PERTURBATION FEEDBACK APPROACHES IN STOCHASTIC OPTIMAL CONTROL : APPLICATIONS TO MODEL-BASED AND MODEL-FREE PROBLEMS IN ROBOTICS

A Thesis

by

KARTHIKEYA SHARMA PARUNANDI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Suman Chakravorty
Co-Chair of Committee,	Dileep Kalathil
Committee Members,	Manoranjan Majji
	Dylan Shell
Head of Department,	Rodney Bowersox

December 2019

Major Subject: Aerospace Engineering

ABSTRACT

Decision making under uncertainty is an important problem in engineering that is traditionally approached differently in each of the Stochastic optimal control, Reinforcement learning and Motion planning disciplines. One prominent challenge that is common to all is the 'curse of dimensionality' $i.e,$ the complexity of the problem scaling exponentially as the state dimension increases. As a consequence, traditional stochastic optimal control methods that attempt to obtain an optimal feedback policy for nonlinear systems are computationally intractable. This thesis explores the application of a near-optimal decoupling principle to obtain tractable solutions in both model-based and model-free problems in robotics.

The thesis begins with the derivation of a near-optimal decoupling principle between the open loop plan and the closed loop linear feedback gains, based on the analysis performed with the second-order expansion of the cost-to-go function. This leads to a deterministic perturbation feedback control based solution to fully observable stochastic optimal control problems. Basing on this idea of near-optimal decoupling, a model-based trajectory optimization algorithm called the 'Trajectory-optimized Perturbation Feedback Controller' (T-PFC) is proposed. Rather than aiming to solve for the general optimal policy, this algorithm solves for an open-loop trajectory first, followed by the feedback that is automatically entailed by the algorithm from the open-loop plan. The performance is compared against a set of baselines in several difficult robotic planning and control examples that show near identical performance to non-linear model predictive control (NMPC) while requiring much lesser computational effort.

Next, we turn on to the investigation of the model-free version of the problem, where a policy is learnt from the data, without incorporating system's theoretical model. We present a novel decoupled data-based control (D2C) algorithm that addresses this problem using a decoupled 'open loop - closed loop' approach. First, an open-loop deterministic trajectory optimization problem is solved using a black-box simulation model of the dynamical system. Then, a closed loop control is developed around this open loop trajectory by linearization of the dynamics about this nominal

trajectory. By virtue of linearization, a linear quadratic regulator based algorithm is used for the demonstration of the closed loop control. Simulation performance suggests a significant reduction in training time compared to other state of the art reinforcement learning algorithms.

Finally, an alternative method for solving the open-loop trajectory in D2C is presented (called as 'D2C-2.0'). Stemming from the idea of model-based 'Differential Dynamic Programming' (DDP), it possesses second-order convergence property (under certain assumptions) and hence is significantly faster to compute the solution than the original D2C algorithm. An efficient way of sampling from the environment to convert it to a model-free algorithm, along with the suitable line-search and regularization schemes are presented. Comparisons are made with the original version of D2C and a state-of-the-art reinforcement learning algorithm using a variety of examples in the MuJoCo simulator. In conclusion, limitations for each of the above methods are discussed and accordingly, some possible directions have been provided for the future work.

DEDICATION

In memory of $Dr.$ M. Balamuralikrishna, whose fascinating personality and prodigious works have been a great source of inspiration to me.

ACKNOWLEDGMENTS

I wish to extend my sincere appreciation to all those who supported, influenced and taught me valuable lessons (academic or otherwise) during my Masters at Texas A&M. I would like to start by thanking my advisor, Prof. Suman Chakravorty, for providing me all kinds of academic, financial and moral support to the best of his capacities. His guidance and support was instrumental in the great learning curve that I had over the past couple of years. In a world, especially in academic circles, where asking stupid questions is sometimes looked down upon, his way of answering my questions (though they are naive, trivial or stupid sometimes) with patience is my biggest takeaway from him. That gave me the courage and freedom to pursue beyond the ongoing research project and continuously discuss the same with him.

I would like to thank Dr. Dileep Kalathil for introducing me to the fascinating field of Reinforcement Learning. Attending his course and working with him in the D2C project allowed me to gain many interesting insights into understanding its connections with the theory of stochastic optimal control. I sincerely acknowledge his contribution in D2C that is included in the corresponding chapter in this thesis.

I'd like to express my deep appreciations to Dr. Manoranjan Majji and Dr. Dylan Shell for accepting the responsibilities of being on my thesis committee. I take this occasion to mention that their courses were truly helpful in my research and sometimes even in my job interviews.

I'd like to thank my senior in the lab, Saurav Agarwal, for his valuable advice in career and research, and also for teaching me the importance of good coding practices. Thanks to Ran for the fruitful discussions we had during our collaboration in D2C project. I sincerely acknowledge his work in the formulation and experimental results of D2C that is a part of this thesis. Also thanks to my other labmates, Naveed and Kartik, for sharing about your latest research and for your cooperation in multiple instances.

I'm grateful to many other fellow graduate students at Texas A&M who've helped me to a great deal. In particular, I'd like to thank my friend, Roshan Thomas, from whom I learnt playing

Raquetball, took some advanced Piano lessons and learnt about various aspects of how research and academia work. Special thanks to Haresh for giving me a great time in discussing research, technology, music and career. A special shout-out to my other buddies in the department - Raman Goyal, Vedang, Vaishnav and Niladri.

I'd like to thank the administrative staff in the department and at the university level that helped me in smoothly carrying out the paperwork at various stages. In particular, I thank Rowe Gail. She was the first person I'd reach for a query or any activities within the department. It was a pleasure working with her during my time as a member of AEGSC.

Last but not the least, I cannot thank enough my parents and my brother for their constant encouragement and support.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis (or) dissertation committee consisting of Professor Suman Chakravorty, Professor Dileep Kalathil of the Department of Electrical and Computer Engineering, Professor Manoranjan Majji of the Department of Aerospace Engineering and Professor Dylan Shell of the Department of Computer Science and Engineering.

A portion of the preliminary analysis of Chapter 2 is contributed by Professor Suman Chakravorty. Proofs for the theorems and lemmas in Chapter 3 are jointly provided by Professor Suman Chakravorty and Professor Dileep Kalathil. They also contributed to the other sections in the same chapter by proof reading and modifying or adding to the paragraphs. The description of the D2C algorithm and the results from its implementation in this chapter are provided by Ran Wang of the Department of Aerospace Engineering.

All other work conducted for the thesis (or) dissertation was completed by the student independently.

Funding Sources

Graduate study was partly funded by NSF ECCS grant 1637889 and I-Corps grant 1740544.

NOMENCLATURE

OGAPS	Office of Graduate and Professional Studies at Texas A&M University
RL	Reinforcement Learning
MPC	Model Predictive Control
OCP	Optimal Control Problem
RRT	Rapidly-exploring Random Tree
SDE	Stochastic Differential Equation
HJB	Hamilton-Jacobi-Bellman
T-PFC	Trajectory-optimized Perturbation Feedback Controller
NMPC	Nonlinear Model Predictive Control
ROS	Robot Operating System
KKT	Karush-Kuhn-Tucker
T-LQR	Trajectory-optimized Linear Quadratic Regulator
DDP	Differential Dynamic Programming
ILQR	Iterative Linear Quadratic Regulator
NLP	Non-Linear Programming

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND LITERATURE REVIEW

Decision-making under uncertainty is a well-studied problem since the origins of dynamic programming in 1950s for its wide range of applications across various disciplines of engineering [1]. It is studied under the subject of Stochastic optimal control for deriving control laws to control dynamic systems under various kinds of uncertainties [1]. In other words, it is concerned with controlling a system whose behavior is not completely predictable. The uncertainty could arise from either of the noise in an applied control signal, sensor measurement, theoretical model or other ignored environmental effects. The first challenge one faces in designing a control policy is to convert the raw information obtained from the system into a comprehensible and a meaningful entity. This is generally categorized under the problems of *state estimation* and *system identification*. The second challenge is to design a right control policy among several possible policies, in order for the system to achieve desired behavior. Hence, the three important aspects of decision-making under uncertainty are state estimation, system identification and stochastic control [1]. As this thesis deals with the class of problems that are considered under fully observed scenarios, it is assumed that we have a perfect state information or reasonably good state estimates that don't severely affect the proposed approaches when executed. Hence, we aim to address the stochastic control aspect and also briefly touch upon the system identification process. The following subsections provide a brief overview of the standard methods in each of the Stochastic optimal control, Motion Planning and Reinforcement Learning fields.

1.1 Stochastic Optimal Control

Let $\mathbf{x_t} \in \mathbb{R}^{n_x}$, $\mathbf{u_t} \in \mathbb{R}^{n_u}$, $\mu_t : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_u}$, $\mathbf{w_t} \in \mathbb{R}^{n_w}$ denote the state of the system, the control signal, the policy and the uncertainty parameter at time t respectively. Let $U_t(.) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_w} \rightarrow \mathbb{R}$ represent the utility function and $f_t(.) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_w} \rightarrow \mathbb{R}^{n_x}$ be the system dynamical model respectively, both at time t. Then, the general problem of stochastic optimal control is given as shown in Problem 1 [2]. Note that $g(.)$ is a function of the state and

control that represents general equality or inequality constraints. Also, $\mathbf{x_{init}}$ is assumed to be the initial state of the system.

Problem 1: Stochastic optimal control problem

$$\underset{\mu_0,\mu_1,\cdots,\mu_{N-1}}{\text{minimize}} \; U_H(\mathbf{x_H}) + \sum_{t=0}^{H-1} U_t(\mathbf{x_t}, \mu_t(\mathbf{x_t}), \mathbf{w_t})$$

such that

$$\mathbf{x_{t+1}} = f_t(\mathbf{x_t},\ \mu(\mathbf{x_t}), \mathbf{w_t}),$$

$$g_t(\mathbf{x_t}, \mu(\mathbf{x_t})) <= 0$$

$$\mathbf{x_0} = \mathbf{x_{init}}$$

Throughout this thesis, we consider that the noise is Gaussian distributed and is additive in nature. This is a common assumption made in order to better analyze the effects of noise. Although multiplicative noise is the next common case [3], it is out of the scope as far as thesis is concerned.

1.2 Motion Planning

Motion planning, as the name suggests, is concerned with driving the state of a system from its current state to a goal state under certain constraints such as obstacle avoidance, actuator saturation, non-holonomic constraints etc. As the state of a system grows, so does its complexity. As a result, motion planning is typically performed in a hierarchy of planners. For example, most motion planners have a global planner and a local planner [4]. Sampling based planners are a popular set of methods for the former and trajectory optimization is performed for the later. The following subsections review each of these planners. Note that in the scope of this work, we only consider static obstacles although some of the techniques presented in this thesis are mildly applicable in dynamic cases, typically by re-planning.

1.2.1 Sampling-based Planners

Sampling-based planners rely on random sampling in the collision-free configuration space and have achieved widespread popularity for offering several path planning algorithms that are probabilistically complete. One such popular class of algorithms is the 'Rapidly Exploring Random Trees' (RRTs) [5]. It is an incremental sampling and searching approach that works by continually building a tree until it reaches a pre-defined vicinity of the goal. It is probabilistically complete [4], which means, if it is feasible to find a path, the algorithm is bound to find one. The catch here being that it might take a very long time to result in a satisfactory solution. Also, it doesn't offer any guarantees on the optimality of the path. RRT* (read as 'RRT star') is later proposed in [6] to guarantee the convergence to the optimality, when the number of nodes approach infinity. The optimality is based on the metric based on which the tree is built (for example, Euler's distance). Essentially, it also involves an additional step of rewiring the tree after connecting the new sample to the nearest node in the tree. Though it guarantees optimality, it is slower for online computation as compared to the original RRT [4]. There are several variants that have emerged later to address the above issues, such as

Though RRT is fast and to an extent, practical for online path planning, it doesn't take into consideration the non-holonomic path constraints that restrict the movement of the robots. Non-holonomic constraints, by definition, are the differential constraints that cannot be directly integrated to remove time derivatives of the state variables. In informal terms, these constraints not only depend on the current state, but also on the history of states of the system. Majority of the interesting problems in Robotic path planning lie in dealing with the non-holonomic constraints [4]. [7] paved a way for incorporating these constraints while planning a path. In other words, the algorithm results not only in a path, but one that is feasible for the given system to travel along. Hence, it is used to generate an initial guess for some trajectory optimization problems in this thesis.

1.2.2 Trajectory Optimization

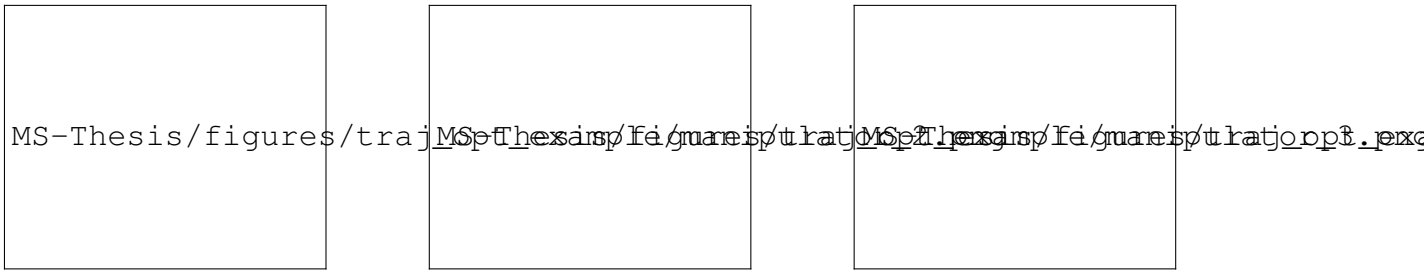
Trajectory optimization in motion planning is the process of obtaining a trajectory that minimizes a user-specified cost function while satisfying some constraints. It is formulated as an optimal control problem and the available methods can be categorized under two types : (i) Direct methods and (ii) Indirect methods [8].

Direct methods discretize the optimization problem and feed it to a non-linear program (NLP) solver to solve the optimization problem. The dynamical model of the system and other constraints involving the state and the control are passed as constraints to the optimization problem. This is also called as 'direct transcription', and is generally solved using one of the (i) Collocation methods or (ii) Direxct shooting methods (ii) Pseudospectral methods. Direct collocation methods are implicitly Runge-Kutta methods and typically approximate the states and the controls using piece-wise polynomials [8] such as trapezoidal collocation scheme, Hermite-Simpson scheme etc. Shooting methods parameterize the optimization problem entirely in terms of the control sequence and the states are generated by a forward rollout of the control trajectory. On the other hand, pseudospectral methods represent the entire trajectory as a *global* polynomial with orthogonally collocated points. This is in contrast with the local collocation methods in which the number of collocation points are varied and the degree of the polynomial is fixed. Whereas in the pseudospectral methods, the degree of polynomial is varied with the number of collocation points being fixed [9]. Originally developed to solve the computational fluid dynamical problems, they have gained popularity due to their exponentially convergence properties [9]. Popular pseudospectral methods have employed Lagrange, Legendre and Chebyshev polynomials as basis functions. Consequently, the three popular methods are (i) Legendre pseudospectral method (ii) Chebyshev pseudospectral method and (iii) Jacobi pseudospectral method (Gauss-Lobatto formulation) [9].

Indirect methods, instead of transcribing the original problem to a non-linear program, start with the analytical construction of the sufficient conditions for local optimality. They are then discretized to obtain a numerical solution. To paraphrase the difference with direct methods from [8], "Direct methods discretize and then optimize, whereas indirect methods optimize and then

discretize". Due to this, they tend to result in more accurate solutions [8]. But on the other hand, they require better initializations of the optimization variables and become complicated with the inclusion of various kinds of constraints. Such scenarios are precisely where the direct methods fare well [8].

Differential Dynamic Programming (DDP) based methods have recently gained popularity due to the minor but important modifications made to its original formulation by [3].It is an indirect shooting method, in that, the trajectory optimization problem is iteratively solved using a set of equations resulted from the local optimality condition. The local optimality condition finds its origins in the well-celebrated Pontryagin's principle [10]. While the original formulation involved computation of the second order derivatives of the dynamics terms, modified methods such as ILQR [3] [11] [10] [12] dropped them by compensating with Gauss-Newton approximations, regularization and line search schemes. The consequence of it is that it has faster convergence and can also be fit into the MPC framework. This advantage of ILQR forms the basis of the D2C-2.0 approach, a model-free reinforcement learning problem that is presented in this thesis.



(a) (b) (c)

Figure 1.1: Illustration of the control sequence obtained by trajectory optimization being executed on a 6-DOF UR5 manipulator. Arrows (in pink) indicate the end effector's pose during the course of its path traversal(shown in green).

1.3 Reinforcement Learning

The field of 'Reinforcement Learning' (hereon also called as RL) is concerned with obtaining policies for a system to achieve the desired behavior. A policy, here, is a mapping from a state to an action. It could be a look-up table as in discrete state spaces or a function of the general state, such that given the distribution of the current state, it engenders an action to be taken. The desired behavior is obtained by defining a reward corresponding to every state and the action taken at that state, and then looking for policies that maximize the cumulative reward over time. This is similar to other domains, say, optimal control where we look for a control law that minimizes the cumulative cost or a utility maximum problem in Microeconomics [13]. Reinforcement Learning differs in that, it aims to solve for such problems where the state transition model is not known or no assumptions are made about it. This is where the learning component of RL seeps in. Experiments (or simulations, for that matter!) are conducted for a system to explore its environment in a trial-and-error fashion. This lets the system learn more about the relation between its current state, the action, next state and the reward obtained in transition. As it explores its surroundings, it 'discovers' the behavior that is driving it to maximize the cumulative reward. This resembles the way humans and animals learn their behavior [13]. Their activities are driven by Dopamine, a neuro-transmitter in their brain that is involved in reward processing. Similarly, for any system here, the more it explores, better are the chances of finding an optimal such policy. However, this is a never-ending problem in most cases, in that, the state-space is large enough in most real-world problems that it is not possible to explore in its entirety. So, the policies have to be designed from the incomplete information that the system has about its environment. In other words, it has to exploit from the available information to come up with the best possible policy. This dilemma is widely prevalent in the broad field of RL and is popularly called as the 'Exploration vs. Exploitation Dilemma'.

1.3.1 Markov Decision Processes

As discussed in the earlier subsection, the crux of the RL lies around the notion of decision-making or obtaining policies. Finite Markov Decision Process (MDP) is a discrete time stochastic control process that provides a mathematical framework to model and formalize sequential decision-making [13]. They are an extension of Markov chains along with the addition of 'actions' at each 'state' guided by 'rewards'.

In MDPs, the decision maker is called the `agent`. Everything else surrounding it is the `environment`. The agent interacts with its environment by taking a discrete action a_t, and by observing the change in its state s_t and the corresponding reward r_t. This is shown in the figure 1.2. The transitions in the state can be described by its state transition model as $p(s_{t+1}|s_t, a_t)$ and are guided by the Markov property which says that given the current state, the history of past states is irrelevant in determining the next state. Mathematically, if S_t denotes the random variable of the state at time t and s_t be its corresponding value, then, $p(S_{t+1} = s_{t+1}|S_t = s_t) = p(S_{t+1} = s_{t+1}|S_t = s_t, S_{t-1} = s_{t-1}, \ldots, S_0 = s_0)$. To summarize, a Markov Decision Process (MDP) is a tuple of states - S, actions - A, rewards - R and a transition model T.

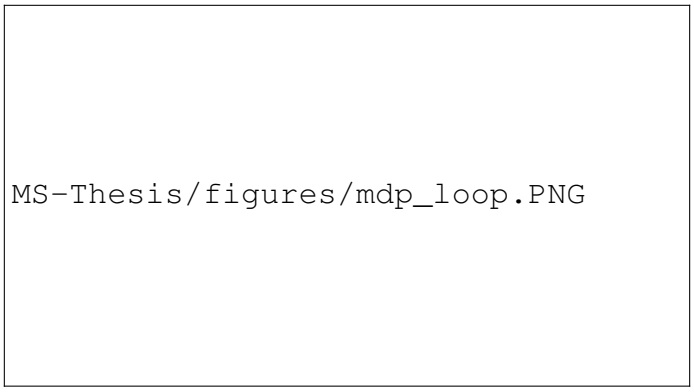


Figure 1.2: Markov Decision Process (MDP) : Agent takes an 'action' that is applied on the environment. In return, environment gives the information regarding the state (assuming fully-observed scenario) and the reward corresponding to the state-action pair.

1.3.2 Model-free RL methods

Model-free RL is concerned with learning the optimal policy without directly learning the transition probability in the process. Q-learning is a popular model-free reinforcement learning algorithm that tries to learn the state-action value function, also called the Q-function. The most important step in Q-learning is a value iteration update over the current average based on the new information, which is as follows [13]:

$$Q^{new}(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma\, max_a Q(s_{t+1}, a)),$$

where, s_t, a_t, r_t are the state, action and reward at time t respectively. γ and α are the discount factor and the learning rate respectively.

Methods based on Q-learning, especially those that used powerful function approximators such as neural networks to represent the Q-function witnessed a great success in the past decade [14] [15] [16] [17] [18]. However, they are all pertained to discrete action-spaces and are not directly applicable to continuous domains, where most of the real-world robotics problems are located. Policy gradient based algorithms are widely used in order to deal with the continuous action spaces [19]. Here, a general stochastic policy, say $\pi_\theta(a_t|s_t)(= Pr(A_t = a_t|S_t = s_t; \theta_t = \theta))$, is parameterized in terms of a set of parameters θ, where a_t is the action at time t and s_t is the state at time t. The performance objective can be written as $J(\pi_\theta) = \mathbb{E}_{\pi_\theta}[\sum_{i=t}^{\infty} r_i(s_i, \pi_\theta(s_i))\gamma^{i-t}|\pi_\theta]$. Since the policy is parameterized in terms of θ, we can rewrite the performance objective just as a function of θ as $J(\theta)$. The most straightforward way to determine the parameters θ is by gradient descent, as follows: $\theta_{t+1} = \theta_t + \alpha\nabla_{\theta_t} J(\theta_t)$. However, it is challenging to directly computing the above gradient. The *policy gradient theorem* has an elegant simplification to the above gradient as follows [19] [13]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^\pi,\ a\sim\pi_\theta}[\nabla_\theta log(\pi_\theta(a|s;\theta)Q^\pi(s,a)]$$

or

$$\nabla_\theta J(\theta) \propto \sum_s \rho^\pi(s) \sum_a Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s;\theta)$$

A notable fact from the above equation is that the policy gradient does not depend on the gradient of the state distribution. This makes it practically convenient to apply in model-free continuous reinforcement learning problems. A large number of widely successful algorithms [20] [19] [21] [22] [23] in the continuous action spaces that have emerged from the so-called actor-critic architecture are based on policy gradient theorem. Deep Deterministic Policy Gradient (DDPG) [20] is one such popular algorithm that is used as a baseline for benchmarking the approaches proposed in this thesis. More in-depth details about the algorithm are provided in the appendix - A.0.3.

2. T-PFC : TRAJECTORY-OPTIMIZED PERTURBATION FEEDBACK APPROACH

Note : This chapter is reprinted from our published paper with the same title. [1]

2.1 Introduction

Stochastic optimal control is concerned with obtaining control laws under uncertainty, minimizing a user-defined cost function while being compliant with its model and constraints. This problem frequently arises in robotics, where, planning a robot's motion under sensor, actuator and environmental limitations is vital to achieve a commanded task. Online planning methods such as Model Predictive Control (MPC) have become popular of late over offline methods for their accuracy and ability to deal with uncertainty. However, approaches within the MPC framework take a toll on the on-board computational resources for recursively solving the optimal control problem. On the other hand, offline solutions are susceptible to drift, and cannot deal with a dynamic environment. In this work, we propose a composite approach that merges the merits of both approaches i.e, computation off-line and a robust feedback control online, while re-planning, unlike in MPC, is performed only rarely, and is typically required only beyond moderate levels of noise.

The main contributions of this chapter are as follows:

(a) to demonstrate the decoupling between the deterministic open-loop and the closed loop feedback control of perturbations, in a fully-observed stochastic optimal setting, that is near-optimal,

(b) to propose a novel method based on the aforementioned decoupling principle to deal with robotic stochastic optimal control problem, and

(c) to draw comparisons between the proposed approach and the non-linear MPC framework, aimed at re-examining the widespread use of non-linear MPC in robotic planning and control.

[1] Reprinted with permission from "T-PFC : Trajectory-Optimized Perturbation Feedback Approach" by K.S. Parunandi et al., 2019. Robotics and Automation - Letters, vol. 4, no. 4, pp. 3457-3464. Copyright 2019 by IEEE.

2.2 Related Work

In fully observable systems, decision-making is typically modeled as a Markov Decision Process (MDP). Methods that try to solve MDPs using dynamic programming/HJB face the 'curse of dimensionality' in high-dimensional spaces while discretizing the state space [24]. Hence, most successful/practical methods are based on Pontryagin's maximum principle [25] though it results in locally optimal solutions. Iterative methods such as ILQG [11], DDP [26] and stochastic DDP [27] fall under this category. They expand the optimal cost-to-go and the system dynamics about a nominal, which is updated with every iteration. ILQG relies on the quadratic expansion of the cost-to-go and a linear expansion of system dynamics. DDP/stochastic-DDP considers the quadratic approximation of both. The convergence of these methods is similar to Newton's method. These methods generally optimize the open loop and the linear feedback gain together in an iterative fashion. Differently, in our approach, owing to the decoupling, the open loop optimal control sequence is obtained using a state-of-the-art Nonlinear Programming (NLP) solver, and given this open loop sequence, the optimal feedback gain is obtained using the "decoupled" gain equations. This, in turn, avoids, the expensive recursive Ricatti solutions required by ILQG and DDP techniques.

Model Predictive Control (MPC) is a popular planning and control framework in robotics. It bypasses the curse of dimensionality by repeatedly generating open-loop controls through the numerical solution of a finite horizon constrained optimal control problem at every discrete time-step [28]. Initially employed in chemical process industry [29], MPC has found widespread application in robotics owing to its ability to handle nonlinearity and constraints. Currently, this framework is well-established in the field and has demonstrated success in diverse range of problems including manipulation [30], visual servoing [30], and motion planning. In robotic motion planning, MPC is widely in use for motion planning of mobile robots, manipulators, humanoids and aerial robots such as quadrotors [31]. Despite its merits, it can be computationally very expensive, especially in context of robot planning and control, since (a) unlike in process industries, typical robotic systems demand re-planning online at high frequency, (b) most systems have highly non-linear dynamical

models and (c) constraints apply both on state and controls. Hence, the nonlinear-MPC (NMPC) poses a number of challenges in practical implementation [32]. Lighter variants of MPC such as LMPC, explicit MPC [32], tube-based MPC [33] and other simplified NMPC-based methods [34][32] have emerged. However, LMPC gradually induces uncertainties and fails for highly non-linear systems where the range of linearization is narrow and inadequate [28]. Explicit MPC is not practical for higher state and input states due to expensive memory requirements [32]. In [35], the authors proposed a decoupling principle under a small noise assumption and demonstrated first order near optimality of the decoupled control law for general non-linear systems.

This chapter derives a near-optimal decoupling principle that consists of a nominal open loop controls sequence along with a precisely defined linear feedback law dependent on the open loop. The latter is derived using a perturbation expansion of the Dynamic Programming equation, that is near optimal to second order, and hence, can work for even moderate noise levels. Further, we perform an extensive empirical comparison of our proposed technique [36], termed as the "Trajectory-optimized Perturbation Feedback Control (T-PFC)", with the NMPC technique, that shows near identical performance up to moderate noise levels, while taking approximately as much as $100\times$ lesser time than NMPC to execute in some examples.

2.3 Problem Formulation and Preliminaries

This section outlines the details of the system considered and the problem statement.

2.3.1 System Description

Let $\mathbf{x_t} \in \mathcal{X} \subset \mathbb{R}^{n_x}$ and $\mathbf{u_t} \in \mathcal{U} \subset \mathbb{R}^{n_u}$ denote the system state and the control input at time t respectively, with $\mathcal{X}$ and $\mathcal{U}$ being corresponding vector spaces. We consider a control-affine nonlinear state propagation model as $\mathbf{x_{t+1}} = f(\mathbf{x_t}) + g(\mathbf{x_t})\mathbf{u_t} + \epsilon\sqrt{dT}\omega_\mathbf{t}$, where, $\omega_\mathbf{t} \in \mathcal{N}(0, \mathbf{I})$ is an i.i.d. zero mean Gaussian noise with variance $\mathbf{I}$. It is derived from the noiseless continuous model: $\dot{\mathbf{x}}_\mathbf{t} = \bar{f}(\mathbf{x_t}) + \bar{g}(\mathbf{x_t})\mathbf{u_t}$, as follows: Let dT be the discretization time for the continuous time

Stochastic Differential Equation (SDE) : $\mathbf{dx} = \bar{f}(\mathbf{x})dT + \bar{g}(\mathbf{x})\mathbf{u}dT + \epsilon\mathbf{dw}$, where ϵ is a scaling factor and $\mathbf{w}$ denotes a Wiener process (Standard Brownian motion). It is assumed throughout this chapter that ϵ is small (small noise assumption). Note that $\mathbf{dw} = \sqrt{dT}\omega_\mathbf{t}$. The discrete time dynamics are obtained from the SDE as follows: $f(\mathbf{x_t}) = \mathbf{x_t} + \bar{f}(\mathbf{x_t})dT$, $g(\mathbf{x_t}) = \bar{g}(\mathbf{x_t})dT$ and the noise term becomes $\epsilon\sqrt{dT}\omega_\mathbf{t}$, where $\omega_\mathbf{t}$ are standard normal random variables. The reason we explicitly introduce the discretization time dT will become clear later in this section. It is assumed from hereon that $O(dT^2)$ terms are negligible, i.e, the discretization time is small enough.

2.3.2 Problem Description

Given an initial state $\mathbf{x_0}$, the problem of a discrete-time stochastic optimal control [37] for a fully observed control-affine system is to solve

$$\begin{aligned} &\min_{\pi} \ \mathop{\mathbb{E}}_{\omega_t}\left[C_N(\mathbf{x_N}) + \sum_{t=0}^{N-1} C_t(\mathbf{x_t}, \mathbf{u_t})\right] \\ s.t \quad &\mathbf{x_{t+1}} = f(\mathbf{x_t}) + g(\mathbf{x_t})\mathbf{u_t} + \epsilon\sqrt{dT}\omega_t \end{aligned} \tag{2.1}$$

for a sequence of admissible control policies $\pi = \{\pi_0, \pi_1, ..\pi_t, ., \pi_{N-1}\}$, where $\pi_t : \mathcal{X} \to \mathcal{U}$, $C_t : \mathcal{X} \times \mathcal{U} \to \mathcal{R}$ denotes the incremental cost function and $C_K : \mathcal{X} \to \mathcal{R}$, the terminal cost. The definition of a policy π_t at time t gives the relation between the state and the control signal at time t as : $\mathbf{u_t} = \pi_t(\mathbf{x_t})$.

2.3.3 Definitions

Let $(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})$ represent the nominal trajectory of the system, with its state propagation described by the model, $\bar{\mathbf{x}}_\mathbf{t+1} = f(\bar{\mathbf{x}}_\mathbf{t}) + g(\bar{\mathbf{x}}_\mathbf{t})\bar{\mathbf{u}}_\mathbf{t}$. Let $(\delta\mathbf{x_t}, \delta\mathbf{u_t})$ denote the perturbation from its nominal trajectory at $(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})$, defined by $\delta\mathbf{x_t} = \mathbf{x_t} - \bar{\mathbf{x}}_\mathbf{t}, \delta\mathbf{u_t} = \mathbf{u_t} - \bar{\mathbf{u}}_\mathbf{t}$. Now, by Taylor's expansion of the state propagation model about $(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})$ and the zero mean $\mathbf{w_t}$, the state perturbation at time t can be written as $\delta\mathbf{x_{t+1}} = A_t\delta\mathbf{x_t} + B_t\delta\mathbf{u_t} + \epsilon\sqrt{\mathbf{dT}}\omega_\mathbf{t} + r_t$, where $A_t = \frac{\partial f(\mathbf{x_t})}{\partial \mathbf{x_t}}|_{\bar{\mathbf{x}}_\mathbf{t}} + \frac{\partial g(\mathbf{x_t})}{\partial \mathbf{x_t}}|_{\bar{\mathbf{x}}_\mathbf{t}}\bar{\mathbf{u}}_\mathbf{t}$, $B_t = g(\bar{\mathbf{x}}_\mathbf{t})$ and r_t represents higher order terms in $\delta\mathbf{x_t}$ and $\delta\mathbf{u_t}$.

Let $\bar{J}_t(\mathbf{x_t})$ denote the optimal cost-to-go function at time t from $\mathbf{x_t}$ for the deterministic prob-

lem (i.e, $\epsilon = 0$), and $\bar{J}_t^{\epsilon}(\mathbf{x_t})$ denote the optimal cost-to-go function of the stochastic problem. We expand the deterministic cost-to-go quadratically about the nominal state in terms of state perturbations as $\bar{J}_t(\mathbf{x_t}) = \bar{J}_t(\bar{\mathbf{x}}_\mathbf{t}) + G_t\delta\mathbf{x_t} + \frac{1}{2}\delta\mathbf{x_t}^\intercal P_t\delta\mathbf{x_t} + q_t$, where, $G_t = \frac{\partial \bar{J}_t(\mathbf{x_t})}{\partial \mathbf{x_t}}^\intercal|_{\bar{\mathbf{x}}_\mathbf{t}}$, $P_t = \frac{\partial^2 \bar{J}_t(\mathbf{x_t})}{\partial^2 \mathbf{x_t}}|_{\bar{\mathbf{x}}_\mathbf{t}}$ and q_t denotes the higher order terms.

Finally, we consider a step cost function of the form $C_t(\mathbf{x_t}, \mathbf{u_t}) = l(\mathbf{x_t}) + \frac{1}{2}\mathbf{u_t}^\intercal R\mathbf{u_t}$ and let $L_t = \frac{\partial l(\mathbf{x_t})}{\partial \mathbf{x_t}}|_{\bar{\mathbf{x}}_\mathbf{t}}$ and $L_{tt} = \frac{\partial^2 l(\mathbf{x_t})}{\partial^2 \mathbf{x_t}}|_{\bar{\mathbf{x}}_\mathbf{t}}$. Using the definitions above, we assume that the functions $f(\mathbf{x_t})$, $\bar{J}(\mathbf{x_t})$ and $l(\mathbf{x_t})$ are sufficiently smooth over their domains such that the requisite derivatives exist and are uniformly bounded.

2.4 Methodology

This section states a near-optimal decoupling principle that forms the basis of the T-PFC algorithm presented in the next section. Our program in this section shall be as follows:

- *Decoupling:* First, we shall show that the optimal open loop control sequence of the deterministic problem (given by the gains G_t) can be designed independent of the closed loop gains determined by P_t, i.e, the P_t do not affect the G_t equations for an optimal control sequence in the deterministic problem.

- *Step A:* Next, we shall only keep the first two terms in the optimal deterministic feedback law, i.e., $\mathbf{u_t^l} = \bar{\mathbf{u}}_\mathbf{t} + K_t\delta\mathbf{x_t}$, and show that the closed loop performance of the truncated linear law is within $O(\epsilon^2 dT)$ of the full deterministic feedback law when applied to the *stochastic system*.

- *Step B:* Finally, we will appeal to a result by Fleming [38] that shows that the closed loop performance of the full deterministic law applied to the stochastic system is within $O(\epsilon^4 dT)$ of the optimal stochastic closed loop, and show that the stochastic closed loop performance of the truncated linear feedback law is within $O(\epsilon^2 dT)$ of the optimal stochastic closed loop

The scheme above is encapsulated in Fig. 2.1.

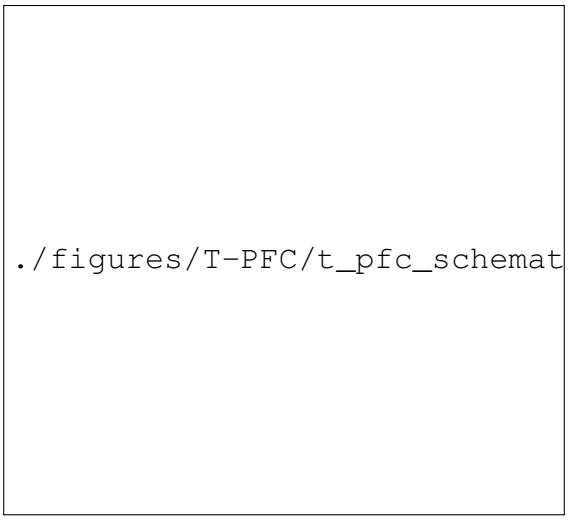


Figure 2.1: Schematic of the Near-Optimal Decoupling Principle

2.4.1 A Near Optimal Decoupling Principle

Proposition 1: Given an optimal nominal trajectory, the backward evolutions of the deterministic gain G_t and the covariance P_t of the optimal cost-to-go function $\bar{J}_t(\mathbf{x_t})$, initiated with $G_N = \frac{\partial \bar{C}_N(\mathbf{x_N})}{\partial \mathbf{x_N}}^\intercal|_{\bar{\mathbf{x}}_\mathbf{N}}$ and $P_N = \frac{\partial^2 \bar{C}_N(\mathbf{x_N})}{\partial^2 \mathbf{x_N}}|_{\bar{\mathbf{x}}_\mathbf{N}}$ respectively, are as follows:

$$G_t = L_t + G_{t+1} A_t \tag{2.2}$$

$$P_t = L_{tt} + A_t^\intercal P_{t+1} A_t - K_t^\intercal S_t K_t + G_{t+1} \otimes \tilde{R}_{t,xx} \tag{2.3}$$

for $t = \{0, 1, ..., N-1\}$, where,

$S_t = (R_t + B_t^\intercal P_{t+1} B_t), K_t = -S_t^{-1}(B_t^\intercal P_{t+1} A_t + (G_{t+1} \otimes \tilde{R}_{t,xu})^\intercal)$,

$\tilde{R}_{t,xx} = \nabla^2_{xx} f(\mathbf{x_t})|_{\bar{\mathbf{x}}_\mathbf{t}} + \nabla^2_{xx} g(\mathbf{x_t})|_{\bar{\mathbf{x}}_\mathbf{t}} \bar{\mathbf{u}}_\mathbf{t}$,

$\tilde{R}_{t,xu} = \nabla^2_{xu}(f(\mathbf{x_t}) + g(\mathbf{x_t})\mathbf{u_t})|_{\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t}}$

where ∇^2_{xx} represents the Hessian of a vector-valued function w.r.t. x and $\otimes$ denotes the tensor product.

Proof: Let $\bar{J}_t(\mathbf{x_t})$ be the optimal cost-to-go from the state $\mathbf{x_t}$. By definition and dynamic program-

ming principle respectively, it can be written as

$$\bar{J}_t(\mathbf{x_t}) = \min_{\mathbf{u_t}} J_t(\mathbf{x_t}, \mathbf{u_t}) = \min_{\mathbf{u_t}} \{C_t(\mathbf{x_t}, \mathbf{u_t}) + \bar{J}_{t+1}(\mathbf{x_{t+1}})\}$$

By Taylor's expansion of $\bar{J}_{t+1}(\mathbf{x_{t+1}})$ about $\bar{\mathbf{x}}_{\mathbf{t+1}}$ *i.e,* the nominal state at time $t+1$,

$$\bar{J}_{t+1}(\mathbf{x_{t+1}}) = \bar{J}_{t+1}(\bar{\mathbf{x}}_{\mathbf{t+1}}) + G_{t+1}\delta\mathbf{x_{t+1}} + \frac{1}{2}\delta\mathbf{x_{t+1}}^\intercal P_{t+1}\delta\mathbf{x_{t+1}} + q_{t+1}(\delta\mathbf{x_{t+1}}).$$

Substituting $\delta\mathbf{x_{t+1}} = A_t\delta\mathbf{x_t} + B_t\delta\mathbf{u_t} + r_t(\delta\mathbf{x_t}, \delta\mathbf{u_t})$ in the above expansion,

$$\begin{aligned}\bar{J}_{t+1}(\mathbf{x_{t+1}}) = \; & \bar{J}_{t+1}(\bar{\mathbf{x}}_{\mathbf{t+1}}) + G_{t+1}(A_t\delta\mathbf{x_t} + B_t\delta\mathbf{u_t} + r_t(\delta\mathbf{x_t}, \delta\mathbf{u_t}))+ \\ & (A_t\delta\mathbf{x_t} + B_t\delta\mathbf{u_t} + r_t(\delta\mathbf{x_t}, \delta\mathbf{u_t}))^\intercal P_{t+1}(A_t\delta\mathbf{x_t} + B_t\delta\mathbf{u_t} + r_t(\delta\mathbf{x_t}, \delta\mathbf{u_t})) \\ & + q_{t+1}(\delta\mathbf{x_{t+1}})\end{aligned}$$

Similarly, expand the incremental cost at time t about the nominal state,

$$\begin{aligned}C_t(\mathbf{x_t}, \mathbf{u_t}) = \; & \bar{l}_t + L_t\delta\mathbf{x_t} + \frac{1}{2}\delta\mathbf{x_t}^\intercal L_{tt}\delta\mathbf{x_t} + \frac{1}{2}\delta\mathbf{u_t}^\intercal R_t\bar{\mathbf{u}}_\mathbf{t} + \frac{1}{2}\bar{\mathbf{u}}_\mathbf{t}^\intercal R_t\delta\mathbf{u_t} + \frac{1}{2}\delta\mathbf{u_t}^\intercal R_t\delta\mathbf{u_t} \\ & + \frac{1}{2}\bar{\mathbf{u}}_\mathbf{t}^\intercal R_t\bar{\mathbf{u}}_\mathbf{t} + s_t(\delta\mathbf{x_t})\end{aligned}$$

$$\begin{aligned}J_t(\mathbf{x_t}, \mathbf{u_t}) = \; & \overbrace{[\bar{l}_t + \frac{1}{2}\bar{\mathbf{u}}_\mathbf{t}^\intercal R_t\bar{\mathbf{u}}_\mathbf{t} + \bar{J}_{t+1}(\bar{\mathbf{x}}_{\mathbf{t+1}})]}^{\bar{J}_t(\bar{\mathbf{x}}_\mathbf{t})} + \delta\mathbf{u_t}^\intercal(B_t^\intercal\frac{P_{t+1}}{2}B_t + \frac{1}{2}R_t)\delta\mathbf{u_t} + \\ & \delta\mathbf{u_t}^\intercal(B_t^\intercal\frac{P_{t+1}}{2}A_t\delta\mathbf{x_t} + \frac{1}{2}R_t\bar{\mathbf{u}}_\mathbf{t} + B_t^\intercal\frac{P_{t+1}}{2}r_t) + \\ & (\delta\mathbf{x_t}^\intercal A_t^\intercal\frac{P_{t+1}}{2}B_t + \frac{1}{2}\bar{\mathbf{u}}_\mathbf{t}R_t + r_t^\intercal\frac{P_{t+1}}{2}B_t + G_{t+1}B_t)\delta\mathbf{u_t} + \delta\mathbf{x_t}^\intercal A_t^\intercal\frac{P_{t+1}}{2}A_t\delta\mathbf{x_t} + \\ & \delta\mathbf{x_t}^\intercal\frac{P_{t+1}}{2}A_t^\intercal r_t + (r_t^\intercal\frac{P_{t+1}}{2}A_t + G_{t+1}A_t)\delta\mathbf{x_t} + \\ & r_t^\intercal\frac{P_{t+1}}{2}r_t + G_{t+1}r_t + q_t.\end{aligned}$$

$$\begin{aligned}\text{Now, } \min_{\mathbf{u_t}} J_t(\mathbf{x_t}, \mathbf{u_t}) &= \min_{\bar{\mathbf{u}}_\mathbf{t},\, \delta\mathbf{u_t}} J_t(\bar{\mathbf{x}}_\mathbf{t} + \delta\mathbf{x_t}, \bar{\mathbf{u}}_\mathbf{t} + \delta\mathbf{u_t}) \\ &= \min_{\bar{\mathbf{u}}_\mathbf{t}} J_t(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t}) + \min_{\delta\mathbf{u_t}} H_t(\delta\mathbf{x_t}, \delta\mathbf{u_t})\end{aligned} \tag{2.4}$$

First order optimality: At every state and control $(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})$ in the optimal nominal trajectory at time t, it follows from the minimum principle that

$$\frac{\partial C_t(\mathbf{x_t}, \mathbf{u_t})}{\partial \mathbf{u_t}} + \frac{\partial g(\mathbf{x_t})}{\partial \mathbf{u_t}}^\intercal \frac{\partial \bar{J}_{t+1}(\mathbf{x_{t+1}})}{\partial \mathbf{x_{t+1}}} = 0$$

$$\Rightarrow R_t \bar{\mathbf{u}}_\mathbf{t} + B_t^\intercal G_{t+1}^\intercal = 0 \tag{2.5}$$

By setting $\frac{\partial H_t(\delta\mathbf{x_t}, \delta\mathbf{u_t})}{\partial \delta\mathbf{u_t}} = 0$ and solving for optimal $\delta\mathbf{u_t}$, we get:

$$\begin{aligned}\delta\mathbf{u_t} &= -S_t^{-1}(R_t\bar{\mathbf{u}}_\mathbf{t} + B_t^\intercal G_{t+1}^\intercal) - S_t^{-1}(B_t^\intercal P_{t+1} A_t + (G_t \otimes \tilde{R}_{t,xu})^\intercal)\delta\mathbf{x_t} - S_t^{-1}(B_t^\intercal P_{t+1} r_t) \\ &= \underbrace{-S_t^{-1}(B_t^\intercal P_{t+1} A_t + (G_{t+1} \otimes \tilde{R}_{t,xu})^\intercal)}_{K_t}\delta\mathbf{x_t} + \underbrace{S_t^{-1}(-B_t^\intercal P_{t+1} r_t)}_{p_t} \text{ (using (2.5))}\end{aligned} \tag{2.6}$$

where, $S_t = R_t + B_t^\intercal P_{t+1} B_t$.

$$\Rightarrow \delta\mathbf{u_t} = K_t \delta\mathbf{x_t} + p_t.$$

Substituting it in the expansion of J_t and regrouping the terms based on the order of $\delta\mathbf{x_t}$ (till 2^{nd} order), we obtain:

$$\begin{aligned}\bar{J}_t(\mathbf{x_t}) = \; & \bar{J}_t(\bar{\mathbf{x}}_\mathbf{t}) + (L_t + (R_t\bar{\mathbf{u}}_\mathbf{t} + B_t^\intercal G_{t+1}^\intercal)K_t + G_{t+1}A_t)\delta\mathbf{x_t} + \\ & \frac{1}{2}\delta\mathbf{x_t}^\intercal(L_{tt} + A_t^\intercal P_{t+1} A_t - K_t^\intercal S_t K_t + G_{t+1} \otimes \tilde{R}_{t,xx})\delta\mathbf{x_t}\end{aligned}$$

By substituting the equality from (2.5) in the above expression, we obtain

$$\bar{J}_t(\mathbf{x_t}) = \bar{J}_t(\bar{\mathbf{x}}_\mathbf{t}) + (L_t + G_{t+1}A_t)\delta\mathbf{x_t} + \frac{1}{2}\delta\mathbf{x_t}^\intercal(L_{tt} + A_t^\intercal P_{t+1} A_t - K_t^\intercal S_t K_t + G_{t+1} \otimes \tilde{R}_{t,xx})\delta\mathbf{x_t}$$

Expanding the LHS about the optimal nominal state up to second order,

$$\bar{J}_t(\bar{\mathbf{x}}_\mathbf{t}) + G_t\delta\mathbf{x_t} + \frac{1}{2}\delta\mathbf{x_t}P_t\delta\mathbf{x_t} = \bar{J}_t(\bar{\mathbf{x}}_\mathbf{t}) + (L_t + G_{t+1}A_t)\delta\mathbf{x_t} + \frac{1}{2}\delta\mathbf{x_t}^\intercal(L_{tt} + A_t^\intercal P_{t+1}A_t - K_t^\intercal S_t K_t + G_{t+1} \otimes \tilde{R}_{t,xx})\delta\mathbf{x_t}$$

Balancing the first and the second order terms result in the following equations:

$$G_t = L_t + G_{t+1}A_t \tag{2.7}$$

$$P_t = L_{tt} + A_t^\intercal P_{t+1}A_t - K_t^\intercal S_t K_t + G_{t+1} \otimes \tilde{R}_{t,xx} \tag{2.8}$$

Note that P_t being the hessian of the cost-to-go function (which is scalar) is a symmetric matrix. In essence, the key step in the proof of proposition-1 is in realizing that when the nominal trajectory is optimal, the term corresponding to the open-loop control trajectory ($R\bar{\mathbf{u}}_\mathbf{t} + B_t^\intercal G_{t+1}^\intercal$) vanishes in deriving an expression for perturbed control ($\delta\mathbf{u_t}$) as shown in equation (2.6) and thereafter. This means that the dependency of the perturbed variables in the design of the nominal trajectory is nullified, resulting in equations (2.2) and (2.3). It may be noted here that equation (2.2) corresponds to the co-state equation following the first order optimality conditions over an optimal nominal trajectory, whereas equation (2.3) is a discrete time dynamic Riccati-like equation dictating the feedback law design. The consequence of the above result is that the second order sensitivity matrix in the expansion of the cost, P_t which determines the feedback gain K_t, doesn't influence the first order sensitivity matrix G_t (the co-state) that determines the optimal open-loop sequence. Thus, the decoupling between the nominal and linear feedback holds true. In other words, the design of an optimal control policy in a fully-observed problem as in (2.1) can be decoupled into the design of an open-loop deterministic nominal ($\bar{\mathbf{x}}_\mathbf{t}$, $\bar{\mathbf{u}}_\mathbf{t}$) and then a linear feedback law whose coefficients can be extracted through a recursive backpropagation of (2.2) and (2.3), but which is nonetheless near optimal to second order ($O(\epsilon^2 dT)$) as we shall show below.

Step A. Let the optimal deterministic feedback law for the deterministic system ($\epsilon = 0$) be given by: $\mathbf{u_t}(\mathbf{x_t}) = \bar{\mathbf{u}}_\mathbf{t} + K_t\delta\mathbf{x_t} + R(\delta\mathbf{x_t})$. The result above gives us the recursive equations required

to solve for $\bar{\mathbf{u}}_\mathbf{t}$ in terms of G_t, and K_t in terms of P_t. Consider the truncated linear feedback law, i.e., $\mathbf{u}^\mathbf{l}_\mathbf{t}(\mathbf{x_t}) = \bar{\mathbf{u}}_\mathbf{t} + K_t\delta\mathbf{x_t}$. Now, we shall apply the control laws $\mathbf{u_t}(.)$ and $\mathbf{u}^\mathbf{l}_\mathbf{t}(.)$ to the stochastic system ($\epsilon \neq 0$) and compare the closed loop performance. It can be shown that the state perturbations from the nominal under the optimal deterministic law evolve according to $\delta\mathbf{x_{t+1}} = \bar{A}_t\delta\mathbf{x_t} + B_tR(\delta\mathbf{x_t}) + S_t(\delta\mathbf{x_t}) + \epsilon\sqrt{dT}\omega_t$, while that under the truncated linear law evolves according to $\delta\mathbf{x}^\mathbf{l}_\mathbf{t+1} = \bar{A}_t\delta\mathbf{x}^\mathbf{l}_\mathbf{t} + S_t(\delta\mathbf{x}^\mathbf{l}_\mathbf{t}) + \epsilon\sqrt{dT}\omega_t$, where $\bar{A}_t = A_t + B_tK_t$ is the linear closed loop part, and $S_t(.)$ are the second and higher order terms in the dynamics. The closed loop cost-to-go under the full deterministic feedback law is then given by: $\bar{J}_k(\mathbf{x_k}) = E[\sum_{t=k}^N c(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t}) + C^1_t\delta\mathbf{x_t} + H_t(\delta\mathbf{x_t})]$, and that for the truncated linear law is given by: $\bar{J}^l_k(\mathbf{x_k}) = E[\sum_{t=k}^N c(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t}) + C^1_t\delta\mathbf{x}^\mathbf{l}_\mathbf{t} + H_t(\delta\mathbf{x}^\mathbf{l}_\mathbf{t})]$, where C^1_t is the first order coefficient of the step cost expansion that depend only on the nominal $(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})$, and $H_t(.)$ denote second and higher order terms of the expansions. Then $\bar{J}_k(\mathbf{x_k}) - \bar{J}^l_k(\mathbf{x_k}) = \sum_{t=k}^N \underbrace{E[C^1_t(\delta\mathbf{x_t} - \delta\mathbf{x}^\mathbf{l}_\mathbf{t})]}_{T_1} + \underbrace{\sum_{t=k}^N E[H_t(\delta\mathbf{x_t}) - H_t(\delta\mathbf{x}^\mathbf{l}_\mathbf{t})]}_{T_2}$. Consider the deviation between the two closed loops $\delta\mathbf{x_t} - \delta\mathbf{x}^\mathbf{l}_\mathbf{t} = \bar{A}_t(\delta\mathbf{x_t} - \delta\mathbf{x}^\mathbf{l}_\mathbf{t}) + B_tR_t(\delta\mathbf{x_t}) + S_t(\delta\mathbf{x_t}) - S_t(\delta\mathbf{x}^\mathbf{l}_\mathbf{t})$, where note that $||R_t(\delta\mathbf{x_t})|| = O(\epsilon^2 dT)$, as are $||S_t(\delta\mathbf{x}^\mathbf{l}_\mathbf{t})||$ and $||S_t(\delta\mathbf{x_t})||$ since they consist of second and higher order terms in the feedback law and the dynamics respectively, when $\epsilon\sqrt{dT}$ is small. Therefore, it follows that the closed loop state deviation between the full deterministic and the truncated linear law is $||\delta\mathbf{x_t} - \delta\mathbf{x}^\mathbf{l}_\mathbf{t}|| = O(\epsilon^2 dT)$. Further, it is also true that $\delta\mathbf{x_t}$ and $\delta\mathbf{x}^\mathbf{l}_\mathbf{t}$ are both $O(\epsilon\sqrt{dT})$. Hence, using the above it follows that terms $T_1 + T_2$ is $O(\epsilon^2 dT)$. *Therefore, it follows that the difference in the closed loop performance of the full deterministic feedback law and the truncated linear feedback law is* $|\bar{J}_k(\mathbf{x_k}) - \bar{J}^l_k(\mathbf{x_k})| = O(\epsilon^2 dT)$.

Step B: Now, we shall establish the closeness of the optimal stochastic closed loop and the stochastic closed loop under the truncated linear feedback law. First, we recount a seminal result due to Fleming [38] regarding the "goodness" of the deterministic feedback law for the stochastic system. Fleming considered the continuous time SDE: $d\mathbf{x} = \bar{f}(\mathbf{x})dt + g(\mathbf{x})\mathbf{u}dt + \epsilon d\mathbf{w}$. Let the cost-to-go of the optimal stochastic closed loop be given by $\bar{J}^\epsilon(t, \mathbf{x})$, and let the cost-to-go of the closed loop under the deterministic law be given by $\bar{J}(t, \mathbf{x})$. Then, it is shown that the functions $\bar{J}^\epsilon$ and $\bar{J}$ have

the following perturbation expansion in terms of ϵ: $\bar{J}^{\epsilon} = \varphi + \epsilon^2\theta + \epsilon^4\chi$,and $\bar{J} = \varphi + \epsilon^2\theta + \epsilon^4\chi'$, where φ, θ,χ and χ' are functions of $(t, \mathbf{x})$. Therefore, it follows that the difference in the closed loop performance between the optimal stochastic and optimal deterministic law *when applied to the stochastic system* is $O(\epsilon^4)$!

If we adapt this result to our discrete time case with a time discretization dT, where $O(dT^2)$ is negligible, then the difference between the true stochastic closed loop performance and that under the deterministic optimal law, $|J_t^{\epsilon}(\mathbf{x_t}) - J_t(\mathbf{x_t})| = O(\epsilon^4 dT)$. Thus, using the above result and the result form step A, it follows that *difference between the closed loop performance of the truncated linear feedback law and that of the optimal stochastic closed loop,* $|J_t^{\epsilon}(\mathbf{x_t}) - J_t^{l}(\mathbf{x_T})| = O(\epsilon^2 dT)$ *at the least.* This establishes the near optimality of the truncated linear feedback loop.

A note on comparison with ILQG/DDP: The condition in (2.2) is precisely when the ILQG/ DDP algorithms are deemed to have converged. However, that does not imply that the feedback gain at that stage for ILQG/ DDP is the same as that in Eq. (2.3), and in fact, the feedback gains of ILQG/ DDP are different from that in (2.3) as we shall see in our examples. The basic idea in the development above is to design an open loop optimal sequence, and then design a feedback gain according to (2.3), it is in this second step that we differ from ILQG/ DDP (which are methods to get open loop optimal sequences and make no claims about the feedback gains).

2.4.2 Trajectory-optimized Perturbation Feedback Control (T-PFC)

In this section, we formalize the Trajectory-optimized Perturbation Feedback Control (T-PFC) method based on the decoupling principle of the previous section.

2.4.2.1 Nominal Trajectory Design

The optimal nominal trajectory can be designed by solving the deterministic equivalent of problem (2.1), which can be formulated as an open-loop optimization problem as follows:

$$\min_{\bar{\mathbf{u}}} \left[C_N(\bar{\mathbf{x}}_\mathbf{N}) + \sum_{t=0}^{N-1} C_t(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t}) \right]$$

$$s.t \quad \bar{\mathbf{x}}_{\mathbf{t+1}} = f(\bar{\mathbf{x}}_\mathbf{t}) + g(\bar{\mathbf{x}}_\mathbf{t})\bar{\mathbf{u}}_\mathbf{t}$$

where, $\check{\mathbf{u}} = \{\bar{\mathbf{u}}_\mathbf{0}, \bar{\mathbf{u}}_\mathbf{1}, ..\bar{\mathbf{u}}_{\mathbf{N-1}}\}$. This is a design problem that can be solved by a standard NLP solver. The resultant open-loop control sequence together with a sequence of states obtained through a forward simulation of noise-free dynamics constitute the nominal trajectory.

Constraints on the state and the control can be incorporated in the above problem as follows:

State constraints: Nonlinear state constraints such as obstacle avoidance can be dealt by imposing exponential penalty cost as barrier functions. Obstacles can be circumscribed by Minimum Volume Enclosing Ellipsoids (MVEE) [39] that enclose a polygon given its vertices. Such kind of barrier functions can be formulated by [39] [40]: $C_{obs}(\bar{\mathbf{x}}_\mathbf{t}) = \sum_{m=1}^{n} \Gamma_m \exp(-\rho_m(\bar{\mathbf{x}}_\mathbf{t} - c^m)^\intercal \mathcal{E}^m(\bar{\mathbf{x}}_\mathbf{t} - c^m))$, where, c^m and $\mathcal{E}$ correspond to the center and geometric shape parameters of the m^{th} ellipsoid respectively. Γ_m and ρ_m are the scaling factors. Obstacles are assimilated into the problem by adding $C_{obs}(\bar{\mathbf{x}}_\mathbf{t})$ to the incremental cost $C_t(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})$.

Control bounds: Control bounds can safely be incorporated while designing the optimal nominal trajectory as hard constraints in the NLP solver. In this case, the constraints are linear in control inputs and let us assume they are of the form $F_t\bar{\mathbf{u}}_\mathbf{t} + H_t \leq 0$. The modified incremental cost function can be written as $C'_t(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t}) = C_t(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t}) + \mu_t(F_t\bar{\mathbf{u}}_\mathbf{t} + H_t)$, where μ_t is a lagrange multiplier for the aforementioned linear constraint. The first order condition (2.5) is then modified to $R_t\bar{\mathbf{u}}_\mathbf{t} + B_t^\intercal G_{t+1}^\intercal + F_t^\intercal \mu_t^\intercal = 0$ using KKT conditions [41], which upon utilizing in the derivation of expression for $\delta\mathbf{u}_\mathbf{t}$ nullifies the influence of μ_t. Hence, equations (2.2), (2.3) and (2.5) will remain the same.

2.4.2.2 Linear Feedback Controller Design

Given a nominal trajectory $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$, a linear perturbation feedback controller around it is designed by pre-computing the feedback gains. The sequence of K_t is determined by a backward pass of G_t and P_t as described by (2.2) and (2.3). The linear feedback control input is given by $\delta\mathbf{u}_\mathbf{t} = K_t\delta\mathbf{x}_\mathbf{t}$. Hence, $\mathbf{u}_\mathbf{t} = \bar{\mathbf{u}}_\mathbf{t} + \delta\mathbf{u}_\mathbf{t} = \bar{\mathbf{u}}_\mathbf{t} + K_t(\mathbf{x}_\mathbf{t} - \bar{\mathbf{x}}_\mathbf{t})$ forms the near-optimal online control policy. Algorithm-1 outlines the complete T-PFC algorithm.

Re-planning: At any point of time during the execution, if the cost deviates beyond a threshold from the nominal cost $i.e, C_{Th}$, a re-planning can be initiated.

Algorithm 1: T-PFC

Input: Initial State - $\mathbf{x_0}$, Goal State - $\mathbf{x_f}$, Time-step Δt, Horizon - N, System and environment parameters - $\mathcal{P}$;
$t \leftarrow 0$;
/* Run until the current state is in ε proximity to the goal */
while $\|\mathbf{x_t} - \mathbf{x_f}\| < \varepsilon$ **do**
  /* Plan at $t = 0$ and re-plan when the cost deviation exceeds a threshold or if not within the goal proximity at $t = N - 1$. */
  if *t == 0* or *Cost fraction* $> C_{Th}$ or *t == N-1* **then**
    /* Open-loop sequence */
    $(\bar{\mathbf{x}}_{\mathbf{t:N-1}}, \bar{\mathbf{u}}_{\mathbf{t:N-1}}) \leftarrow$Plan$(\mathbf{x_t}, \mathcal{P}, \mathbf{x_f})$
    /* Closed-loop parameters */
    Compute parameters:$\{P_{t:N-1}, G_{t:N-1}, K_{t:N-1}\}$
  end if
  Policy evaluation: $\mathbf{u_t} \leftarrow \bar{\mathbf{u}}_\mathbf{t} + K_t(\mathbf{x_t} - \bar{\mathbf{x}}_\mathbf{t})$
  Process: $\mathbf{x_{t+1}} \leftarrow f(\mathbf{x_t}) + g(\mathbf{x_t})\mathbf{u_t} + \epsilon\omega_\mathbf{t}$
  $t \leftarrow t + 1$
end while

2.5 Example Applications

This section demonstrates T-PFC in simulation with three examples. The `Gazebo` [42] robotics simulator is used as a simulation platform in interface with `ROS` middleware [43]. Numerical optimization is performed using the `Casadi` [44] framework employing the `Ipopt` [45] NLP software. A feasible trajectory generated by the non-holonomic version of the RRT algorithm [5] is fed into the optimizer for an initial guess. Simulations are carried out in a computer equipped with an Intel Core i7 2.80GHz octa-core processor. The results presented in each example are averaged from a set of 100 Monte Carlo simulations for a range of tolerable noise levels ϵ. The proposed approach has been implemented to the problem of motion planning under process noise in the dynamical model to obtain the cost plots and then simulated in a physics engine on a realistic robot model for further analysis.

Noise characterization: Process noise is modeled as a standard Brownian noise added to the system model with a standard deviation of $\epsilon\sqrt{dT}$. Since it is assumed to be additive Gaussian and

i.i.d. (even w.r.t. the noise in other state variables), it could account for various kinds of uncertainties including that of parametric, model and the actuator. ϵ is a scaling parameter that is varied to analyze the influence of the magnitude of the noise. Other case-specific parameters are provided in Table II.

For simulation, we use realistic physical robot models in a physics engine in an obstacle-filled environment along with moderate contact friction ($\mu = 0.9$) and drag, which are unaccounted for in our system model. Apart from this model uncertainty, we also introduce actuator noise through an additive Gaussian of standard deviation $\epsilon\sigma_t$, where σ_t is $\|\mathbf{u_s}\|_\infty$.

2.5.1 Car-like Robot

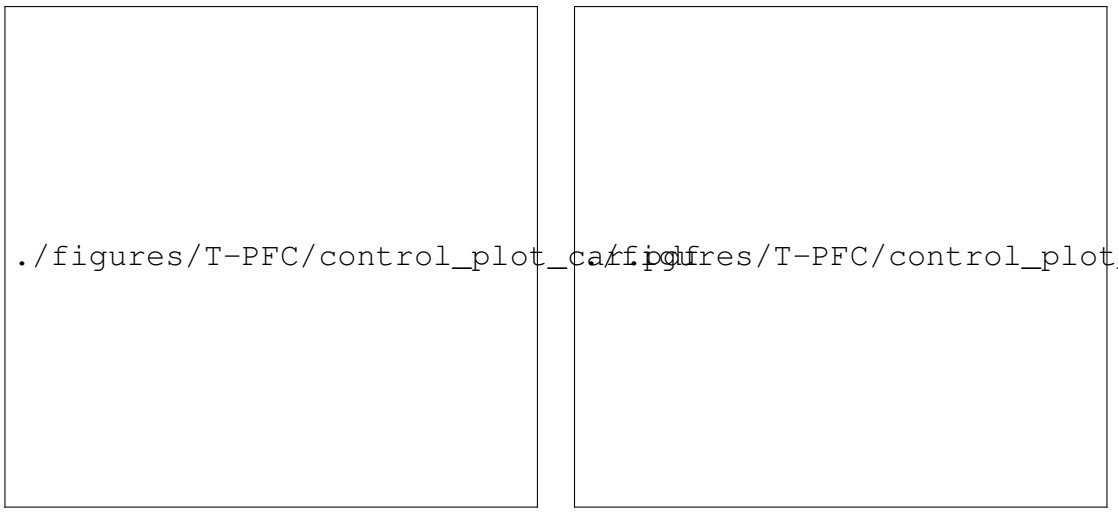
A 4-D model of a car-like robot with its state described by $(x_t, y_t, \theta_t, \phi_t)^\intercal$ is considered. For a control input constituting of the driving and the steering angular velocities, $(u_t, w_t)^\intercal$, the state propagation model is as follows:

$$\dot{x} = u\ cos(\theta), \quad \dot{\theta} = \frac{u}{L}\ tan(\phi)$$
$$\dot{y} = u\ sin(\theta), \quad \dot{\phi} = \omega$$

Fig. 4 shows an example path taken by a car-like robot in an environment filled with 8 obstacles enclosed in MVEEs. Plots in Fig. 2.3 indicate the averaged magnitude of both the nominal and the total control signals at $\epsilon = 0.25$. The standard deviation of the averaged total control sequence, in both plots, from the nominal is less than one percent of it.

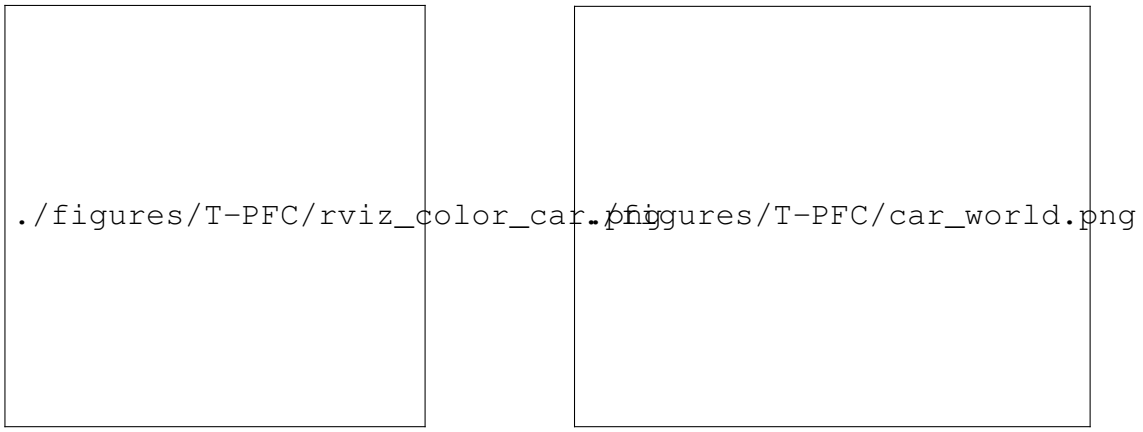
2.5.2 Car-like Robot with Trailers

With n trailers attached to a car-like robot, the state of a car-like-robot is augmented by n dimensions, each additional entry describing the heading angle of the corresponding trailer. In the



(a) (b)

Figure 2.2: Optimal nominal and total control inputs (averaged) at ϵ = 0.25 for (a) a car-like robot and (b) car with trailers



(a) Rviz trajectory (b) Robot's world in Gazebo

Figure 2.3: Motion Planning of a car-like robot using T-PFC for an additive control noise of standard deviation = 25% of the norm of saturation controls $i.e$, $\epsilon = 0.25$. The axes along and perpendicular to the robot's trajectory are indicated in red and green colors respectively.

current example, n = 2 trailers are considered and their heading angles are given by [5]:

$$\dot{\theta}_1 = \frac{u}{L} sin(\theta - \theta_1)$$
$$\dot{\theta}_2 = \frac{u}{L} cos(\theta - \theta_1) sin(\theta_1 - \theta_2)$$

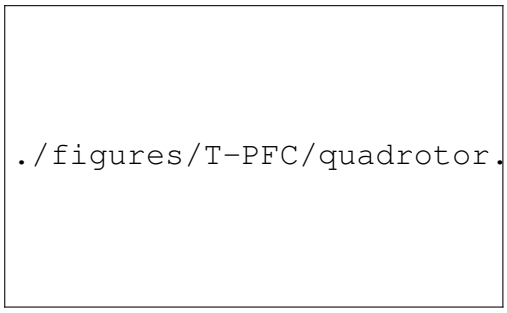
Hence, the robot has six degrees of freedom. Its world is considered to be composed of four obstacles as shown in Fig. 2.4. The robot, its environment and its trajectory shown are at $\epsilon = 0.25$.

(a) Rviz trajectory (b) Robot's world in Gazebo

Figure 2.4: Motion planning of a car with trailers using T-PFC for an additive control noise of standard deviation set to 25% of the norm of saturation controls $i.e$, $\epsilon = 0.25$. The axes along and perpendicular to the robot's trajectory are indicated in red and green colors respectively.

2.5.3 3D Quadrotor

The 12 DOF state of a Quadrotor comprises of its position, orientation and corresponding rates - $(\mathbf{x_t}, \theta_\mathbf{t}, \mathbf{v_t}, \omega_\mathbf{t})^\intercal$. Forces and torques in its body frame are external inputs in the equations below. However, in the real world (and also in `Gazebo` simulation shown here) the control input is typically fed into the motors. Hence, we consider rotor velocities as the control input, which can be obtained by a linear transformation of forces and torques in body frame. The state propagation



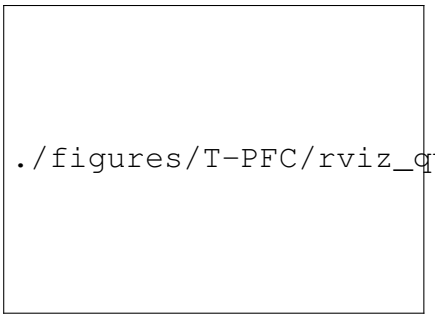


Figure 2.5: (a) Quadrotor's world in Gazebo - green boxes represent its initial and final positions respectively. (b) Example trajectory in rviz

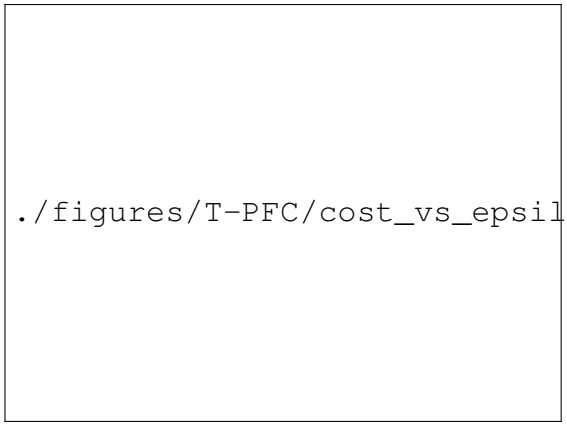
model is then given by the following [46]:

$$
\begin{aligned}
\dot{\mathbf{x}}_\mathbf{t} &= \mathbf{v}_\mathbf{t}, \\
\dot{\mathbf{v}}_\mathbf{t} &= \mathbf{g} + \frac{1}{m}(R_{\theta_t}\mathbf{F}_\mathbf{bt} - k_d\mathbf{v}_\mathbf{t}) \\
\dot{\theta}_\mathbf{t} &= J_w^{-1}\omega_\mathbf{t}, \\
\dot{\omega}_\mathbf{t} &= I_c^{-1}(\tau_\mathbf{t} - \omega_t \times I_c\omega_t)
\end{aligned}
$$

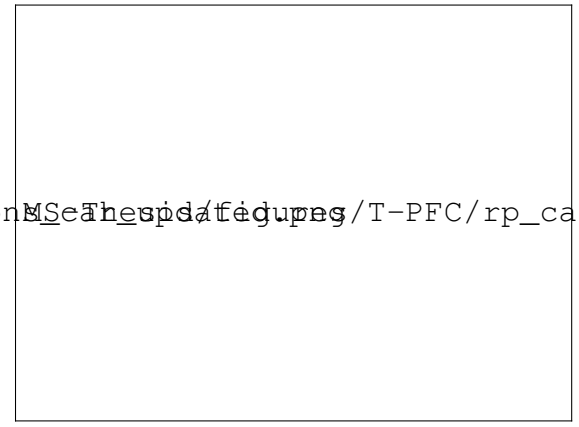
Simulations are performed using an AR.drone model [47] in an environment containing a cylindrical obstacle as shown in Fig. 2.5.

Table 2.1: Average run-time of algorithms in seconds

Robot type	MPC	T-LQR	ILQG	T-PFC
Car-like	447.89	4.48	161	4.52
Car with trailers	384.42	4.11	146	4.24
Quadrotor	71	3.33	49	3.5



(a) Cost comparison



(b) No. of replannings for $\epsilon > 0.25$

Figure 2.6: (a) Cost evolution over a feasible range of ϵ for a car-like robot, where ϵ is a measure of the noise in the system. Note that the performance of T-PFC is close to NMPC for a wide range of noise levels, while T-PFC takes approximately $100\times$ less time to execute (see Table I). (b) No. of re-plannings for above-moderate noise levels in the car-like robot simulation in gazebo using T-PFC is still around 8 times less than NMPC. Note that the re-planning for T-PFC starts at 2 = 0.25, whereas the no. of re-plannings for NMPC is always its horizon i.e, 229.

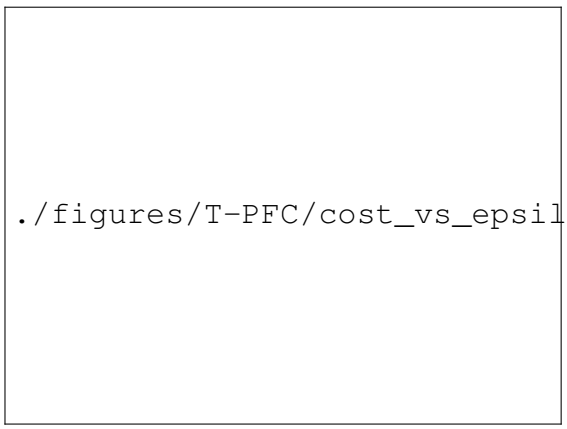


Figure 2.7: Cost evolution over a feasible range of ϵ for (a) car with trailers robot and (b) 3D Quadrotor.

Table 2.2: Simulation Parameters

	Car-like	Car with trailers	Quadrotor
$\mathbf{x_0}$	$(0,0,0,0)^\intercal$	$(0,0,0,0,0,0)^\intercal$	$(0,0,0.08,0,$ $0,0,0,0,0,0,0,0)'$
$\mathbf{x_f}$	$(5,5,0,0)^\intercal$	$(5,6,0,0,0,0)^\intercal$	$(2.6,2.4,1.75,$ $0,0,0)^\intercal$
$N, \Delta t$	229, 0.1s	180, 0.1s	60, 0.1s
Control bounds	$\mathbf{u_s^1} = (0.7, -0.7)$ $\mathbf{u_s^2} = (-1.3, 1.3)$	$\mathbf{u_s^1} = (0.7, -0.7)$ $\mathbf{u_s^2} = (-1.3, 1.3)$	$\mathbf{u_s^1} = (20, -20)$ $\mathbf{u_s^i} = (1, -1)$ $i = 2, 3, 4$

2.6 Discussion and Comparison of Methods

This section empirically details the implications of the decoupling principle and the T-PFC from the examples in the previous section. Further, we make a comparison here with the Nonlinear MPC (NMPC) [32], T-LQR [35] and ILQG [11]. Average cost incurred, rate of re-planning and time-taken for an execution are chosen as the performance criteria.

Nonlinear MPC: A deterministic NMPC is implemented with a typical OCP formulation, by re-solving it at every time-step. The NMPC variant implemented here is summarized in Algorithm-2. The prediction horizon is taken as $N - i$ at the i^{th} time-step. In other words, planning is performed all the way till the end rather than for next few time-steps as in typical MPC. This is done for two reasons:

(1) The control sequence obtained this way is equivalent to the deterministic optimal control law that includes higher order terms of feedback control. We wish to juxtapose it with T-PFC that only has a linear feedback (first-order).

(2) Due to high penalty cost of multiple barrier functions, the optimizer is prone to failures with smaller prediction horizons. Also, by the above arrangement, it follows from Bellman's Principle of Optimality that the predicted open-loop control input will be equal to the optimal feedback policy [32]. Therefore, this also results in nominal stability.

Algorithm 2: NMPC

Input: Initial State - $\mathbf{x_0}$, Goal State - $\mathbf{x_f}$, Horizon - N, System and environment parameters - $\mathcal{P}$;

$t \leftarrow 0$;

while $t < N$ **do**

  $(\bar{\mathbf{x}}_{\mathbf{t:N-1}}, \bar{\mathbf{u}}_{\mathbf{t:N-1}}) \leftarrow$ Plan$(\mathbf{x_t}, \mathbf{u_t}, N-t, \mathbf{x_f}, \mathcal{P})$;

  Process: $\mathbf{x_{t+1}} \leftarrow f(\mathbf{x_t}) + g(\mathbf{x_t})\bar{\mathbf{u}}_\mathbf{t} + \epsilon\omega_\mathbf{t}$

  $t \leftarrow t+1$;

end while

T-LQR: T-LQR is implemented using the same nominal cost as T-PFC. However, the cost parameters of the LQR are tuned entirely separately from the nominal cost [35].

ILQG: ILQG is initiated with the same initial guess as the above three methods. Since the cost contains exponential terms from the barrier functions, it is crucial to carefully choose right parameters for regularization and line search. Regularization is performed by penalizing state deviations in a quadratic modification schedule and an improved line search, both as mentioned in [11]. The feedback gains computed at the final iteration is used for feedback control against noise on top of the resulting open-loop trajectory.

Comparison: From Fig. 2.6 and 2.7, the average cost incurred for the systems in each simulation via T-PFC is close to that incurred through an NMPC approach. In other words, the cost accumulated by our perturbation linear feedback approach is nearly the same as that accumulated by an optimal deterministic control law over the feasible range of ϵ for T-PFC. T-LQR being based on the first order cost approximation, the cost rapidly diverges with increase in the noise level as reflected in Figs. 2.6 and 2.7. On the other hand, as ILQG doesn't make any claims regarding feedback, it is expected and is also clear from the same plots that the performance deteriorates rapidly with noise.

Table 2.1 shows the average time taken to execute an episode with each of the algorithms with no intermediate re-planning. The total execution time taken by NMPC is nearly 100 times the T-PFC in the most complex of the examples considered. The low online computational demand of T-PFC makes it scalable to implement in systems with higher dimensional state-space. ILQG, although has near-second order convergence property, is slower than NLP solver-based methods

such as T-PFC and T-LQR. This is because ILQG is derived from the quadratic approximation of the cost-to-go function and hence is slower for non-quadratic costs and highly nonlinear systems.

Another challenging aspect in the implementation of NMPC is generating initial guesses for online optimization. With a number of obstacle constraints or barrier functions, the NMPC optimizer fails to converge to a solution with trivial initializations and even with warm-starting, more so at higher noise levels. In contrast, T-PFC typically solves the optimization problem only once and hence, a one-time initialization is sufficient for the execution. Fig. 2.6 (b) shows the average rate of re-planning for example-1. Until $\epsilon = 0.25$, no re-planning was necessary in the example of a car-like robot. From Fig. 2.6 (b), it is evident that even at above-moderate levels of noise, the re-planning frequency is still eight times lesser than that required for an NMPC.

Unlike T-LQR, T-PFC also handles the residual second order terms of cost-to-go as well as system dynamics. This way, tuning is also bypassed as the feedback adapts itself according to the nominal cost. In contrast, T-LQR can apply aggressive controls during feedback depending on LQR parameter-tuning. T-PFC in an attempt to reduce the overall cost, generates smooth and small controls relative to its nominal. This is noticeable in Fig. 2.2. Also, this fact plays an advantage when the nominal control is on the constraint boundary and it is undesirable for the perturbation control to deviate significantly from the nominal.

The advantage of decoupling between the optimal nominal and the perturbation feedback law is clear when compared with ILQG. Parameter tuning in ILQG for regularization and line-search involves trial and error regulation and is often time consuming to searching for the right set of parameters to every given system, especially when the cost function is non-quadratic and non-polynomial. On the other hand, an NLP solver (using, say, interior-point methods) can be conveniently used in a black box fashion in perturbation feedback approaches such as T-PFC (or even T-LQR) without needing any fine-tuning to result in a deterministic control policy.

Small noise assumption: Though the theory is valid for small noise cases $i.e,$ for small epsilons, empirical results suggest a greater range of stability i.e, stability holds even for moderate levels of noise. As long as the noise falls in this range, a precise knowledge of the magnitude of noise is

irrelevant as T-PFC is insensitive to noise levels.

Is deterministic NMPC necessary? Since the MPC framework is broad and there are several ad-hoc techniques that could efficiently solve NMPC, answering this question requires a much deeper analysis. However, our central observation is that the T-PFC (and even T-LQR) method has near identical performance with deterministic NMPC in problems that mandate long horizons. They are also orders of magnitude computationally efficient, both according to the decoupling theory, as well as empirically, based on the problems that we have considered here. In such cases, why not use perturbation feedback techniques instead of NMPC at least until the noise levels predicate frequent re-planning?

3. D2C : DECOUPLED DATA BASED APPROACH FOR LEARNING TO CONTROL NONLINEAR DYNAMICAL SYSTEMS

3.1 Introduction

Controlling an unknown dynamical system adaptively has a rich history in control literature [1] [48]. This classical literature provides rigorous analysis about the asymptotic performance and the stability of the closed loop system. Classical adaptive control literature mainly focuses on non-stochastic systems [49] [50]. Stochastic adaptive control literature mostly addresses tractable models like linear quadratic regulator (LQR) where Riccati equation based closed form expressions are available for the optimal control law. Optimal control of an unknown nonlinear dynamical system with continuous state space and continuous action space is a significantly more challenging problem. Even with a known model, computing an optimal control law requires solving a dynamic programming problem. The 'curse of dimensionality' associated with dynamic programming makes solving such problems computationally intractable, except under special structural assumptions on the underlying system. *Learning to control* problems where the model of the system is unknown also suffer from this computational complexity issues, in addition to the usual identifiability problems in adaptive control.

Last few years have seen significant progresses in deep neural netwoks based reinforcement learning approaches for controlling unknown dynamical systems, with applications in many areas like playing games [51], locomotion [20] and robotic hand manipulation [52]. A number of new algorithms that show promising performance are proposed [22] [21] [53] and various improvements and innovations have been continuously developed. However, despite excellent performance on a number of tasks, reinforcement learning (RL) is still considered very data intensive. The training time for such algorithms are typically really large. Moreover, high variance and reproducibility issues on the performance are also reported [54]. While there have been some attempts to improve the sample efficiency [55], a systematic approach is still lacking.

This chapter proposes a novel decoupled data based control (D2C) algorithm for learning to control an unknown nonlinear dynamical system. The approach introduces a rigorous decoupling of the open loop (planning) problem from the closed loop (feedback control) problem. This decoupling allows us to come up with a highly sample efficient approach to solve the problem in a completely data based fashion. Our approach proceeds in two steps: (i) first, we optimize the nominal open loop trajectory of the system using a blackbox simulation model, (ii) then we identify the linear system governing perturbations from the nominal trajectory using random input-output perturbation data, and design an LQR controller for this linearized system. We show that the performance of D2C algorithm is approximately optimal, in the sense that the decoupled design is near optimal to second order in a suitably defined noise parameter. Moreover, simulation performance suggests significant reduction in training time compared to other state of the art algorithms.

3.2 Related Work

The solution approaches to the problem of controlling an unknown dynamical systems can be divided into two broad classes, (i) model-based methods and (ii) model-free methods.

In the model-based methods, many techniques [56] rely on a discretization of the underlying state and action space, and hence, run into the curse of dimensionality, the fact that the computational complexity grows exponentially with the dimension of the state space of the problem. The most computationally efficient among these techniques are trajectory-based methods such as differential dynamic programming (DDP) [26] [27] which linearizes the dynamics and the cost-to-go function around a given nominal trajectory, and designs a local feedback controller using DP. The iterative linear quadratic Gaussian (ILQG) [11] [57], which is closely related to DDP, considers the first order expansion of the dynamics (in DDP, a second order expansion is considered), and designs the feedback controller using Riccati-like equations, and is shown to be computationally more efficient. In both approaches, the control policy is executed to compute a new nominal trajectory, and the procedure is repeated until convergence.

Model-free methods, more popularly known as approximate dynamic programming [58] [37] or reinforcement learning (RL) methods [13], seek to improve the control policy by repeated in-

teractions with the environment, while observing the system's responses. The repeated interactions, or learning trials, allow these algorithms to compute the solution of the dynamic programming problem (optimal value/Q-value function or optimal policy) without explicitly constructing the model of the unknown dynamical system. Standard RL algorithms are broadly divided into value-based methods, like Q-learning, and policy-based methods, like policy gradient algorithms. Recently, function approximation using deep neural networks has significantly improved the performance of reinforcement learning algorithm, leading to a growing class of literature on 'deep reinforcement learning'. Despite the success, the amount of samples and training time required still seem prohibitive. On the other hand, works such as [59] demonstrated that simple policies such as the ones with linear parameterization showed a promising performance comparable to benchmark results obtained by policies represented using deep neural networks. This chapter presents our D2C method and carries a detailed analysis with some benchmark deep RL results along the same line specifically aiming to focus on aspects such as simplicity, data efficiency, reliability of training and reproducibility of results.

The rest of the chapter is organized as follows. In section 3.3, the basic problem formulation is outlined. In subsection 3.4.1, a decoupling result which solves the MDP in a "decoupled open loop-closed loop " fashion is briefly summarized. In subsection 3.4.4, we propose a decoupled data based control algorithm, with discussions of implementation problems. In section 3.5, we test the proposed approach using four typical benchmarking examples with comparisons to a state of the art RL technique.

3.3 Problem Formulation and Preliminaries

3.3.1 Problem Description

Consider the following discrete time nonlinear stochastic dynamical system:

$$\mathbf{x_{t+1}} = h(\mathbf{x_t}, \mathbf{u_t}, \mathbf{w_t}), \tag{3.1}$$

where $\mathbf{x_t} \in \mathcal{X} \in \mathbb{R}^{n_x}$, $\mathbf{u_t} \in \mathcal{U} \in \mathbb{R}^{n_u}$ are the state measurement and control vector at time k, respectively. The process noise $\mathbf{w_t}$ is assumed as zero-mean, uncorrelated Gaussian white noise, with covariance W.

The *optimal stochastic control* problem is to find the the control policy $\pi^o = \{\pi_1^o, \pi_2^o, \cdots, \pi_{T-1}^o\}$ such that the expected cumulative cost is minimized, i.e.,

$$\begin{aligned} \pi^o &= \arg\min_{\pi} \ \tilde{J}^{\pi}(\mathbf{x}), \ \text{ where,} \\ \tilde{J}^{\pi}(\mathbf{x}) &= \mathbb{E}_{\pi}\left[\sum_{t=1}^{T-1} c(\mathbf{x_t}, \mathbf{u_t}) + c_T(\mathbf{x_T}) | \mathbf{x_1} = \mathbf{x}\right], \end{aligned} \tag{3.2}$$

$u_t = \pi_t(\mathbf{x_t})$, $c(\cdot,\cdot)$ is the instantaneous cost function, and $c_T(\cdot)$ is the terminal cost function. In the following, we assume that the initial state x_1 is fixed, and denote $\tilde{J}^{\pi}(\mathbf{x})$ simply as $\tilde{J}^{\pi}$.

If the function $h(\cdot,\cdot,\cdot)$ is known exactly, then the optimal control law π^o can be computed using dynamic programming method. However, as noted before, this can be often computationally intractable. Moreover, when h is unknown, designing an optimal closed loop control law is a much more challenging problem. In the following, a data based decoupled approach is proposed for solving (3.2) when h is unknown.

3.4 Methodology

The central idea of our approach is that rather than directly finding the closed loop control law which requires solving a dynamic programming problem, we aim to address the original stochastic control problem in a "decoupled open loop - closed loop" fashion. In this approach: i) we solve an open loop deterministic optimization problem to obtain an optimal nominal trajectory in a model-free fashion, and then ii) we design a closed loop controller for the resulting linearized time-varying system around the optimal nominal trajectory, in a model-based fashion. This 'divide and conquer' strategy can be shown to be extremely effective. In this context, our major contributions are: 1) we show a near optimal parameterization of the feedback policy in terms of an open loop control sequence, and a linear feedback control law, 2) we show rigorously that the open loop and closed loop learning can be decoupled, which 3) results in the D2C algorithm that is highly data efficient

when compared to state of the art RL techniques. This chapter is a rigorous extension of our preliminary work [60, 61], in particular, it includes a stronger decoupling result, and an extensive empirical evaluation of the D2C algorithm with state of the art RL implementations on standard benchmark problems.

3.4.1 A Near Optimal Decoupling Principle

We first outline a near-optimal decoupling principle in stochastic optimal control that paves the way for the D2C algorithm described in Section 3.4.4.

We make the following assumptions for the simplicity of illustration. We assume that the dynamics given in (3.1) can be written in the form

$$\mathbf{x_{t+1}} = f(\mathbf{x_t}) + B_t(\mathbf{u_t} + \epsilon \mathbf{w_t}), \tag{3.3}$$

where $\epsilon < 1$ is a small parameter. We also assume that the instantaneous cost $c(\cdot, \cdot)$ has the following simple form,

$$c(\mathbf{x}, \mathbf{u}) = l(\mathbf{x}) + \frac{1}{2}\mathbf{u^T} R \mathbf{u}. \tag{3.4}$$

It is worth emphasizing that these assumptions such as quadratic control cost and affine in control dynamics are purely for the simplicity of treatment. These assumptions can be omitted at the cost of increased notational complexity.

3.4.2 Linearization w.r.t. Nominal Trajectory

Consider a noiseless version of the system dynamics given by (3.3). We denote the "nominal" state trajectory as $\bar{\mathbf{x}}_\mathbf{t}$ and the "nominal" control as $\bar{\mathbf{u}}_\mathbf{t}$ where $\mathbf{u_t} = \pi_t(\mathbf{x_t})$, where $\pi = (\pi_t)_{t=1}^{T-1}$ is a given control policy. The resulting dynamics without noise is given by $\bar{\mathbf{x}}_{\mathbf{t+1}} = f(\bar{\mathbf{x}}_\mathbf{t}) + B_t \bar{\mathbf{u}}_\mathbf{t}$.

Assuming that $f(\cdot)$ and $\pi_t(\cdot)$ are sufficiently smooth, we can linearize the dynamics about the

nominal trajectory. Denoting $\delta\mathbf{x_t} = \mathbf{x_t} - \bar{\mathbf{x}}_\mathbf{t}, \delta\mathbf{u_t} = \mathbf{u_t} - \bar{\mathbf{u}}_\mathbf{t}$, we can express,

$$\delta\mathbf{x_{t+1}} = A_t\delta\mathbf{x_t} + B_t\delta\mathbf{u_t} + S_t(\delta\mathbf{x_t}) + \epsilon\mathbf{w_t}, \tag{3.5}$$

$$\delta\mathbf{u_t} = K_t\delta\mathbf{x_t} + \tilde{S}_t(\delta\mathbf{x_t}), \tag{3.6}$$

where $A_t = \frac{\partial f}{\partial \mathbf{x}}|_{\bar{\mathbf{x}}_\mathbf{t}}$, $K_t = \frac{\partial \pi_t}{\partial \mathbf{x}}|_{\bar{\mathbf{x}}_\mathbf{t}}$, and $S_t(\cdot), \tilde{S}_t(\cdot)$ are second and higher order terms in the respective expansions. Similarly, we can linearize the instantaneous cost $c(\mathbf{x_t}, \mathbf{u_t})$ about the nominal values $(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})$ as,

$$\begin{aligned} c(\mathbf{x_t}, \mathbf{u_t}) = l(\bar{\mathbf{x}}_\mathbf{t}) + L_t\delta\mathbf{x_t} + H_t(\delta\mathbf{x_t}) + \\ \frac{1}{2}\bar{\mathbf{u}}_\mathbf{t}^T R\bar{\mathbf{u}}_\mathbf{t} + \delta\mathbf{u_t}^T R\bar{\mathbf{u}}_\mathbf{t} + \delta\mathbf{u_t}^T R\delta\mathbf{u_t}, \end{aligned} \tag{3.7}$$

$$c_T(\mathbf{x_T}) = c_T(\bar{\mathbf{x}}_\mathbf{T}) + C_T\delta\mathbf{x_T} + H_T(\delta\mathbf{x_T}), \tag{3.8}$$

where $L_t = \frac{\partial l}{\partial \mathbf{x}}|_{\bar{\mathbf{x}}_\mathbf{t}}$, $C_T = \frac{\partial c_T}{\partial \mathbf{x}}|_{\bar{\mathbf{x}}_\mathbf{t}}$, and $H_t(\cdot)$ and $H_T(\cdot)$ are second and higher order terms in the respective expansions.

Using (3.5) and (3.6), we can write the closed loop dynamics of the trajectory $(\delta\mathbf{x_t})_{t=1}^T$ as,

$$\delta\mathbf{x_{t+1}} = \underbrace{(A_t + B_tK_t)}_{\bar{A}_t}\delta\mathbf{x_t} + \underbrace{\{B_t\tilde{S}_t(\delta\mathbf{x_t}) + S_t(\delta\mathbf{x_t})\}}_{\bar{S}_t(\delta\mathbf{x_t})} + \epsilon\mathbf{w_t}, \tag{3.9}$$

where $\bar{A}_t$ represents the linear part of the closed loop systems and the term $\bar{S}_t(.)$ represents the second and higher order terms in the closed loop system. Similarly, the closed loop incremental cost given in (3.7) can be expressed as

$$\begin{aligned} c(\mathbf{x_t}, \mathbf{u_t}) = \underbrace{\{l(\bar{\mathbf{x}}_\mathbf{t}) + \frac{1}{2}\bar{\mathbf{u}}_\mathbf{t}^T R\bar{\mathbf{u}}_\mathbf{t}\}}_{\bar{c}_t} + \underbrace{[L_t + \bar{\mathbf{u}}_\mathbf{t}^T RK_t]}_{\bar{C}_t}\delta\mathbf{x_t} \\ + \underbrace{(K_t\delta\mathbf{x_t} + \tilde{S}_t(\delta\mathbf{x_t}))'R(K_t\delta\mathbf{x_t} + \tilde{S}_t(\delta\mathbf{x_t}))}_{\bar{H}_t(\delta\mathbf{x_t})}. \end{aligned} \tag{3.10}$$

Therefore, the cumulative cost of any given closed loop trajectory $(x_t, u_t)_{t=1}^{T}$ can be expressed as,

$$\begin{aligned} J^{\pi} &= \sum_{t=1}^{T-1} c(\mathbf{x_t}, \mathbf{u_t} = \pi_t(\mathbf{x_t})) + c_T(\mathbf{x_T}) \\ &= \sum_{t=1}^{T} \bar{c}_t + \sum_{t=1}^{T} \bar{C}_t \delta \mathbf{x_t} + \sum_{t=1}^{T} \bar{H}_t(\delta \mathbf{x_t}), \end{aligned} \tag{3.11}$$

where $\bar{c}_T = c_T(\bar{\mathbf{x}}_\mathbf{T}), \bar{C}_T = C_T$.

We first show the following result.

Lemma 1. *The state perturbation equation* $\delta \mathbf{x_{t+1}} = \bar{A}_t \delta \mathbf{x_t} + \bar{S}_t(\delta \mathbf{x_t}) + \epsilon \mathbf{w_t}$ *given in* (3.9) *can be equivalently characterized as*

$$\delta \mathbf{x_t} = \delta \mathbf{x_t^l} + \bar{\bar{S}}_t, \ \delta \mathbf{x_{t+1}^l} = \bar{A}_t \delta \mathbf{x_t^l} + \epsilon \mathbf{w_t} \tag{3.12}$$

where $\bar{\bar{S}}_t$ *is an* $O(\epsilon^2)$ *function that depends on the entire noise history* $\{\mathbf{w_0}, \mathbf{w_1}, \cdots \mathbf{w_t}\}$ *and* $\delta \mathbf{x_t^l}$ *evolves according to the linear closed loop system.*

Proof is provided in the appendix- A.0.1 [62].

Using (3.12) in (3.11), we can obtain the cumulative cost of any given closed loop trajectory as,

$$J^{\pi} = \underbrace{\sum_{t=1}^{T} \bar{c}_t}_{\bar{J}^{\pi}} + \underbrace{\sum_{t=1}^{T} \bar{C}_t \delta \mathbf{x_t}^l}_{\delta J_1^{\pi}} + \underbrace{\sum_{t=1}^{T} \bar{H}_t(\delta \mathbf{x_t}) + \bar{C}_t \bar{\bar{S}}_t}_{\delta J_2^{\pi}}. \tag{3.13}$$

Now, we show the following important result.

Proposition 1.

$$\tilde{J}^{\pi} = \mathbb{E}[J^{\pi}] = \bar{J}^{\pi} + O(\epsilon^2),$$

$$\mathit{Var}(J^{\pi}) = \underbrace{\mathit{Var}(\delta J_1^{\pi})}_{O(\epsilon^2)} + O(\epsilon^4).$$

From (3.13), we get,

$$\begin{aligned}\tilde{J}^{\pi} = \mathbb{E}[J^{\pi}] &= \mathbb{E}[\bar{J}^{\pi} + \delta J_1^{\pi} + \delta J_2^{\pi}], \\ &= \bar{J}^{\pi} + \mathbb{E}[\delta J_2^{\pi}] = \bar{J}^{\pi} + O(\epsilon^2),\end{aligned} \tag{3.14}$$

The first equality in the last line of the equations before follows from the fact that $\mathbb{E}[\delta \mathbf{x_t}^l] = 0$, since its the linear part of the state perturbation driven by white noise and by definition $\delta \mathbf{x_1}^l = 0$.The second equality follows form the fact that δJ_2^{π} is an $O(\epsilon^2)$ function. Now,

$$\begin{aligned}\text{Var}(J^{\pi}) &= \mathbb{E}[J^{\pi} - \tilde{J}^{\pi}]^2 \\ &= \mathbb{E}[\bar{J}_0^{\pi} + \delta J_1^{\pi} + \delta J_2^{\pi} - \bar{J}_0^{\pi} - \delta \tilde{J}_2^{\pi}]^2 \\ &= \text{Var}(\delta J_1^{\pi}) + \text{Var}(\delta J_2^{\pi}) + 2\mathbb{E}[\delta J_1^{\pi} \delta J_2^{\pi}].\end{aligned} \tag{3.15}$$

Since δJ_2^{π} is $O(\epsilon^2)$, $\text{Var}(\delta J_2^{\pi})$ is an $O(\epsilon^4)$ function. It can be shown that $\mathbb{E}[\delta J_1^{\pi} \delta J_2^{\pi}]$ is $O(\epsilon^4)$ as well (proof is given [62]). Finally $\text{Var}(\delta J_1^{\pi})$ is an $O(\epsilon^2)$ function because $\delta \mathbf{x}^{\mathbf{l}}$ is an $O(\epsilon)$ function. Combining these, we will get the desired result.

The following observations can now be made from Proposition 1.

Remark 1 (Expected cost-to-go)**.** *Recall that* $\mathbf{u_t} = \pi_t(\mathbf{x_t}) = \bar{\mathbf{u}}_\mathbf{t} + K_t \delta \mathbf{x_t} + \tilde{S}_t(\delta \mathbf{x_t})$*. However, note that due to Proposition 1, the expected cost-to-go,* $\tilde{J}^{\pi}$*, is determined almost solely (within* $O(\epsilon^2)$*) by the nominal control action sequence* $\bar{\mathbf{u}}_\mathbf{t}$*. In other words, the linear and higher order feedback terms have only* $O(\epsilon^2)$ *effect on the expected cost-to-go function.*

Remark 2 (Variance of cost-to-go)**.** *Given the nominal control action* $\bar{\mathbf{u}}_\mathbf{t}$*, the variance of the cost-to-go, which is* $O(\epsilon^2)$*, is determined overwhelmingly (within* $O(\epsilon^4)$*) by the linear feedback term* $K_t\delta\mathbf{x_t}$*, i.e., by the variance of the linear perturbation of the cost-to-go,* δJ_1^π*, under the linear closed loop system* $\delta\mathbf{x}^\mathbf{l}_{\mathbf{t+1}} = (A_t + B_tK_t)\delta\mathbf{x}^\mathbf{l}_\mathbf{t} + \epsilon\mathbf{w_t}$*.*

3.4.3 Decoupled Approach for Closed Loop Control

Proposition 1 and the remarks above suggest that an open loop control super imposed with a closed loop control for the perturbed linear system may be approximately optimal. We delineate this idea below.

Open Loop Design. First, we design an optimal (open loop) control sequence $\bar{u}_t^*$ for the noiseless system. More precisely,

$$\begin{aligned}(\bar{\mathbf{u}}^*_\mathbf{t})_{t=1}^{T-1} &= \arg\min_{(\tilde{\mathbf{u}}_\mathbf{t})_{t=1}^{T-1}} \sum_{t=1}^{T-1} c(\bar{\mathbf{x}}_\mathbf{t}, \tilde{\mathbf{u}}_\mathbf{t}) + c_T(\bar{\mathbf{x}}_\mathbf{T}), \\ \bar{\mathbf{x}}_{\mathbf{t+1}} &= f(\bar{\mathbf{x}}_\mathbf{t}) + B_t\tilde{\mathbf{u}}_\mathbf{t}.\end{aligned} \tag{3.16}$$

We will discuss the details of this open loop design in Section 3.4.4.

Closed Loop Design. We find the optimal feedback gain K_t^* such that the variance of the linear closed loop system around the nominal path, $(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}^*_\mathbf{t})$, from the open loop design above, is minimized.

$$\begin{aligned}(K_t^*)_{t=1}^{T-1} &= \arg\min_{(K_t)_{t=1}^{T-1}} \text{Var}(\delta J_1^\pi), \\ \delta J_1^\pi &= \sum_{t=1}^{T} \bar{C}_t {\mathbf{x}_\mathbf{t}}^l, \\ \delta {\mathbf{x}_{\mathbf{t+1}}}^l &= (A_t + B_tK_t)\delta {\mathbf{x}_\mathbf{t}}^l + \epsilon\mathbf{w_t}.\end{aligned} \tag{3.17}$$

We now characterize the approximate closed loop policy below.

Proposition 2. *Construct a closed loop policy*

$$\pi_t^*(\mathbf{x_t}) = \bar{\mathbf{u}}_\mathbf{t}^* + K_t^* \delta \mathbf{x_t}, \tag{3.18}$$

where $\bar{\mathbf{u}}_\mathbf{t}^*$ *is the solution of the open loop problem* (3.16)*, and* K_t^* *is the solution of the closed loop problem* (3.17)*. Let* π^o *be the optimal closed loop policy. Then,*

$$|\tilde{J}^{\pi *} - \tilde{J}^{\pi^o}| = O(\epsilon^2).$$

Furthermore, among all policies with nominal control action $\bar{\mathbf{u}}_\mathbf{t}^*$*, the variance of the cost-to-go under policy* π_t^**, is within* $O(\epsilon^4)$ *of the variance of the policy with the minimum variance.*

We have

$$\begin{aligned} \tilde{J}^{\pi^*} - \tilde{J}^{\pi^o} &= \tilde{J}^{\pi^*} - \bar{J}^{\pi^*} + \bar{J}^{\pi^*} - \tilde{J}^{\pi^o} \\ &\leq \tilde{J}^{\pi^*} - \bar{J}^{\pi^*} + \bar{J}^{\pi^o} - \tilde{J}^{\pi^o} \end{aligned}$$

The inequality above is due the fact that $\bar{J}^{\pi^*} \leq \bar{J}^{\pi^o}$, by definition of π^*. Now, using Proposition 1, we have that $|\tilde{J}^{\pi^*} - \bar{J}^{\pi^*}| = O(\epsilon^2)$, and $|\tilde{J}^{\pi^o} - \bar{J}^{\pi^o}| = O(\epsilon^2)$. Also, by definition, we have $\tilde{J}^{\pi^o} \leq \tilde{J}^{\pi^*}$. Then, from the above inequality, we get

$$|\tilde{J}^{\pi^*} - \tilde{J}^{\pi^o}| \leq |\tilde{J}^{\pi^*} - \bar{J}^{\pi^*}| + |\bar{J}^{\pi^o} - \tilde{J}^{\pi^o}| = O(\epsilon^2)$$

A similar argument holds for the variance as well.

Unfortunately, there is no standard solution to the closed loop problem (3.17) due to the non additive nature of the cost function $\text{Var}(\delta J_1^\pi)$. Therefore, we solve a standard LQR problem as a surrogate, and the effect is again one of reducing the variance of the cost-to-go by reducing the variance of the closed loop trajectories.

Approximate Closed Loop Problem. We solve the following LQR problem for suitably defined

cost function weighting factors Q_t, R_t:

$$\min_{(\delta \mathbf{u_t})_{t=1}^{T}} \mathbb{E}[\sum_{t=1}^{T-1} \delta \mathbf{x_t}^T Q_t \delta \mathbf{x_t} + \delta \mathbf{u_t}^T R_t \delta \mathbf{u_t} + \delta \mathbf{x_T}^T Q_T \delta \mathbf{x_t}],$$

$$\delta \mathbf{x_{t+1}} = A_t \delta \mathbf{x_t} + B_t \delta \mathbf{u_t} + \epsilon \mathbf{w_t}. \tag{3.19}$$

The solution to the above problem furnishes us a feedback gain $\hat{K}_t^*$ which we can use in the place of the true variance minimizing gain K_t^*.

Remark 3. *Proposition 1 states that the expected cost-to-go of the problem is dominated by the nominal cost-to-go. Therefore, even an open loop policy consisting of simply the nominal control action is within $O(\epsilon^2)$ of the optimal expected cost-to-go. However, the plan with the optimal feedback gain K_t^* is strictly better than the open loop plan in that it has a lower variance in terms of the cost to go. Furthermore, solving the approximate closed loop problem using the surrogate LQR problem, we can expect a lower variance of the cost-to-go function as well.*

3.4.4 D2C : Decoupled Data-based Control

This section presents the decoupled data-based control (D2C) algorithm formalizing the ideas from the previous section. First, it solves a model-free deterministic optimization problem to obtain the optimal nominal sequence. Then, a linear feedback controller is designed to track the open-loop trajectory. To summarize, D2C has three primary steps as follows:

1. Solve for the open-loop control sequence by using information from multiple episodic roll-outs of the system in a simulator followed by gradient descent.

2. Linearize the system around the optimal nominal trajectory and obtain the parameters of the resulting linear time-varying system, also called as system identification.

3. Design an LQR controller that tracks the optimal nominal control sequence. The linear feedback law combined with the open-loop control sequence constitutes the policy.

Each of the above steps are described in detail as follows:

3.4.4.1 Open-loop Optimal Trajectory

A first order gradient descent based algorithm is proposed here for solving the open loop optimization problem given in (3.16), where the underlying dynamic model is used as a blackbox, and the necessary gradients are found from a sequence of input perturbation experiment data using standard least square.

Denote the initial guess of the control sequence as $U^{(0)} = \{\bar{\mathbf{u}}_\mathbf{t}^{(0)}\}_{t=1}^{T}$, and the corresponding states $\mathcal{X}^{(0)} = \{\bar{\mathbf{x}}_\mathbf{t}^{(0)}\}_{t=1}^{T}$. The control policy is updated iteratively via

$$U^{(n+1)} = U^{(n)} - \alpha \nabla_U \bar{J}|_{\mathcal{X}^{(n)},U^{(n)}}, \tag{3.20}$$

where $U^{(n)} = \{\bar{\mathbf{u}}_\mathbf{t}^{(n)}\}_{t=1}^{T}$ denotes the control sequence in the n^{th} iteration, $\mathcal{X}^{(n)} = \{\bar{\mathbf{x}}_\mathbf{t}^{(n)}\}_{t=1}^{T}$ denotes the corresponding states, and α is the step size parameter. As $\bar{J}|_{\mathcal{X}^{(n)},U^{(n)}}$ is the expected cumulative cost under control sequence $U^{(n)}$ and corresponding states $\mathcal{X}^{(n)}$, the gradient vector is defined as

$$\nabla_U \bar{J}|_{\mathcal{X}^{(n)},U^{(n)}} = \left(\frac{\partial \bar{J}}{\partial \mathbf{u_1}} \quad \frac{\partial \bar{J}}{\partial \mathbf{u_2}} \quad \cdots \quad \frac{\partial \bar{J}}{\partial \mathbf{u_T}} \right) |_{\mathcal{X}^{(n)},U^{(n)}}, \tag{3.21}$$

which is the gradient of the expected cumulative cost w.r.t the control sequence after n iterations. The following paragraph elaborates on how to estimate the above gradient.

Let us define a rollout to be an episode in the simulation that starts from the initial settings to the end of the horizon with a control sequence. For each iteration, multiple rollouts are conducted sequentially with both the expected cumulative cost and the gradient vector updated iteratively after each rollout. During one iteration for the control sequence, the expected cumulative cost is calculated as

$$\bar{J}|^{j+1}_{\mathcal{X}^{(n)},U^{(n)}} = (1 - \frac{1}{j})\bar{J}|^{j}_{\mathcal{X}^{(n)},U^{(n)}} + \frac{1}{j}(J|_{\mathcal{X}^{j,(n)},U^{j,(n)}}), \tag{3.22}$$

where j denotes the j^{th} rollout within the current iteration process of control sequence. $\bar{J}|^{j}_{\mathcal{X}^{(n)},U^{(n)}}$

is the expected cumulative cost after j rollouts while $J|_{\mathcal{X}^{j,(n)},U^{j,(n)}}$ denotes the cost of the j^{th} rollout under control sequence $U^{j,(n)}$ and corresponding states $\mathcal{X}^{j,(n)}$. Note that $U^{j,(n)} = \{\bar{\mathbf{u}}_\mathbf{t}^{(n)} + \delta\mathbf{u}_\mathbf{t}^{\mathbf{j},(\mathbf{n})}\}_{t=1}^{T}$ where $\{\delta\mathbf{u}_\mathbf{t}^{\mathbf{j},(\mathbf{n})}\}_{t=1}^{T}$ is the zero-mean, i.i.d Gaussian noise added as perturbation to the control sequence $U^{(n)}$.

Then the gradient vector is calculated in a similar sequential manner as

$$\nabla_U \bar{J}|^{j+1}_{\mathcal{X}^{(n)},U^{(n)}} = (1-\frac{1}{j})\nabla_U \bar{J}|^{j}_{\mathcal{X}^{(n)},U^{(n)}} + \frac{1}{j\sigma_{\delta u}}(J|_{\mathcal{X}^{j,(n)},U^{j,(n)}} - \bar{J}|^{j+1}_{\mathcal{X}^{(n)},U^{(n)}})(U^{j,(n)} - U^{(n)}),$$

where $\sigma_{\delta u}$ is the variance of the control perturbation and $\nabla_U \bar{J}|^{j+1}_{\mathcal{X}^{(n)},U^{(n)}}$ denotes the gradient vector after j rollouts. Note that after each rollout, both the expected cumulative cost and the gradient vector are updated. The rollout number m in one iteration for the control sequence is decided by the convergence of both the expected cumulative cost and the gradient vector. After m rollouts, the control sequence is updated by equation (3.20) in which $\nabla_U \bar{J}|_{\mathcal{X}^{(n)},U^{(n)}}$ is estimated by $\nabla_U \bar{J}|^{m}_{\mathcal{X}^{(n)},U^{(n)}}$. Keep doing this until the cost converges and the optimized nominal control sequence is $\{\bar{\mathbf{u}}_\mathbf{t}^{*}\}_{t=1}^{T} = \{\bar{\mathbf{u}}_\mathbf{t}^{(\mathbf{N-1})}\}_{t=1}^{T}$.

Higher order approaches other than gradient descent are possible. However, for a general system, the gradient descent approach is easy to implement. Also it is memory efficient and highly amenable to parallelization as a result of our sequential approach.

3.4.4.2 System Identification

Closed loop control design specified in (3.17) or the approximate closed loop control design specified in (3.19) requires the knowledge of the parameters $A_t, B_t, 1 \leq t \leq T$, of the perturbed linear system. We propose a linear time variant (LTV) system identification procedure to estimate these parameters.

First start from perturbed linear system given by equation (3.19). Using only first order information and estimate the system parameters A_t, B_t with the following form

$$\delta\mathbf{x_{t+1}} = \hat{A}_t\delta\mathbf{x_t} + \hat{B}_t\delta\mathbf{u_t}, \tag{3.23}$$

rewrite with augmented matrix

$$\delta \mathbf{x_{t+1}} = [\hat{A}_t \mid \hat{B}_t] \begin{bmatrix} \delta \mathbf{x_t} \\ \delta \mathbf{u_t} \end{bmatrix}, \tag{3.24}$$

Now write out their components for each iteration in vector form as,

$$\begin{aligned} Y &= [\delta \mathbf{x_{t+1}^0} \delta \mathbf{x_{t+1}^1} \cdots \delta \mathbf{x_{t+1}^{N-1}}], \\ X &= \begin{bmatrix} \delta \mathbf{x_t^0} & \delta \mathbf{x_t^1} & \cdots & \delta \mathbf{x_t^{N-1}} \\ \delta \mathbf{u_t^0} & \delta \mathbf{u_t^1} & \cdots & \delta \mathbf{u_t^{N-1}} \end{bmatrix}, \\ Y &= [\hat{A}_t \mid \hat{B}_t] X, \end{aligned} \tag{3.25}$$

where N is the total iteration number. $\delta \mathbf{x_{t+1}^n}$ denotes the output state deviation , $\delta \mathbf{x_t^n}$ denotes the input state perturbations and $\delta \mathbf{u_t^n}$ denotes the input control perturbations at time t of the n^{th} iteration. All the perturbations are zero-mean, i.i.d, Gaussian random vectors whose covariance matrix is σI where I is the identity matrix and σ is a scalar. Note that here one iteration only has one rollout.

The next step is to apply the perturbed control $\{\bar{\mathbf{u}}_\mathbf{t}^* + \delta \mathbf{u_t^n}\}_{t=1}^T$ to the system and collect input-output experiment data in a blackbox simulation manner.

Using the least square method $\hat{A}_t$ and $\hat{B}_t$ can be calculated in the following procedure

$$YX' = [\hat{A}_t \mid \hat{B}_t] XX', \tag{3.26}$$

As the perturbations are zero-mean, i.i.d, Gaussian random noise, $XX' = \sigma NI$. Remember N is

the total iteration number. Then

$$
\begin{aligned}
[\hat{A}_t \mid \hat{B}_t] &= \frac{1}{\sigma N} Y X' \\
&= \frac{1}{\sigma N} [\delta \mathbf{x}_{\mathbf{t+1}}^{\mathbf{0}} \delta \mathbf{x}_{\mathbf{t+1}}^{\mathbf{1}} \cdots \delta \mathbf{x}_{\mathbf{t+1}}^{\mathbf{N-1}}] \begin{bmatrix} (\delta \mathbf{x}_{\mathbf{t}}^{\mathbf{0}})' & (\delta \mathbf{u}_{\mathbf{t}}^{\mathbf{0}})' \\ (\delta \mathbf{x}_{\mathbf{t}}^{\mathbf{1}})' & (\delta \mathbf{u}_{\mathbf{t}}^{\mathbf{1}})' \\ \vdots & \vdots \\ (\delta \mathbf{x}_{\mathbf{t}}^{\mathbf{N-1}})^T & (\delta \mathbf{u}_{\mathbf{t}}^{\mathbf{N-1}})^T \end{bmatrix}
\end{aligned} \tag{3.27}
$$

The calculation procedure can also be done in a sequential way similar to the update of the gradient vector in the open-loop optimization algorithm. Therefore it is highly amenable to parallelization and memory efficient.

3.4.4.3 Design of the Linear Feedback Law

Given the parameter estimate of the perturbed linear system, we solve the closed loop control problem given in (3.19). This is a standard LQR problem. By solving the Riccati equation, we can get the closed-loop optimal feedback gain K_t^*. The details of the design is standard and is omitted here.

3.4.4.4 D2C Algorithm: Summary

The Decoupled Data Based Control (D2C) Algorithm is summarized in Algorithm 3.

3.5 Discussion and Comparison of Methods

In this section, we compare the D2C approach with the well-known deep reinforcement learning algorithm - Deep Deterministic Policy Gradient (DDPG) [20]. For the comparison, we evaluate both the methods in the following three aspects:

- **Data efficiency in training** - the amount of data sampling and storage required to achieve a desired task.
- **Robustness to noise** - the deviation from the predefined task due to random noise in process in the testing stage.

Algorithm 3: D2C Algorithm

1) Solve the deterministic open-loop optimization problem for optimal open loop nominal control sequence and trajectory $(\{\bar{\mathbf{u}}_\mathbf{t}^*\}_{t=1}^T, \{\bar{\mathbf{x}}_\mathbf{t}^*\}_{t=1}^T)$ using gradient descent method (Section 3.4.4.1).
2) Identify the LTV system $(\hat{A}_t, \hat{B}_t)$ via least square estimation (Section 3.4.4.2).
3) Solve the Riccati equations using estimated LTV system equation for feedback gain $\{K_t^*\}_{t=1}^T$.
4) Set $t = 1$, given initial state $\mathbf{x_1} = \bar{\mathbf{x}}_\mathbf{1}^*$ and state deviation $\delta\mathbf{x_1} = 0$.
while $t \leq T$ **do**

$$\begin{aligned} \mathbf{u_t} &= \bar{\mathbf{u}}_\mathbf{t}^* + K_t^* \delta\mathbf{x_t}, \\ \mathbf{x_{t+1}} &= f(\mathbf{x_t}) + B_t(\mathbf{u_t} + \epsilon\mathbf{w_t}), \\ \delta\mathbf{x_{t+1}} &= \mathbf{x_{t+1}} - \bar{\mathbf{x}}_\mathbf{t+1}^* \end{aligned} \tag{3.28}$$

$t = t + 1$.
end while

- **Ease of training** - the challenges involved in training with either of the data-based approaches.

3.5.1 Tasks and Implementation

We tested our method with four benchmark tasks, all implemented in MuJoCo simulator [63]: Inverted pendulum, Cart-pole, 3-link swimmer and 6-link swimmer [64]. Each of the systems and their tasks are briefly defined as follows:

1. `Inverted pendulum` : A swing-up task of this 2D system from its downright initial position is considered.

2. `Cart-pole` : The state of a 4D under-actuated cart-pole comprises of the angle of the pole, cart's horizontal position and their rates. Within a given horizon, the task is to swing-up the pole and balance it at the middle of the rail by applying a horizontal force on the cart.

3. `3-link Swimmer` : The 3-link swimmer model has 5 degrees of freedom and together with their rates, the system is described by 10 state variables. The task is to solve the planning and control problem from a given initial state to the goal position located at the center of the

ball. Controls can only be applied in the form of torques to two joints. Hence, it is under-actuated by 3 DOF.

4. `6-link Swimmer` : The task with a 6-link swimmer model is similar to that defined in the 3-link case. However, with 6 links, it has 8 degrees of freedom and hence, 16 state variables, controlled by 5 joint motors.

5. `Fish` : The fish model moves in 3D space, the torso is a rigid body with 6 DOF. The system is described by 26 dimensions of states and 6 control channels. Controls are applied in the form of torques to the joints that connects the fins and tails with the torso. The rotation of the torso is described using quaternions.

D2C implementation is done in three stages corresponding to those mentioned in the previous section and 'MuJoCo Pro 150' is used as the environment for simulating the blackbox model. An off-the-shelf implementation of DDPG provided by `Keras-RL` [65] library has been customized for our simulations. It may be noted that the structure of the reward function is formulated to optimize the performance of DDPG and hence, different from that used in D2C. However, the episode length (or horizon) and the time-discretization step is held identical. Although continuous RL algorithms such as DDPG learn the policy and thereby interpolating it to the entire state space, we consider the task-based experiments in a finite limited time-frame window approach. Training is performed until the neural networks' loss is converged. Hence, though the episodic reward does seem to converge earlier, that itself may not indicate that the policy is converged.

For fair comparison, 'Episodic reward/cost fraction' is considered with both methods. It is defined as the fraction of reward obtained in an episode during training w.r.t the nominal episodic reward (converged reward). Note that the words 'reward' and 'cost' are being used interchangeably due to their different notions in optimal control and RL literature respectively, though they achieve the same objective. For simplicity, one is considered the negative of the other.

3.5.2 Performance Comparison

Data-efficiency: As mentioned above, an efficient training is one that requires minimal data

sampling in order to achieve the same behavior. One way of measuring it is to collate the times taken for the episodic cost (or reward) to converge during training. Plots in Fig. 3.1 show the training process with both methods on the systems considered. Each plot shows the training curve of one experiment. The curve marked as original is the actual training curve reflecting the original reward data. The one marked as filtered is the curve after smoothing out the spikes to show better view of the reward trend as the training goes. Table 3.1 delineates the times taken for training respectively. As the system identification and feedback gain calculation in case of D2C take only a small portion of time, the total time comparison in (Table 3.1) shows that D2C learns the optimal policy substantially faster than DDPG and hence, has a better data efficiency.

Robustness to noise: As with canonical off-policy RL algorithms, DDPG requires that an exploration noise be added to the policy, during training. Given that the training adapts the policy to various levels of noise, combined with hours of intense training and a nonlinear policy output, it is expected that it is more robust towards noise as is evident from Figs. 3.2 (c) and 3.2 (d). However, from plots in Figs. 3.2 (a) and (b), it is evident that in some systems the performance of D2C is on par with or better than DDPG. It may also be noted that the error variance in D2C method increases abruptly when the noise level is higher than a threshold and drives the system too far away from the nominal trajectory that the LQR controller cannot fix it. This could be considered as a drawback for D2C. However, it must be noted that the range of noise levels (up until 100 % of the maximum control signal) that we are considering here is far beyond what is typically encountered in practical scenarios. Hence, even in swimmer examples, the performance of D2C is tolerable to a reasonable extent of noise in the system. This is further demonstrated in Fig. 3.3. Here we compared the episodic cost during testing between the open-loop policy applied along and the closed-loop policy. In the first case, the nominal control solution from open-loop optimization is applied without the feedback control. So the perturbation drives the model off the nominal trajectory and increases the episodic cost as the noise level increases. It can be noted that when the D2C closed-loop policy is applied, the episodic cost is much smaller in the entire range of noise level we considered, which shows a great improvement compared to the open-loop policy

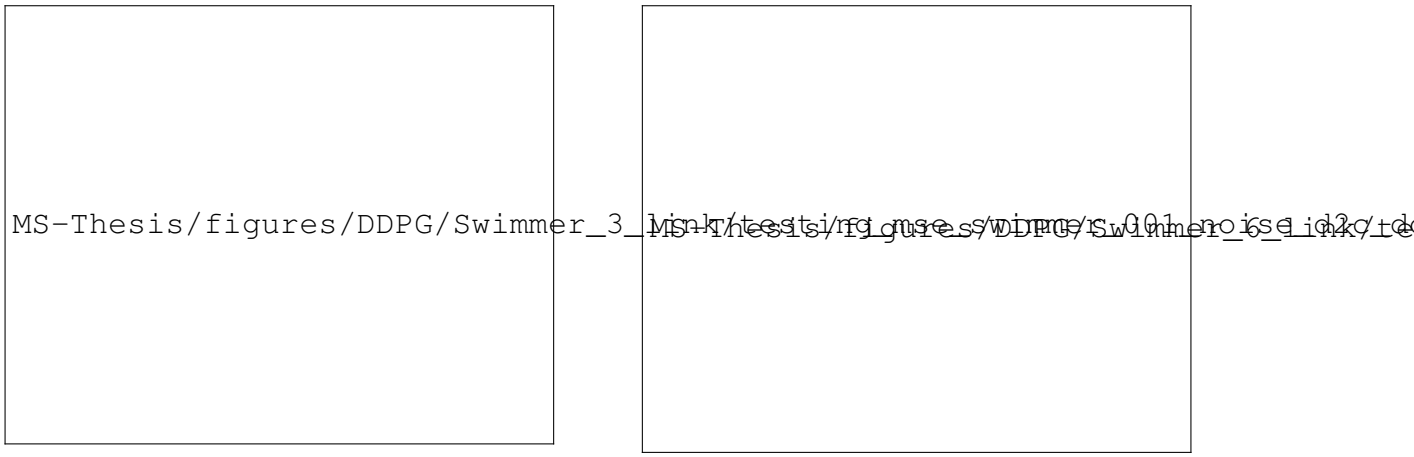
and proves the robustness of our method. Moreover, in the D2C column of Table 3.1, it is shown that the more than ninety percent of total training time is consumed in the open-loop training part. Therefore, in the implementation of D2C, most of the training time is cost in the MPC style open-loop optimization whereas the closed-loop part makes a great improvement in performance within a small time budget.

Ease of training: The ease of training is often an ignored topic in analyzing a reinforcement learning (RL) algorithm. By this, we mean the challenges associated with its implementation. As with many RL algorithms that involve neural networks, DDPG has no guarantees for policy convergence. As a result, the implementation often involves tuning a number of hyper-parameters and a careful reward shaping in a trial and error fashion, which is even more time-consuming given that their successful implementation already involves significant time and computational resources.

To elucidate the ease of training from an empirical perspective, the exploration noise that is required for training in DDPG mandates the system to operate with a shorter time-step than a threshold, beyond which the simulation fails due to an unbearable magnitude of control actions into the system. For this, we train both the swimmers in one such case (with $\Delta t = 0.01$ sec) till it fails and execute the intermediate policy. Fig. 3.4 shows the plot in the testing-stage with both methods. It is evident from the terminal state mean-squared error at zero noise level that the nominal trajectory of DDPG is incomplete and its policy failed to reach the goal. The effect is more pronounced in the higher-dimensional 6-link swimmer system (Fig. 3.4b), where the DDPG's policy can be deemed to be downright broken. Note, from Table 3.1, that the systems have been trained with DDPG for a time that is more than thrice with the 3-link swimmer and 4 times with the 6-link swimmer. On the other hand, under same conditions, the seamless training of D2C results in a working policy with even greater data-efficiency.

Table 3.1: Simulation parameters and training outcomes

System	Steps per episode	Time-step (in sec.)	Training time (in sec.) D2C Open-loop	D2C Closed-loop	DDPG
Inverted Pendulum	30	0.1	12.9	< 0.1	2261.15
Cart pole	30	0.1	15.0	1.33	6306.7
3-link Swimmer	1600 800	0.005 0.01	7861.0 4001.0	13.1 4.6	38833.64 13280.7*
6-link Swimmer	1500 900	0.006 0.01	9489.3 3585.4	26.5 16.4	88160 15797.2*
Fish	1200	0.005	6011.2	75.6	124367.6



(a) 3-link Swimmer

(b) 6-link Swimmer

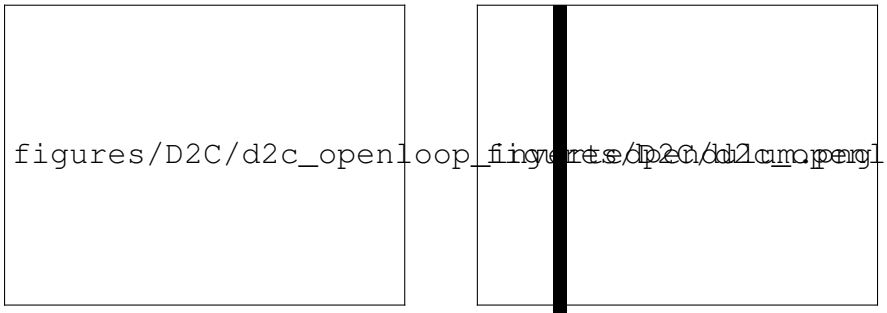
Figure 3.4: D2C vs DDPG at $\Delta t = 0.01s$

3.5.3 Reproducibility

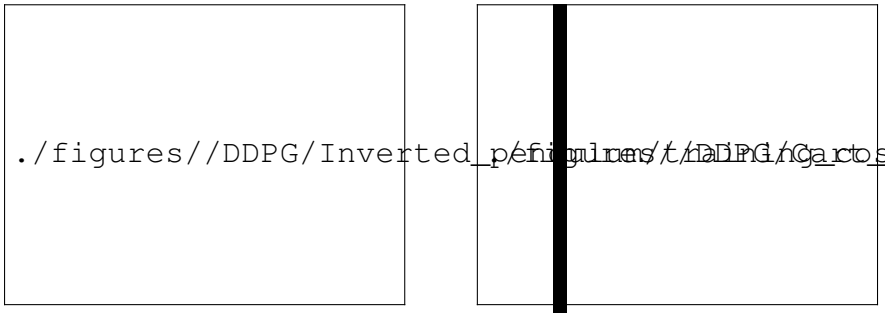
Reproducibility is still a major challenge that the field of reinforcement learning (RL) is yet to overcome. Despite a significant progress in recent times, the difficulty in reproducing the results of the existing work made the reports of improvements over state-of-the-art RL methods questionable.

In the recent years, people have realized this problem and some have conducted research on

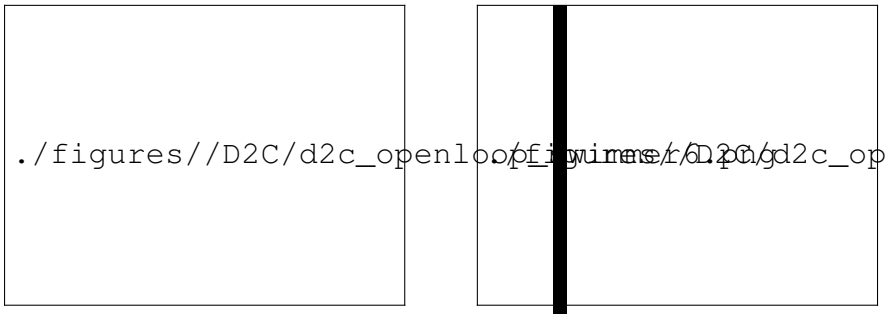
the challenges caused by reproducibility, proper experimental techniques and reporting procedures [54]. According to their work, the reproducibility of RL can be affected by extrinsic factors like hyperparameters or code base and intrinsic factors such as effects of random seeds or non-determinism in benchmark environments. In order to fairly compare different methods and confirm improvements, the effects of some factors mentioned above can be eliminated by optimizing the parameters, running more trials and reporting all the parameters and experimental setup. Despite this, the effects of random seeds still adds randomness in parameter tuning and makes the procedure more time consuming. In alignment with their proposed reporting procedures, we test the reproducibility of our method by conducting multiple training sessions with the same hyperparameters. Fig. 3.5 shows the mean and the standard deviation of the episodic cost data during training run 16 times each. For the inverted pendulum model and the cart-pole model, the results of all the training experiments are almost the same. Even for more complex model like the 6-link swimmer and the fish, the training is stable and the variance is small. Fig. 3.6 compares D2C with DDPG in the 3-link swimmer environment. Both algorithms run 4 repeated training experiments. It is evident that the variation of D2C is small and stable through out the training whereas DDPG has a large variance even after it seems to be converged (note that the variable on y-axis is not the absolute cost, but is scaled w.r.t. averaged cost during testing). After they both converge, the variation of D2C is still smaller than DDPG. It is evident that given the set of hyperparameters, D2C always results in the same policy (with a very small variance) unlike the results of the baseline RL algorithms also reported in [66]. This shows that D2C is more reliable and stable in training, thus has an advantage in reproducibility. This is an important feature in parameter tuning as the effects of changed parameters can be clearly demonstrated which makes selecting the best parameter set more efficient. Finally, in the spirit of reproducibility, our hyperparameters are reported for each example in the appendix and our codebase can be accessed from our repository by clicking here.



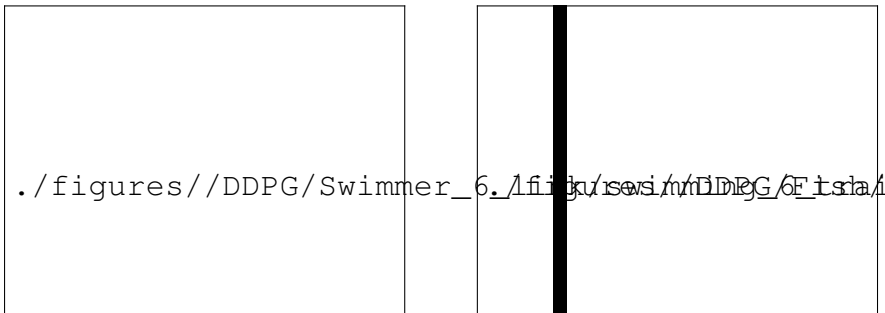
(a) Inverted Pendulum - D2C (b) Cart-Pole - D2C



(c) Inverted Pendulum - DDPG (d) Cart-Pole - DDPG

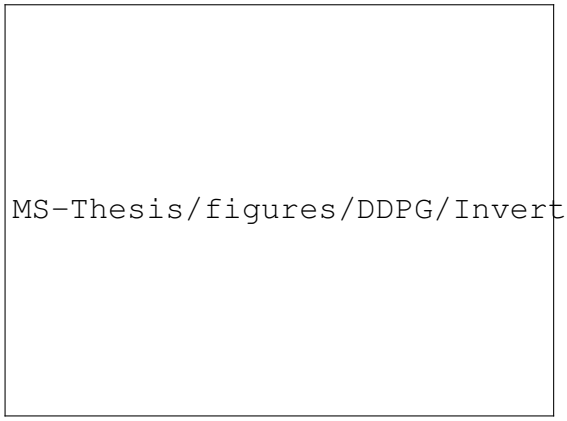


(e) 6-link Swimmer - D2C (f) Fish - D2C

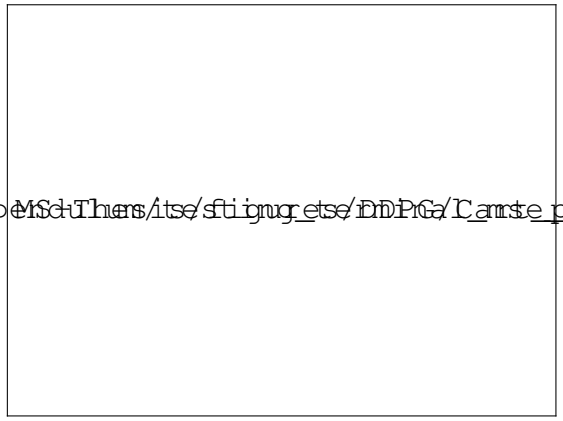


(g) 6-link Swimmer - DDPG (h) Fish - DDPG

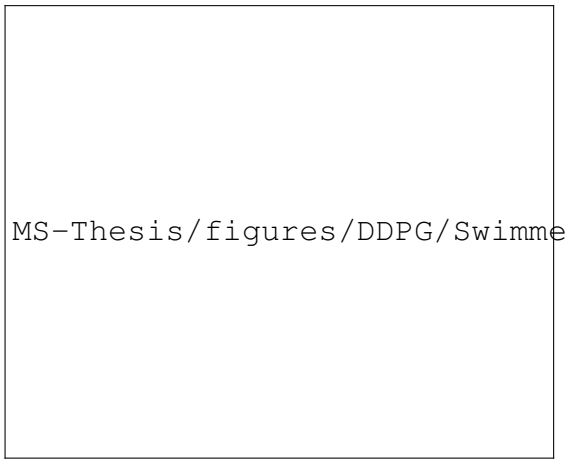
Figure 3.1: Episodic reward fraction vs time taken during training



(a) Inverted Pendulum



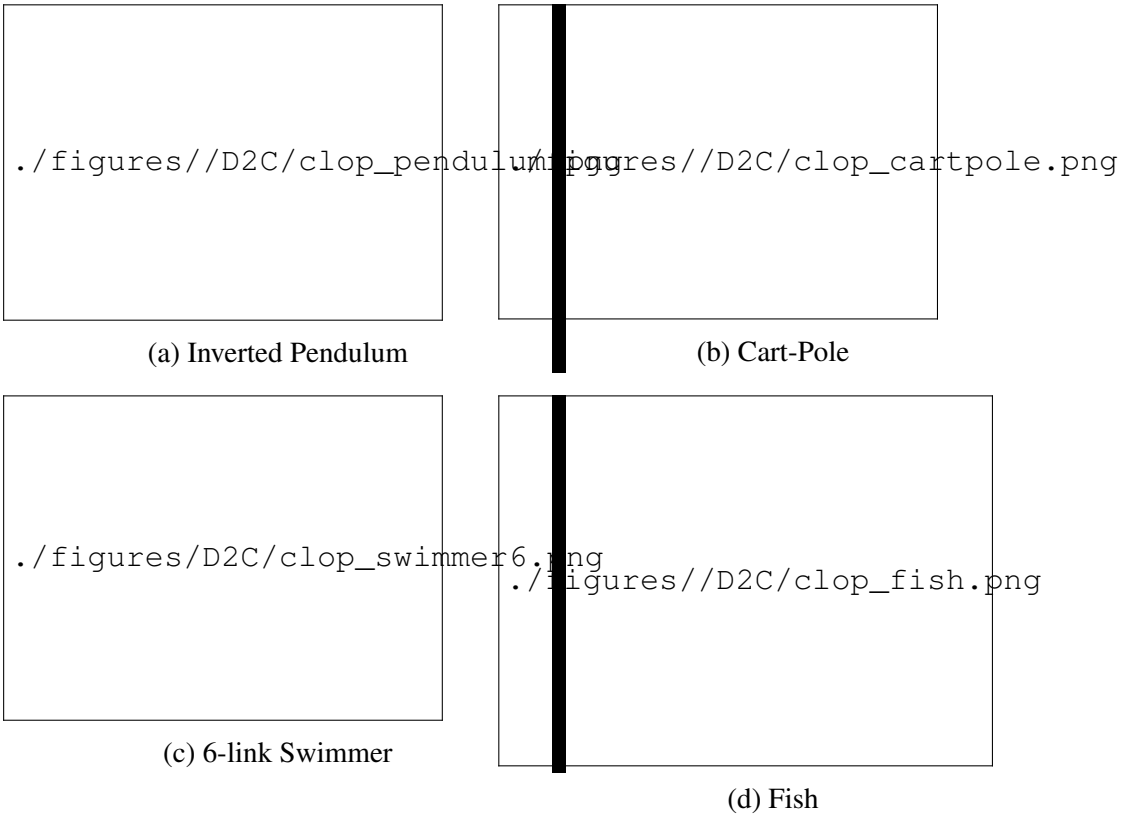
(b) Cart-Pole



(c) 6-link Swimmer

(d) Fish

Figure 3.2: Terminal MSE vs noise level during testing



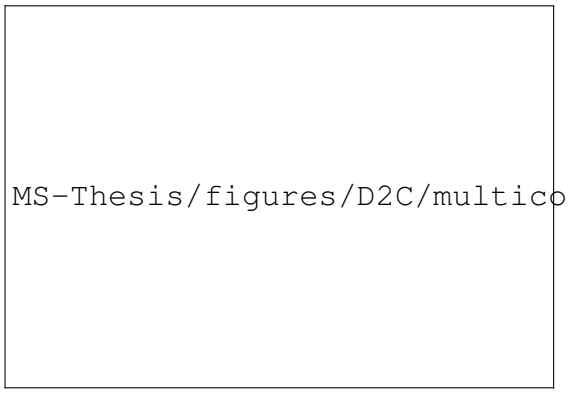
(a) Inverted Pendulum

(b) Cart-Pole

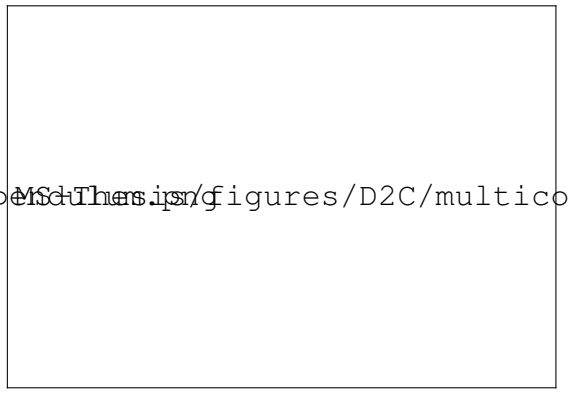
(c) 6-link Swimmer

(d) Fish

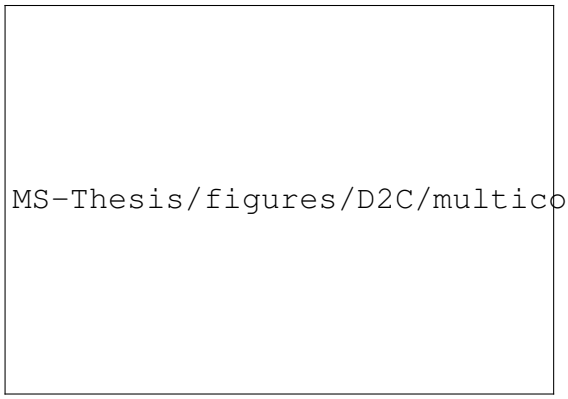
Figure 3.3: Averaged episodic reward fraction vs noise level during testing for D2C



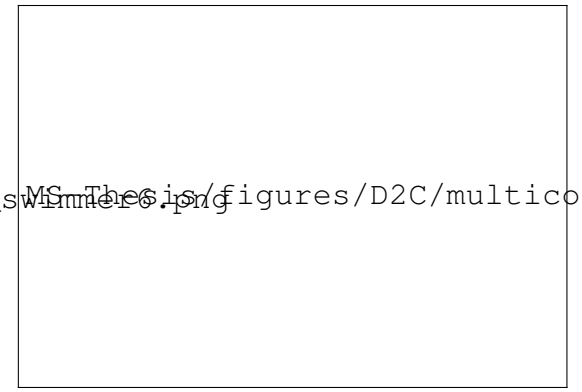
(a) Inverted Pendulum



(b) Cart-Pole

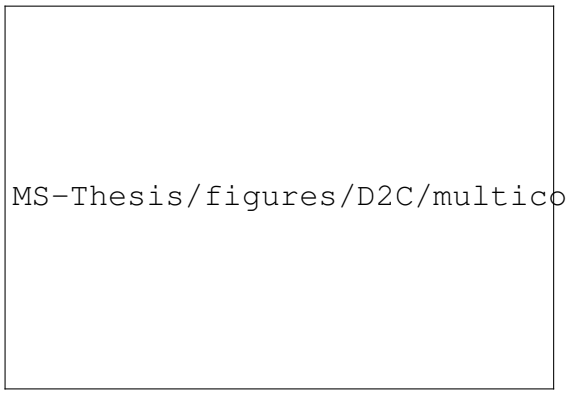


(c) 6-link Swimmer



(d) Fish

Figure 3.5: Averaged episodic reward fraction vs time taken during training for D2C



(a) D2C openloop training

(b) DDPG training

Figure 3.6: Averaged episodic reward fraction vs time taken during 4 training sessions for a 3-link swimmer for D2C vs DDPG

4. D2C 2.0 : MODEL-FREE DDP-BASED APPROACH FOR FASTER OPEN-LOOP TRAJECTORY COMPUTATION

4.1 Introduction

Previous chapter introduced a model-free algorithm called as 'D2C', whose action policy is a combination of an optimal nominal trajectory and a linear feedback law wrapped around it. It is evident from the outcome of the simulations and their analysis that the majority of the algorithmic run time is spent on computing for the open-loop sequence. In retrospect, one conspicuous observation that stood out was the slow convergence of the episodic cost. This is expected given the fact that the algorithm involved gradient descent, which, by its nature, is first-order convergent. Current chapter deals with an alternative method to obtain the open-loop trajectory using a formulation based on an existing model-based algorithm called 'Differential Dynamic Programming' (DDP) [26]. The chapter re-derives DDP first and describes a sample-efficient way to compute model parameters based on the formulation of DDP. It is then followed by example applications, comparison with the original D2C and a discussion on the merits and demerits of the approach.

4.2 Related Work

Differential Dynamic Programming (DDP) is a class of iterative algorithms for trajectory optimization. First proposed by D.Q. Mayne in 1966 [26], one salient aspect of DDP is that it exhibits quadratic convergence for a general non-linear optimal control problem and can be solved in a single step for linear systems. The derivation is similar to the Newton's method for optimization problems. In spite of its origins in 60s, it gained popularity only in the last decade due to the success of the modified algorithm called ILQR [3]. Though DDP (theoretically) guarantees a quadratic convergence, some of the terms in it involve computing second order derivatives of the system dynamical models. Since the dynamical models of most systems are multi-valued vector functions and their second order derivatives being third order tensors, DDP in its original form was not effective for practical implementation. ILQR [3][11] dropped these terms and introduced

regularization schemes [12] to handle the absence of these computationally expensive terms. This resulted in a faster but also stable convergence to the optimal solution. We take advantage of these properties in the formulation of open-loop trajectory design in the current chapter.

Finite difference method is a popular way to numerically estimate the Jacobians (and even the Hessians) of a function. Typically, a forward Euler scheme is chosen to independently determine each element (*i.e,* gradient) in a Jacobian matrix. Spall introduced the 'Stochastic Perturbation Method for Efficient Optimization' (SPSA) [67] method that only evaluates the function twice to calculate the Jacobian of the cost function. In this chapter, a similar formulation is derived to compute the Jacobians online through the least squares method in a central-difference scheme.

4.3 Preliminaries

Let $\mathbf{x_t} \in \mathcal{X} \in \mathbb{R}^{n_x}$ represent the state of a system and $\mathbf{u_t} \in \mathcal{X} \in \mathbb{R}^{n_u}$ be the control signal at time t respectively. Let us define the notion of a 'nominal' trajectory as follows - any trajectory (state and control sequence) that is feasible (for example, by satisfying the kinematic constraints for a non-holonomic system) for the given system is called as a nominal trajectory. Let us represent the nominal trajectory variables with *bars* over them $i.e,$ $\bar{\mathbf{x}}_\mathbf{t}$ and $\bar{\mathbf{u}}_\mathbf{t}$, both at time t. So, the nominal trajectory is given by, $\mathbb{T}_{nominal} = \{\bar{\mathbf{x}}_{\mathbf{0:N}}, \bar{\mathbf{u}}_{\mathbf{0:N-1}}\}$.

Let $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}^{n_x}$ denote the state transition model of the system. Let $\pi_t : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_u}$ be the policy to be applied on the system. Let $l_t : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}$ be the incremental cost function. Let $V_t^{\pi_t} : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$ represent the cost-to-go function under the policy π_t and $Q_t^{\pi_t} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \rightarrow \mathbb{R}$ be the corresponding action-value function, both at time t. Finally, let us distinctly denote the optimal variables with by having $*$ in their subscripts. In other words, $\mathbb{T}^*_{nominal} = \{\bar{\mathbf{x}}^*_{\mathbf{0:N}}, \bar{\mathbf{u}}^*_{\mathbf{0:N-1}}\}$ is the optimal nominal trajectory, π_t^* is the optimal policy, $V_t^*(.)$ is the optimal cost-to-go function and $Q_t^*(.)$ is the corresponding action-value function. The following equations present some of the important relations between the defined variables (assuming a deterministic system and hence, a deterministic policy):

$$V_t^{\pi_t}(\bar{\mathbf{x}}_\mathbf{t}) = \; Q_t^{\pi_t}(\bar{\mathbf{x}}_\mathbf{t}, \pi_\mathbf{t}(\bar{\mathbf{x}}_\mathbf{t})) \quad \text{(by definition)} \tag{4.1}$$

$$V_t^*(\bar{\mathbf{x}}_\mathbf{t}) = \min_{\bar{\mathbf{u}}_\mathbf{t}} Q_t^*(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t}) = \min_{\pi_\mathbf{t}} Q_t^{\pi_t}(\bar{\mathbf{x}}_\mathbf{t}, \pi_t(\bar{\mathbf{x}}_\mathbf{t})) \quad \text{(by definition)} \tag{4.2}$$

Figure 4.1 illustrates an arbitrary nominal trajectory and the converged optimal trajectory in DDP for a car-like robot's motion planned from $(x_0, y_0, \theta_0, \phi_0)^T = (0, 0, 0, 0)^T$ to $(x_N, y_N, \theta_N, \phi_N)^T = (5, 5, 0, 0)^T$, where N is the length of the horizon, (x, y) denoting the 2D position and (θ, ϕ) being the heading and the steering angles respectively.

Figure 4.1: Illustration of a sub-optimal nominal trajectory and an optimal trajectory (obtained after convergence) in DDP. The start and the goal locations are marked in blue heptagrams.

4.4 Methodology

This section presents a comprehensive derivation of the model-free DDP-based algorithm. It starts by revisiting the basic derivation of DDP/ILQG [26] [11] [12] [68]. The derivation is followed by a preliminary analysis to later propose a sample-efficient approach to solve for the Jacobians in context of DDP. The final equations resulted in DDP are straightforward on a theoretical front. However, that is practical for implementation only when supplemented by a right selection of regularization scheme, line search mechanism and initialization. These are discussed later in the

section which is then followed by a subsection on linear feedback design and how it fits within the framework of D2C.

4.4.1 Revisiting the Derivation of DDP

Let us initialize with an arbitrary policy π_t. Evaluating this policy on the system (also referred to as 'forward pass' later) results in $\mathbb{T}_{nominal} = \{\bar{\mathbf{x}}_{\mathbf{0:N}}, \bar{\mathbf{u}}_{\mathbf{0:N-1}}\}$, where $\bar{\mathbf{x}}_{\mathbf{0}}$ is the given initial state of the system and $\bar{\mathbf{u}}_{\mathbf{t}} = \pi_t(\bar{\mathbf{x}}_{\mathbf{t}})$. Let $\mathbf{x_t} = \bar{\mathbf{x}}_{\mathbf{t}} + \delta\mathbf{x_t}$ be a small deviation from the nominal state $\bar{\mathbf{x}}_{\mathbf{t}}$ and the corresponding control be $\mathbf{u_t} = \bar{\mathbf{u}}_{\mathbf{t}} + \delta\mathbf{u_t}$. For simplicity of notation, $Q_t^{\pi_t}()$. is being replaced by Q_t in the following derivation. By Taylor's expansion around $(\bar{\mathbf{x}}_{\mathbf{t}}, \bar{\mathbf{u}}_{\mathbf{t}})$ up to two orders, we obtain the following:

$$
\begin{aligned}
Q_t(\mathbf{x_t}, \mathbf{u_t}) &= Q_t(\bar{\mathbf{x}}_{\mathbf{t}} + \delta\mathbf{x_t}, \bar{\mathbf{u}}_{\mathbf{t}} + \delta\mathbf{u_t}) \\
&= Q_t(\bar{\mathbf{x}}_{\mathbf{t}}, \bar{\mathbf{u}}_{\mathbf{t}}) + \begin{bmatrix} Q_{\mathbf{x_t}} & Q_{\mathbf{u_t}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t} \\ \delta\mathbf{u_t} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} (\delta\mathbf{x_t})^\intercal & \delta\mathbf{u_t}^\intercal \end{bmatrix} \begin{bmatrix} Q_{\mathbf{x_t x_t}} & Q_{\mathbf{x_t u_t}} \\ Q_{\mathbf{u_t x_t}} & Q_{\mathbf{u_t u_t}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t} \\ \delta\mathbf{u_t} \end{bmatrix}
\end{aligned}
\tag{4.3}
$$

where, $Q_{\mathbf{x_t}} = \frac{\partial Q_t}{\partial \mathbf{x_t}}\Big|_{(\bar{\mathbf{x}}_{\mathbf{t}}, \bar{\mathbf{u}}_{\mathbf{t}})}$, $Q_{\mathbf{u_t}} = \frac{\partial Q_t}{\partial \mathbf{x_t}}\Big|_{(\bar{\mathbf{x}}_{\mathbf{t}}, \bar{\mathbf{u}}_{\mathbf{t}})}$, $Q_{\mathbf{x_t x_t}} = \frac{\partial^2 Q_t}{\partial \mathbf{x_t} \partial \mathbf{x_t}}\Big|_{(\bar{\mathbf{x}}_{\mathbf{t}}, \bar{\mathbf{u}}_{\mathbf{t}})}$, $Q_{\mathbf{u_t u_t}} = \frac{\partial^2 Q_t}{\partial \mathbf{u_t} \partial \mathbf{u_t}}\Big|_{(\bar{\mathbf{x}}_{\mathbf{t}}, \bar{\mathbf{u}}_{\mathbf{t}})}$, $Q_{\mathbf{u_t x_t}} = \frac{\partial^2 Q_t}{\partial \mathbf{u_t} \partial \mathbf{x_t}}\Big|_{(\bar{\mathbf{x}}_{\mathbf{t}}, \bar{\mathbf{u}}_{\mathbf{t}})}$ and $Q_{\mathbf{x_t u_t}} = \frac{\partial^2 Q_t}{\partial \mathbf{x_t} \partial \mathbf{u_t}}\Big|_{(\bar{\mathbf{x}}_{\mathbf{t}}, \bar{\mathbf{u}}_{\mathbf{t}})}$. Please note that the notation for $Q_t(.)$ has not been changed in spite of the quadratic approximation, for simplicity of notations.

Now, given $\mathbb{T}_{nominal}$, we look for an incremental control law, $\delta\mathbf{u_t}$, that minimizes $Q_t(\mathbf{x_t}, \mathbf{u_t})$. In other words, solve for $\delta\mathbf{u_t}$ such that $\min_{\delta\mathbf{u_t}} Q_t(\bar{\mathbf{x}}_{\mathbf{t}} + \delta\mathbf{x_t}, \bar{\mathbf{u}}_{\mathbf{t}} + \delta\mathbf{u_t})$. Ideally, if the structure of the Q-function is known, we could try to solve for a $\delta\mathbf{u_t}$ that will directly result in $\mathbb{T}^*_{nominal}$ from $\mathbb{T}_{nominal}$, just in one step. However, for a general Q_t, this is not directly solvable. But, since we consider the local quadratic approximation as in equation (4.6), we can solve for $\delta\mathbf{u_t}$ that will improve the nominal trajectory, in each iteration, reaching towards the optimal nominal trajectory. This is similar to Newton's method for root-finding.

$$
\min_{\delta\mathbf{u_t}} Q_t(\bar{\mathbf{x}}_{\mathbf{t}} + \delta\mathbf{x_t}, \bar{\mathbf{u}}_{\mathbf{t}} + \delta\mathbf{u_t}) \implies \frac{\partial Q_t}{\partial \delta\mathbf{u_t}} = 0 \tag{4.4}
$$

Solving equation (4.4) results in an incremental linear control law $\delta\mathbf{u_t} = K_t\delta\mathbf{x_t} + k_t$, where, $K_t = -Q^{-1}_{\mathbf{u_t u_t}}Q_{\mathbf{u_t}}$ and $k_t = -Q^{-1}_{\mathbf{u_t u_t}}Q_{\mathbf{u_t x_t}}$.

$$\begin{aligned}\delta\mathbf{u_t} = K_t\delta\mathbf{x_t} + k_t \implies \mathbf{u_t} &= \bar{\mathbf{u}}_\mathbf{t} + K_t(\mathbf{x_t} - \bar{\mathbf{x}}_\mathbf{t}) + k_t \\ \implies \pi_t(\mathbf{x_t}) &= \pi_t(\bar{\mathbf{x}}_\mathbf{t}) + K_t(\mathbf{x_t} - \bar{\mathbf{x}}_\mathbf{t}) + k_t\end{aligned} \tag{4.5}$$

Substituting the new policy from equation (4.5) in the Q function results in the following:

$$\begin{aligned}Q_t(\mathbf{x_t}, \pi_t(\mathbf{x_t})) = \; & Q_t(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t}) + \begin{bmatrix} Q_{\mathbf{x_t}} & Q_{\mathbf{u_t}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t} \\ K_t\delta\mathbf{x_t} + k_t \end{bmatrix} + \\ & \frac{1}{2}\begin{bmatrix} \delta\mathbf{x_t}^T & (K_t\delta\mathbf{x_t} + k_t)^T \end{bmatrix} \begin{bmatrix} Q_{\mathbf{x_t x_t}} & Q_{\mathbf{x_t u_t}} \\ Q_{\mathbf{u_t x_t}} & Q_{\mathbf{u_t u_t}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t} \\ K_t\delta\mathbf{x_t} + k_t \end{bmatrix}. \\ = \; & (Q_t(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t}) + k_t^T Q_{\mathbf{u_t u_t}} k_t) + \\ & (Q_{\mathbf{x_t}} + Q_{\mathbf{u_t}}K_t + k^T Q_{\mathbf{x_t u_t}} + k_t^T Q_{\mathbf{u_t u_t}} K_t)\delta\mathbf{x_t} + \\ & \frac{1}{2}(\delta\mathbf{x_t})^T(Q_{\mathbf{x_t x_t}} + Q_{\mathbf{x_t u_t}}K_t + K_t^T Q_{\mathbf{u_t x_t}} + K_t^T Q_{\mathbf{u_t u_t}} K_t)\delta\mathbf{x_t}.\end{aligned} \tag{4.6}$$

The above equation inherently assumed that $Q_{\mathbf{u_t x_t}} = Q_{\mathbf{x_t u_t}}$ (since $Q_t(.)$ is assumed to be continuous) and a well-known property that if A and B are arbitrary matrices such that their product, AB, is a scalar, then, $AB = (AB)^T = B^T A^T$.

From equation (4.1), we know that $Q_t(\mathbf{x_t}, \mathbf{u_t}) = Q_t(\mathbf{x_t}, \pi_t(\mathbf{x_t})) = V_t(\mathbf{x_t})$. Also, by definition, $V_t(\mathbf{x_t}) = l_t(\mathbf{x_t}, \mathbf{u_t}) + V_{t+1}(\mathbf{f}(\mathbf{x_t}))$. By Taylor's expansion of $V_t(.)$ around $(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})$, we obtain the

following:

$$
\begin{aligned}
Q_t(\mathbf{x_t}, \mathbf{u_t}) &= l_t(\mathbf{x_t}, \mathbf{u_t}) + V_{t+1}(\mathbf{f}(\mathbf{x_t})) \\
&= \bar{l}_t + V_{t+1}(f(\bar{\mathbf{x}}_\mathbf{t})) + (l_{\mathbf{x_t}} + f_{\mathbf{x_t}}^T V'_{\mathbf{x_{t+1}}})\delta\mathbf{x_t} + (l_{\mathbf{u_t}} + f_{\mathbf{u_t}}^T V'_{\mathbf{x_{t+1}}})\delta\mathbf{u_t} + \\
&\quad \frac{1}{2}\delta\mathbf{x_t}^T(l_{\mathbf{x_t x_t}} + f_{\mathbf{x_t}}^T V'_{\mathbf{x_{t+1}x_{t+1}}} f_{\mathbf{x_t}} + V'_{\mathbf{x_{t+1}}} \otimes f_{\mathbf{x_t x_t}})\delta\mathbf{x_t} + \\
&\quad \delta\mathbf{u_t}^T(l_{\mathbf{u_t x_t}} + f_{\mathbf{u_t}}^T V'_{\mathbf{x_{t+1}x_{t+1}}} f'_{\mathbf{x_t}} + V'_{\mathbf{x_{t+1}}} \otimes f_{\mathbf{u_t x_t}})\delta\mathbf{x_t} + \\
&\quad \frac{1}{2}\delta\mathbf{u_t}^T(l_{\mathbf{u_t u_t}} + f_{\mathbf{u_t}}^T V'_{\mathbf{x_{t+1}x_{t+1}}} f_{\mathbf{u_t}} + V'_{\mathbf{x_{t+1}}} \otimes f_{\mathbf{u_t u_t}})\delta\mathbf{u_t},
\end{aligned} \tag{4.7}
$$

where, $V'_{\mathbf{x_{t+1}}} = \frac{\partial V_{t+1}}{\partial \mathbf{x_{t+1}}}\Big|_{(\bar{\mathbf{x}}_{\mathbf{t+1}}, \bar{\mathbf{u}}_{\mathbf{t+1}})}$, $V'_{\mathbf{x_{t+1}x_{t+1}}} = \frac{\partial^2 V_{t+1}}{\partial^2 \mathbf{x_{t+1}}}\Big|_{(\bar{\mathbf{x}}_{\mathbf{t+1}}, \bar{\mathbf{u}}_{\mathbf{t+1}})}$, $l_{\mathbf{x_t}} = \frac{\partial l_t}{\partial \mathbf{x_t}}\Big|_{(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})}$, $l_{\mathbf{u_t}} = \frac{\partial l_t}{\partial \mathbf{u_t}}\Big|_{(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})}$, $l_{\mathbf{u_t u_t}} = \frac{\partial^2 l_t}{\partial^2 \mathbf{u_t}}\Big|_{(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})}$, $f_{\mathbf{x_t}} = \frac{\partial f}{\partial \mathbf{x_t}}\Big|_{(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})}$, $f_{\mathbf{u_t}} = \frac{\partial f}{\partial \mathbf{u_t}}\Big|_{(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})}$, $f_{\mathbf{x_t x_t}} = \frac{\partial^2 f}{\partial^2 \mathbf{x_t}}\Big|_{(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})}$, $f_{\mathbf{u_t u_t}} = \frac{\partial^2 f}{\partial^2 \mathbf{u_t}}\Big|_{(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})}$ and $f_{\mathbf{u_t x_t}} = \frac{\partial^2 f}{\partial \mathbf{x_t} \partial \mathbf{u_t}}\Big|_{(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})}$. Note the 'prime' here over 'V' indicates V_{t+1} $i.e$, at a time-step higher. This is a slight abuse of notation to avoid populating its subscript with a number of variables. From equation (4.7) and (4.6), we have :

$$
\begin{aligned}
Q_{\mathbf{x_t}} &= l_{\mathbf{x_t}} + f_{\mathbf{x_t}}^T V'_{\mathbf{x_{t+1}}} \\
Q_{\mathbf{u_t}} &= l_{\mathbf{u_t}} + f_{\mathbf{u_t}}^T V'_{\mathbf{x_{t+1}}} \\
Q_{\mathbf{x_t x_t}} &= l_{\mathbf{x_t x_t}} + f_{\mathbf{x_t}}^T V'_{\mathbf{x_{t+1}x_{t+1}}} f_{\mathbf{x_t}} + V'_{\mathbf{x_{t+1}}} \otimes f_{\mathbf{x_t x_t}} \\
Q_{\mathbf{u_t x_t}} &= l_{\mathbf{u_t x_t}} + f_{\mathbf{u_t}}^T V'_{\mathbf{x_{t+1}x_{t+1}}} f_{\mathbf{x_t}} + V'_{\mathbf{x_{t+1}}} \otimes f_{\mathbf{u_t x_t}} \\
Q_{\mathbf{u_t u_t}} &= l_{\mathbf{u_t u_t}} + f_{\mathbf{u_t}}^T V'_{\mathbf{x_{t+1}x_{t+1}}} f_{\mathbf{u_t}} + V'_{\mathbf{x_{t+1}}} \otimes f_{\mathbf{u_t u_t}}
\end{aligned} \tag{4.8}
$$

$\otimes$ in the above equation denotes tensor product, as the second order derivatives of dynamics following it are third order tensors. Now, the difference between DDP and ILQR/ILQG is in whether or not these terms are included. Though they make the equations complete, it is later found that computing them is expensive and given the quadratic approximation, ignoring them, in fact, makes the convergence towards the solution faster.

4.4.1.1 Regularization

The computation of K_t and k_t in the equation (4.5) require an inversion of the $Q_{\mathbf{u_t u_t}}$ matrix. Since the derivation is based upon the quadratic approximation of the Q function, $Q_{\mathbf{u_t u_t}}$ is not guaranteed to be positive semi-definite. $Q_{\mathbf{u_t u_t}}$ being negative definite is worse than slower descent in the cost, because that implies the problem is ill- posed and the algorithm cannot proceed from this point. In order to deal with it, [11] introduced two following different fixes:

- **The Levenberg-Marquardt trick:** replacing $Q_{\mathbf{u_t u_t}}$ with $Q_{\mathbf{u_t u_t}} + (\epsilon - \lambda_{min}(Q_{\mathbf{u_t u_t}})I_{n_u \times n_u}$, where $\lambda_{min}(Q_{\mathbf{u_t u_t}})$ is the minimum eigenvalue of $Q_{\mathbf{u_t u_t}}$ and $\epsilon > 0$.

- performing the eigenvalue decomposition such that V is the eigenvector matrix and D is the diagonal matrix filled with corresponding eigenvalues. Then, $Q_{\mathbf{u_t u_t}} = VDV^T$. By replacing the diagonal elements in D smaller than ϵ with $\epsilon > 0$, the inversion of $Q_{\mathbf{u_t u_t}}$ is simply given by $Q^{-1}_{\mathbf{u_t u_t}} = VD^{-1}V^T$.

[12] proposed an improved regularization scheme over the Levenberg-Marquardt scheme. It penalizes deviations from the states rather than the controls. Even as μ increases, this scheme ensures that the new trajectory is well within the vicinity of the previous trajectory. Hence, this scheme is followed in the current implementation. The updated $Q_{\mathbf{u_t x_t}}$ and $Q_{\mathbf{u_t u_t}}$ are shown in the algorithm 5. It is to be noted that adding extra terms in the Q function equations such as above, alters the value of the cost-to-go, V and its derivatives w.r.t. $\mathbf{x_t}$. It is reflected in the corresponding equations in algorithm 5. μ is initialized with a small positive constant and increased by a factor when $Q_{\mathbf{u_t u_t}}$ is negative definite, until $Q_{\mathbf{u_t u_t}}$ is positive definite. The implementation follows the quadratic modification schedule proposed in [12].

4.4.1.2 Line Search

The backward pass step results in an open-loop parameter k_t and a feedback parameter K_t for the forward pass in the next iteration. In a general nonlinear system, the new policy obtained could cause the new trajectory to substantially deviate from the previous trajectory and might result

in the divergence of the cost. This can be precluded by introducing a scaling parameter α that is multiplied with k_t. α can be varied by verifying if the new cost is within a predetermined bound and backtracking accordingly. Current implementation follows the line search scheme of [12], where the new trajectory is deemed to be acceptable if it satisfies the following criteria: An estimate of the expected total-cost reduction in the line-search is given by

$$\Delta cost(\alpha) = \alpha \sum_{t=1}^{N-1} k_t^T Q_{\mathbf{u_t}} + \frac{\alpha^2}{2} \sum_{t=1}^{N-1} k_t^T Q_{\mathbf{u_t u_t}} k_t$$

$$\frac{(cost(\mathbb{T}^{new}) - cost(\mathbb{T}^{current}))}{\Delta cost(\alpha)} < c,$$

where c is a constant parameter that can be tuned for a given system and α (although this is rare in practice). If this is not satisfied, we backtrack $i.e$, reduce α and re-iterate through the same step as done above.

Now, the advantage with the above result w.r.t. making it model-free is that the equations involved in it are explicit in system dynamics and their gradients. Hence, in order to make it a model-free algorithm, it is sufficient if we could explicitly obtain the estimates of Jacobians and Hessians. A sample-efficient way of doing it is described in the next subsection.

4.4.2 Estimation of Jacobians and Hessians in a Model-free Setting

We make a preliminary analysis on the number of evaluations required in the above algorithm. If the horizon is 'h' and the number of iterations required to converge are 'n', the algorithm requires us to estimate each of the $f_{\mathbf{x_t}}, f_{\mathbf{u_t}}, f_{\mathbf{x_t x_t}}, f_{\mathbf{u_t x_t}}$ and $f_{\mathbf{u_t u_t}}$ 'nh' times.

$$f(\bar{\mathbf{x}}_\mathbf{t}+\delta\mathbf{x_t},\bar{\mathbf{u}}_\mathbf{t}+\delta\mathbf{u_t}) = f(\bar{\mathbf{x}}_\mathbf{t},\bar{\mathbf{u}}_\mathbf{t}) + \begin{bmatrix} f_{\mathbf{x_t}} & f_{\mathbf{u_t}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t} \\ \delta\mathbf{u_t} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta\mathbf{x_t}^T & \delta\mathbf{u_t}^T \end{bmatrix} \begin{bmatrix} f_{\mathbf{x_t x_t}} & f_{\mathbf{x_t u_t}} \\ f_{\mathbf{u_t x_t}} & f_{\mathbf{u_t u_t}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t} \\ \delta\mathbf{u_t} \end{bmatrix} + O(\|\delta\mathbf{x_t}\|^3 + \|\delta\mathbf{u_t}\|^3)$$

$$f(\bar{\mathbf{x}}_\mathbf{t}-\delta\mathbf{x_t},\bar{\mathbf{u}}_\mathbf{t}-\delta\mathbf{u_t}) = f(\bar{\mathbf{x}}_\mathbf{t},\bar{\mathbf{u}}_\mathbf{t}) - \begin{bmatrix} f_{\mathbf{x_t}} & f_{\mathbf{u_t}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t} \\ \delta\mathbf{u_t} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta\mathbf{x_t}^T & \delta\mathbf{u_t}^T \end{bmatrix} \begin{bmatrix} f_{\mathbf{x_t x_t}} & f_{\mathbf{x_t u_t}} \\ f_{\mathbf{u_t x_t}} & f_{\mathbf{u_t u_t}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t} \\ \delta\mathbf{u_t} \end{bmatrix} + O(\|\delta\mathbf{x_t}\|^3 + \|\delta\mathbf{u_t}\|^3)$$

Subtracting the above equations on both sides results in the following :

$$f(\bar{\mathbf{x}}_\mathbf{t}+\delta\mathbf{x_t},\bar{\mathbf{u}}_\mathbf{t}+\delta\mathbf{u_t}) - f(\bar{\mathbf{x}}_\mathbf{t}-\delta\mathbf{x_t},\bar{\mathbf{u}}_\mathbf{t}-\delta\mathbf{u_t}) = 2 \begin{bmatrix} f_{\mathbf{x_t}} & f_{\mathbf{u_t}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t} \\ \delta\mathbf{u_t} \end{bmatrix} + O(\|\delta\mathbf{x_t}\|^3 + \|\delta\mathbf{u_t}\|^3) \quad (4.9)$$

Multiplying by $\begin{bmatrix} \delta\mathbf{x_t}^T & \delta\mathbf{u_t}^T \end{bmatrix}$ on both sides to the above equation:

$$\begin{aligned} f(\bar{\mathbf{x}}_\mathbf{t}+\delta\mathbf{x_t},\bar{\mathbf{u}}_\mathbf{t}+\delta\mathbf{u_t}) - f(\bar{\mathbf{x}}_\mathbf{t}-\delta\mathbf{x_t},\bar{\mathbf{u}}_\mathbf{t}-\delta\mathbf{u_t}) \begin{bmatrix} \delta\mathbf{x_t}^T & \delta\mathbf{u_t}^T \end{bmatrix} &= 2 \begin{bmatrix} f_{\mathbf{x_t}} & f_{\mathbf{u_t}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t} \\ \delta\mathbf{u_t} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t}^T & \delta\mathbf{u_t}^T \end{bmatrix} + \\ & O(\|\delta\mathbf{x_t}\|^4 + \|\delta\mathbf{u_t}\|^4) \\ &= 2 \begin{bmatrix} f_{\mathbf{x_t}} & f_{\mathbf{u_t}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t}\delta\mathbf{x_t}^T & \delta\mathbf{x_t}\delta\mathbf{u_t}^T \\ \delta\mathbf{u_t}\delta\mathbf{x_t}^T & \delta\mathbf{u_t}\delta\mathbf{u_t}^T \end{bmatrix} + \\ & O(\|\delta\mathbf{x_t}\|^4 + \|\delta\mathbf{u_t}\|^4) \end{aligned}$$

Assuming that $\begin{bmatrix} \delta\mathbf{x_t}\delta\mathbf{x_t}^T & \delta\mathbf{x_t}\delta\mathbf{u_t}^T \\ \delta\mathbf{u_t}\delta\mathbf{x_t}^T & \delta\mathbf{u_t}\delta\mathbf{u_t}^T \end{bmatrix}$ is invertible (which will later be proved to be true by making some assumptions!), let us perform inversions on either sides of the above equation as follows:

$$f(\bar{\mathbf{x}}_\mathbf{t} + \delta\mathbf{x_t}, \bar{\mathbf{u}}_\mathbf{t} + \delta\mathbf{u_t}) - f(\bar{\mathbf{x}}_\mathbf{t} - \delta\mathbf{x_t}, \bar{\mathbf{u}}_\mathbf{t} - \delta\mathbf{u_t}) \begin{bmatrix} \delta\mathbf{x_t}^T & \delta\mathbf{u_t}^T \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t}\delta\mathbf{x_t}^T & \delta\mathbf{x_t}\delta\mathbf{u_t}^T \\ \delta\mathbf{u_t}\delta\mathbf{x_t}^T & \delta\mathbf{u_t}\delta\mathbf{u_t}^T \end{bmatrix}^{-1} =$$

$$2 \begin{bmatrix} f_{\mathbf{x_t}} & f_{\mathbf{u_t}} \end{bmatrix} + O(\|\delta\mathbf{x_t}\|^2 + \|\delta\mathbf{u_t}\|^2)$$

$\Longrightarrow$

$$\begin{bmatrix} f_{\mathbf{x_t}} & f_{\mathbf{u_t}} \end{bmatrix} =$$

$$\frac{1}{2} \left[f(\bar{\mathbf{x}}_\mathbf{t} + \delta\mathbf{x_t}, \bar{\mathbf{u}}_\mathbf{t} + \delta\mathbf{u_t}) - f(\bar{\mathbf{x}}_\mathbf{t} - \delta\mathbf{x_t}, \bar{\mathbf{u}}_\mathbf{t} - \delta\mathbf{u_t}) \right] \begin{bmatrix} \delta\mathbf{x_t}^T & \delta\mathbf{u_t}^T \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t}\delta\mathbf{x_t}^T & \delta\mathbf{x_t}\delta\mathbf{u_t}^T \\ \delta\mathbf{u_t}\delta\mathbf{x_t}^T & \delta\mathbf{u_t}\delta\mathbf{u_t}^T \end{bmatrix}^{-1} +$$

$$O(\|\delta\mathbf{x_t}\|^2 + \|\delta\mathbf{u_t}\|^2) \tag{4.10}$$

Equation (4.10) is a way to solve for Jacobians, $f_{\mathbf{x_t}}$ and $f_{\mathbf{u_t}}$, simultaneously. It is noted that the above formulation requires only 2 evaluations of $f(.)$, given the nominal state and control $(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})$. The remaining terms (also, the error in the evaluation) are of the order that is quadratic in $\delta\mathbf{x_t}$ and $\delta\mathbf{u_t}$. The following extends equation (4.10) to be used in practical implementations (sampling).

We are free to choose the distribution of $\delta\mathbf{x_t}$ and $\delta\mathbf{x_t}$. As mentioned before, let us assume both are i.i.d. Gaussian distributed random variables with zero mean and a standard deviation of σ. This ensures that $\begin{bmatrix} \delta\mathbf{x_t}\delta\mathbf{x_t}^T & \delta\mathbf{x_t}\delta\mathbf{u_t}^T \\ \delta\mathbf{u_t}\delta\mathbf{x_t}^T & \delta\mathbf{u_t}\delta\mathbf{u_t}^T \end{bmatrix}$ is invertible. More on the advantage of using this distribution will be elaborated in the next paragraph. Let 'n_s' be the number of samples for each of the random variables, $\delta\mathbf{x_t}$ and $\delta\mathbf{u_t}$, as $\delta\mathbf{X_t} = \begin{bmatrix} \delta\mathbf{x_t^1} & \delta\mathbf{x_t^2} & \dots & \delta\mathbf{x_t^{n_s}} \end{bmatrix}$ and $\delta\mathbf{U_t} = \begin{bmatrix} \delta\mathbf{u_t^1} & \delta\mathbf{u_t^2} & \dots & \delta\mathbf{u_t^{n_s}} \end{bmatrix}$, respectively. Then $\begin{bmatrix} f_{\mathbf{x_t}} & f_{\mathbf{u_t}} \end{bmatrix}$ is given by the following :

$$\begin{bmatrix} f_{\mathbf{x_t}} & f_{\mathbf{u_t}} \end{bmatrix} =$$
$$\begin{bmatrix} (f(\bar{\mathbf{x}}_\mathbf{t} + \delta\mathbf{x_t^1}, \bar{\mathbf{u}}_\mathbf{t} + \delta\mathbf{u_t^1}) - f(\bar{\mathbf{x}}_\mathbf{t} - \delta\mathbf{x_t^1}, \bar{\mathbf{u}}_\mathbf{t} - \delta\mathbf{u_t^1}))^T \\ (f(\bar{\mathbf{x}}_\mathbf{t} + \delta\mathbf{x_t^2}, \bar{\mathbf{u}}_\mathbf{t} + \delta\mathbf{u_t^2}) - f(\bar{\mathbf{x}}_\mathbf{t} - \delta\mathbf{x_t^2}, \bar{\mathbf{u}}_\mathbf{t} - \delta\mathbf{u_t^2}))^T \\ \vdots \\ (f(\bar{\mathbf{x}}_\mathbf{t} + \delta\mathbf{x_t^{n_s}}, \bar{\mathbf{u}}_\mathbf{t} + \delta\mathbf{u_t^{n_s}}) - f(\bar{\mathbf{x}}_\mathbf{t} - \delta\mathbf{x_t^{n_s}}, \bar{\mathbf{u}}_\mathbf{t} - \delta\mathbf{u_t^{n_s}}))^T \end{bmatrix}^T \begin{bmatrix} \delta\mathbf{X_t}^T & \delta\mathbf{U_t}^T \end{bmatrix} \begin{bmatrix} \delta\mathbf{X_t}\delta\mathbf{X_t}^T & \delta\mathbf{X_t}\delta\mathbf{U_t}^T \\ \delta\mathbf{U_t}\delta\mathbf{X_t}^T & \delta\mathbf{U_t}\delta\mathbf{U_t}^T \end{bmatrix}^{-1}$$

(4.11)

Let us consider the terms in the matrix $\begin{bmatrix} \delta\mathbf{X_t}\delta\mathbf{X_t}^T & \delta\mathbf{X_t}\delta\mathbf{U_t}^T \\ \delta\mathbf{U_t}\delta\mathbf{X_t}^T & \delta\mathbf{U_t}\delta\mathbf{U_t}^T \end{bmatrix}$:

$$\delta\mathbf{X_t}\delta\mathbf{X_t}^T = \begin{bmatrix} \delta\mathbf{x_t^1} & \delta\mathbf{x_t^2} & \dots & \delta\mathbf{x_t^{n_s}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x_t^1}^T \\ \delta\mathbf{x_t^2}^T \\ \vdots \\ \delta\mathbf{x_t^{n_s}}^T \end{bmatrix} = \sum_{i=1}^{n_s} \delta\mathbf{x_t}^i \delta\mathbf{x_t}^{iT}$$

Similarly, $\delta\mathbf{U_t}\delta\mathbf{U_t}^T = \sum_{i=1}^{n_s} \delta\mathbf{u_t}^i \delta\mathbf{u_t}^{iT}$, $\delta\mathbf{U_t}\delta\mathbf{X_t}^T = \sum_{i=1}^{n_s} \delta\mathbf{u_t}^i \delta\mathbf{x_t}^{iT}$ and $\delta\mathbf{X_t}\delta\mathbf{U_t}^T = \sum_{i=1}^{n_s} \delta\mathbf{x_t}^i \delta\mathbf{u_t}^{iT}$.

From the definition of sample variance, we can write the above matrix as

$$\begin{aligned} \begin{bmatrix} \delta\mathbf{X_t}\delta\mathbf{X_t}^T & \delta\mathbf{X_t}\delta\mathbf{U_t}^T \\ \delta\mathbf{U_t}\delta\mathbf{X_t}^T & \delta\mathbf{U_t}\delta\mathbf{U_t}^T \end{bmatrix} &= \begin{bmatrix} \sum_{i=1}^{n_s} \delta\mathbf{x_t}^i \delta\mathbf{x_t}^{iT} & \sum_{i=1}^{n_s} \delta\mathbf{x_t}^i \delta\mathbf{u_t}^{iT} \\ \sum_{i=1}^{n_s} \delta\mathbf{u_t}^i \delta\mathbf{x_t}^{iT} & \sum_{i=1}^{n_s} \delta\mathbf{u_t}^i \delta\mathbf{u_t}^{iT} \end{bmatrix} \\ &\approx \begin{bmatrix} \sigma^2(n_s - 1)\mathbf{I}_{n_x} & 0_{n_x \times n_u} \\ 0_{n_u \times n_x} & \sigma^2(n_s - 1)\mathbf{I}_{n_u} \end{bmatrix} = \sigma^2(n_s - 1)\mathbf{I}_{(n_x+n_u)\times(n_x+n_u)} \end{aligned}$$

Given that we have high enough number of samples 'n_s' (typically, 30 to 40 is sufficient), the above approximation holds good. Since the inversion of an identity matrix is trivial and always exists, this means, the above matrix is invertible in equation (4.11). Thus, one can calculate $f_\mathbf{x}$ and $f_\mathbf{u}$ this

way during the backward pass. The entire algorithm to determine the optimal nominal trajectory in a model-free fashion is summarized together in Algorithm 4, Algorithm 5 and Algorithm 6.

Algorithm 4: Open-loop trajectory optimization via model-free DDP

Input: Initial State - $\mathbf{x_0}$, System and cost parameters - $\mathcal{P}$;
`/* Initialize the iteration number, k, to 1.*/`
$k \leftarrow 1$
$forward\_pass\_flag$ = true
`/* Run until the difference in costs between subsequent iterations is less an` ϵ `fraction of the former cost.*/`
while $k == 1$ or $(cost(\mathbb{T}^k_{nom})/cost(\mathbb{T}^{k-1}_{nom})) < 1 + \epsilon$ **do**
  `/*Each iteration has a backward pass followed by a forward pass.*/`
  $\{k^k_{0:N-1}, K^k_{0:N-1}\}$, $backward\_pass\_success\_flag$ = Backward Pass($\mathbb{T}^k_{nominal}$, $\mathcal{P}$)
  if *backward_pass_success_flag == true* **then**
    $\mathbb{T}^{k+1}_{nominal}, forward\_pass\_flag$ = Forward Pass($\mathbb{T}^k_{nominal}$, $\{k^k_{0:N-1}, K^k_{0:N-1}\}$, $\mathcal{P}$)
    while *forward_pass_flag == false* **do**
      $\mathbb{T}^{k+1}_{nominal}, forward\_pass\_flag$ = Forward Pass($\mathbb{T}^k_{nominal}$, $\{k^k_{0:N-1}, K^k_{0:N-1}\}$, $\mathcal{P}$)
      Reduce α from $\mathcal{P}$.
    end while
  end if
  else
    `/* Regularization step */`
    Increase μ from $\mathcal{P}$.
  end if
  $k \leftarrow k + 1$
  $\mathbb{T}^*_{nominal} \leftarrow \mathbb{T}^{k+1}_{nominal}$
end while
return $\mathbb{T}^*_{nominal}$

4.5 Example Applications

This section presents the results concerning the implementation of D2C-2.0 described in the previous sections of this chapter. The algorithm has been implemented on the systems that are chosen in the previous chapter. This is in order to be able to perform a comprehensive comparison with them, as shown in the next section. To recall from the previous chapter, the following are the systems and their tasks : pendulum swing up, cart-pole swing up and motion planning of a 3-link

Algorithm 5: Backward Pass

Input: Nominal trajectory - $\mathbb{T}^k_{nominal}$, previous iteration policy parameters - $\{k_{0:N-1}, K_{0:N-1}\}$, horizon - N and system and cost parameters - $\mathcal{P}$;

`/* Backward pass starts from the final time-step i.e, N-1.*/`

$t \leftarrow N-1$

Compute $V_{\mathbf{x_N}}$ and $V_{\mathbf{x_N x_N}}$ using boundary conditions.

`/*Keep a copy of the previous policy gains.*/`

$k\_old \leftarrow k_{0:N}$ and $K\_old \leftarrow K_{0:N}$.

while $t >= 0$ **do**

`/*Obtain the Jacobians from the simulator rollouts as shown in equation (4.11):*/`

$f_{\mathbf{x_t}}, f_{\mathbf{u_t}} \leftarrow model\_free\_jacobian(\bar{\mathbf{x}}_\mathbf{t}, \bar{\mathbf{u}}_\mathbf{t})$

`/*Obtain the partials of the Q function as follows:*/`

$$
\begin{aligned}
Q_{\mathbf{x_t}} &= l_{\mathbf{x_t}} + f^T_{\mathbf{x_t}} V'_{\mathbf{x_{t+1}}} \\
Q_{\mathbf{u_t}} &= l_{\mathbf{u_t}} + f^T_{\mathbf{u_t}} V'_{\mathbf{x_{t+1}}} \\
Q_{\mathbf{x_t x_t}} &= l_{\mathbf{x_t x_t}} + f^T_{\mathbf{x_t}} V'_{\mathbf{x_{t+1} x_{t+1}}} f_{\mathbf{x_t}} \\
Q_{\mathbf{u_t x_t}} &= l_{\mathbf{u_t x_t}} + f^T_{\mathbf{u_t}} (V'_{\mathbf{x_{t+1} x_{t+1}}} + \mu I_{n_x \times n_x}) f_{\mathbf{x_t}} \\
Q_{\mathbf{u_t u_t}} &= l_{\mathbf{u_t u_t}} + f^T_{\mathbf{u_t}} (V'_{\mathbf{x_{t+1} x_{t+1}}} + \mu I_{n_x \times n_x}) f_{\mathbf{u_t}}
\end{aligned}
$$

if $Q_{\mathbf{u_t u_t}}$ is positive-definite **then**

$$
\begin{aligned}
k_t &= -Q^{-1}_{\mathbf{u_t u_t}} Q_{\mathbf{u_t}}, \\
K_t &= -Q^{-1}_{\mathbf{u_t u_t}} Q_{\mathbf{u_t x_t}}.
\end{aligned}
$$

end if

else

`/*If` $Q_{\mathbf{u_t u_t}}$ `is not positive-definite, then, abort the backward pass.*/`

return $\{k\_old, K\_old\}$, false.

end if

`/*Obtain the partials of the value function` V_t `as follows:*/`

$$
\begin{aligned}
V_{\mathbf{x_t}} &= Q_{\mathbf{x_t}} + K^T_t Q_{\mathbf{u_t u_t}} k_t + K^T_t Q_{\mathbf{u_t}} + Q^T_{\mathbf{u_t x_t}} k_t, \\
V_{\mathbf{x_t x_t}} &= Q_{\mathbf{x_t x_t}} + K^T_t Q_{\mathbf{u_t u_t}} K_t + K^T_t Q_{\mathbf{u_t x_t}} + Q^T_{\mathbf{u_t x_t}} K_t.
\end{aligned}
$$

$t \leftarrow t-1$

end while

$k\_new = k_{0:N-1}$

$K\_new = K_{0:N-1}$

return $\{k\_new, K\_new\}$, true.

Algorithm 6: Forward Pass

Input: Nominal trajectory - $\mathbb{T}^k_{nominal}$, previous iteration policy parameters - $\{k_{0:N-1}, K_{0:N-1}\}$ and system and cost parameters - $\mathcal{P}$;

`/* Unpack the previous nominal trajectory.*/`

$\{\mathbf{x_t^{prev}}, \mathbf{u_t^{prev}}\} \leftarrow \mathbb{T}^k_{nominal}$

`/* Initialize time to 0.*/`

$t \leftarrow 0$;

while $t < N$ **do**

`/*`α `is the line-search parameter.*/`

`/*Simulate one step forward in a simulator.*/`

$$\mathbf{u_t} = \mathbf{u_t^{prev}} + \alpha k_t + K_t(\mathbf{x_t} - \mathbf{x_t^{prev}})$$
$$\mathbf{x_{t+1}} = simulate\_forward\_step(\mathbf{x_t}, \mathbf{u_t})$$

$t \leftarrow t + 1$

end while

$\mathbb{T}^{k+1}_{nominal} \leftarrow \{\mathbf{x_{0:N}}, \mathbf{u_{0:N-1}}\}$

if $\mathbb{T}^{k+1}_{nominal}$ is an acceptable new nominal trajectory from $\mathbb{T}^k_{nominal}$ **then**

return $\mathbb{T}^{k+1}_{nominal}$, true.

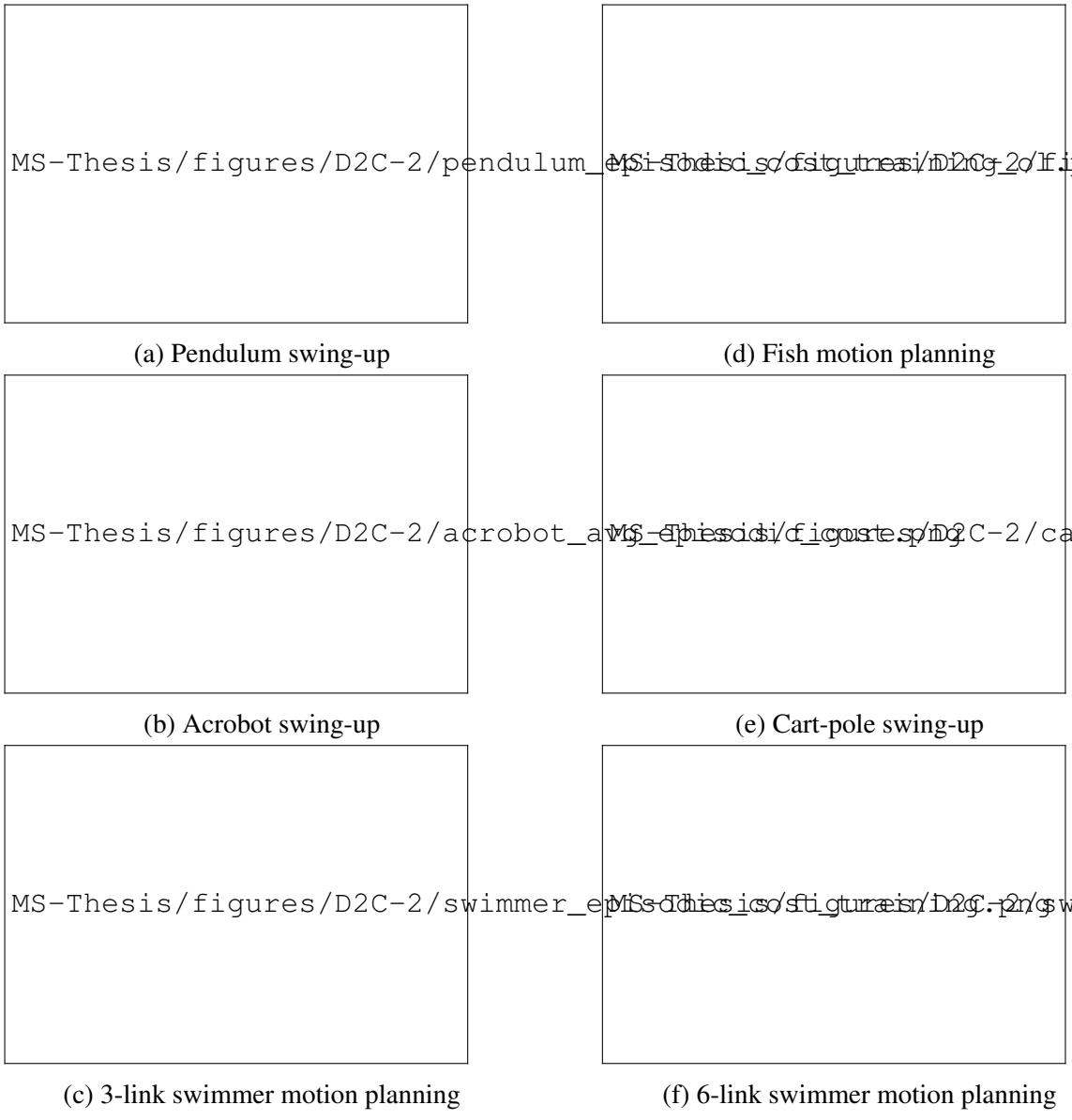
end if

else

return $\mathbb{T}^k_{nominal}$, false.

end if

swimmer, 6-link swimmer and fish. Figure 4.2 shows the open-loop training plots for each of the systems. As each iteration corresponds to an episode of training, each training plot comprises of averaged episodic costs over 5 trials of training.



(a) Pendulum swing-up

(d) Fish motion planning

(b) Acrobot swing-up

(e) Cart-pole swing-up

(c) 3-link swimmer motion planning

(f) 6-link swimmer motion planning

Figure 4.2: Training for the optimal policy in D2C - 2.0

4.6 Discussion and Comparison of Methods

It is already discussed on how D2C fares well w.r.t. DDPG in training time. In order to have a fair comparison between the original D2C and D2C-2.0, the same cost function is used. A consequence of this is that it will not result in the best performances for each of the methods. Rather, we are interested in analyzing the performances under an arbitrary cost function which performs the requisite task.

Table 4.1: Comparison of the simulation parameters and training outcomes of D2C-2.0 with other baselines

System	State dimen-sion	Control dimen-sion	Steps per episode	Time-step (in sec.)	Training time (in sec.)			
					D2C		DDPG	D2C-2.0
					Open-loop	Closed-loop		
Inverted Pendulum	2	1	30	0.1	12.9	< 0.1	2261.15	0.33
Cart pole	4	1	30	0.1	15.0	1.33	6306.7	1.62
3-link Swimmer	10	2	800	0.01	4001.0	4.6	38833.64	186.2
6-link Swimmer	16	5	900	0.01	3585.4	16.4	88160	127.2
Fish	27	5	1200	0.005	6011.2	75.6	124367.6	54.8

Table 4.1 shows the training time comparing the original D2C, a state-of-the-art reinforcement learning (DDPG) and D2C-2.0. Note that this is different from the training time of D2C shown in table 3.1. This is because, as emphasized in the previous paragraph, D2C presented in the previous chapter and that in the current chapter are different implementations. Now, comparing D2C-2.0 with the original D2C, it is evident from the table that for simple and lower dimensional systems such as pendulum and cart pole the policy can be almost calculated online (parallelization could make it much more faster). It is due to the tendency of DDP to quickly converge if the linearization of system models has bigger basin of attraction (such as pendulum, which is, in fact, almost often

considered to be linear about its rest position). Whereas for a higher dimensional system such as in the case of a robotic fish, D2C-2.0 took 311 seconds while the original D2C took around 6096 seconds, which is approximately 20 times faster. The table makes it evident that D2C-2.0 clearly exhibits a faster convergence property.

A major reason behind the performance differences stems from their formulation. D2C is based on gradient descent which is a linearly convergent algorithm. It is a generic way of solving an optimization problem that doesn't take advantage of the structure in the problem. In other words, it is a direct method. On the other hand, the equations involved in D2C-2.0 arise from the optimality conditions that inherently exploit the recursive optimal structure in the problem. It is identical to the Newton's method and hence, exhibits near second order convergence properties. Moreover, the equations are explicit in the Jacobians of the system model. As a result, it is being possible to solve for it *independently* and in a sample efficient manner, at every time-step of each iteration.

Another notable advantage comes from the fact that though this is a second order convergence based method, we are ignoring the second order derivatives of the dynamical model. Earlier, it is noted that this is a conscious decision that is observed to drastically improve the speed of convergence in case of model-based problems, provided it is sufficiently compensated with the regularization and line search schemes. Hence, we don't need to account for such computationally intensive variables.

Now, as mentioned in the related work section of this chapter, earlier works have attempted to tackle the idea of model-free ILQR and have mostly confined to finite-differences for Jacobian computation. Table 4.2 shows the comparison of per-iteration computational times between 'finite-differences' and 'linear least squares with central difference formulation'. It is clearly evident that, as the dimension of the state space increases, the method of finite-differences requires many more function evaluations, and hence, our LLS-CD method is much more efficient.

To end the discussion on the global vs local policy approximation dilemma (while comparing with DDPG), how well deep RL methods interpolate their policy to the entire continuous state

Table 4.2: Comparison of the computational times (in sec.) per iteration in seconds (averaged over 5 runs).

System	FD	LLS-CD
Inverted Pendulum	0.017	0.033
Cart pole	0.0315	0.0463
Acrobot	0.554	0.625
3-link Swimmer	4.39	1.86
6-link Swimmer	14.43	1.278
Fish	34.75	2.74

and action spaces remains an open question. Now, consider that the results shown in the Table 4.1 are based on serial implementations on an off-the-shelf computer and D2C-2.0 can be highly parallelizable. By looking at the order of magnitude of numbers in the same table, we expect with reasonable augmentation of computational power by parallelization, D2C-2.0 could offer a near-real time solution in high dimensional problems. In such cases, one could rely on the near-optimal policy described in this paper for low noises, and re-solve for the open-loop trajectory online by the approach presented in this paper along with the attendant feedback, whenever the noise takes the method out of the region of attraction of the linear feedback policy. This will be our motivation for the future work.

5. CONCLUSIONS

It is established that in a fully-observed scenario, a deterministic action policy can be split into an optimal nominal sequence and a feedback that tracks the nominal in order to maintain the cost within a tube around the nominal. As a result, we proposed the 'T-PFC' algorithm. It is shown to be maintaining low cost, has low online computation and hence, faster execution. This makes our approach tractable in systems with higher dimensional states. Like NMPC, the nominal trajectory design of T-PFC also allows for the inclusion of constraints as described. We have empirically shown that the overall control signals are very close to the saturation boundary, if not with-in, when the nominal is at saturation. Also, T-PFC works with minimal number of re-plannings even at medium noise levels, as against to the traditional principle of deterministic NMPC to re-plan in a recurrent fashion irrespective of the noise levels. Coming to its limitations, the following are noteworthy:

- T-PFC assumes a control-affine system and the cost to be in a specific form. Though many robotic systems are affine in controls, methods like T-LQR have an edge by considering a general nonlinear system.

- Though T-LQR does not fare well on the cost incurred, it offers a flexibility to tune the feedback parameters according to ones needs, even if that means sacrificing the optimality.

Future model-based works can focus on overriding the assumptions made w.r.t. the system model and the form of the cost function. Another important direction could be to exploring this idea of decoupling to partially-observed systems and also dealing with nonlinear hard inequality constraints.

In the context of model-free data-based solutions, we proposed a near-optimal control algorithm under fully observed conditions from an alternative theoretical perspective and showed that our method is able to scale-up to higher dimensional state-space without any knowledge about the system model. Due to sequential calculation used in the open-loop optimization and the system

identification, it is shown that D2C is memory efficient and also convenient for parallelization. We tested its performance and juxtaposed them with a state-of-the-art deep RL technique - DDPG. From the results, our method has conspicuous advantages over DDPG in terms of data efficiency and ease of training. The robustness of D2C is also better in some cases, but has scope for further development by employing more sophisticated feedback methods and ensuring that the data efficiency is not compromised. We also believe further drastic reduction in the planning time can be achieved by parallelization and a more sophisticated parametrization of the open loop problem.

It is evident from the simulations that methods such as D2C are able to achieve their goals accurately whereas DDPG consumes inordinate amount of time in 'fine-tuning' their behavior towards the goal. However, we also note that, by doing this, DDPG is tentatively exploring over the entire state-space and can result in a better generic policy. Another drawback with D2C over canonical RL algorithms is that the cost could be stuck in a local minimum, whereas DDPG due to the nature of stochastic gradient descent in training neural networks, can potentially reach the globally optimal solution. Nevertheless, this approach is rather aimed to signify the potential of decoupling based approaches such as D2C in a reinforcement learning paradigm and recognizes the need for more hybrid approaches that complement the merits of each.

A closer inspection into the D2C algorithm revealed that there is a great scope to improve the computational efficiency in the design of the open-loop plan. Hence, as a sequel to the D2C algorithm, we proposed the D2C-2.0 algorithm which, as the simulation results show, drastically reduced the run-time of the algorithm as a whole. In fact, in simpler problems such as the pendulum and the cart-pole swing-up, the algorithm run truly online, which means we have a global feedback solution (not to be confused with the globally optimal solution). However, it should be noted that the same formulation that resulted in the faster (second-order) convergence is likely to slow down with the increasing state dimension beyond a threshold. This is due to the presence of large number of variables that are of polynomial order in the state dimension. Also, the algebraic expressions involved in each iteration, for example, the computation of the model's Jacobians or the inversion of large matrices are expected to scale up poorly. In such instances, it might be worth revisiting

the original way of solving for the open-loop trajectory. This is an interesting problem to explore for the future work. Problems with very high dimensions can be found in non traditional sub-disciplines of robotics such as tensegrity structures. Another direction could be to make use of the third order tensor terms in the formulation of DDP for rapid convergence. In other words, it is clear from the behavior of the training plot in D2C-2.0 that the first few iterations witness a drastic reduction in the cost, while the remaining iterations are spent in 'fine-tuning'. To apply the second-order convergence property throughout training, it may be worth investigating the utility of the second order derivatives of the model.

REFERENCES

[1] P. R. Kumar and P. Varaiya, *Stochastic systems: Estimation, identification, and adaptive control*, vol. 75. SIAM, 2015.

[2] D. P. Bertsekas, *Dynamic Programming and Stochastic Control*. Academic Press, 1976.

[3] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," *International Conference on Informatics in Control, Automation and Robotics*, vol. 1, pp. 222–229, 2004.

[4] S. Lavalle, *Planning algorithms*. Cambridge University Press, 2006.

[5] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," tech. rep., Iowa State University, 1998.

[6] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[7] S. M. LaValle and J. James J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[8] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017.

[9] A. Rao, "A survey of numerical methods for optimal control," *Advances in the Astronautical Sciences*, vol. 135, pp. 497–528, 2010.

[10] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *International Conference on Robotics and Automation (ICRA)*, pp. 1168–1175, 2014.

[11] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *American Control Conference*, pp. 300–306, 2005.

[12] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *International Conference on Intelligent Robots and Systems*, pp. 4906–4913, 2012.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. G Bellemare, A. Graves, M. Riedmiller, A. K Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 2015.

[15] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *AAAI Fall Symposia*, 2015.

[16] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI Conference on Artificial Intelligence*, 2015.

[17] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning*, 2015.

[18] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *AAAI Conference on Artificial Intelligence*, 2018.

[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, vol. 32, pp. 387–395, 2014.

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

[21] S. John, L. Sergey, M. Philipp, J. Michael I., and A. Pieter, "Trust region policy optimization," *arXiv:1502.05477*, 2017.

[22] W. Yuhuai, M. Elman, L. Shun, G. Roger, and B. Jimmy, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," *arXiv:1708.05144*, 2017.

[23] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv:1801.01290*, 2018.

[24] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[25] R. E. Kopp, "Pontryagin's maximum principle," *Mathematics in Science and Engineering*, vol. 5, pp. 255–279, 1962.

[26] D. H. Jacobson, *Differential dynamic programming*. Elsevier, New york, 1970.

[27] E. T. E. Theodorou, Y. Tassa, "Stochastic differential dynamic programming," in *American Control Conference*, pp. 1125–1132, 2010.

[28] P. Rutquist, *Methods for Stochastic Optimal Control*. PhD thesis, Chalmers University of Technology, 2017.

[29] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, pp. 2967–2986, 2014.

[30] N. Cazy, P. Wieber, P. R. Giordano, and F. Chaumette, "Visual servoing using model predictive control to assist multiple trajectory tracking," in *International Conference on Robotics and Automation (ICRA)*, pp. 2057–2064, 2017.

[31] G. Garimella, M. Sheckells, and M. Kobilarov, "Robust obstacle avoidance for aerial platforms using adaptive model predictive control," in *International Conference on Robotics and Automation (ICRA)*, pp. 5876–5882, 2017.

[32] B. Kouvaritakis and M. Cannon, "Non-linear predictive control: Theory and practice," *Bibliovault OAI Repository, the University of Chicago Press*, 2001.

[33] I. W.Langson, S. Raković, and D. Mayne, "Robust model predictive control using tubes," *Automatica*, vol. 40, pp. 125–133, 2004.

[34] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *International Conference on Robotics and Automation (ICRA)*, pp. 1714–1721, 2017.

[35] M. Rafieisakhaei, S. Chakravorty, and P. R. Kumar, "A near-optimal decoupling principle for nonlinear stochastic systems arising in robotic path planning and control," in *Conference on Decision and Control (CDC)*, pp. 1–6, 2017.

[36] K. S. Parunandi and S. Chakravorty, "T-pfc: A trajectory-optimized perturbation feedback control approach," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3457–3464, 2019.

[37] D. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific, 3 ed., 2007.

[38] W. H. Fleming, "Stochastic control for small noise intensities," *SIAM Journal on Control and Optimization*, vol. 9, pp. 473–517, 1971.

[39] N. Moshtagh, "Minimum volume enclosing ellipsoid," *http://www.mathworks.com/matlabcentral/fileexchange/9542*, 2006.

[40] M. Rafieisakhaei, S. Chakravorty, and P. R. Kumar, "T-lqg: Closed-loop belief space planning via trajectory-optimized lqg," in *International Conference on Robotics and Automation (ICRA)*, pp. 649–656, 2017.

[41] V. Nevistic and J. A. Primbs, "Constrained nonlinear optimal control: A converse hjb approach," tech. rep., 1996.

[42] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2149–2154, 2004.

[43] M. Quigley, B. P. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," tech. rep., 2009.

[44] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," tech. rep., 2018.

[45] A. Wächter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[46] T. Luukkonen, "Modelling and control of quadcopter," tech. rep., School of science, Aalto university, 2011.

[47] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *RotorS - A modular gazebo MAV simulator framework*, vol. 625, pp. 595–625. 2016.

[48] P. A. Ioannou and J. Sun, *Robust adaptive control*. Courier Corporation, 2012.

[49] K. J. Åström and B. Wittenmark, *Adaptive control*. Courier Corporation, 2013.

[50] S. Sastry and M. Bodson, *Adaptive control: stability, convergence and robustness*. Courier Corporation, 2011.

[51] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[52] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[53] S. John, W. Filip, D. Prafulla, R. Alec, and K. Oleg, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[54] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *AAAI Conference on Artificial Intelligence*, 2018.

[55] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-prop: Sample-efficient policy gradient with an off-policy critic," *arXiv:1611.02247*, 2016.

[56] M. Falcone, "Recent results in the approximation of nonlinear optimal control problems," in *Large-Scale Scientific Computing*, 2013.

[57] W. Li and E. Todorov, "Iterative linearization methods for approximately optimal control and estimation of non-linear stochastic system," *International Journal of Control*, vol. 80, no. 9, pp. 1439–1453, 2007.

[58] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*. John Wiley & Sons, 2007.

[59] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade, "Towards generalization and simplicity in continuous control," in *Advances in Neural Information Processing Systems*, pp. 6550–6561, Curran Associates, Inc., 2017.

[60] D. Yu, M. Rafieisakhaei, and S. Chakravorty, "Stochastic feedback control of systems with unknown nonlinear dynamics," in *Conference on Decision and Control(CDC)*, 2017.

[61] M. Rafieisakhaei, S. Chakravorty, and P. R. Kumar, "A near-optimal separation principle for nonlinear stochastic systems arising in robotic path planning and control," in *Conference on Decision and Control(CDC)*, 2017.

[62] R. Wang, K. Parunandi, D. Yu, D. Kalathil, and S. Chakravorty, "Decoupled data based approach for learning to control nonlinear dynamical systems," *arXiv:1904.08361*, 2019.

[63] T. Emanuel, E. Tom, and Y. Tassa, "Mujoco: A physics engine for model-based control," *International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[64] T. Yuval and *et al.*, "Deepmind control suite," *arXiv:1801.00690*, 2018.

[65] M. Plappert, "Keras-rl." `https://github.com/keras-rl/keras-rl`, 2016.

[66] M. G. D. P. Riashat Islam, Peter Henderson, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," *Reproducibility in Machine Learning Workshop, ICML*, 2017.

[67] J. C. Spall, "An overview of the simultaneous perturbation method for efficient optimization," *Johns Hopkins APL Technical Digest*, vol. 19, no. 4, pp. 482–492, 1998.

[68] Z. Xie, C. K. Liu, and K. Hauser, "Differential dynamic programming with nonlinear constraints," in *International Conference on Robotics and Automation (ICRA)*, pp. 695–702, 2017.

APPENDIX A

A.0.1 Proof of Lemma 1

We proceed by induction. The first general instance of the recursion occurs at $t = 3$. It can be shown that:

$$\delta\mathbf{x_3} = \underbrace{(\bar{A}_2\bar{A}_1(\epsilon w_0) + \bar{A}_2(\epsilon w_1) + \epsilon w_2)}_{\delta x_3^l} + \underbrace{\{\bar{A}_2\bar{S}_1(\epsilon w_0) + \bar{S}_2(\bar{A}_1(\epsilon w_0) + \bar{S}_2(\bar{A}_1(\epsilon w_0) + \epsilon w_1 + \bar{S}_1(\epsilon w_0))\}}_{\bar{\bar{S}}_3}. \tag{A.1}$$

Noting that $\bar{S}_1(.)$ and $\bar{S}_2(.)$ are second and higher order terms, it follows that $\bar{\bar{S}}_3$ is $O(\epsilon^2)$. Suppose now that $\delta\mathbf{x_t} = \delta\mathbf{x_t^l} + \bar{\bar{S}}_t$ where $\bar{\bar{S}}_t$ is $O(\epsilon^2)$. Then:

$$\begin{aligned}\delta\mathbf{x_{t+1}} &= \bar{A}_{t+1}(\delta\mathbf{x_t^l} + \bar{\bar{S}}_t) + \epsilon w_t + \bar{S}_{t+1}(\delta\mathbf{x_t}),\\ &= \underbrace{(\bar{A}_{t+1}\delta\mathbf{x_t^l} + \epsilon w_t)}_{\delta\mathbf{x_{t+1}^l}} + \underbrace{\{\bar{A}_{t+1}\bar{\bar{S}}_t + \bar{S}_{t+1}(\delta\mathbf{x_t})\}}_{\bar{\bar{S}}_{t+1}}.\end{aligned} \tag{A.2}$$

Noting that $\bar{S}_{t+1}$ is $O(\epsilon^2)$ and that $\bar{\bar{S}}_{t+1}$ is $O(\epsilon^2)$ by assumption, the result follows.

A.0.2 Lemma 2

Lemma 2. *Let δJ_1^π, δJ_2^π be as defined in* (3.13)*. Then, $\mathbb{E}[\delta J_1 \delta J_2]$ is an $O(\epsilon^4)$ function.*

In the following, we suppress the explicit dependence on π for δJ_1^π and δJ_2^π for notational convenience. Recall that $\delta J_1 = \sum_{t=0}^{T} c_t^x \delta\mathbf{x_t^l}$, and $\delta J_2 = \sum_{t=0}^{T} \bar{H}_t(\delta\mathbf{x_t}) + c_t^x \bar{\bar{S}}_t$. For notational convenience, let us consider the scalar case, the vector case follows readily at the expense of more elaborate notation. Let us first consider $\bar{\bar{S}}_2$. We have that $\bar{\bar{S}}_2 = \bar{A}_2\bar{S}_1(\epsilon w_0) + \bar{S}_2(\bar{A}_1(\epsilon w_0) + \epsilon w_1 +$

$\bar{S}_1(\epsilon w_0))$. Then, it follows that:

$$\bar{\bar{S}}_2 = \bar{A}_2 \bar{S}_1^{(2)}(\epsilon w_0)^2 + \bar{S}_2^{(2)}(\bar{A}_1 \epsilon w_0 + \epsilon w_1)^2 + O(\epsilon^3), \tag{A.3}$$

where $\bar{S}_t^{(2)}$ represents the coefficient of the second order term in the expansion of $\bar{S}_t$. A similar observation holds for $H_2(\delta \mathbf{x_2})$ in that:

$$\bar{H}_2(\delta \mathbf{x_2}) = \bar{H}_2^{(2)}(\bar{A}_1(\epsilon w_0) + \epsilon w_1)^2 + O(\epsilon^3), \tag{A.4}$$

where $\bar{H}_t^{(2)}$ is the coefficient of the second order term in the expansion of $\bar{H}_t$. Note that $\epsilon w_0 = \delta \mathbf{x_1^1}$ and $\bar{A}_1(\epsilon w_0) + \epsilon w_1 = \delta \mathbf{x_2^1}$. Therefore, it follows that we may write:

$$\bar{H}_t(\delta \mathbf{x_t}) + C_t^x \bar{\bar{S}}_t = \sum_{\tau=0}^{t-1} q_{t,\tau}(\delta \mathbf{x}_\tau^1)^2 + O(\epsilon^3), \tag{A.5}$$

for suitably defined coefficients $q_{t,\tau}$. Therefore, it follows that

$$\begin{aligned} \delta J_2 &= \sum_{t=1}^{T} \bar{H}_t(\delta \mathbf{x_t}) + C_t^x \bar{\bar{S}}_t \\ &= \sum_{\tau=0}^{T} \bar{q}_{T,\tau}(\delta \mathbf{x}_\tau^1)^2 + O(\epsilon^3), \end{aligned} \tag{A.6}$$

for suitably defined $\bar{q}_{T,\tau}$. Therefore:

$$\delta J_1 \delta J_2 = \sum_{t,\tau} C_\tau^x (\delta \mathbf{x}_\tau^1) \bar{q}_{T,t} (\delta \mathbf{x_t^1})^2 + O(\epsilon^4). \tag{A.7}$$

Taking expectations on both sides:

$$E[\delta J_1 \delta J_2] = \sum_{t,\tau} C_\tau^x \bar{q}_{T,t} E[\delta \mathbf{x}_\tau^1 (\delta \mathbf{x_t^1})^2] + O(\epsilon^4). \tag{A.8}$$

Break $\delta \mathbf{x_t^l} = (\delta \mathbf{x_t^l} - \delta \mathbf{x_\tau^l}) + \delta \mathbf{x_\tau^l}$, assuming $\tau < t$. Then, it follows that:

$$\begin{aligned} E[\delta \mathbf{x_\tau^l}(\delta \mathbf{x_t^l})^2] &= E[\delta \mathbf{x_\tau^l}(\delta \mathbf{x_t^l} - \delta \mathbf{x_\tau^l})^2] + E[(\delta \mathbf{x_\tau^l})^3] \\ &\quad + 2E[(\delta \mathbf{x_t^l} - \delta \mathbf{x_\tau^l})(\delta \mathbf{x_\tau^l})^2] \\ &= E[(\delta \mathbf{x_\tau^l})^3], \end{aligned} \tag{A.9}$$

where the first and last terms in the first equality drop out due to the independence of the increment $(\delta \mathbf{x_t^l} - \delta \mathbf{x_\tau^l})$ from $\delta \mathbf{x_\tau^l}$, and the fact that $E[\delta \mathbf{x_t^l} - \delta \mathbf{x_\tau^l}] = 0$ and $E[\delta \mathbf{x_\tau^l}] = 0$. Since $\delta \mathbf{x_\tau^l}$ is the state of the linear system $\delta \mathbf{x_{t+1}} = \bar{A}_t \delta \mathbf{x_t^l} + \epsilon w_t$, it may again be shown that:

$$E[\delta x_\tau^l]^3 = \sum_{s_1, s_2, s_3} \Phi_{\tau, s_1} \Phi_{\tau, s_2} \Phi_{\tau, s_3} E[w_{s_1} w_{s_2} w_{s_3}], \tag{A.10}$$

where $\Phi_{t,\tau}$ represents the state transitions operator between times τ and t, and follows from the closed loop dynamics. Now, due to the independence of the noise terms w_t, it follows that $E[w_{s_1} w_{s_2} w_{s_3}] = 0$ regardless of s_1, s_2, s_3.

An analogous argument as above can be repeated for the case when $\tau > t$. Therefore, it follows that $E[\delta J_1 \delta J_2] = O(\epsilon^4)$.

A.0.3 DDPG Algorithm

Deep Deterministic Policy Gradient (DDPG) is a policy-gradient based off-policy reinforcement learning algorithm that operates in continuous state and action spaces. It relies on two function approximation networks one each for the actor and the critic. The critic network estimates the $Q(s, a)$ value given the state and the action taken, while the actor network engenders a policy given the current state. Neural networks are employed to represent the networks.

The off-policy characteristic of the algorithm employs a separate behavioural policy by introducing additive noise to the policy output obtained from the actor network. The critic network minimizes loss based on the temporal-difference (TD) error and the actor network uses the deterministic policy gradient theorem to update its policy gradient as shown below:

Table A.1: D2C parameters

System	Std of noise	Stepsize	Cost parameters*		
			Q	Q_t	R
Inverted Pendulum	0.0005	0.00018	0	700	0
Cart pole	0.07	0.005	10	(200, 100, 500, 100)	0
3-link Swimmer	0.2	0.022	2.5	2000	0.001
6-link Swimmer	0.2	0.018	2	200	0.001
Fish	0.05	0.0004	0.3	260	0.1

* Q is the incremental cost matrix, Q_t is the terminal cost matrix and R is the control cost matrix, all of which are diagonal matrices. If the diagonal elements have the same value, only one of them is presented in the table, otherwise all diagonal value are presented. The cost function structure is specified in the openloop training code on our github repository.

Critic update by minimizing the loss:

$$L = \frac{1}{N}\Sigma_i(y_i - Q(s_i, a_i|\theta^Q))^2$$

Actor policy gradient update:

$$\nabla_{\theta^\mu} \approx \frac{1}{N}\Sigma_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

The actor and the critic networks consists of two hidden layers with the first layer containing 400 ('*relu*' activated) units followed by the second layer containing 300 ('*relu*' activated) units. The output layer of the actor network has the number of ('tanh' activated) units equal to that of the number of actions in the action space.

Target networks one each for the actor and the critic are employed for a gradual update of network parameters, thereby reducing the oscillations and a better training stability. The target networks are updated at $\tau = 0.001$. Experience replay is another technique that improves the

stability of training by training the network with a batch of randomized data samples from its experience. We have used a batch size of 32 for the inverted pendulum and the cartpole examples, whereas it is 64 for the rest. Finally, the networks are compiled using Adams' optimizer with a learning rate of 0.001.

To account for state-space exploration, the behavioural policy consists of an off-policy term arising from a random process. We obtain discrete samples from Ornstein-Uhlenbeck (OU) process to generate noise as followed in the original DDPG method. The OU process has mean-reverting property and produces temporally correlated noise samples as follows:

$$dx_t = \Theta(\mu - x_t)dt + \sigma dW$$

where Θ indicates how fast the process reverts to mean, μ is the equilibrium or the mean value and σ corresponds to the degree of volatility of the process. Θ is set to 0.15, μ to 0 and σ is annealed from 0.35 to 0.05 over the training process.