians on Riemannian manifolds: Applications for robot learning and adaptive control”. In: *IEEE Robotics & Automation Magazine* 27.2 (2020), pp. 33–45 (cit. on p. 18).

[CFH20] Jan Carius, Farbod Farshidian, and Marco Hutter. “Mpc-net: A first principles guided policy search”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2897–2904 (cit. on p. 26).

[CM+19] Rohan Budhiraja Carlos Mastalli, Nicolas Mansard, et al. *Crocoddyl: a fast and flexible optimal control library for robot control under contact sequence*. https://github.com/loco-3d/crocoddyl/wikis/home. 2019 (cit. on p. 38).

[Car+17] Justin Carpentier, Andrea Del Prete, Steve Tonneau, et al. “Multi-contact locomotion of legged robots in complex environments–the loco3d project”. In: *RSS workshop on challenges in dynamic legged locomotion*. 2017, 3p (cit. on p. 110).

[CM18] Justin Carpentier and Nicolas Mansard. “Analytical derivatives of rigid body dynamics algorithms”. In: *Robotics: Science and systems (RSS 2018)*. 2018 (cit. on p. 112).

[Car+19] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, et al. “The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives”. In: *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2019, pp. 614–619 (cit. on pp. 11, 25, 85, 111).

[Car+16] Justin Carpentier, Steve Tonneau, Maximilien Naveau, Olivier Stasse, and Nicolas Mansard. “A versatile and efficient pattern generator for generalized legged locomotion”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 3555–3561 (cit. on p. 110).

[Che+17] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, et al. “Path integral guided policy search”. In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3381–3388 (cit. on p. 26).

[CA98] Hong Chen and Frank Allgöwer. “A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability”. In: *Automatica* 34.10 (1998), pp. 1205–1217 (cit. on pp. 26, 42, 57).

[Cho+75] Gregory C Chow et al. *Analysis and control of dynamic economic systems*. Wiley, 1975 (cit. on p. 32).

[CL01] Th H Cormen and CE Leiserson. *Rivest RL, Stein C. Introduction to Algorithms*. 2001 (cit. on p. 14).

[CB21] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. `http://pybullet.org`. 2016–2021 (cit. on p. 85).

[CB89] James E Cuthrell and Lorenz T Biegler. “Simultaneous optimization and solution methods for batch reactor control profiles”. In: *Computers & Chemical Engineering* 13.1-2 (1989), pp. 49–62 (cit. on p. 20).

[Cza+17] Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. “Sobolev training for neural networks”. In: *Advances in Neural Information Processing Systems* 30 (2017) (cit. on p. 28).

[Dan+21] Ewen Dantec, Rohan Budhiraja, Adria Roig, et al. “Whole body model predictive control with a memory of motion: Experiments on a torque-controlled talos”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 8202–8208 (cit. on p. 24).

[DKT19] Robin Deits, Twan Koolen, and Russ Tedrake. “LVIS: Learning from value function intervals for contact-aware robot controllers”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 7762–7768 (cit. on p. 26).

[DHO22] Yaofeng Desmond Zhong, Jiequn Han, and Georgia Olympia Brikis. “Differentiable Physics Simulations with Contacts: Do They Have Correct Gradients wrt Position, Velocity and Control?” In: *arXiv e-prints* (2022), arXiv–2207 (cit. on p. 107).

[Di +18] Jared Di Carlo, Patrick M Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. “Dynamic locomotion in the mit cheetah 3 through convex model-predictive control”. In: *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2018, pp. 1–9 (cit. on p. 23).

[Die+06] Moritz Diehl, Hans Georg Bock, Holger Diedam, and P-B Wieber. “Fast direct multiple shooting algorithms for optimal robot control”. In: *Fast motions in biomechanics and robotics*. Springer, 2006, pp. 65–93 (cit. on p. 20).

[DBS05] Moritz Diehl, Hans Georg Bock, and Johannes P Schlöder. “A real-time iteration scheme for nonlinear optimization in optimal feedback control”. In: *SIAM Journal on control and optimization* 43.5 (2005), pp. 1714–1736 (cit. on p. 24).

[Dre02] Stuart Dreyfus. “Richard Bellman on the birth of dynamic programming”. In: *Operations Research* 50.1 (2002), pp. 48–51 (cit. on p. 39).

[DZZ21] Luke Drnach, John Z Zhang, and Ye Zhao. “Mediating between contact feasibility and robustness of trajectory optimization through chance complementarity constraints”. In: *Frontiers in Robotics and AI* 8 (2021) (cit. on p. 24).

[Dub+20] Alexis Duburcq, Yann Chevaleyre, Nicolas Bredeche, and Guilhem Boéris. “Online trajectory planning through combined trajectory optimization and function approximation: Application to the exoskeleton Atalante”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 3756–3762 (cit. on p. 56).

[Dul+21] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, et al. “Challenges of real-world reinforcement learning: definitions, benchmarks and analysis”. In: *Machine Learning* 110.9 (2021), pp. 2419–2468 (cit. on p. 19).

[ETT11] Tom Erez, Yuval Tassa, and Emanuel Todorov. “Infinite-Horizon Model Predictive Control for Periodic Tasks with Contacts”. In: *Robotics: Science and Systems*. 2011 (cit. on p. 26).

[FA12] Michael Fairbank and Eduardo Alonso. “Value-gradient learning”. In: *The 2012 international joint conference on neural networks (ijcnn)*. IEEE. 2012, pp. 1–8 (cit. on p. 45).

[FAP12] Michael Fairbank, Eduardo Alonso, and Danil Prokhorov. “Simple and fast calculation of the second-order gradients for globalized dual heuristic dynamic programming in neural networks”. In: *IEEE transactions on neural networks and learning systems* 23.10 (2012), pp. 1671–1676 (cit. on p. 45).

[Fea14] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014 (cit. on pp. 11, 111).

[Fer+17] Pierre Fernbach, Steve Tonneau, Andrea Del Prete, and Michel Taıx. “A kinodynamic steering-method for legged multi-contact locomotion”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3701–3707 (cit. on p. 110).

[FLA16] Chelsea Finn, Sergey Levine, and Pieter Abbeel. “Guided cost learning: Deep inverse optimal control via policy optimization”. In: *International conference on machine learning*. PMLR. 2016, pp. 49–58 (cit. on p. 18).

[Flo07] Razvan V Florian. “Correct equations for the dynamics of the cart-pole system”. In: *Center for Cognitive and Neural Studies (Coneural), Romania* (2007) (cit. on p. 63).

[Gar+09] MA Porta Garcia, Oscar Montiel, Oscar Castillo, Roberto Sepulveda, and Patricia Melin. “Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation”. In: *Applied Soft Computing* 9.3 (2009), pp. 1102–1110 (cit. on p. 18).

[Ged11] Gilbert Gede. “The direct collocation method for optimal control”. In: *UC Davis* 26 (2011) (cit. on p. 20).

[GM16] Mathieu Geisert and Nicolas Mansard. “Trajectory generation for quadrotor based systems using numerical optimal control”. In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 2958–2964 (cit. on p. 24).

[Ger03] Matthias Gerdts. “Direct shooting method for the numerical solution of higher-index DAE optimal control problems”. In: *Journal of Optimization Theory and Applications* 117.2 (2003), pp. 267–294 (cit. on p. 20).

[Ger+21] Jan E Gerken, Jimmy Aronsson, Oscar Carlsson, et al. “Geometric deep learning and equivariant neural networks”. In: *arXiv preprint arXiv:2105.13926* (2021) (cit. on p. 107).

[GMS05] Philip E Gill, Walter Murray, and Michael A Saunders. “SNOPT: An SQP algorithm for large-scale constrained optimization”. In: *SIAM review* 47.1 (2005), pp. 99–131 (cit. on p. 20).

[GHO99] Gene H Golub, Per Christian Hansen, and Dianne P O'Leary. "Tikhonov regularization and total least squares". In: *SIAM journal on matrix analysis and applications* 21.1 (1999), pp. 185–194 (cit. on p. 37).

[GDY19] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. "Hamiltonian neural networks". In: *Advances in neural information processing systems* 32 (2019) (cit. on p. 107).

[Gro+12] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. "A survey of actor-critic reinforcement learning: Standard and natural policy gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1291–1307 (cit. on p. 17).

[Gul90] Vijaykumar Gullapalli. "A stochastic reinforcement learning algorithm for learning real-valued functions". In: *Neural networks* 3.6 (1990), pp. 671–692 (cit. on p. 17).

[Haa18] Tuomas Haarnoja. *Acquiring diverse robot skills via maximum entropy deep reinforcement learning*. University of California, Berkeley, 2018 (cit. on p. 19).

[Haa+19] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, et al. "Learning to Walk Via Deep Reinforcement Learning". In: *Robotics: Science and Systems XV*. RSS. 2019 (cit. on p. 17).

[Haa+17] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. "Reinforcement learning with deep energy-based policies". In: *International conference on machine learning*. PMLR. 2017, pp. 1352–1361 (cit. on p. 19).

[Haa+18] Tuomas Haarnoja, Aurick Zhou, P. Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *ICML*. 2018 (cit. on p. 17).

[HMD15] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *arXiv preprint arXiv:1510.00149* (2015) (cit. on p. 44).

[Hen+18] Peter Henderson, Riashat Islam, Philip Bachman, et al. "Deep reinforcement learning that matters". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018 (cit. on p. 17).

[HRP18] Peter Henderson, Joshua Romoff, and Joelle Pineau. "Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods". In: *arXiv preprint arXiv:1810.02525* (2018) (cit. on pp. 18, 19).

[HSS12] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *Cited on* 14.8 (2012), p. 2 (cit. on p. 18).

[HVD+15] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* 2.7 (2015) (cit. on p. 44).

[Hoc98] Sepp Hochreiter. “The vanishing gradient problem during learning recurrent neural nets and problem solutions”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116 (cit. on p. 55).

[HFH20] David Hoeller, Farbod Farshidian, and Marco Hutter. “Deep Value Model Predictive Control”. In: *Conference on Robot Learning*. CoRL. 2020 (cit. on p. 17).

[Hor91] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural networks* 4.2 (1991), pp. 251–257 (cit. on p. 44).

[HD10] Boris Houska and Moritz Diehl. “Robustness and stability optimization of power generating kite systems in a periodic pumping mode”. In: *2010 IEEE International Conference on Control Applications*. IEEE. 2010, pp. 2172–2177 (cit. on p. 24).

[HFD11] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. “ACADO toolkit—An open-source framework for automatic control and dynamic optimization”. In: *Optimal Control Applications and Methods* 32.3 (2011), pp. 298–312 (cit. on p. 20).

[How+22] Taylor A Howell, Simon Le Cleac’h, J Zico Kolter, Mac Schwager, and Zachary Manchester. “Dojo: A Differentiable Simulator for Robotics”. In: *arXiv preprint arXiv:2203.00806* (2022) (cit. on p. 107).

[HL02] Bei Hu and Arno Linnemann. “Toward infinite-horizon optimality in nonlinear model predictive control”. In: *IEEE Transactions on Automatic Control* 47.4 (2002), pp. 679–682 (cit. on pp. 26, 57).

[Hub+11] M Huba, S Skogestad, M Fikar, et al. “Selected topics on constrained and nonlinear control”. In: *Slovakia, ROSA. Dolnỳ Kubın* (2011) (cit. on p. 23).

[Hut+17] Marco Hutter, Christian Gehring, Andreas Lauber, et al. “Anymal-toward legged robots for harsh environments”. In: *Advanced Robotics* 31.17 (2017), pp. 918–931 (cit. on p. 24).

[HLH18] Jemin Hwangbo, Joonho Lee, and Marco Hutter. “Per-contact iteration method for solving contact dynamics”. In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 895–902 (cit. on p. 24).

[Ily+20] Andrew Ilyas, Logan Engstrom, Shibani Santurkar, et al. “A closer look at deep policy gradients”. In: *International Conference on Learning Representations*. ICLR. 2020 (cit. on pp. 28, 82).

[Isl+15] Md Amirul Islam et al. “A comparative study on numerical solutions of initial value problems (IVP) for ordinary differential equations (ODE) with Euler and Runge Kutta Methods”. In: *American Journal of computational mathematics* 5.03 (2015), p. 393 (cit. on p. 20).

[Jad+17] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, et al. “Decoupled neural interfaces using synthetic gradients”. In: *International conference on machine learning*. PMLR. 2017, pp. 1627–1635 (cit. on p. 44).

[Joh+19] Tobias Johannink, Shikhar Bahl, Ashvin Nair, et al. “Residual reinforcement learning for robot control”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 6023–6029 (cit. on p. 18).

[Kah+17] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. “Plato: Policy learning using adaptive trajectory optimization”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 3342–3349 (cit. on p. 26).

[Kal+18] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, et al. “Scalable deep reinforcement learning for vision-based robotic manipulation”. In: *Conference on Robot Learning*. PMLR. 2018, pp. 651–673 (cit. on p. 1).

[Kam+12] Takashi Kamihigashi et al. *Existence and uniqueness of a fixed point for the Bellman operator in deterministic dynamic programming*. Tech. rep. 2012 (cit. on p. 28).

[Kat+16] Kapil D Katyal, Edward W Staley, Matthew S Johannes, et al. “In-hand robotic manipulation via deep reinforcement learning”. In: *Proceedings of the Workshop on Deep Learning for Action and Interaction, in Conjunction with Annual Conference on Neural Information Processing Systems, Barcelona, Spain*. Vol. 9. 2016 (cit. on p. 18).

[Kel17] Matthew Kelly. “Transcription Methods for Trajectory Optimization: a beginners tutorial”. In: (July 2017) (cit. on p. 20).

[Kel15] Matthew P Kelly. “Transcription methods for trajectory optimization”. In: *Tutorial, Cornell University, Feb* (2015) (cit. on p. 21).

[Kim+19] Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. “Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control”. In: *arXiv preprint arXiv:1909.06586* (2019) (cit. on p. 25).

[Kim+20] Jae In Kim, Mineui Hong, Kyungjae Lee, et al. “Learning to walk a tripod mobile robot using nonlinear soft vibration actuators with entropy adaptive reinforcement learning”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2317–2324 (cit. on p. 18).

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015) (cit. on p. 17).

[Koe+15] Jonas Koenemann, Andrea Del Prete, Yuval Tassa, et al. “Whole-body model-predictive control applied to the HRP-2 humanoid”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 3346–3351 (cit. on p. 23).

[KT03] Vijay R Konda and John N Tsitsiklis. “Onactor-critic algorithms”. In: *SIAM journal on Control and Optimization* 42.4 (2003), pp. 1143–1166 (cit. on pp. 16, 17).

[Kop62] Richard E Kopp. “Pontryagin maximum principle”. In: *Mathematics in Science and Engineering*. Vol. 5. Elsevier, 1962, pp. 255–279 (cit. on p. 13).

[KZG22] Arash Bahari Kordabad, Mario Zanon, and Sébastien Gros. “Equivalency of Optimality Criteria of Markov Decision Process and Model Predictive Control”. In: *arXiv preprint arXiv:2210.04302* (2022) (cit. on p. 22).

[KB06] Serdar Kucuk and Zafer Bingul. *Robot kinematics: Forward and inverse kinematics*. INTECH Open Access Publisher London, UK, 2006 (cit. on p. 69).

[Kui+16] Scott Kuindersma, Robin Deits, Maurice Fallon, et al. “Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot”. In: *Autonomous robots* 40.3 (2016), pp. 429–455 (cit. on p. 23).

[KKR19] Harshat Kumar, Alec Koppel, and Alejandro Ribeiro. “On the sample complexity of actor-critic method for reinforcement learning with function approximation”. In: *arXiv preprint arXiv:1910.08412* (2019) (cit. on p. 1).

[Lam11] Moulay Tahar Lamchich. *Torque control*. BoD–Books on Demand, 2011 (cit. on p. 85).

[LBS22] Jee-eun Lee, Tirthankar Bandyopadhyay, and Luis Sentis. “Adaptive robot climbing with magnetic feet in unknown slippery structure”. In: *Frontiers in Robotics and AI* (2022), p. 205 (cit. on p. 24).

[LO97] Jeong-Woo Lee and Jun-Ho Oh. “Hybrid learning of mapping and its Jacobian in multilayer neural networks”. In: *Neural computation* 9.5 (1997), pp. 937–958 (cit. on p. 45).

[LRM22] Peter Q Lee, Vidyasagar Rajendran, and Katja Mombaur. “Optimization-Based Motion Generation for Buzzwire Tasks With the REEM-C Humanoid Robot”. In: *Frontiers in Robotics and AI* 9 (2022) (cit. on p. 24).

[Lem+20] Teguh Santoso Lembono, Carlos Mastalli, Pierre Fernbach, Nicolas Mansard, and Sylvain Calinon. “Learning how to walk: Warm-starting optimal control solver with memory of motion”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 1357–1363 (cit. on pp. 24, 26).

[LKS15] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. “DeepMPC: Learning deep latent features for model predictive control.” In: *Robotics: Science and Systems*. Vol. 10. Rome, Italy. 2015 (cit. on p. 25).

[Lev+16] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. “End-to-end training of deep visuomotor policies”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373 (cit. on p. 26).

[LK13] Sergey Levine and Vladlen Koltun. “Guided policy search”. In: *International conference on machine learning*. PMLR. 2013, pp. 1–9 (cit. on p. 26).

[LK14] Sergey Levine and Vladlen Koltun. “Learning complex neural network policies with trajectory optimization”. In: *International Conference on Machine Learning*. PMLR. 2014, pp. 829–837 (cit. on p. 26).

[LS92] Li-zhi Liao and Christine A Shoemaker. *Advantages of differential dynamic programming over Newton’s method for discrete-time optimal control problems*. Tech. rep. Cornell University, 1992 (cit. on pp. 32, 33).

[Lid+22a] Quentin Le Lidec, Wilson Jallet, Ivan Laptev, Cordelia Schmid, and Justin Carpentier. “Enforcing the consensus between Trajectory Optimization and Policy Learning for precise robot control”. In: *arXiv preprint arXiv:2209.09006* (2022) (cit. on pp. 24, 26, 56).

[Lid+22b] Quentin Le Lidec, Louis Montaut, Cordelia Schmid, Ivan Laptev, and Justin Carpentier. “Augmenting differentiable physics with randomized smoothing”. In: *arXiv preprint arXiv:2206.11884* (2022) (cit. on p. 108).

[Lil+15] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015) (cit. on p. 18).

[Lon+18] Pinxin Long, Tingxiang Fan, Xinyi Liao, et al. “Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6252–6259 (cit. on p. 18).

[Low+19] Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. “Plan online, learn offline: Efficient learning and exploration via model-based control”. In: *Int. Conf. on Learning Representations*. 2019 (cit. on pp. 25, 38).

[Lut+21] Michael Lutter, Shie Mannor, Jan Peters, Dieter Fox, and Animesh Garg. “Value iteration in continuous actions, states and time”. In: *arXiv preprint arXiv:2105.04682* (2021) (cit. on p. 38).

[Mah+18a] A Rupam Mahmood, Dmytro Korenkevych, Brent J Komer, and James Bergstra. “Setting up a reinforcement learning task with a real-world robot”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4635–4640 (cit. on p. 19).

[Mah+18b] A Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra. “Benchmarking reinforcement learning algorithms on real-world robots”. In: *Conference on robot learning*. PMLR. 2018, pp. 561–591 (cit. on p. 19).

[MAW07] Theodore W Manikas, Kaveh Ashenayi, and Roger L Wainwright. “Genetic algorithms for autonomous robot navigation”. In: *IEEE Instrumentation & Measurement Magazine* 10.6 (2007), pp. 26–31 (cit. on p. 18).

[Man19] N Mansard. “Feasibility-prone differential dynamic programming is ddp a multiple shooting algorithm?” In: *LAAS, Toulouse, Tech. Rep* (2019) (cit. on p. 112).

[Man+18] 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)*. IEEE. 2018, pp. 2986–2993 (cit. on pp. 24, 26, 56).

[Mas+20] Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, et al. “Crocoddyl: An efficient and versatile framework for multi-contact optimal control”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 2536–2542 (cit. on pp. 110, 111).

[Mas93] Ryusuke Masuoka. “Noise robustness of EBNN learning”. In: *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*. Vol. 2. IEEE. 1993, pp. 1665–1668 (cit. on p. 45).

[May73] David Q Mayne. “Differential dynamic programming–a unified approach to the optimization of dynamic systems”. In: *Control and Dynamic Systems*. Vol. 10. Elsevier, 1973 (cit. on pp. 4, 20, 32).

[MM90] DQ Mayne and Hi Michalska. “An implementable receding horizon controller for stabilization of nonlinear systems”. In: *29th IEEE Conference on Decision and Control*. IEEE. 1990, pp. 3396–3397 (cit. on p. 57).

[Men+19] Russell Mendonca, Abhishek Gupta, Rosen Kralev, et al. “Guided meta-policy search”. In: *Advances in Neural Information Processing Systems* 32 (2019) (cit. on p. 26).

[MM93] Hannah Michalska and David Q Mayne. “Robust receding horizon control of constrained nonlinear systems”. In: *IEEE transactions on automatic control* 38.11 (1993), pp. 1623–1633 (cit. on p. 57).

[MZP22] Enrico Mingo Hoffman, Chengxu Zhou, and Matteo Parigi Polverini. “Editorial: Advancements in trajectory optimization and model predictive control for legged systems”. In: *Frontiers in Robotics and AI* 9 (2022) (cit. on p. 23).

[MT92] Tom M Mitchell and Sebastian B Thrun. “Explanation-based neural network learning for robot control”. In: *Advances in neural information processing systems* 5 (1992) (cit. on p. 45).

[ML16] William H Montgomery and Sergey Levine. “Guided policy search via approximate mirror descent”. In: *Advances in Neural Information Processing Systems* 29 (2016) (cit. on p. 26).

[MA00] Andrew Moore and Christopher Atkeson. “The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces”. In: *Machine Learning* 21 (May 2000) (cit. on p. 1).

[MT14] Igor Mordatch and Emo Todorov. “Combining the benefits of function approximation and trajectory optimization.” In: *Robotics: Science and Systems*. Vol. 4. 2014 (cit. on pp. 26, 27, 56).

[Mor78] Jorge J Moré. “The Levenberg-Marquardt algorithm: implementation and theory”. In: *Numerical analysis*. Springer, 1978, pp. 105–116 (cit. on p. 37).

[Nai02] D Subbaram Naidu. *Optimal control systems*. CRC press, 2002 (cit. on p. 14).

[Neu+16] Michael Neunert, Cédric De Crousaz, Fadri Furrer, et al. “Fast nonlinear model predictive control for unified trajectory optimization and tracking”. In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 1398–1404 (cit. on p. 24).

[Neu+18] Michael Neunert, Markus Stäuble, Markus Giftthaler, et al. “Whole-body nonlinear model predictive control through contacts for quadrupeds”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1458–1465 (cit. on p. 23).

[NZZ19] Chunyu Nie, Zewei Zheng, and Ming Zhu. “Three-dimensional path-following control of a robotic airship with reinforcement learning”. In: *International Journal of Aerospace Engineering* 2019 (2019) (cit. on p. 18).

[NKH16] Iram Noreen, Amna Khan, and Zulfiqar Habib. “Optimal path planning using RRT* based approaches: a survey and future directions”. In: *International Journal of Advanced Computer Science and Applications* 7.11 (2016) (cit. on p. 24).

[Ota+19] Kei Ota, Devesh K Jha, Tomoaki Oiki, et al. “Trajectory optimization for unknown constrained systems using reinforcement learning”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 3487–3494 (cit. on p. 18).

[Par+22] Amit Parag, Sébastien Kleff, Léo Saci, Nicolas Mansard, and Olivier Stasse. “Value learning from trajectory optimization and Sobolev descent: A step toward reinforcement learning with superlinear convergence properties”. In: *International Conference on Robotics and Automation*. 2022 (cit. on pp. 18, 28).

[Pen92] Shige Peng. “A generalized dynamic programming principle and Hamilton-Jacobi-Bellman equation”. In: *Stochastics: An International Journal of Probability and Stochastic Processes* 38.2 (1992), pp. 119–134 (cit. on p. 13).

[PS08] Jan Peters and Stefan Schaal. “Reinforcement learning of motor skills with policy gradients”. In: *Neural networks* 21.4 (2008), pp. 682–697 (cit. on p. 17).

[PSC22] Emmanuel Pignat, João Silvério, and Sylvain Calinon. “Learning from demonstration using products of experts: Applications to manipulation and task prioritization”. In: *The International Journal of Robotics Research* 41.2 (2022), pp. 163–188 (cit. on p. 18).

[PY93] Elijah Polak and TH Yang. “Moving horizon control of linear systems with input saturation and plant uncertainty part 1. robustness”. In: *International Journal of Control* 58.3 (1993), pp. 613–638 (cit. on p. 57).

[Put94] ML Puterman. “Finite-horizon Markov decision processes”. In: *Markov Decision Processes: Discrete Stochastic Dynamic Programming. New York: Wiley-Interscience* (1994), pp. 78–9 (cit. on p. 42).

[RMM94] James B Rawlings, Edward S Meadows, and Kenneth R Muske. “Nonlinear model predictive control: A tutorial and survey”. In: *IFAC Proceedings Volumes* 27.2 (1994), pp. 185–197 (cit. on p. 22).

[Rif+11] Salah Rifai, Grégoire Mesnil, Pascal Vincent, et al. “Higher order contractive auto-encoder”. In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2011, pp. 645–660 (cit. on p. 45).

[Rös19] Christoph Rösmann. “Time-optimal nonlinear model predictive control”. PhD thesis. 2019 (cit. on p. 20).

[RN94] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994 (cit. on p. 17).

[Rus+15] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, et al. “Policy distillation”. In: *arXiv preprint arXiv:1511.06295* (2015) (cit. on p. 44).

[Sch02] Ralf Schoknecht. “Optimality of reinforcement learning algorithms with linear function approximation”. In: *Advances in neural information processing systems* 15 (2002) (cit. on p. 17).

[Sch+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. “Proximal policy optimization algorithms”. In: (2017) (cit. on pp. 17, 18, 28).

[SR98] Pierre OM Scokaert and James B Rawlings. “Constrained linear quadratic regulation”. In: *IEEE Transactions on automatic control* 43.8 (1998), pp. 1163–1169 (cit. on p. 38).

[ST17] Ravid Shwartz-Ziv and Naftali Tishby. “Opening the Black Box of Deep Neural Networks via Information”. In: (Mar. 2017) (cit. on p. 28).

[Sim+91] Patrice Simard, Bernard Victorri, Yann LeCun, and John Denker. “Tangent prop-a formalism for specifying selected invariances in an adaptive network”. In: *Advances in neural information processing systems* 4 (1991) (cit. on p. 45).

[Sma+22] F Smaldone, Nicola Scianca, Leonardo Lanari, and Giuseppe Oriolo. “From Walking to Running: 3D Humanoid Gait Generation via MPC”. In: *Frontiers in Robotics and AI* 9 (2022) (cit. on p. 23).

[SWT22] Oswin So, Ziyi Wang, and Evangelos A Theodorou. “Maximum entropy differential dynamic programming”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 3422–3428 (cit. on p. 24).

[Str00] Malcolm Strens. “A Bayesian framework for reinforcement learning”. In: *ICML*. Vol. 2000. 2000, pp. 943–950 (cit. on p. 1).

[Sut88] Richard S Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine learning* 3.1 (1988), pp. 9–44 (cit. on p. 17).

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (cit. on pp. 1, 16, 28).

[Tag+22] Andrea Tagliabue, Dong-Ki Kim, Michael Everett, and Jonathan P How. “Efficient guided policy search via imitation of robust tube MPC”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 462–468 (cit. on p. 26).

[Tam+17] Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. “Learning from the hindsight plan—episodic mpc improvement”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 336–343 (cit. on p. 25).

[Tan+18] Jie Tan, Tingnan Zhang, Erwin Coumans, et al. “Sim-to-Real: Learning Agile Locomotion For Quadruped Robots”. In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania, June 2018 (cit. on p. 17).

[TE07] Yuval Tassa and Tom Erez. “Least squares solutions of the HJB equation with neural network value-function approximators”. In: *IEEE transactions on neural networks* 18.4 (2007), pp. 1031–1041 (cit. on p. 45).

[TMT14] Yuval Tassa, Nicolas Mansard, and Emo Todorov. “Control-limited differential dynamic programming”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1168–1175 (cit. on pp. 32, 112).

[Ten+09] Srinivas Tennety, Saurabh Sarkar, Ernest L Hall, and Manish Kumar. “Support vector machines based mobile robot path planning in an unknown environment”. In: *Dynamic Systems and Control Conference*. Vol. 48920. 2009, pp. 395–401 (cit. on p. 18).

[TPB01] Naftali Tishby, Fernando Pereira, and William Bialek. “The Information Bottleneck Method”. In: *Proceedings of the 37th Allerton Conference on Communication, Control and Computation* 49 (July 2001) (cit. on p. 28).

[TZ15] Naftali Tishby and Noga Zaslavsky. “Deep learning and the information bottleneck principle”. In: *2015 ieee information theory workshop (itw)*. IEEE. 2015, pp. 1–5 (cit. on p. 28).

[TET12] 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*. IEEE. 2012, pp. 5026–5033 (cit. on p. 24).

[Ton+18] Steve Tonneau, Andrea Del Prete, Julien Pettré, et al. “An efficient acyclic contact planner for multiped robots”. In: *IEEE Transactions on Robotics* 34.3 (2018), pp. 586–601 (cit. on p. 110).

[Too+93] Symbolic Math Toolbox et al. “Matlab”. In: *Mathworks Inc* (1993) (cit. on p. 20).

[Tou09] Marc Toussaint. “Robot trajectory optimization using approximate inference”. In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 1049–1056 (cit. on p. 56).

[Vas10] Massimiliano Vasile. “Finite elements in time: a direct transcription method for optimal control problems”. In: *AIAA/AAS Astrodynamics Specialist Conference*. 2010, p. 8275 (cit. on p. 20).

[Vie+18] Julian Viereck, Jules Kozolinsky, Alexander Herzog, and Ludovic Righetti. “Learning a structured neural network policy for a hopping task”. In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 4092–4099 (cit. on p. 26).

[VMR22] Julian Viereck, Avadesh Meduri, and Ludovic Righetti. “ValueNetQP: Learned one-step optimal control for legged locomotion”. In: *Learning for Dynamics and Control Conference*. PMLR. 2022, pp. 931–942 (cit. on p. 26).

[WB06] Andreas Wächter and Lorenz T Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical programming* 106.1 (2006), pp. 25–57 (cit. on p. 20).

[WFK22] Ke Wang, Hengyi Fei, and Petar Kormushev. “Fast Online Optimization for Terrain-Blind Bipedal Robot Walking with a Decoupled Actuated SLIP Model”. In: *Frontiers in Robotics and AI* 9 (2022) (cit. on p. 23).

[Wan+18] Xin Wang, Wenhan Xiong, Hongmin Wang, and William Yang Wang. “Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 37–53 (cit. on p. 18).

[WD92] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3 (1992), pp. 279–292 (cit. on p. 17).

[Wat89] Christopher John Cornish Hellaby Watkins. “Learning from delayed rewards”. In: (1989) (cit. on p. 1).

[Wei+18] Minggao Wei, Song Wang, Jinfan Zheng, and Dan Chen. “UGV navigation optimization aided by reinforcement learning-based path tracking”. In: *IEEE Access* 6 (2018), pp. 57814–57825 (cit. on p. 18).

[Wen+20] Qingsong Wen, Liang Sun, Fan Yang, et al. “Time series data augmentation for deep learning: A survey”. In: *arXiv preprint arXiv:2002.12478* (2020) (cit. on p. 55).

[Wer92] Paul Werbos. “Approximate dynamic programming for realtime control and neural modelling”. In: *Handbook of intelligent control: neural, fuzzy and adaptive approaches* (1992), pp. 493–525 (cit. on p. 45).

[Wil92] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3 (1992), pp. 229–256 (cit. on p. 17).

[WK88] Andrew Witkin and Michael Kass. “Spacetime constraints”. In: *ACM Siggraph Computer Graphics* 22.4 (1988), pp. 159–168 (cit. on p. 2).

[WAP17] Anqi Wu, Mikio C Aoi, and Jonathan W Pillow. “Exploiting gradients and Hessians in Bayesian optimization and Bayesian quadrature”. In: *arXiv preprint arXiv:1704.00060* (2017) (cit. on p. 45).

[Xia+19] Jiadong Xiao, Lin Li, Yanbiao Zou, and Tie Zhang. “Reinforcement learning for robotic time-optimal path tracking using prior knowledge”. In: *arXiv preprint arXiv:1907.00388* (2019) (cit. on p. 18).

[Xie+19] Zhaoming Xie, Patrick Clary, Jeremy Dao, et al. “Iterative reinforcement learning based design of dynamic locomotion skills for cassie”. In: *arXiv preprint arXiv:1903.09537* (2019) (cit. on p. 1).

[XLH17] Zhaoming Xie, C Karen Liu, and Kris Hauser. “Differential dynamic programming with nonlinear constraints”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 695–702 (cit. on pp. 4, 33).

[Yah+17] Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. “Collective robot reinforcement learning with distributed asynchronous guided policy search”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 79–86 (cit. on p. 26).

[Yin19] Xue Ying. “An overview of overfitting and its solutions”. In: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing. 2019, p. 022022 (cit. on p. 86).

[Yu+19] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. “A review of recurrent neural networks: LSTM cells and network architectures”. In: *Neural computation* 31.7 (2019), pp. 1235–1270 (cit. on p. 55).

[ZK16] Sergey Zagoruyko and Nikos Komodakis. “Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer”. In: *arXiv preprint arXiv:1612.03928* (2016) (cit. on p. 45).

[Zha+16] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. “Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search”. In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 528–535 (cit. on p. 26).

[Zha+19] Zhizheng Zhang, Jiale Chen, Zhibo Chen, and Weiping Li. “Asynchronous episodic deep deterministic policy gradient: Toward continuous control in computationally complex environments”. In: *IEEE transactions on cybernetics* 51.2 (2019), pp. 604–613 (cit. on p. 26).

[Zho+13] Mingyuan Zhong, Mikala Johnson, Yuval Tassa, Tom Erez, and Emanuel Todorov. “Value function approximation and model predictive control”. In: *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE. 2013, pp. 100–107 (cit. on pp. 25, 56, 57).

[Zhu+19] Henry Zhu, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar. “Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 3651–3657 (cit. on p. 1).

List of Figures

2.1 Shooting Methods 21

3.1 Sobolev Regression 46
3.2 Adaptive sampling for the Manipulator Arm for the pose reaching task. 48
3.3 Example State trajectory computed DDP 49
3.4 Illustration of refine sub-sampling approach to generate a dataset for the Manipulator Pose reaching task, shown here in End-Effector Space. 50

4.1 Unicycle 62
4.2 Cart-Pole 63
4.3 Pendulum 64
4.4 Manipulator Arm 65

5.1 Predicted Value functions for Unicycle, Cart-Pole, Pendulum 70
5.2 Validation Losses for Unicycle, Cart-Pole, Pendulum 71
5.3 Bellman Residuals for Unicycle, Cart-Pole, Pendulum 71
5.4 Relation between T and ∂PVP_v for Unicycle 73
5.5 Relation between T and ∂PVP_v for Cart-Pole 73
5.6 Relation between T and ∂PVP_v for Pendulum 74
5.7 State trajectories computed by ∂PVP_v and DDP for different initial preview horizons T. 75
5.8 Mean Squared Errors for Unicycle, Cart-Pole, Pendulum under noise . . 76
5.9 Trajectories computed by ∂PVP_v 77
5.10 Mean Squared Errors for Unicycle, Cart-Pole, Pendulum 79
5.11 Sobolev Loss Curves 80
5.12 Sobolev and Classical Training : Loss Curves 80

5.13 Gradients of neural networks 81
5.14 Comparison with PPO 83
5.15 State trajectories computed by ∂PVP$_v$ and PPO 84
5.16 End-Effector trajectories computed by ∂PVP$_v$ and DDP 88
5.17 Bellman Residuals for manipulator arm 89
5.18 Illustration of predictions, value, and gradients of value, for the manipulator arm. 90
5.19 Illustration of MSE for manipulator arm in MPC. 91
5.20 ∂PVP$_v with refine sub-sampling$ 92

6.1 Warmstarts provided by U_γ, X_β for the goal-reaching task from different starting configurations in the Unicycle problem. 97
6.2 Histogram of rollouts under different precision factors 98
6.3 Cart-Pole : warmstart and roll-out - ∂PVP and DDP. 99
6.4 Pendulum : warm-start and roll-out - ∂PVP and DDP. 100
6.5 7 *dof* Manipulator pose reaching task : warm-start and roll-out 101

8.1 Crocoddyl Whole Body Contact Sequence 110
8.2 Crocoddyl Solver Schema 111

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Declaration

I solemnly swear that I am up to no good !

Toulouse, May 24, 2023

Amit Parag