MODEL PREDICTIVE PATH INTEGRAL CONTROL: THEORETICAL FOUNDATIONS AND APPLICATIONS TO AUTONOMOUS DRIVING

A Dissertation
Presented to
The Academic Faculty

By

Grady R. Williams

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology

May 2019

MODEL PREDICTIVE PATH INTEGRAL CONTROL: THEORETICAL FOUNDATIONS AND APPLICATIONS TO AUTONOMOUS DRIVING

There is nothing so powerful as truth - and often nothing so strange.

Daniel Webster

To my Grandfather, Dr. Grady F. Williams.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Prof. Evangelos Theodorou, for his continuous support since my first day at Georgia Tech. Without his knowledge, guidance, and openness, this research would not have been possible. In addition to an amazing teacher, he has been a tremendous mentor during the past five years and has helped me grow as both a student and person.

Besides my advisor, I would like to thank the rest of my thesis committe: Prof. James Rehg, Prof. Magnus Egerstedt, Prof. Byron Boots, and Prof. Emo Todorov, for their insights, encouragement, and feedback.

My sincere thanks also goes to Prof. Tom Daniel and Prof. Eric Rombokas who introduced me to robotics research as an undergraduate, and helped guide me on the path to pursuing a PhD in robotics. I would also like to thank Christopher Swierczewski, who was one who first introduced me to independent research.

I thank my current and former labmates at Georgia Tech in the Autonomous Control and Decision Systems lab: Kamil Saigol, Yunpeng Pan, Manan Gandhi, Ethan Evans, Marcus Pereira, George Boutselis, Keuntaek Lee, Pat Wang, David Fan, Harleen Brar, and Kaivalya Bakshi. I would also like to thank my collaborators on the AutoRally project in the Computational Perception Lab: Brian Goldfain and Paul Drews.

I would like to thank my family, especially my parents: Joe Williams and Liz Broderick, who have always offered their support and who helped encourage me to pursue a PhD. Most of all, I would like to thank my Fiancée, Elsa Ayala, who has been with me throughout all of the highs and lows of graduate school, and who has always offered me her love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis presents a new approach for stochastic model predictive (optimal) control: model predictive path integral control, which is based on massive parallel sampling of control trajectories. We first show the theoretical foundations of model predictive path integral control, which are based on a combination of path integral control theory and an information theoretic interpretation of stochastic optimal control. We then apply the method to high speed autonomous driving on a 1/5 scale vehicle, and analyze the performance and robustness of the method. Extensive experimental results are used to identify and solve key problems relating to robustness of the approach, which leads to a robust stochastic model predictive control algorithm capable of consistently pushing the limits of performance on the 1/5 scale vehicle.

CHAPTER 1
INTRODUCTION

Autonomous robotic systems operating in dynamic, uncontrolled environments have the potential to revolutionize modern industry. The deployment of these systems could increase the quality and efficiency of solutions to current problems by augmenting the strength of workers, and they could create new services by performing tasks that are too dangerous or otherwise not economically feasible for human workers. This could in turn lead to drastic improvements in the quality of life for people living in highly automated societies. However, in order to get to this point, there has to be sufficiently advanced technology available so that creating and deploying robotic systems to solve real world tasks can be easily done.

In order to operate effectively, autonomous robotic systems need to be able to perceive their environment through their sensors, create high-level plans based on an interpretation of the environment, and then effectively control their actuators in order to carry out their plans. In this thesis we are concerned with the latter of these: once a high-level course of action has been decided how can an autonomous system use its actuators to best carry it out? This is an inherently difficult problem, and it has very high real world stakes. The interaction between the robot and the physical world can be governed by complex physics, and interactions with the physical world are almost always noisy and uncertain. Additionally, many of the problems we would like robots to solve have task critical constraints, and violation of these constraints can be catastrophic.

While the goal of ubiquitous autonomous robots increasing productivity and quality of life is nearly as old as the computer itself, in the last couple of decades the computational tools available to achieve this goal have exploded, and previously unthinkable algorithmic approaches are now routine. The clearest example of this technological shift is the advent of deep learning for robot perception, which would not be possible without the massive parallel computing power of graphics processing units (GPUs). In this thesis, high performance computing with GPUs is also the key tool which enable the development of our algorithms. But instead of utilizing GPUs for sensor processing tasks critical in perception, we apply them to massive forward simulation of possible futures (trajectories) for the robot. The advent of fast forward simulation on GPUs enables new algorithms, that were once wildly intractable, to be applied in the context of autonomous control.

1.1 Motivating Problem

The mathematics and algorithms that we develop in this thesis can be applied to controlling general autonomous systems. However, the primary system that we consider is the autonomous racing platform developed at Georgia Tech known as AutoRally [1], which is pictured in Fig. 1.1. AutoRally vehicles are 1/5 scaled autonomous driving platforms. The 1/5 scale size of the platforms makes the systems large enough to carry powerful sensing and computing hardware, and the dynamics better resemble that of a full-scale vehicle than the smaller 1/10 scale vehicles commonly used in laboratories and educational settings. On the other hand, the 1/5 scale is still small enough the vehicle can be handled by a small team and survive crashes that are inevitable when conducting experimental research.

The canonical task that we consider for the AutoRally is to navigate a dirt track as fast as possible while staying within the track boundaries. Utilizing the AutoRally system on this task serves as a source of inspiration for our desired controller properties, provides a method of experimentally proving our algorithms, and enables us to observe problems relating to performance and robustness that would be difficult to discover on less capable systems. Utilizing the AutoRally also enforces a degree of practicality and computational feasibility on the algorithms we develop, since real-world trials are not cheap, and compute resources are limited.

One of the main advantages of focusing on the AutoRally system is the direct appli-

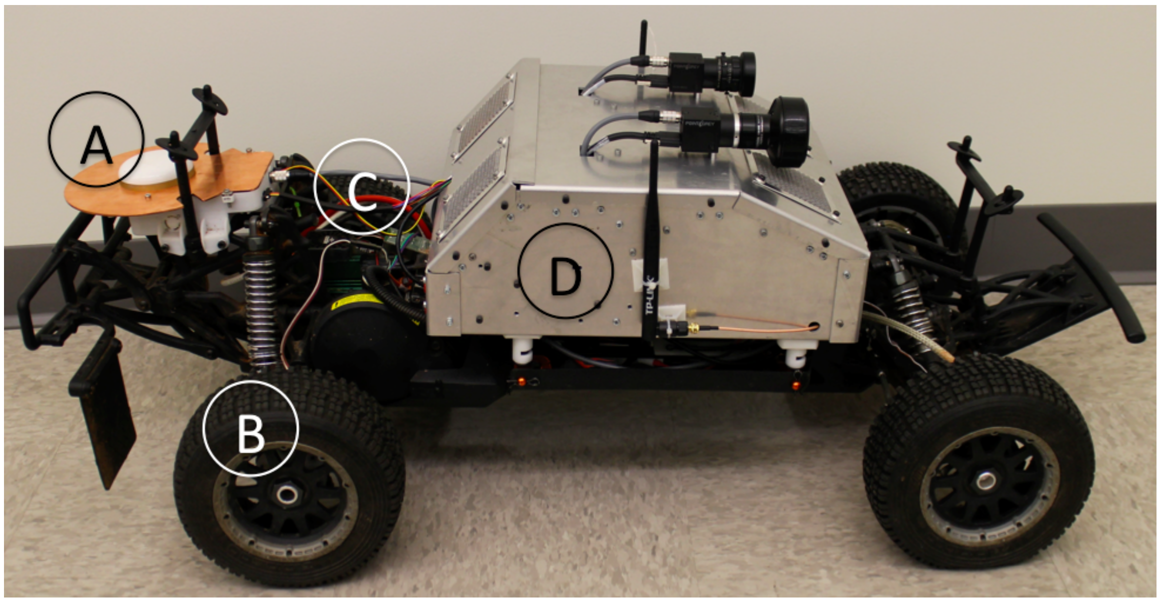


Figure 1.1: AutoRally vehicle. The vehicle is large enough to carry all of the sensors and computing required for autonomous driving on-board. A: High accuracy GPS enclosure for localization. B: Wheel speed sensors. C: High performance electric motor capable of over 50 mph. D: Rugged computer and battery enclose with camera mounts. The on-board computer carries an Nvidia GTX 1050 Ti GPU.

cability of algorithms developed on the AutoRally to full-size autonomous driving. Autonomous driving is now one of the most important sub-fields in robotics and artificial intelligence, and the introduction of safe autonomous driving systems is predicted to have a widespread positive impact on society [2, 3]. While existing control methodologies have proven to be effective for many standard vehicle tasks such as lane keeping, turning, and parking, there is an important frontier of control at the limits of vehicle performance that has not been fully addressed by prior work. This is important, since many collisions can be avoided or mitigated by performing an appropriate aggressive maneuver, and high-speed autonomous driving on the AutoRally serves as a direct proxy problem for these types of tasks.

The main potential drawback of heavily relying on a single experimental system, like the AutoRally, is the concern that the resulting algorithms will not be applicable to any other system. However, in the case of this thesis, this potential drawback has some mitigating features. The first is that the mathematics behind the algorithms are completely general, in other words there are no vehicle specific assumptions relating to the AutoRally that limit the algorithmic generality. The other key feature is that the task we attempt to solve with the AutoRally: maneuvering around a certain area at high speeds while satisfying constraints, is a standard type of task that we would like autonomous robots to be able to perform. So, although we perform all of our real-world experiments on the AutoRally platform, there is reason to believe that the algorithms developed in this thesis could have broader applicability, and we do spend some time analyzing performance on related (simulated) tasks such as quadrotor navigation and helicopter landing.

1.2 Controller Desiderata

There are a few key properties that we need an autonomous racing controller to have in order to effectively handle the AutoRally vehicle. The properties that we want in a control algorithm, which we outline here, guide the work in the rest of this thesis. In addition to driving the AutoRally vehicle around a dirt track as fast as possible, we want our approach to be generalizable enough that it could be plausibly applied to related systems (e.g. a full-scale car driving on a highway).

The first requirement that we have for an autonomous racing system is the ability to control the vehicle up to its limits of handling. This trait is necessary since, in order to drive as fast as possible, the vehicle must push up against these limits. This is especially important when cornering at high speeds: if the vehicle goes too fast it can spin out or roll over but playing it safe and going to slow is not desirable in our racing context. In order to satisfy this requirement, our system needs to be able to operate in a fast control loop, understand the non-linear dynamics of the vehicle, and understand the consequences of its current actions far (several seconds) into the future.

The next requirement that we want is for the system to be able to satisfy constraints. In the AutoRally case, this means that the vehicle needs to stay on the track. This is difficult because the fastest path around the track often requires driving very close to the boundaries.

In order to drive as fast as possible, the system therefore needs to know exactly where the boundary is and know not to cross it. In a more general setting, satisfying constraints could mean avoiding other vehicles, pedestrians, or obstacles. For an autonomous driving system in the real world, achieving these tasks is critical, and throughout this thesis we focus much of our attention in this area.

A final consideration is that we would like all planning and control to take place in real-time. This means that we are not allowed any expensive pre-planning steps before executing a maneuver, as done in [4] for instance. The rationale for this restriction is that such an approach would not be available on a full-size, realistic, deployment of an autonomous vehicle. This is because the environment cannot be entirely known beforehand in an on-road setting (e.g. the location of obstacles and other vehicles are unknown). Even though using an approach involving an expensive pre-planning phase would be possible for the AutoRally system on the tracks that we have available, we avoid this approach since it has clear scalability issues.

1.3 Existing Solutions

There are a variety of existing approaches that we could try and utilize in order to accomplish our goal of autonomous racing. However, each one of them has a deficiency with respect to the desired properties of a controller that we outlined in the previous section. Here, we review the existing approaches that we could use for controlling autonomous vehicles, and argue that no existing method is suitable for the designated task.

Hierarchical Methods

The dominant paradigm in the autonomous driving literature is to use a hierarchical approach to control which splits the control architecture into two layers: a motion planning layer which decides on a kinematically feasible trajectory for the robot, and then a control layer which attempts to follow the path decided on by the motion planner [5]. This was the paradigm utilized in the early days of autonomous driving (during and before the DARPA Challenges) [6, 7, 8, 9], and it is still the dominant method utilized in current autonomous driving systems (e.g. Apollo [10], and Autoware [11]). The appeal behind this architecture is that the motion planning layer can utilize simplified dynamic or kinematic constraints, as opposed to considering a full non-linear vehicle model. Popular constraints utilized in the planning layer are curvature constraints (e.g. [12], [13]), unicycle type models [14], or kinematic bicycle models [15]. In addition to making the problem computationally tractable, the separation of planning and control into separate layers enables the application of familiar techniques from computer science (e.g. graph search or quadratic programming) to be applied, which in turn enables various theoretical results about complexity and completeness to be obtained. Then, once the motion planning layer produces a reference trajectory, the control layer need only follow that reference trajectory and feedback controllers can then be designed, for which results about stability can be obtained.

The issue with this approach is that, in general, the path generated by the motion planning layer will not be dynamically feasible. This means that, regardless of whatever theoretical results about stability the controller possesses, there will be tracking error between the desired and actual trajectory. In the case that a vehicle is safely within its handling limits, this error may not be significant enough to be of concern, however in the case of autonomous racing this error is usually prohibitive. For instance, in early experiments with the AutoRally utilizing a waypoint planner and a tracking controller (we used a pure pursuit controller which is similar to the controller used in [11]), we found that the system was not able to operate close to the limits of handling.

There are modifications to this hierarchical control scheme that can be used for racing. Instead of using a simple constraint model, these methods utilize the non-linear dynamics of the vehicle to solve an optimization problem which generates an optimal racing line [4, 16]. The difference is that these are *much* more computationally intensive than the planners which utilize simplified constraints, which means that they cannot be run online. Instead, the track must be known ahead of time (including all possible obstacles), then the solution is generated for that specific track, and then the solution is executed using a feedback controller. Although this is effective for racing, this approach is not directly translatable to autonomous driving in the real world since it does not run in real-time. Moreover, these methods require precise knowledge of the physical parameters of the vehicle and the road surface, in the case of off-road driving, this knowledge cannot be obtained ahead of time.

To summarize, even though the hierarchical approach to planning and control is the dominant paradigm for autonomous driving, it is not straightforward to apply it to the task of autonomous racing. The methods which run in real-time do not perform well at the limits of handling, and methods which can perform at the limits of handling do not run in real-time. Note that we are not necessarily advocating for the dismissal of this hierarchical approach entirely, and high-level motion plans can still be useful even in the case of autonomous racing (long term planning and strategy for instance). However, for racing, the low-level feedback controller needs to be more capable than pure tracking feedback control laws which have no understanding of the high-level task besides a reference trajectory. This is because constraints can be violated when tracking errors accumulate, unless the low-level controller has some understanding of the task constraints as well.

Constrained Non-Linear MPC

Another possible solution, besides the traditional hierarchical approach, is to utilize a model predictive controller which can consider task related constraints during optimization. The MPC controller could then be directed to follow the track center-line (or a quickly generated racing line) as fast as possible, without violating the track boundaries. This is the approach we take, however we do so with a new type of model predictive controller based on stochastic model predictive controller. There are two previously existing classes of solutions which could be applied in this setting: MPC based on non-linear programming, and MPC based on differential dynamic programming.

In the non-linear programming approach, the dynamics and task related objectives are encoded as constraints, and then a constrained optimization problem is solved which typically seeks to minimize some overarching objective subject to the specified constraints. In theory, this is an excellent approach to autonomous racing. We can use any non-linear dynamics that we like, which means that we will be able to operate close to the limits of handling, and the method naturally handles state and control constraints. In practice, however, the non-linear programming approach comes with some significant drawbacks.

The primary issue is that there are not methods for exactly solving non-linear programs, and the computational cost of approximate methods is prohibitive. In [17] and [18] highly optimized MPC methods based on non-linear programming methods are utilized to control autonomous vehicles with full non-linear vehicle models. However, even for these highly optimized implementations, the maximum planning horizon these methods can achieve is small (10 - 20 timesteps), and the control frequency is large (around 20 Hz). This means that these methods may not be able to react fast enough (or plan long enough) in order to fully utilize the dynamics of the vehicle.

Another issue with these methods is that the high degree of problem specific customization required to achieve real-time performance makes them difficult to work with. This is especially the case for the AutoRally since the off-road vehicle dynamics are significantly different from standard on-road models, and significant customization would be required. It should be noted that this is still an active area of research, and progress is being made. However, at the current time, it is not possible to take an off-the-shelf nonlinear-programming algorithm and use it as an MPC controller for high-speed autonomous racing.

Another possible existing solution that we consider is MPC based on differential dynamic programming (DDP). The original DDP method presented in [19] made quadratic approximations of the costs and dynamics, which enables the value function for an optimal control problem to be approximated. Once the value function is computed, it can be utilized to compute the optimal control. A simpler, more efficient variant of DDP that is commonly used in the current robotics literature is iterative Linear Quadratic Gaussian control (iLQG) [20, 21]. In iLQG a linear approximation of the dynamics is taken instead of a quadratic approximation, although a quadratic approximation is still taken for the cost function.

DDP/iLQG methods have been applied to a vast number of robotic systems, in domains ranging from dynamic walking [22], helicopter control [23], and autonomous ground robots [24]. Model predictive control variants of DDP/iLQG have achieved notable success as well, in particular [25] achieves real time model predictive control for humanoid robots using iLQG. The primary issue with DDP/iLQG is its ability to handle hard constraints. Although there are effective variants for handling control constraints [26], state constraints cannot be directly handled. Instead, state constraints must be encoded using smooth cost functions, which then need to be quadratized, and ensuring that the quadratic approximation is valid (i.e. results in a positive definite Hessian) takes some care [21].

In practice, usually it is not too difficult to tune a smooth cost function which achieves a given desired behavior (i.e. satisfies task constraints). However, in our case using a smooth cost function to encode constraints is not ideal: our problem definition ensures that we will come very close to constraint boundaries, therefore, we would like to be able to very precisely encode them so that appropriate evasive action can be taken if necessary. Moreover, real world autonomous driving requires the consideration of dozens of constraints simultaneously, and tuning soft costs may not be scalable to this case.

Reinforcement Learning

A last possible approach we could take is reinforcement learning. Reinforcement learning can roughly be divided into two groups: model-free reinforcement learning (directly learning a policy), and model-based reinforcement learning (learning a dynamics model and using the model to learn a policy).

Model-free approaches to RL such as policy gradient methods have been successfully applied to many challenging robotics tasks [27, 28, 29, 30, 31, 32, 33, 34]. These approaches typically require an expert demonstration to initialize the learning process, followed by many interactions with the actual robotic system. Unfortunately, model-free approaches often require a large amount of data from these interactions, which limits their applicability. For many robotic systems, it is feasible to allow the robot to fail repeatedly when interacting with the system. Unfortunately, this is not the case with high-speed autonomous driving. Although the AutoRally platform is reasonably robust to crashing, repeated crashes at high speeds will inevitably damage the system.

On the other hand, model-based RL approaches first learn a model of the system and then use the learned model [35, 36] to generate a controller. Interacting with the system can enable a highly accurate dynamics model to be learned, which can enable very precise control. Our method can be thought of as a combination of model-based RL with stochastic model predictive control.

1.4 Thesis Outline

In this thesis, we develop a new algorithmic approach to model predictive control that is applicable to the autonomous racing problem. The method is a sampling-based approach to stochastic (optimal) model predictive control. Unlike the nonlinear programming and DDP/iLQG approaches to model predictive control, sampling based methods do not require making linear or quadratic approximations to the dynamics and costs. Moreover, since sampling-based methods are gradient free, there is considerable flexibility allowed when creating cost functions. These factors make sampling-based methods good candidates to handle non-linear dynamics and constraints.

The difficulty with sampling-based methods, and the reason that they have not been explored in the context of MPC until now, is the computational cost. In order to be effective, a sampling based MPC controller must be able to sample thousands of trajectories in real-time. Until recently this was impossible. However, with the advent of massively parallel graphics processing unit (GPU) computing architectures, it has now become feasible to run sampling based MPC in real-time.

Chapters 3 and 4 deal with the theoretical foundations for sampling based control. Chapter 3 gives an overview of path integral control theory, which shows how the optimal control can be interpreted as a cost-weighted average over sampled paths. Chapter 4 develops an information theoretic approach to control, which shows how cost-weighted averaging relates to an optimal control *distribution*, and discusses the relationship between the optimal control distribution and the optimal control. The theoretical tools from chapters 3 and 4 are applied to develop a novel model predictive control algorithm, Model Predictive Path Integral Control (MPPI), in chapter 5. The performance of the algorithm is analyzed on several simulation systems, as well as the real-world AutoRally vehicle in chapter 6. Portions of the content in these chapters are drawn from [37, 38, 39, 40].

By the end of chapter 6, we have a highly capable algorithmic approach for autonomous racing. However, there are some important robustness issues that the experiments in chapter 6 reveal. We found that MPPI needs a well-tuned cost function in order drive at high speeds and be robust to external disturbances. Chapter 7 analyzes why these robustness issues emerge, and analyzes a preliminary solution. The ideas from chapter 7 are then further developed in chapter 8, which lead to a robust version of the MPPI algorithm that is highly capable and has the desired properties that we previously outlined. The content of chapter 7 corresponds to the material in [41].

Chapter 9 discusses learning and adaptation of neural networks for the AutoRally vehicle dynamics, which is a key component in the MPPI algorithm, and then chapter 10 discusses some future directions for this research. Parts of the content in these chapters corresponds to [42, 43].

CHAPTER 2
MATHEMATICAL BACKGROUND

Throughout this thesis we will rely on some mathematical objects that may not be widely known among roboticists. We give a brief overview of three essential tools here: stochastic differential equations, the stochastic HJB equation, and Radon-Nikodym derivatives. The reader already familiar with these tools can safely skip this chapter.

2.1 Stochastic Differential Equations

In standard optimal control theory, we deal with ordinary differential equations of the form:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t, t).$$

The analogous mathematical objects in stochastic optimal control are *stochastic* differential equations (SDEs). Stochastic differential equations are defined by a deterministic part (called drift) and a stochastic part (called diffusion), they are commonly written in the form:

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t, t)\mathrm{d}t + \mathbf{B}(\mathbf{x}_t, \mathbf{u}_t, t)\mathrm{d}\mathbf{w}, \tag{2.1}$$

This equation is really just shorthand for the integral description of an SDE:

$$\mathbf{x}_t = \mathbf{x}_{t_0} + \int_{t_0}^{t} \mathbf{f}(\mathbf{x}_s, \mathbf{u}_s, s)\mathrm{d}s + \int_{t_0}^{t} \mathbf{B}(\mathbf{x}_s, \mathbf{u}_s, s)\mathrm{d}\mathbf{w}. \tag{2.2}$$

The mathematics required to construct these integrals is quite involved. We do not attempt to give an overview here, and refer the interested reader to [44]. In this thesis, we will not often attempt to directly manipulate or solve SDEs, however we often need to forward simulate the them. Approximate forward simulation of SDEs can be achieved via Euler-

Maruyama integration [45], which is a generalization of Euler integration to the stochastic case. The discretization takes the form:

$$\Delta \mathbf{x} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t, t)\Delta t + \mathbf{B}(\mathbf{x}_t, \mathbf{u}_t, t)\Delta W,$$

The term ΔW is a Gaussian random variable with variance Δt. This means that $\Delta W = \epsilon\sqrt{\Delta t}$, where ϵ is a standard normal random variable. So, the final discretization takes the form:

$$\Delta \mathbf{x} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t, t)\Delta t + \mathbf{B}(\mathbf{x}_t, \mathbf{u}_t, t)\epsilon\sqrt{\Delta t}. \tag{2.3}$$

2.2 Stochastic Hamilton-Jacobi-Bellman Equation

The Hamilton-Jacobi-Bellman (HJB) equation is foundational to modern optimal control theory. The stochastic Hamilton-Jacobi-Bellman equation is equally important in the stochastic optimal control setting, so we review its derivation here. See [46] for a complete introduction to both the deterministic and stochastic HJB equations. We start by considering a stochastic differential equation in the form of Eq. (2.1), a running cost defined by $\mathcal{L}(\mathbf{x}, \mathbf{u}, t)$, and a terminal cost $\phi(\mathbf{x}, t)$. The value function for a stochastic optimal control problem is defined as:

$$V(\mathbf{x}, t) = \min_{\mathbf{u}} \mathbb{E}_{\mathbb{Q}}\left[\phi(\mathbf{x}_T, T) + \int_t^T \mathcal{L}(\mathbf{x}_s, \mathbf{u}_s, s)\mathrm{d}s\right], \tag{2.4}$$

where the expectation is taken with respect to paths generated by the stochastic dynamics starting at the initial condition $\mathbf{x}$. Now let $(\mathbf{x}^*, \mathbf{u}^*)$ denote the optimal state and control values respectively, we can then re-write the right-hand term as:

$$V(\mathbf{x}, t) = \mathbb{E}_{\mathbb{Q}}\left[\phi(\mathbf{x}_T^*, T) + \int_t^T \mathcal{L}(\mathbf{x}_s^*, \mathbf{u}_s^*, s)\mathrm{d}s\right].$$

Note that $\mathbf{x}_t$ must equal $\mathbf{x}$, since it cannot be changed by the dynamics. Therefore it must be the optimal value (since it is the only possible value), and we have:

$$V(\mathbf{x}_t^*, t) = \mathbb{E}_{\mathbb{Q}}\left[\phi(\mathbf{x}_T^*, T) + \int_t^T \mathcal{L}(\mathbf{x}_s^*, \mathbf{u}_s^*, s)\mathrm{d}s\right].$$

Now, using the Bellman optimality principle, we have:

$$V(\mathbf{x}_t^*, t) = \mathbb{E}_{\mathbb{Q}}\big[\mathcal{L}(\mathbf{x}_t^*, \mathbf{u}_t^*, t)\Delta t + V(\mathbf{x}_{t+\Delta t}^*, t + \Delta t)\big],$$

Next, we can re-arrange this equation significantly:

$$\begin{aligned}
V(\mathbf{x}_t^*, t) &= \mathcal{L}(\mathbf{x}_t^*, \mathbf{u}_t^*, t)\Delta t + \mathbb{E}_{\mathbb{Q}}\big[V(\mathbf{x}_{t+\Delta t}^*, t + \Delta t)\big],\\
V(\mathbf{x}_t^*, t) - \mathbb{E}_{\mathbb{Q}}\big[V(\mathbf{x}_{t+\Delta t}^*, t + \Delta t)\big] &= \mathcal{L}(\mathbf{x}_t^*, \mathbf{u}_t^*, t)\Delta t,\\
\mathbb{E}_{\mathbb{Q}}\big[V(\mathbf{x}_{t+\Delta t}^*, t + \Delta t) - V(\mathbf{x}_t^*, t)\big] &= -\mathcal{L}(\mathbf{x}_t^*, \mathbf{u}_t^*, t)\Delta t,
\end{aligned}$$

and then, in the limit as $\Delta t \to 0$, we have:

$$\mathbb{E}_{\mathbb{Q}}[\mathrm{d}V(\mathbf{x}_t^*, t)] = -\mathcal{L}(\mathbf{x}_t^*, \mathbf{u}_t^*, t)\mathrm{d}t \tag{2.5}$$

The term inside the expectation is the derivative of a function with respect to a stochastic process. In order to compute this, we need a generalization of the chain rule known as Ito's Lemma. Ito's lemma states that:

$$\mathrm{d}V(\mathbf{x}_t^*, t) = \left(\frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x}_t^*, \mathbf{u}_t^*) + \frac{1}{2}\mathrm{tr}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}}V_{\mathbf{x}\mathbf{x}}\right)\right)\mathrm{d}t + \frac{\partial V}{\partial \mathbf{x}}\mathbf{B}(\mathbf{x}_t, \mathbf{u}_t)\mathrm{d}\mathbf{w}. \tag{2.6}$$

Taking the expectation of this results in:

$$\mathbb{E}_{\mathbb{Q}}[\mathrm{d}V(\mathbf{x}_t^*, t)] = \left(\frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x}_t^*, \mathbf{u}_t^*) + \frac{1}{2}\mathrm{tr}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}}V_{\mathbf{x}\mathbf{x}}\right)\right)\mathrm{d}t,$$

since the only stochastic term has an expectation of zero with respect to $\mathbb{Q}$. Now we can insert this result back into Eq. (2.5), which results in:

$$
\begin{aligned}
&\left(\frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x}_t^*, \mathbf{u}_t^*) + \frac{1}{2}\mathrm{tr}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}} V_{\mathbf{xx}}\right)\right)\mathrm{d}t = -\mathcal{L}(\mathbf{x}_t^*, \mathbf{u}_t^*, t)\mathrm{d}t,\\
&\frac{\partial V}{\partial t} + \frac{\partial V}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x}_t^*, \mathbf{u}_t^*) + \frac{1}{2}\mathrm{tr}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}} V_{\mathbf{xx}}\right) = -\mathcal{L}(\mathbf{x}_t^*, \mathbf{u}_t^*, t),\\
&\frac{\partial V}{\partial t} = -\mathcal{L}(\mathbf{x}_t^*, \mathbf{u}_t^*, t) - \frac{\partial V}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x}_t^*, \mathbf{u}_t^*) - \frac{1}{2}\mathrm{tr}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}} V_{\mathbf{xx}}\right),\\
&-\frac{\partial V}{\partial t} = \mathcal{L}(\mathbf{x}_t^*, \mathbf{u}_t^*, t) + \frac{\partial V}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x}_t^*, \mathbf{u}_t^*) + \frac{1}{2}\mathrm{tr}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}} V_{\mathbf{xx}}\right),\\
&-\frac{\partial V}{\partial t} = \min_{u}\left(\mathcal{L}(\mathbf{x}, \mathbf{u}, t) + \frac{\partial V}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x}, \mathbf{u}) + \frac{1}{2}\mathrm{tr}\left(\mathbf{B}\mathbf{B}^{\mathrm{T}} V_{\mathbf{xx}}\right)\right) \qquad (2.7)
\end{aligned}
$$

The last equation (Eq. (2.7)) is the stochastic Hamilton-Jacobi-Bellman (stochastic HJB) equation. Note that this is a backwards in time process with the boundary condition $V(\mathbf{x}_T, T) = \phi(\mathbf{x}_T, t)$.

2.3 Radon-Nikodym Theorem

When dealing with probabilities, it is often necessary or convenient to transform an expression originally written with respect to one probability distribution to an expression written with respect to a different distribution. This is especially true when handling expectations. Using the measure-theoretic interpretation of probability, what we need is a tool for relating integrals taken with respect to one probability measure to integrals taken with a different measure. This tool is provided by the Radon-Nikodym theorem, which we review here, see [47] for full detail.

A measure space is defined by a set $\mathcal{X}$, subsets of $\mathcal{X}$ denoted $\mathcal{B}$ which form a σ-algebra of $\mathcal{X}$, and a measure μ. The measure space is then denoted by the tuple $(\mathcal{X}, \mathcal{B}, \mu)$. Now consider another measure defined over the same space, $(\mathcal{X}, \mathcal{B}, \nu)$, the Radon-Nikodym theorem states:

Theorem 1. *Let $(\mathcal{X}, \mathcal{B}, \mu)$ be a measure space, and let ν be a measure defined on $(\mathcal{X}, \mathcal{B})$.*

If, for each set $A \in \mathcal{B}$, the following holds:

$$\int_A \mathrm{d}\mu = 0 \implies \int_A \mathrm{d}\nu = 0. \tag{2.8}$$

Then there is a non-negative measurable function f such that for each set $E \in \mathcal{B}$ we have:

$$\int_E \mathrm{d}\nu = \int_E f \mathrm{d}\mu. \tag{2.9}$$

The function f is unique in the sense that if g is another measurable function which satisfies Eq. (2.9)*, then $f = g$ except for possibly on a set of measure zero with respect to μ.*

The condition in Eq. (2.8) is called *absolute continuity*, and it is denoted as $\nu << \mu$. A common convention, which we follow in this thesis, is to denote the function f as:

$$f = \frac{\mathrm{d}\nu}{\mathrm{d}\mu}, \tag{2.10}$$

and to then refer to $\frac{\mathrm{d}\nu}{\mathrm{d}\mu}$ as the Radon-Nikodym derivative (it has nothing to do with differential calculus). Note that the fact that the Radon-Nikodym is unique is significant, since it enables us to uniquely define measures using an existing measure and a non-negative function. Radon-Nikodym derivatives have a few important properties, which we list here:

i) If $\nu << \mu$ and f is a non-negative measurable function, then:

$$\int f \mathrm{d}\nu = \int f \frac{\mathrm{d}\nu}{\mathrm{d}\mu} \mathrm{d}\mu$$

ii) If $\nu << \mu << \lambda$, then:

$$\frac{\mathrm{d}\nu}{\mathrm{d}\lambda} = \frac{\mathrm{d}\nu}{\mathrm{d}\mu}\frac{\mathrm{d}\mu}{\mathrm{d}\lambda}$$

iii) If $\nu << \mu$ and $\mu << \nu$, then:

$$\frac{\mathrm{d}\nu}{\mathrm{d}\mu} = \left(\frac{\mathrm{d}\mu}{\mathrm{d}\nu}\right)^{-1}$$

The example of a Radon-Nikodym derivative that is most familiar is the case of a probability density function. Consider a probability measure $\mathbb{P}$, and the let $p(x)$ denote its corresponding density. For a set A we often say:

$$\mathbb{P}(A) = \int_A p(x)\mathrm{d}x,$$

Where the right-hand integral is taken with respect to the Lebesgue measure. Equivalently, we could say:

$$\int_A \mathrm{d}\mathbb{P} = \int_A p(x)\mathrm{d}x.$$

Thus, $p(x)$ is playing the role of the Radon-Nikodym derivative between the Lebesgue measure and the probability measure defined by $\mathbb{P}$ (we would denote it as $p(x) = \frac{\mathrm{d}\mathbb{P}}{\mathrm{d}x}$).

Another, less familiar, Radon-Nikodym derivative that we heavily utilize is the Radon-Nikodym derivative between the probability distributions induced by two stochastic differential equations. Consider the case where we have:

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + \mathbf{B}(\mathbf{x}, t)\mathrm{d}\mathbf{w}, \tag{2.11}$$

$$\mathrm{d}\mathbf{x} = \left(\mathbf{f}(\mathbf{x}, t) + \mathbf{G}(\mathbf{x}, t)\mathbf{u}(t)\right)\mathrm{d}t + \mathbf{B}(\mathbf{x}, t)\mathrm{d}\mathbf{w}. \tag{2.12}$$

Each one of these stochastic processes defines a probability measure (on infinite dimensional space). Suppose that we denote $\mathbb{P}$ as the probability measure associated with Eq. (2.11) and let $\mathbb{Q}$ be the measure associated with Eq. (2.12). The the existence and value of $\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}}$ is provided by Girsanov's theorem [48], which provides the following result:

$$\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} = \exp\left(\mathcal{D}(\tau, \mathbf{u}(\cdot))\right). \tag{2.13}$$

The term $\mathcal{D}(\tau, \mathbf{u})$ is the sum of a stochastic integral and a standard integral. The stochastic integral depends on which brownian motion is used to perform the integration. One option is to perform the integration with respect to brownian motion from the probability measure in the numerator ($\mathbb{P}$). We denote this brownian motion as $\mathrm{d}\mathbf{w}^{(0)}$, the $\mathcal{D}(\tau, \mathbf{u})$ term in this case is:

$$\mathcal{D}(\tau, \mathbf{u}(\cdot)) = -\int_0^T \mathbf{u}_t^{\mathrm{T}} \mathbf{G}(\mathbf{x}_t, t)^{\mathrm{T}} \Sigma(\mathbf{x}_t, t)^{-1} \mathbf{B}(\mathbf{x}_t, t) \mathrm{d}\mathbf{w}^{(0)} \\ + \frac{1}{2} \int_0^T \mathbf{u}_t^{\mathrm{T}} \mathbf{G}(\mathbf{x}_t, t)^{\mathrm{T}} \Sigma(\mathbf{x}_t, t)^{-1} \mathbf{G}(\mathbf{x}_t, t) \mathbf{u}_t \mathrm{d}t. \quad (2.14)$$

Alternatively, we can perform the integration with respect to the brownian motion from the probability measure in the denominator ($\mathbb{Q}$), which we denote as $\mathrm{d}\mathbf{w}^{(1)}$. These two brownian motions are related via the expression:

$$\mathbf{B}(\mathbf{x}_t, t)\mathrm{d}\mathbf{w}^{(0)} = \mathbf{G}(\mathbf{x}_t, t)\mathbf{u}_t \mathrm{d}t + \mathbf{B}(\mathbf{x}_t, t)\mathrm{d}\mathbf{w}^{(1)}. \quad (2.15)$$

Now if we insert Eq. (2.14) into Eq. (2.15), we get a similar expression in terms of brownian motion for $\mathbb{Q}$:

$$\mathcal{D}(\tau, \mathbf{u}(\cdot)) = -\int_0^T \mathbf{u}_t^{\mathrm{T}} \mathbf{G}(\mathbf{x}_t, t)^{\mathrm{T}} \Sigma(\mathbf{x}_t, t)^{-1} \mathbf{B}(\mathbf{x}_t, t) \mathrm{d}\mathbf{w}^{(1)} \\ - \frac{1}{2} \int_0^T \mathbf{u}_t^{\mathrm{T}} \mathbf{G}(\mathbf{x}_t, t)^{\mathrm{T}} \Sigma(\mathbf{x}_t, t)^{-1} \mathbf{G}(\mathbf{x}_t, t) \mathbf{u}_t \mathrm{d}t. \quad (2.16)$$

Notice that the only difference is that the sign is flipped on the deterministic portion of the integral. In these equations $\Sigma = \mathbf{B}\mathbf{B}^{\mathrm{T}}$, in many cases $\mathbf{B}$ has more rows than columns, in which case Σ does not have full rank and is not invertible. This is called a degenerate diffusion process, in this case it is possible to pick out a non-degenerate subsystem (see Appendix A), and then use the non-degenerate sub-system to define the Radon-Nikodym derivative.

CHAPTER 3
PATH INTEGRAL CONTROL THEORY

In this chapter we review and slightly generalize the path integral approach to stochastic optimal control. The path integral control framework was originally developed by Kappen in [49, 50], and was first applied to robotics in [34]. In this setting, a path integral is an expectation over all system trajectories (not to be confused with a line integral). The key element of path integral control theory is the usage of the Feynman-Kac equation [51] to transform the stochastic Hamilton-Jacobi-Bellman (stochastic HJB) equation into an expectation over system trajectories. This is the same transformation used in the path integral formulation of quantum mechanics, hence the usage of the name "path integral" to describe the resulting expectation. The transformation of the stochastic HJB equation into a path integral is critical to this thesis because it provides a bridge between the information theoretic framework that we develop later, and which works with expectations and probability distributions, with traditional stochastic optimal control theory that deals in partial differential equations.

Before diving into the derivation of path integral control, it is worth discussing what the result is and is not. The result in path integral control is a formula for the optimal control at the current time and state in terms of paths randomly sampled from the system dynamics. Because there is no explicit parameterization of a feedback control law, and because the solution is calculated using open loop sampling of trajectories, it is tempting to interpret path integral control as a type of open loop trajectory optimization. This is especially the case since the path integral formula for the optimal control at the current time and state can be heuristically applied to future timesteps in order to generate an entire control sequence. However, the interpretation of path integral control as open-loop trajectory optimization is incorrect. Path integral control theory is developed within the stochastic HJB equation framework, which deals with optimal feedback control, as such, the result is an optimal feedback controller. The issue is that the feedback controller is very difficult to implement, since it requires sampling many trajectories in a fast control loop. We will discuss these computational issues in more detail in later sections. In this chapter we are only concerned with the mathematical theory.

We first review the stochastic HJB-PDE, and how it can be used to compute the optimal control in the control-affine case. We then show how to convert the value function into a path integral and finally derive the path integral form of the optimal controls.

3.1 Control-Affine Stochastic Optimal Control

In this section we review how the stochastic HJB equation can be used to generate an optimal feedback control law for a control-affine system. Let $\mathbf{x}_t = \mathbf{x}(t) \in \mathbb{R}^N$ denote the state of a dynamical system at time t, $\mathbf{u}(\mathbf{x}_t, t) \in \mathbb{R}^m$ denotes a control input for the system, $\tau : [t_0, T] \to \mathbb{R}^n$ represents a trajectory of the system, and $\mathrm{d}\mathbf{w} \in \mathbb{R}^p$ is a brownian disturbance. We suppose that the dynamics have the form:

$$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}_t, t) + \mathbf{G}(\mathbf{x}_t, t)\mathbf{u}(\mathbf{x}_t, t)\right] \mathrm{d}t + \mathbf{B}(\mathbf{x}_t, t)\mathrm{d}\mathbf{w}. \tag{3.1}$$

This is called a *control-affine* system since controls affect the system through a linear transformation. Expectations over trajectories taken with respect to Eq. (3.1) are denoted as $\mathbb{E}_{\mathbb{Q}}[\cdot]$. Now, suppose that the cost function for the optimal control problem has a quadratic control cost and an arbitrary state-dependent cost. We denote $\phi(\mathbf{x}_T)$ as a final the terminal cost, and $q(\mathbf{x}_t, t)$ as a running cost which together define the state-dependent cost. The control cost matrix is $\mathbf{R}(\mathbf{x}_t, t)$, and is required to be a positive definite matrix. We allow $\mathbf{R}(\mathbf{x}_t, t)$ to be time and state dependent as long as it remains positive definite for all values of $\mathbf{x}$ and t. The value function $V(\mathbf{x}_t, t)$ for this optimal control problem is then:

$$V(\mathbf{x}_t, t) = \min_{\mathbf{u}} \mathbb{E}_{\mathbb{Q}}\left[\phi(\mathbf{x}_T, T) + \int_t^T \left(q(\mathbf{x}_t, t) + \frac{1}{2}\mathbf{u}(\mathbf{x}_t, t)^{\mathrm{T}}\mathbf{R}(\mathbf{x}_t, t)\mathbf{u}(\mathbf{x}_t, t)\right)\mathrm{d}t\right]. \quad (3.2)$$

The Stochastic Hamilton-Jacobi-Bellman equation and boundary condition for this type of system is then:

$$-\partial_t V = \min_{\mathbf{u}}\left[(\mathbf{f} + \mathbf{G}\mathbf{u})^{\mathrm{T}}\nabla_{\mathbf{x}}V + \frac{1}{2}\mathrm{tr}(\mathbf{B}\mathbf{B}^{\mathrm{T}}\nabla_{\mathbf{xx}}V) + q + \frac{1}{2}\mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u}\right] \quad (3.3)$$
$$V(\mathbf{x}_T, T) = \phi(\mathbf{x}_T).$$

Note that we have dropped the functional arguments for convenience. By noting that the expression inside the parenthesis is convex with respect to the controls $\mathbf{u}$, we can find the minimum by taking the gradient of the expression inside the brackets in Eq. (3.3) with respect to $\mathbf{u}$ and setting it to zero:

$$0 = \nabla_{\mathbf{u}}((\mathbf{f} + \mathbf{G}\mathbf{u})^{\mathrm{T}}\nabla_{\mathbf{x}}V + \frac{1}{2}\mathrm{tr}(\mathbf{B}\mathbf{B}^{\mathrm{T}}\nabla_{\mathbf{xx}}V) + q + \frac{1}{2}\mathbf{u}^{\mathrm{T}}\mathbf{R}\mathbf{u}),$$
$$0 = \mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V + \mathbf{R}\mathbf{u},$$

and so the minimum is found as:

$$\mathbf{u}^* = -\mathbf{R}^{-1}\mathbf{G}^T\nabla_{\mathbf{x}}V. \quad (3.4)$$

Next, we take this equation for $\mathbf{u}^*$ and plug it back into the stochastic HJB equation. After some simplifications, we are left with:

$$-\partial_t V = q + \mathbf{f}^{\mathrm{T}}\nabla_{\mathbf{x}}V - \frac{1}{2}\nabla_{\mathbf{x}}V^{\mathrm{T}}\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V + \frac{1}{2}\mathrm{tr}(\mathbf{B}\mathbf{B}^{\mathrm{T}}\nabla_{\mathbf{xx}}V) \quad (3.5)$$
$$V(\mathbf{x}_T, T) = \phi(\mathbf{x}_T).$$

Equation (3.4) along with Eq. (3.5) provide the necessary mathematical tools to find the stochastic optimal control. In order to compute the optimal control, one needs to solve the partial differential equation defined in Eq. (3.5) to find the value function, and then compute the gradient of the value function and plug the result into Eq. (3.4). Unfortunately, Eq. (3.4) is a, potentially high-dimensional, non-linear partial differential equation, and standard solution techniques suffer from the *curse of dimensionality* in that they scale exponentially with the size of the state space. This motivates the need to find alternative solution methods.

3.2 Transformation to a Path Integral

In this section we describe convert the stochastic HJB into a path integral using the Feynman-Kac formula. The Feynman-Kac formula relates *linear* partial differential equations to expectations. Thus, we first need to transform the stochastic HJB-PDE into a linear PDE, which requires and extra assumption on the relationship between noise and control costs of the system.

Value Function to Desirability Function

The first step in converting Eq. (3.5) into a linear PDE is to make the following exponential transformation of the value function:

$$V(\mathbf{x}, t) = -\lambda \log(\Psi(\mathbf{x}, t)) \tag{3.6}$$

The function Ψ is called the desirability function. In order to continue we need the derivatives of $V(\mathbf{x}, t)$ in terms of $\Psi(\mathbf{x}, t)$. The derivatives with respect to $\mathbf{x}$ and time are clearly:

$$\begin{aligned} \partial_t V &= -\frac{\lambda}{\Psi}\partial_t \Psi, \\ V_{\mathbf{x}} &= -\frac{\lambda}{\Psi}\Psi_{\mathbf{x}}. \end{aligned}$$

Computing a nice expression for the Hessian matrix ($V_{\mathbf{xx}}$) in terms of Ψ is a little more difficult. The ij_{th} entry of $V_{\mathbf{xx}}$ in terms of Ψ is:

$$[\nabla_{\mathbf{xx}} V]_{ij} = -\lambda \left(\frac{\Psi \frac{\partial \Psi}{\partial \mathbf{x}_i \partial \mathbf{x}_j} - \frac{\partial \Psi}{\partial \mathbf{x}_i} \frac{\partial \Psi}{\partial \mathbf{x}_j}}{\Psi^2} \right),$$

with this equation in hand we can compactly write the Hessian matrix as:

$$V_{\mathbf{xx}} = -\frac{\lambda}{\Psi} \Psi_{\mathbf{xx}} + \frac{\lambda}{\Psi^2} \Psi_{\mathbf{x}} \Psi_{\mathbf{x}}^{\mathrm{T}}.$$

The next step in the transformation is to re-write Eq. (3.5) in terms of Ψ in order to get a partial differential equation with respect to the desirability function as opposed to the value function. Inserting the appropriate terms for V, $\partial_t V$, $V_{\mathbf{x}}$, and $V_{\mathbf{xx}}$ yields the following equation:

$$\lambda \frac{\partial_t \Psi}{\Psi} = q - \lambda \frac{\mathbf{f}^{\mathrm{T}} \Psi_{\mathbf{x}}}{\Psi} + \frac{\lambda}{2} \frac{\Psi_{\mathbf{x}}^{\mathrm{T}}}{\Psi} \mathbf{G} \mathbf{R}^{-1} \mathbf{G}^{\mathrm{T}} - \lambda \frac{\Psi_{\mathbf{x}}}{\Psi} - \frac{1}{2} \mathrm{tr}(\mathbf{B}\mathbf{B}^{\mathrm{T}} (\frac{\lambda}{\Psi} \Psi_{\mathbf{xx}})) + \frac{\lambda}{2\Psi^2} \mathrm{tr}(\mathbf{B}\mathbf{B}^{\mathrm{T}} \Psi_{\mathbf{x}} \Psi_{\mathbf{x}}^{\mathrm{T}}),$$

now, using basic properties of the trace, we can re-write the last term as:

$$\mathrm{tr}(\mathbf{B}\mathbf{B}^{\mathrm{T}} \Psi_{\mathbf{x}} \Psi_{\mathbf{x}}^{\mathrm{T}}) = \mathrm{tr}(\Psi_{\mathbf{x}}^{\mathrm{T}} \mathbf{B}\mathbf{B}^{\mathrm{T}} \Psi_{\mathbf{x}}),$$

and $\Psi_{\mathbf{x}}^{\mathrm{T}} \mathbf{B}\mathbf{B}^{\mathrm{T}} \nabla_x \Psi_{\mathbf{x}}$ is a scalar so we can drop the trace operator:

$$\mathrm{tr}(\mathbf{B}\mathbf{B}^{\mathrm{T}} \Psi_{\mathbf{x}} \Psi_{\mathbf{x}}^{\mathrm{T}}) = \Psi_{\mathbf{x}}^{\mathrm{T}} \mathbf{B}\mathbf{B}^{\mathrm{T}} \Psi_{\mathbf{x}}.$$

Substituting this expression for the trace, multiplying by $\frac{\Psi}{\lambda}$, and simplifying yields:

$$\partial_t \Psi = \frac{\Psi}{\lambda} q - \mathbf{f}^{\mathrm{T}} \Psi_{\mathbf{x}} - \underline{\frac{\lambda}{2\Psi} \Psi_{\mathbf{x}}^{\mathrm{T}} \mathbf{G} \mathbf{R}^{-1} \mathbf{G}^{\mathrm{T}} \Psi_{\mathbf{x}}} - \frac{1}{2} \mathrm{tr}(\mathbf{B}\mathbf{B}^{\mathrm{T}} \Psi_{\mathbf{x}} \mathbf{x}) + \underline{\frac{1}{2\Psi} \Psi_{\mathbf{x}}^{\mathrm{T}} \mathbf{B}\mathbf{B}^{\mathrm{T}} \Psi_{\mathbf{x}}}. \quad (3.7)$$

Notice that the two underlined terms are both quadratic forms with respect to $\Psi_{\mathbf{x}}$, the only difference being that one is multiplied with respect to $\mathbf{B}\mathbf{B}^{\mathrm{T}}$ and the other with respect to $\lambda\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}$. If we select $\mathbf{R}$ such that $\mathbf{B}\mathbf{B}^{\mathrm{T}} = \lambda\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}$, then the two underlined terms will cancel and what remains is a linear PDE. This leads to the following definition:

Definition 3.2.1. *If, for all* $\mathbf{x} \in \mathbb{R}^n$ *and all* $t \in [0, T]$*, the following equation:*

$$\mathbf{B}(\mathbf{x}, t)\mathbf{B}(\mathbf{x}, t)^{\mathrm{T}} = \lambda\mathbf{G}(\mathbf{x}, t)\mathbf{R}(\mathbf{x}, t)^{-1}\mathbf{G}(\mathbf{x}, t)^{\mathrm{T}} \tag{3.8}$$

holds, then we say that the Stochastic-HJB equation is **Linearizable***.*

Initially, this may seem like an unjustified mathematical manipulation. However, it actually results in a reasonable control cost matrix since this condition implies that the noise in any given state is inversely proportional the control authority that can be exercised over that state. In other words if a state suffers from high variance it is necessary that the state can be directly actuated, and that the cost of actuation is low. Later on, we will see that there is a more direct mathematical reason (in terms of KL-Divergence) for making this assumption as well. If we assume that the stochastic HJB equation is linearizable, then Eq. (3.7) becomes:

$$\partial_t \Psi(\mathbf{x}_t, t) = \frac{\Psi(\mathbf{x}_t, t)}{\lambda} q(\mathbf{x}_t, t) - \mathbf{f}(\mathbf{x}_t, t)^{\mathrm{T}}\Psi_{\mathbf{x}} - \frac{1}{2}\mathrm{tr}(\mathbf{B}(\mathbf{x}_t, t)\mathbf{B}(\mathbf{x}_t, t)^{\mathrm{T}}\Psi_{\mathbf{x}\mathbf{x}}) \tag{3.9}$$

Which is a linear PDE in terms of Ψ. This equations is sometimes referred to as the Chapman-Kolmogorov backward equation.

Application of the Feynman-Kac Lemma

With the stochastic HJB equation transformed into a linear PDE, we can now apply the Feynman-Kac formula, which gives the solution to the PDE for the initial condition $\mathbf{x}_0$ and time $t = 0$ as:

$$\Psi(\mathbf{x}_{t_0}, 0) = \mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}\int_0^T q(\mathbf{x}, t)\,\mathrm{d}t\right)\Psi(\mathbf{x}_T, T)\right]. \tag{3.10}$$

In this equation the expectation is taken with respect to $\mathbb{P}$, which is the probability of trajectories taken with respect to the uncontrolled system dynamics:

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}_t, t)\mathrm{d}t + \mathbf{B}(\mathbf{x}_t, t)\mathrm{d}\mathbf{w}.$$

Next, recognize that the term $\Psi(\mathbf{x}_T)$ is the transformed terminal cost: $e^{-\frac{1}{\lambda}\phi(\mathbf{x}_T)}$, so we can re-write Eq. (3.10) as:

$$\begin{aligned}\Psi(\mathbf{x}_{t_0}, t_0) &= \mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\right], \\ S(\tau) &= \phi(\mathbf{x}_T) + \int_{t_0}^T q(\mathbf{x}_t, t)\mathrm{d}t.\end{aligned} \tag{3.11}$$

where is $S(\tau)$ is the state-dependent cost-to-go of the trajectory.

There are several interesting properties of this equations. The most obvious property is that all of the terms are *forward* in time processes, whereas the stochastic HJB equation is purely backwards in time. Another property to take note of is that the expectation is with respect to the uncontrolled dynamics of the system, and the control costs have completely disappeared. At first inspection, this appears very strange, since the optimal controls are being computed without ever explicitly referring to the controlled dynamics of the system or the control costs. The reason this is possible, is that the controls are linked to the passive dynamics through the assumption between the noise and controls. The optimization is taking into account the control matrix $\mathbf{G}$ and the control costs, but only implicitly through the diffusion matrix $\mathbf{B}$ and inverse temperature λ.

3.3 Optimal Control for Decomposed Systems

Equation (3.11) provides a path integral form for the value function, but in order to compute the optimal control, we need to compute the gradient of the value function with respect to $\mathbf{x}$. Recall that the optimal control took the form:

$$\mathbf{u}^* = -\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V,$$

if we express this equation in terms of the desirability function instead of the value function we get the equation:

$$\mathbf{u}^* = \lambda\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}\frac{\Psi_{\mathbf{x}}}{\Psi}.$$

Thus we need only compute $\Psi_{\mathbf{x}}$ in order to obtain an expression (in terms of a path integral) for the optimal controls. This can be done analytically in the case that the system is decomposed into indirectly and directly actuated components. In this case the control and diffusion matrices take the form:

$$\mathbf{G}(\mathbf{x}_t, t) = \begin{pmatrix} 0 \\ \mathbf{G}_c(\mathbf{x}_t, t) \end{pmatrix}, \quad \mathbf{B}(\mathbf{x}_t, t) = \begin{pmatrix} 0 \\ \mathbf{B}_c(\mathbf{x}_t, t) \end{pmatrix}.$$

Where $\mathbf{G}_c$ and $\mathbf{B}_c$ are full rank for all $(\mathbf{x}, t)$. This is the case originally considered in [34], the reason this decomposition is necessary is that it enables to probability of a trajectory to be easily written out in terms of a Guassian probability and dirac-delta function. This enables to gradient of Ψ to be taken analytically [34], resulting in the path integral form of the optimal controls:

$$\mathbf{u}^*\mathrm{d}t = \mathbf{R}^{-1}\mathbf{G}_c^{\mathrm{T}}\left(\mathbf{G}_c\mathbf{R}^{-1}\mathbf{G}_c^{\mathrm{T}}\right)^{-1}\mathbf{B}_c\frac{\mathbb{E}_{\tilde{\mathbb{P}}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\mathrm{d}\mathbf{w}\right]}{\mathbb{E}_{\tilde{\mathbb{P}}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\right]}. \tag{3.12}$$

This expression consists of three distinct steps: the rightmost term is the expectation of the $\mathrm{d}\mathbf{w}$ weighted according to the state-cost of a trajectory ($S(\tau)$), this term can be thought of as the *optimal disturbance* for the system. The optimal disturbance is then projected into the directly actuated part of the state-space via multiplication with B_c, this results in an optimal state-adjustment, which is then projected into control space and scaled via multiplication with $\mathbf{R}^{-1}\mathbf{G}_c^{\mathrm{T}}\left(\mathbf{G}_c\mathbf{R}^{-1}\mathbf{G}_c^{\mathrm{T}}\right)^{-1}$.

In the case that $\mathbf{B}_c$ and $\mathbf{G}_c$ are square and invertible, this relationship becomes cleaner. In that case we have:

$$\begin{aligned}\mathbf{u}^*\mathrm{d}t &= \mathbf{R}^{-1}\mathbf{G}_c^{\mathrm{T}}\left(\mathbf{G}_c\mathbf{R}^{-1}\mathbf{G}_c^{\mathrm{T}}\right)^{-1}\mathbf{B}_c\frac{\mathbb{E}_{\tilde{\mathbb{P}}}\big[\exp\big(-\frac{1}{\lambda}S(\tau)\big)\,\mathrm{d}\mathbf{w}\big]}{\mathbb{E}_{\tilde{\mathbb{P}}}\big[\exp\big(-\frac{1}{\lambda}S(\tau)\big)\big]},\\ &= \mathbf{R}^{-1}\mathbf{G}_c^{\mathrm{T}}(\mathbf{G}_c^{\mathrm{T}})^{-1}\mathbf{R}\mathbf{G}_c^{-1}\mathbf{B}_c\frac{\mathbb{E}_{\tilde{\mathbb{P}}}\big[\exp\big(-\frac{1}{\lambda}S(\tau)\big)\,\mathrm{d}\mathbf{w}\big]}{\mathbb{E}_{\tilde{\mathbb{P}}}\big[\exp\big(-\frac{1}{\lambda}S(\tau)\big)\big]},\\ &= \mathbf{G}_c^{-1}\mathbf{B}_c\frac{\mathbb{E}_{\tilde{\mathbb{P}}}\big[\exp\big(-\frac{1}{\lambda}S(\tau)\big)\,\mathrm{d}\mathbf{w}\big]}{\mathbb{E}_{\tilde{\mathbb{P}}}\big[\exp\big(-\frac{1}{\lambda}S(\tau)\big)\big]}.\end{aligned}\tag{3.13}$$

and the transformation of the optimal state adjustment to the optimal control from the optimal state adjustment can be achieved simply via multiplication with $\mathbf{G}^{-1}$.

3.4 Optimal Control for General Systems

In the general case we need to be able to handle systems that are not neatly composed into directly and indirectly actuated components. This situation commonly arises when making a control-affine approximation to a fully non-linear system. To solve this problem we take the following approach:

i) We perform a coordinate transformation on the state space which transforms the $\mathbf{G}(\mathbf{x}_t, t)$ and $\mathbf{B}(\mathbf{x}_t, t)$ matrices into a special form for the initial condition.

ii) We then use the results from the previous section to compute the optimal control for the transformed system.

The first step is to perform a coordinate transformation in order to get full rank submatrices from $\mathbf{G}$ and $\mathbf{B}$. In order to make such a coordinate transformation, we have to make the following condition to hold.

Condition 1. *For each $\mathbf{x} \in \mathbb{R}^n$ and $t \in \mathbb{R}$, there exists an invertible matrix A, possibly dependent on $\mathbf{x}$ and t, such that for $\mathbf{G} = \mathbf{G}(\mathbf{x}_t, t)$ and $\mathbf{B} = \mathbf{B}(\mathbf{x}_t, t)$ the following holds:*

$$\mathbf{G} = A^{-1}\begin{pmatrix} 0 \\ \tilde{\mathbf{G}}_c \end{pmatrix}, \ \ \mathbf{B} = A^{-1}\begin{pmatrix} 0 \\ \tilde{\mathbf{B}}_c \end{pmatrix},$$

with $\tilde{\mathbf{G}}_c$, $\tilde{\mathbf{B}}_c$, and $\tilde{\mathbf{B}}_c\tilde{\mathbf{B}}_c^{\mathrm{T}}$ are full rank matrices.

Note that this is *not* saying that there is a single A matrix which makes this decomposition hold for all $(\mathbf{x}, t)$. It is only saying that for each given $(\mathbf{x}, t)$ the control and diffusion matrices for that specific $\mathbf{x}$ and t can be decomposed this way. The main question now is: *when it possible to make this transformation*? It turns out that the linearizable condition on the stochastic HJB equation is sufficient.

Lemma 1. *If $\mathbf{G}$ is not the zero matrix, then condition 1 is implied by the system being linearizable (Def. 3.2.1) .*

Proof. We can place $\mathbf{G}$ in row-echelon form with an appropriate set of elementary matrix multiplications. Therefore, there exists A such that:

$$A\mathbf{G} = \begin{pmatrix} \mathbf{0} \\ \tilde{\mathbf{G}}_c \end{pmatrix}, \quad \mathbf{0} \in \mathbb{R}^{(n-k)\times m}, \ \ \tilde{\mathbf{G}}_c \in \mathbb{R}^{k\times m}.$$

Where $\tilde{\mathbf{G}}_c$ is a full rank matrix (note that this is true generally for all non-zero matrices). We know that $k > 0$ since $\mathbf{G}$ is not the zero matrix. If $n - k = 0$, then the system is already decomposed and we are done. Otherwise, consider the relationship between the noise and control costs:

$$\mathbf{B}\mathbf{B}^{\mathrm{T}} = \lambda\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}},$$

now applying the coordinate transformation we get:

$$A\mathbf{B}\mathbf{B}^{\mathrm{T}}A^{\mathrm{T}} = \lambda A\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}A^{\mathrm{T}},$$

and, if we explicitly write out $A\mathbf{B}$ as: $A\mathbf{B} = \begin{pmatrix} \mathbf{B}_a \\ \tilde{\mathbf{B}}_c \end{pmatrix}$ then we get:

$$\begin{pmatrix} \mathbf{B}_a\mathbf{B}_a^{\mathrm{T}} & \mathbf{B}_a\tilde{\mathbf{B}}_c^{\mathrm{T}} \\ \tilde{\mathbf{B}}_c\mathbf{B}_a^{\mathrm{T}} & \tilde{\mathbf{B}}_c\tilde{\mathbf{B}}_c^{\mathrm{T}} \end{pmatrix} = \lambda \begin{pmatrix} 0 & 0 \\ 0 & \tilde{\mathbf{G}}_c\mathbf{R}^{-1}\tilde{\mathbf{G}}_c^{\mathrm{T}} \end{pmatrix}.$$

But, $\mathbf{B}_a\mathbf{B}_a^{\mathrm{T}} = 0$ if and only if $\mathbf{B}_a = 0$. This shows that $A\mathbf{B}$ is in the correct form, so we need only show that $\tilde{\mathbf{B}}_c$ and $\tilde{\mathbf{B}}_c\tilde{\mathbf{B}}_c^{\mathrm{T}}$ have full rank. We know that $\tilde{\mathbf{B}}_c\tilde{\mathbf{B}}_c^{\mathrm{T}} \in \mathbb{R}^{k\times k}$, and the rank of $\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}$ must be k since $\mathbf{G}$ has rank k and $\mathbf{R}^{-1}$ is positive definite. Therefore $\tilde{\mathbf{B}}_c\tilde{\mathbf{B}}_c^{\mathrm{T}}$ has full rank, which implies that $\tilde{\mathbf{B}}_c$ has full rank as well. □

Now let $A = A(\mathbf{x}_0, t_0)$ be one such transformation for the initial condition $\mathbf{x}_0$ and time t_0. Then we can apply the coordinate transform to the state:

$$\mathbf{z} = A\mathbf{x},$$

and the corresponding transformation for the dynamics is then:

$$\mathrm{d}\mathbf{z} = \big(A\mathbf{f}(A^{-1}\mathbf{z}_t, t) + A\mathbf{G}(A^{-1}\mathbf{z}_t, t)\mathbf{u}\big)\,\mathrm{d}t + A\mathbf{B}(A^{-1}\mathbf{z}_t, t), \tag{3.14}$$

$$\mathrm{d}\mathbf{z} = \Big[\tilde{\mathbf{f}}(\mathbf{z}_t, t) + \tilde{\mathbf{G}}(\mathbf{z}_t, t)\mathbf{u}\Big]\,\mathrm{d}t + \tilde{\mathbf{B}}(\mathbf{z}_t, t)\mathrm{d}\mathbf{w}. \tag{3.15}$$

It is easy to see that this system is equivalent to our original system. The cost function and control matrix are also transformed to:

$$S_z(\tau) = \phi(A^{-1}\mathbf{z}_T) + \int_{t_0}^{T} q(A^{-1}\mathbf{z}_t, t)\mathrm{d}t, \tag{3.16}$$

$$\mathbf{R}_z = \mathbf{R}(A^{-1}\mathbf{z}). \tag{3.17}$$

It is important to note that, although the transformation matrix A is applied to every state, only the initial state is transformed into the special form. The reason why we need the initial condition, and not other states, in this form is because we only have to compute the derivative of Ψ with respect to the current state. Now, let $\tilde{P}$ be the probability measure corresponding to the transformed dynamics. The desirability function is then:

$$\Psi(\mathbf{z}_0, 0) = \mathbb{E}_{\tilde{\mathbb{P}}}\left[\exp\left(-\frac{1}{\lambda}S_z(\tau)\right)\right],$$

and applying our result from the previous section results in the optimal control for the transformed system taking the form:

$$\mathbf{u}^*\mathrm{d}t = \mathbf{R}_z^{-1}\tilde{\mathbf{G}}_c^{\mathrm{T}}\left(\tilde{\mathbf{G}}_c\mathbf{R}_z^{-1}\tilde{\mathbf{G}}_c^{\mathrm{T}}\right)^{-1}\tilde{\mathbf{B}}_c\frac{\mathbb{E}_{\tilde{\mathbb{P}}}\left[\exp\left(-\frac{1}{\lambda}S_z(\tau)\right)\mathrm{d}\mathbf{w}\right]}{\mathbb{E}_{\tilde{\mathbb{P}}}\left[\exp\left(-\frac{1}{\lambda}S_z(\tau)\right)\right]}. \tag{3.18}$$

Which, once again, is a projection of the optimal disturbance into state-space to get the optimal state adjustment, followed by the projection of the optimal state adjustment into control space to get the optimal control. The optimal disturbance is invariant to the particular state parameterization, since different parameterizations produce the same cost cost and path probability for a given set of disturbances. This means that we can compute the optimal disturbance using the standard parameterization instead of the transformed one, so we have:

$$\mathbf{u}^*\mathrm{d}t = \mathbf{R}_z^{-1}\tilde{\mathbf{G}}_c^{\mathrm{T}}\left(\tilde{\mathbf{G}}_c\mathbf{R}_z^{-1}\tilde{\mathbf{G}}_c^{\mathrm{T}}\right)^{-1}\tilde{\mathbf{B}}_c\frac{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\mathrm{d}\mathbf{w}\right]}{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\right]}. \tag{3.19}$$

where the optimal disturbance is now computed using the optimal disturbance computed with the un-transformed dynamics.

Special Case

Equation (3.18) is the form of the optimal control for any control-affine system that satisfies condition 3.2.1. However, in many cases we are interested in under-actuated systems that take the special form:

$$\mathrm{d}\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)\mathrm{d}t + \mathbf{G}(\mathbf{x}_t, t)\left(\mathbf{u}\mathrm{d}t + \Lambda \mathrm{d}\mathbf{w}\right) \tag{3.20}$$

where Λ is a diagonal matrix. In these cases, disturbances enter the system directly through the control input. Additionally, we assume that the system is under-actuated (more states than controls) and that $\mathbf{G}(\mathbf{x}_t, t) \in \mathbb{R}^{n\times m}$ always has full rank (this means that the rank of $\mathbf{G}(\mathbf{x}_t, t)$ is always m). This amounts to a (very light) controllability assumption on the system, since it ensures that all of the actuators always have at least some effect on the behavior of the system. The optimal control for this type of system will be:

$$\mathbf{u}^*\mathrm{d}t = \mathbf{R}_z^{-1}\tilde{\mathbf{G}}_c^{\mathrm{T}}\left(\tilde{\mathbf{G}}_c\mathbf{R}_z^{-1}\tilde{\mathbf{G}}_c^{\mathrm{T}}\right)^{-1}\left(\tilde{\mathbf{G}}_c\Lambda\right)\frac{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\mathrm{d}\mathbf{w}\right]}{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\right]}.$$

Where we have assumed that the system has been appropriately decomposed into indirectly and directly actuated components using a suitable transformation. Given the assumption on $\mathbf{G}(\mathbf{x}_t, t)$, we know that $\tilde{\mathbf{G}}_c$ is invertible since it will be a square full rank matrix. Therefore, we have:

$$\begin{aligned}\mathbf{u}^*\mathrm{d}t &= \mathbf{R}_z^{-1}\tilde{\mathbf{G}}_c^{\mathrm{T}}(\tilde{\mathbf{G}}_c^{\mathrm{T}})^{-1}\mathbf{R}_z\tilde{\mathbf{G}}_c^{-1}\left(\tilde{\mathbf{G}}_c\Lambda\right)\frac{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\mathrm{d}\mathbf{w}\right]}{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\right]},\\ &= \Lambda\frac{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\mathrm{d}\mathbf{w}\right]}{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\right]}\end{aligned} \tag{3.21}$$

So the optimal control is simply the optimal disturbance scaled by the diagonal matrix Λ. The reason for this is that subspace containing the disturbance inputs and the subspace containing the control inputs are the same, so we can skip the various projection steps that are normally required.

3.5 Approximation for Non-Affine Systems

Path integral control requires the dynamics to be control affine (Eq. (3.1)), although many systems can be accurately described with control-affine dynamics, there are many systems for which this is not the case. In particular, if we use function approximation methods such as neural networks to represent the dynamics, the dynamics will not be control-affine (unless we give the network with a special structure, which can be detrimental to performance). Here we show how to approximate the controls for a non-affine system using path integral control theory. We start by assuming fully non-linear dynamics:

$$\mathrm{d}\mathbf{x} = \mathbf{F}(\mathbf{x}, \mathbf{u})\mathrm{d}t, \tag{3.22}$$

Now suppose that we have a control, $\bar{\mathbf{u}}(t) : [0, T] \to \mathbb{R}^m$, we can then make a control-affine approximation of Eq. (3.22) by linearizing around $\bar{\mathbf{u}}(t)$:

$$\begin{aligned} \mathrm{d}\mathbf{x} &= \mathbf{F}(\mathbf{x}_t, \bar{\mathbf{u}}_t + \delta\mathbf{u}_t)\mathrm{d}t, \\ &\approx \left(\mathbf{F}(\mathbf{x}_t, \bar{\mathbf{u}}_t) + \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{x}_t, \bar{\mathbf{u}}_t)\delta\mathbf{u}_t\right)\mathrm{d}t \end{aligned}$$

Now suppose that $\delta\mathbf{u}$ is not deterministic (i.e. there is noise in the actuators), then instead of $\delta\mathbf{u}\mathrm{d}t$ we have:

$$\delta\mathbf{u}\mathrm{d}t + \Lambda\mathrm{d}\mathbf{w},$$

where Λ defines the magnitude of the noise. This leads to the stochastic differential equation:

$$\mathrm{d}\mathbf{x} = \left(\underbrace{\mathbf{F}(\mathbf{x}_t, \bar{\mathbf{u}}_t)}_{\mathbf{f}(\mathbf{x}_t,t)} + \underbrace{\frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{x}_t, \bar{\mathbf{u}}_t)}_{\mathbf{G}(\mathbf{x}_t,t)} \delta \mathbf{u}_t \right) \mathrm{d}t + \underbrace{\frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{x}_t, \bar{\mathbf{u}}_t)}_{\mathbf{B}(\mathbf{x}_t,t)} \Lambda \mathrm{d}\mathbf{w}, \tag{3.23}$$

as the approximation to Eq. (3.22) with noisy actuators. In the following we will assume that we are in the special case where the Jacobian $\frac{\partial \mathbf{F}}{\partial \mathbf{u}}$ is full rank and has more rows than columns. If the cost function is of the form:

$$\mathcal{L}(\mathbf{x}, \mathbf{u}) = q(\mathbf{x}) + \frac{1}{2}\delta \mathbf{u}^{\mathrm{T}} \mathbf{R} \delta \mathbf{u}, \tag{3.24}$$

then we can apply the results from the previous section and obtain the path integral form of the optimal control. This control cost attempts to minimize the control deviation from $\bar{\mathbf{u}}(t)$, which often-times is sensible. However, another important case is when the cost takes the form:

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \mathbf{u}) &= q(\mathbf{x}) + \frac{1}{2}(\bar{\mathbf{u}} + \delta \mathbf{u})^{\mathrm{T}} \mathbf{R} (\bar{\mathbf{u}} + \delta \mathbf{u}), \\ &= q(\mathbf{x}) + \frac{1}{2}\bar{\mathbf{u}}^{\mathrm{T}} \mathbf{R} \bar{\mathbf{u}} + \bar{\mathbf{u}}^{\mathrm{T}} \mathbf{R} \delta \mathbf{u} + \frac{1}{2} \delta \mathbf{u} \mathbf{R} \delta \mathbf{u}, \end{aligned}$$

This cost minimizes the total combined cost, $\bar{\mathbf{u}} + \delta \mathbf{u}$, and is equally important. However, the addition of the cross-term between state and control: $\bar{\mathbf{u}}^{\mathrm{T}} \mathbf{R} \delta \mathbf{u}$, necessitates some modifications to the path integral equations. We detail the necessary modifications and the end result in the following subsection.

Path Integral Control with Cross-Term

The new cross-term will appear in the stochastic-HJB equation, and since the cross-term has a $\delta \mathbf{u}$ in it, it modifies the expression relating the gradient of the value function to the optimal control. In particular, Eq. (3.4) becomes:

$$\delta\mathbf{u}^* = -\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V - \bar{\mathbf{u}} \tag{3.25}$$

Now, when we insert this back into the stochastic HJB equation, instead of Eq. (3.5), we have:

$$\begin{aligned}-\partial_t V = q &+ \frac{1}{2}\bar{\mathbf{u}}^{\mathrm{T}}\mathbf{R}\bar{\mathbf{u}} + \mathbf{f}^{\mathrm{T}}\nabla_{\mathbf{x}}V - \bar{\mathbf{u}}^{\mathrm{T}}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V - \nabla_{\mathbf{x}}V^{\mathrm{T}}\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V + \frac{1}{2}\mathrm{tr}(\mathbf{B}\mathbf{B}^{\mathrm{T}}\nabla\mathbf{xx}V)\\ &+ \bar{\mathbf{u}}^{\mathrm{T}}\mathbf{R}(-\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V - \bar{\mathbf{u}}) + \frac{1}{2}(-\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V - \bar{\mathbf{u}})^{\mathrm{T}}\mathbf{R}(-\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V - \bar{\mathbf{u}}),\end{aligned}$$

expanding this out further yields:

$$\begin{aligned}-\partial_t V = q &+ \underline{\frac{1}{2}\bar{\mathbf{u}}^{\mathrm{T}}\mathbf{R}\bar{\mathbf{u}}} + \mathbf{f}^{\mathrm{T}}\nabla_{\mathbf{x}}V - \underline{\bar{\mathbf{u}}^{\mathrm{T}}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V} - \underline{\nabla_{\mathbf{x}}V^{\mathrm{T}}\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V} + \frac{1}{2}\mathrm{tr}(\mathbf{B}\mathbf{B}^{\mathrm{T}}\nabla\mathbf{xx}V)\\ &- \underline{\bar{\mathbf{u}}^{\mathrm{T}}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V} - \underline{\bar{\mathbf{u}}^{\mathrm{T}}\mathbf{R}\bar{\mathbf{u}}} + \underline{\frac{1}{2}\bar{\mathbf{u}}^{\mathrm{T}}\mathbf{R}\bar{\mathbf{u}}} + \underline{\bar{\mathbf{u}}^{\mathrm{T}}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V} + \underline{\frac{1}{2}\nabla_{\mathbf{x}}V^{\mathrm{T}}\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V},\end{aligned}$$

notice how the same terms appear in multiple places (the underlined terms). Simplifying this equation by canceling results in:

$$-\partial_t V = q + (\mathbf{f} - \mathbf{G}\bar{\mathbf{u}})\,\nabla_{\mathbf{x}}V - \frac{1}{2}\nabla_{\mathbf{x}}V^{\mathrm{T}}\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{\mathrm{T}}\nabla_{\mathbf{x}}V + \frac{1}{2}\mathrm{tr}(\mathbf{B}\mathbf{B}^{\mathrm{T}}\nabla\mathbf{xx}V) \tag{3.26}$$

Notice that this is the same as the original stochastic HJB equation from Eq. (3.5), but with a different first order term. In the linearization of the stochastic HJB equation, only the 2nd order terms play a role, so we can perform the same exponential transformation (Eq. (3.6) and get another Chapman-Kolmogorov backward equation:

$$\partial_t\Psi = q\frac{\Psi}{\lambda} - (\mathbf{f}(\mathbf{x}_t, t) - \mathbf{G}\bar{\mathbf{u}})^{\mathrm{T}}\,\Psi_{\mathbf{x}} - \frac{1}{2}\mathrm{tr}\left(\mathbf{B}(\mathbf{x}_t, t)\mathbf{B}(\mathbf{x}_t, t)^{\mathrm{T}}\Psi_{\mathbf{xx}}\right).$$

When the Feynman-Kac lemma is applied, the zeroth order term defines the cost, the first order term defines the dynamics, and the 2nd order term defines the magnitude of the noise. So, when we take the Feynman-Kac lemma with Eq. (3.27), we end up with:

$$\Psi(\mathbf{x}_{t_0}, 0) = \mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}\int_0^T q(\mathbf{x}, t)\,\mathrm{d}t\right)\Psi(\mathbf{x}_T, T)\right], \tag{3.27}$$

$$\mathbb{P} : \mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}) - \mathbf{G}(\mathbf{x})\bar{\mathbf{u}} + \mathbf{B}(\mathbf{x})\mathrm{d}\mathbf{w}^{(0)}. \tag{3.28}$$

The introduction of the cross-term results in the expectation in the path integral formula changing, but other than that, everything remains the same. However, it turns out that even this change can be “undone” by switching the expectation using the Radon-Nikodym theorem. Define $\mathbb{Q}$ as the probability measure corresponding to:

$$\mathbb{Q} : \mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}) + \mathbf{B}(\mathbf{x})\mathrm{d}\mathbf{w}^{(1)}, \tag{3.29}$$

then, by using the Radon-Nikodym theorem, we have:

$$\Psi(\mathbf{x}_{t_0}, 0) = \mathbb{E}_{\mathbb{Q}}\left[\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}}\exp\left(-\frac{1}{\lambda}\int_0^T q(\mathbf{x}, t)\,\mathrm{d}t\right)\Psi(\mathbf{x}_T, T)\right]. \tag{3.30}$$

Where $\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}}$ is the Radon-Nikodym derivative between $\mathbb{P}$ and $\mathbb{Q}$. In the following we express $\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}}$ using Brownian motion with respect to $\mathbb{Q}$, so we have:

$$\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} = \exp\left(\mathcal{D}(\tau, \mathbf{u}(\cdot))\right),$$
$$\mathcal{D}(\tau, \mathbf{u}(\cdot)) = -\frac{1}{2}\int_0^T \bar{\mathbf{u}}_t^{\mathrm{T}}\tilde{\mathbf{G}}^{\mathrm{T}}\tilde{\Sigma}^{-1}\tilde{\mathbf{G}}\mathbf{u}_t\mathrm{d}t - \int_0^T \bar{\mathbf{u}}_t^{\mathrm{T}}\tilde{\mathbf{G}}^{\mathrm{T}}\tilde{\Sigma}^{-1}\tilde{\mathbf{B}}\mathrm{d}\mathbf{w}^{(1)}$$

Where $\tilde{\mathbf{B}}$, $\tilde{\mathbf{G}}$, and $\tilde{\Sigma}$ define a non-degenerate sub-system (i.e. they are all full rank). Next, we can combine $\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}}$ with the cost term, and define:

$$\tilde{S}(\tau) = \int_0^T q(\mathbf{x}, t) + \frac{\lambda}{2}\mathbf{u}_t^{\mathrm{T}}\tilde{\mathbf{G}}^{\mathrm{T}}\tilde{\Sigma}^{-1}\tilde{\mathbf{G}}\mathbf{u}_t\,\mathrm{d}t + \lambda\int_0^T \mathbf{u}_t^{\mathrm{T}}\tilde{\mathbf{G}}^{\mathrm{T}}\tilde{\Sigma}^{-1}\tilde{\mathbf{B}}\mathrm{d}\mathbf{w}^{(1)} \tag{3.31}$$

But the linearizable condition for the full system in the stochastic HJB equation implies that the same condition holds for any subsystem (since we're just taking rows out of $\mathbf{G}$ and $\mathbf{B}$). This means that we have:

$$\tilde{\mathbf{B}}\tilde{\mathbf{B}}^{\mathrm{T}} = \lambda \tilde{\mathbf{G}}\mathbf{R}^{-1}\tilde{\mathbf{G}}^{\mathrm{T}}, \tag{3.32}$$

but, this is a non-degenerate subsystem and we have assumed that $\mathbf{G}$ is full rank with at least as many rows as columns. This implies that $\tilde{\mathbf{G}}$ is invertible, so we can re-arrange this to get:

$$\lambda \tilde{\mathbf{G}}^{\mathrm{T}} \Big(\underbrace{\tilde{\mathbf{B}}\tilde{\mathbf{B}}^{\mathrm{T}}}_{=\Sigma} \Big)^{-1} \tilde{\mathbf{G}} = \mathbf{R} \tag{3.33}$$

This means that $\tilde{S}(\tau)$ becomes:

$$\tilde{S}(\tau) = \int_0^T q(\mathbf{x}, t) + \frac{1}{2}\bar{\mathbf{u}}^{\mathrm{T}}\mathbf{R}^{-1}\bar{\mathbf{u}}^{\mathrm{T}} \mathrm{d}t + \int_0^T \bar{\mathbf{u}}^{\mathrm{T}}\mathbf{R}\Lambda \mathrm{d}\mathbf{w}^{(1)} \tag{3.34}$$

And, then we can apply our result from earlier to obtain:

$$\delta\mathbf{u}^* \mathrm{d}t = \Lambda \frac{\mathbb{E}_{\mathbb{Q}}\Big[\exp\Big(-\frac{1}{\lambda}\tilde{S}(\tau)\Big)\,\mathrm{d}\mathbf{w}^{(0)}\Big]}{\mathbb{E}_{\mathbb{Q}}\Big[\exp\Big(-\frac{1}{\lambda}\tilde{S}(\tau)\Big)\Big]} - \bar{\mathbf{u}} \tag{3.35}$$

with the slight modification that $-\bar{\mathbf{u}}$ appears due to the modified stochastic HJB equation. One last issue is that the optimal control here is computed with respect to $\mathrm{d}\mathbf{w}^{(0)}$, which is brownian motion with respect to $\mathbb{P}$. We want this formula with respect to brownian motion for $\mathbb{Q}$, which is related (in the special case here) via the equation:

$$\mathrm{d}\mathbf{w}^{(0)} = \Lambda^{-1}\bar{\mathbf{u}} + \mathrm{d}\mathbf{w}^{(1)} \tag{3.36}$$

Replacing $\mathrm{d}\mathbf{w}^{(0)}$ with the correct brownian motion yields:

$$\delta\mathbf{u}^{*}\mathrm{d}t = \Lambda\frac{\mathbb{E}_{\mathbb{Q}}\left[\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau)\right)\mathrm{d}\mathbf{w}^{(1)}\right]}{\mathbb{E}_{\mathbb{Q}}\left[\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau)\right)\right]}. \tag{3.37}$$

Where the $\bar{\mathbf{u}}$ term has canceled. The conclusion of all of this is that nothing changes in the case of a cross term, except that $\tilde{S}(\tau)$ is modified to include the additional cross term. This is the expected result, however it takes a surprising amount of work to actually show this due to having to "undo" the modifications the the uncontrolled dynamics caused by the change in the stochastic HJB equation.

To summarize the results of this section, if we have a system that does not have control-affine dynamics, we can make a control affine approximation (Eq. (3.23)) using some pre-determined control sequence. Once the dynamics are in a control-affine form, we can sample random trajectories around the pre-determined sequence, and compute the optimal disturbance. If the control cost is defined purely in terms of $\delta\mathbf{u}$, then the computation of the optimal disturbance is the same as in the previous section. If the control cost is defined in terms of the total control $\bar{\mathbf{u}} + \delta\mathbf{u}$, an additional cross term is added into the cost used to compute the optimal disturbance. In both cases, after the optimal disturbance is computed, the optimal control deviation is obtained using the same formula as in the previous section.

CHAPTER 4
INFORMATION THEORETIC FRAMEWORK FOR OPTIMAL CONTROL

In the previous chapter, we saw how path integral control theory enables the transformation of stochastic optimal control problems into problems involving estimation of expectations. In this chapter, we show how a related framework can be developed independently of standard stochastic optimal control theory. The theory developed in this chapter utilizes tools from information theory, namely free-energy and KL-Divergence, as opposed to tools normally associated with stochastic optimal control like partial differential equations.

The connection between path integral control theory with the information theoretic notions of free-energy and KL-Divergence were first made in [52], and later expanded on in [53]. Although these earlier works observed that the free-energy was equivalent to the form of the value function in path integral control, they did realize the more direct connection between the optimal distribution and optimal control that we describe here. This connection was first utilized in [37] to create a slightly generalized version of path integral control, and then further expanded on in [39, 40]. Some related work, [54], also utilizes the notion of an optimal distribution to derive an algorithm based on minimizing KL-Divergence. The difference between our work and [54] is that, in our work, the optimal distribution is derived independently of path integral control, which makes for a more general result.

4.1 Free-Energy Relative Entropy Inequalities

Here we define two mathematical quantities which will be essential for the development of the information theoretic control framework. These quantities are the *free-energy* of a control system, and the *relative entropy* between two control systems. The free-energy terminology comes from the thermodynamic quantity known as the Helmholtz free energy, which takes the same form of the free-energy in our case, although the meaning of the variables differs to reflect the fact that we are describing a control system as opposed to a mechanical system (for instance the cost-to-go replaces energy).

Let $S(\cdot)$ denote the cost to go of the trajectory, V is a random variable that can either denote the trajectory itself or some set of variables which generates a trajectory starting at the initial condition $\mathbf{x}_0$. Let $\mathbb{P}$ denote a probability measure over V. We will formalize the description of V and $\mathbb{P}$ later on. Also, we define $\lambda \in \mathbb{R}^+$ as a positive scalar that is called the inverse temperature of the system.

Definition 4.1.1. *Given a cost-to-go function $S(\cdot)$, probability measure $\mathbb{P}$, initial condition $\mathbf{x}_0$, and inverse temperature λ, the* **free-energy** *of a control system is defined as the quantity:*

$$\mathcal{F}\left(S, \mathbb{P}, \mathbf{x}_0, \lambda\right) = -\lambda \log\left(\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(V)\right)\right]\right).$$

Notice that the term inside the expectation is bounded from above by one, so the total expectation is less than one which implies that the free-energy is always positive. Also, notice the similarity between the form of the free-energy and the form of the value function in the linearizable stochastic HJB case.

The next quantity that we need to define is the relative entropy, which is also known as the KL-Divergence. The relative entropy provides a way to measure distances between probability distributions, although it is technically not a distance metric due to it lacking symmetry.

Definition 4.1.2. *Let $\mathbb{P}$ and $\mathbb{Q}$ be two probability distributions, and suppose that the Radon-Nikodym derivative $\frac{\mathrm{d}\mathbb{Q}}{\mathrm{d}\mathbb{P}}$ exists. Then the* **relative entropy** *of $\mathbb{Q}$ with respect to $\mathbb{P}$ is defined as*

$$\mathbb{KL}\left(\mathbb{Q} \parallel \mathbb{P}\right) = \mathbb{E}_{\mathbb{Q}}\left[\log\left(\frac{\mathrm{d}\mathbb{Q}}{\mathrm{d}\mathbb{P}}\right)\right].$$

With the definitions of free-energy and relative entropy, we can derive a lower bound on the cost-to-go of a stochastic optimal control problem. We start by considering the free-energy of a control system:

$$\mathcal{F} = -\lambda \log \left(\mathbb{E}_{\mathbb{P}} \left[\exp \left(-\frac{1}{\lambda} S(V) \right) \right] \right). \tag{4.1}$$

And now we introduce a second probability measure $\mathbb{Q}$, by using the Radon-Nikodym derivative, we can re-write the expectation to be with respect to $\mathbb{Q}$:

$$\mathcal{F} = -\lambda \log \left(\mathbb{E}_{\mathbb{Q}} \left[\exp \left(-\frac{1}{\lambda} S(V) \right) \frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} \right] \right). \tag{4.2}$$

The interpretation of $\mathbb{P}$ is that it is some natural base measure for system trajectories, for instance it could be the probability of a trajectory under the uncontrolled system dynamics. The measure $\mathbb{Q}$ will correspond to a controlled distribution, which means that by actuating the system the measure $\mathbb{Q}$ will change accordingly. It is also possible to view $\mathbb{P}$ and $\mathbb{Q}$ in a Bayesian framework, where $\mathbb{P}$ corresponds to the prior and $\mathbb{Q}$ plays the role of the posterior.

Next, we apply Jensen's inequality to Eq. (4.2). This yields the following:

$$-\lambda \log \left(\mathbb{E}_{\mathbb{Q}} \left[\exp \left(-\frac{1}{\lambda} S(V) \right) \frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} \right] \right) \leq \underbrace{-\lambda \mathbb{E}_{\mathbb{Q}} \left[\log \left(\exp \left(-\frac{1}{\lambda} S(V) \right) \frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} \right) \right]}_{=\star}, \tag{4.3}$$

and next we can simplify the term on the right-hand side of the inequality to get:

$$\begin{aligned}
\star &= -\lambda \mathbb{E}_{\mathbb{Q}} \left[-\frac{1}{\lambda} S(V) + \log \left(\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} \right) \right], && (4.4) \\
&= \mathbb{E}_{\mathbb{Q}}[S(V)] - \lambda \mathbb{E}_{\mathbb{Q}} \left[\log \left(\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} \right) \right], && (4.5) \\
&= \mathbb{E}_{\mathbb{Q}}[S(V)] + \lambda \mathbb{E}_{\mathbb{Q}} \left[\log \left(\frac{\mathrm{d}\mathbb{Q}}{\mathrm{d}\mathbb{P}} \right) \right], && (4.6) \\
&= \mathbb{E}_{\mathbb{Q}}[S(V)] + \lambda \mathbb{KL} \left(\mathbb{Q} \parallel \mathbb{P} \right). && (4.7)
\end{aligned}$$

We then have the following relationship between the free-energy of a system and the relative entropy between a natural base measure ($\mathbb{P}$) and a controlled measure ($\mathbb{Q}$):

Theorem 2. *Let $\mathbb{Q}$ and $\mathbb{P}$ be two measures, and suppose that both $\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}}$ and $\frac{\mathrm{d}\mathbb{Q}}{\mathrm{d}\mathbb{P}}$ exist, then the following inequality holds:*

$$\mathcal{F}\left(S, \mathbb{P}, \mathbf{x}_0, \lambda\right) \leq \mathbb{E}_{\mathbb{Q}}[S(V)] + \lambda \mathbb{KL}\left(\mathbb{Q} \parallel \mathbb{P}\right)$$

Proof. Equations (4.1) - (4.7). □

On the right-hand side of this inequality, we have the expected cost-to-go under the probability measure $\mathbb{Q}$. Under the interpretation that $\mathbb{Q}$ is defined by a control law, this can be thought of as the expected cost-to-go for a certain controller. In addition to this cost-to-go term, there is the KL-Divergence between $\mathbb{Q}$ and $\mathbb{P}$. This term acts as a control cost, in that it enforces the minimal intervention principle [55] by penalizing deviations from the base distribution $\mathbb{P}$.

4.2 Optimal Distribution

Now that we have a lower bound on the cost of a stochastic optimal control problem, a natural question to ask is if there exists a controlled distribution $\mathbb{Q}^*$ which is "optimal" in the sense of achieving the lower bound. Such a distribution would be optimal in the sense that samples drawn from that distribution would have lower cost, in expectation, than any other distribution. It turns out, that such an optimal distribution does indeed exist, and it comes in a relatively simple form:

Theorem 3. *Let $\mathbb{Q}^*$ be a distribution such that the Radon-Nikodym derivative with respect to $\mathbb{P}$ is equal to:*

$$\frac{\mathrm{d}\mathbb{Q}^*}{\mathrm{d}\mathbb{P}} = \frac{\exp\left(-\frac{1}{\lambda}S(V)\right)}{\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(V)\right)\right]},$$

then $\mathbb{Q}^*$ is the optimal distribution in the sense that it achieves the lower bound in Thm. 2.

Proof. Using the form of the Radon-Nikodym derivative between $\mathbb{Q}^*$ and $\mathbb{P}$ we have:

$$\mathbb{KL}\left(\mathbb{Q}^* \parallel \mathbb{P}\right) = -\frac{1}{\lambda}\mathbb{E}_{\mathbb{Q}^*}[S(V)] - \log\left(\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(V)\right)\right]\right).$$

Substituting this equation into the RHS of the inequality in Thm. 2 results in:

$$\mathcal{F}\left(S, \mathbb{P}, \mathbf{x}_0, \lambda\right) \leq \mathbb{E}_{\mathbb{Q}^*}[S(V)] - \mathbb{E}_{\mathbb{Q}^*}[S(V)] - \lambda\log\left(\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(V)\right)\right]\right),$$

and then, after cancelation, we have:

$$\mathcal{F}\left(S, \mathbb{P}, \mathbf{x}_0, \lambda\right) \leq -\lambda\log\left(\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(V)\right)\right]\right) = \mathcal{F}\left(S, \mathbb{P}, \mathbf{x}_0, \lambda\right),$$

which establishes the optimality of $\mathbb{Q}^*$. □

Note that the key to the construction of the optimal distribution in Thm. 4.2 is the augmentation of the base measure with the cost of the state trajectory. As a consequence, control inputs drawn from the optimal distribution achieve a lower cost, in expectation, than any other control distribution. This observation gives us an equivalence between optimizing a control trajectory and sampling from the optimal distribution. We can exploit this equivalence to develop a novel scheme for optimal control: instead of trying to solve for the optimal control by using the stochastic HJB framework, we can try to push the controlled distribution $\mathbb{Q}$ as close as possible to the optimal distribution $\mathbb{Q}^*$ (see Fig. 4.1). If $\mathbb{Q}$ is aligned with $\mathbb{Q}^*$, then sampling from $\mathbb{Q}$ by applying the resulting control input will result in lower cost trajectories than any other control law.

In this, and the previous section, we have been referring to $\mathbb{P}$ and $\mathbb{Q}$ as abstract measures, and not described precisely what the measures represent or how they may change depending on the control input. In order to develop an optimization approach, we need to give both $\mathbb{P}$ and $\mathbb{Q}$ more concrete definitions. One of the benefits of the information theoretic approach is that the definitions of $\mathbb{P}$ and $\mathbb{Q}$ are very flexible, and can change de-

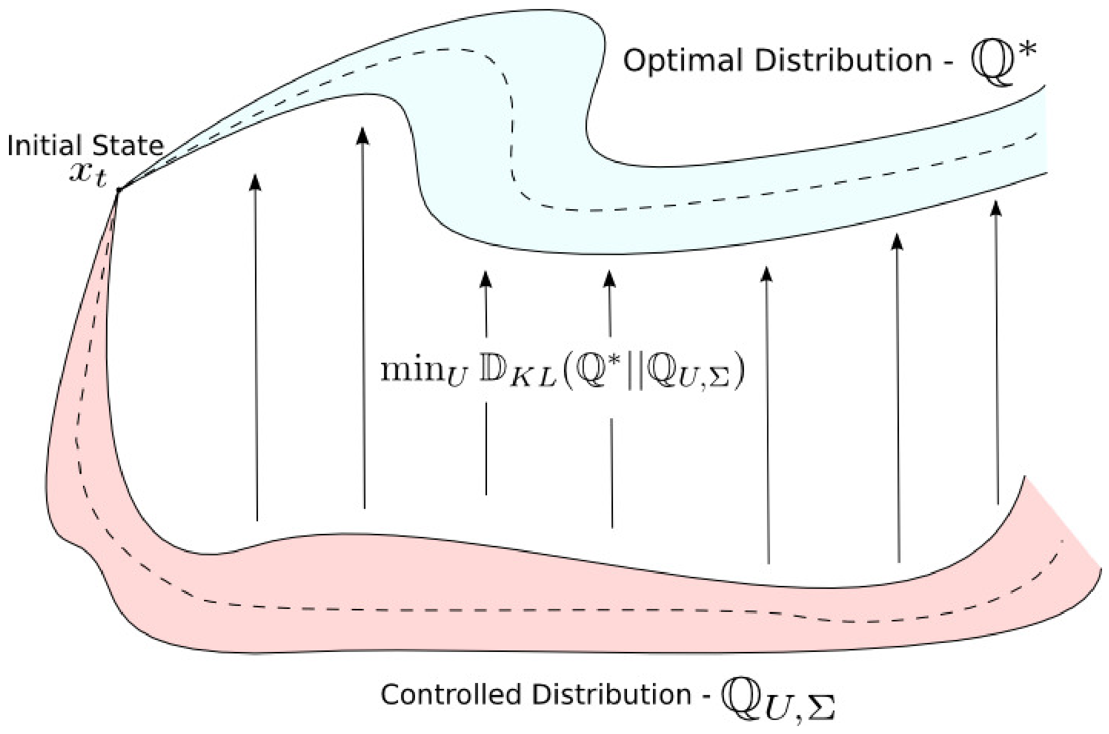

Optimal Distribution - $\mathbb{Q}^*$
Initial State
x_t
$\min_U \mathbb{D}_{KL}(\mathbb{Q}^*||\mathbb{Q}_{U,\Sigma})$
Controlled Distribution - $\mathbb{Q}_{U,\Sigma}$

Figure 4.1: Visualization of the information theoretic control objective of "pushing" the controlled distribution close to the optimal one.

pending on the optimization variables and the control system. In the following sections we parameterize $\mathbb{Q}$ with an open loop sequence of control inputs U and that are disturbed according to some variance Σ. This type of parameterization is denoted as $\mathbb{Q}_{U,\Sigma}$.

4.3 Application to Continuous Time Systems

Here we show how the information-theoretic inequalities and the form of the optimal distribution can be used to develop an optimization scheme for continuous time, control-affine systems. The result ends up being nearly identical to the path integral case, but with some subtle differences. As in the standard path integral control case, we consider stochastic dynamics of the form:

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}_t, t) + \mathbf{G}(\mathbf{x}_t, t)\mathbf{u}_t]\,\mathrm{d}t + \mathbf{B}(\mathbf{x}_t, t)\mathrm{d}\mathbf{w}, \tag{4.8}$$

we will then associate the base distribution $\mathbb{P}$ with the uncontrolled system dynamics. So drawing a sample from $\mathbb{P}$ is equivalent to simulating a trajectory from the system:

$$\mathbb{P} : \mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}_t, t)\mathrm{d}t + \mathbf{B}(\mathbf{x}_t, t)\mathrm{d}\mathbf{w}. \tag{4.9}$$

The controlled distribution will be denoted by $\mathbb{Q}_{U,\Sigma}$, where $U : [t_0, T] \to \mathbb{R}^m$ is a mapping from time to control inputs, and $\Sigma = \mathbf{B}(\mathbf{x}_t, t)\mathbf{B}(\mathbf{x}_t, t)^{\mathrm{T}}$ is the covariance matrix. The control applied to the system at time t is then denoted as $U(t) = \mathbf{u}_t$. Drawing a sample from $\mathbb{Q}_{U,\Sigma}$ is equivalent to simulating a trajectory from the system with inputs given by $U(\cdot)$:

$$\mathbb{Q}_{U,\Sigma} : \mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}_t, t) + \mathbf{G}(\mathbf{x}_t, t)\mathbf{u}_t\right]\mathrm{d}t + \mathbf{B}(\mathbf{x}_t, t)\mathrm{d}\mathbf{w}. \tag{4.10}$$

Next we need to define the cost-to-go. We define a trajectory as a function $[0, T] \to \mathbb{R}^n$, in the previous and subsequent sections, we used V to denote a trajectory. However, in order to keep consistency with previous literature, [37, 52], we will use τ to denote a trajectory here. So $\tau : [0, T] \to \mathbb{R}^n$ denotes a trajectory, and the value of this function at time t is a system state. So we denote $\tau(t) = \mathbf{x}_t$. The cost-to-go of a trajectory τ is then defined as:

$$S(\tau) = \phi(\mathbf{x}_T) + \int_0^T q(\mathbf{x}_t, t)\mathrm{d}t. \tag{4.11}$$

We use the convention that $t = 0$ is always the current time. Notice that the control cost is omitted here, this is because the control cost will naturally appear in the form of the KL-Divergence between the controlled and uncontrolled distribution.

In order to compute the KL-Divergence between $\mathbb{Q}_{U,\Sigma}$ and $\mathbb{P}$, we first need to compute the Radon-Nikodym derivative between the two measures. This is the same Radon-Nikodym derivative as in (2.9). Recall that this can be defined using either the brownian motion with respect to $\mathbb{P}$, or with respect to $\mathbb{Q}$. For now we will use the $\mathcal{D}(\tau, \mathbf{u})$ which depends on $\mathbb{Q}$ (Eq. (2.12)). Applying the KL-Divergence formula on $\frac{\mathrm{d}\mathbb{Q}_{U,\Sigma}}{\mathrm{d}\mathbb{P}}$ yields the following:

$$\mathbb{E}_{\mathbb{Q}_{U,\Sigma}}\left[\frac{\mathrm{d}\mathbb{Q}_{U,\Sigma}}{\mathrm{d}\mathbb{P}}\right] = \mathbb{E}_{\mathbb{Q}_{U,\Sigma}}[-\mathcal{D}(\tau, U)].$$

Where the negative sign is present since we've reversed the order of the Radon-Nikodym derivative. Since the expectation of $\mathrm{d}\mathbf{w}^{(1)}$ with respect to $\mathbb{Q}_{U,\Sigma}$ is zero, this equation simplifies to:

$$\mathbb{KL}\left(\mathbb{Q}_{U,\Sigma} \parallel \mathbb{P}\right) = \mathbb{E}_{\mathbb{Q}(\mathbf{u})}\left[\frac{1}{2}\int_0^T \mathbf{u}_t^{\mathrm{T}}\mathbf{G}(\mathbf{x}_t, t)^{\mathrm{T}}\Sigma(\mathbf{x}_t, t)^{-1}\mathbf{G}(\mathbf{x}_t, t)\mathbf{u}_t \mathrm{d}t\right],$$

and then combining everything results in the following Free-Energy inequality:

$$\mathcal{F}\left(S, \mathbb{P}, \mathbf{x}_0, \lambda\right) \leq \mathbb{E}_{\mathbb{Q}_{U,\Sigma}}\left[S(V) + \frac{\lambda}{2}\int_0^T \mathbf{u}_t^{\mathrm{T}}\mathbf{G}(\mathbf{x}_t, t)^{\mathrm{T}}\Sigma(\mathbf{x}_t, t)^{-1}\mathbf{G}(\mathbf{x}_t, t)\mathbf{u}_t \mathrm{d}t\right]. \quad (4.12)$$

On the right-hand side we have the cost function used in path integral control, and on the left-hand side we have the free-energy (which is the value function) for the system. This means that the optimal distribution from Thm. 3 is optimal with respect to the standard cost used in path integral control. This correspondence is remarkable, the form of the control cost from path integral control theory comes from the desire to linearize the stochastic HJB equations, and although in many cases it is practical, it is not unreasonable to criticize its usage as an ad-hoc mathematical construction. In this framework, however, the control cost appears completely organically from the Radon-Nikodym derivative between the controlled and uncontrolled distribution.

Our goal is to now select U such that $\mathbb{Q}_{U,\Sigma}$ is as close to the optimal distribution as possible. In order to do this we minimize the objective:

$$U^* = \underset{U}{\operatorname{argmin}}\, \mathbb{KL}\left(\mathbb{Q}^* \parallel \mathbb{Q}_{U,\Sigma}\right), \quad (4.13)$$

then applying the definition of relative entropy we have:

$$\mathbb{KL}\left(\mathbb{Q}^* \parallel \mathbb{Q}_{U,\Sigma}\right) = \mathbb{E}_{\mathbb{Q}^*}\left[\log\left(\frac{\mathrm{d}\mathbb{Q}^*}{\mathrm{d}\mathbb{Q}_{U,\Sigma}}\right)\right].$$

In order to optimize this function we need to find an expression for the Radon-Nikodym derivative $\frac{\mathrm{d}\mathbb{Q}^*}{\mathrm{d}\mathbb{Q}_{U,\Sigma}}$. To do this, we apply the chain rule property of Radon-Nikodym derivatives:

$$\frac{\mathrm{d}\mathbb{Q}^*}{\mathrm{d}\mathbb{Q}_{U,\Sigma}} = \frac{\mathrm{d}\mathbb{Q}^*}{\mathrm{d}\mathbb{P}}\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}_{U,\Sigma}}. \tag{4.14}$$

We know what $\frac{\mathrm{d}\mathbb{Q}^*}{\mathrm{d}\mathbb{P}}$ is from the definition of $\mathbb{Q}^*$, and we can use our earlier result from applying Girsanov's theorem to compute $\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}_{U,\Sigma}}$ as:

$$\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}_{U,\Sigma}} = \exp(\mathcal{D}(\tau, U),$$

with $\mathcal{D}(\tau, U)$ defined using Eq. (2.11). This means that we are expressing $\mathcal{D}$ in terms of $\mathrm{d}\mathbf{w}^{(0)}$ which is a Brownian motion with respect to $\mathbb{P}$(i.e. $\mathbb{E}_{\mathbb{P}}\left[\int_0^t \mathrm{d}\mathbf{w}^{(0)}\right] = 0,\ \forall t$). Combining this with Thm. 4.2 and inserting into Eq. (4.14), we get the following expression:

$$\mathbb{KL}\left(\mathbb{Q}^* \parallel \mathbb{Q}_{U,\Sigma}\right) = \mathbb{E}_{\mathbb{Q}^*}\left[\log\left(\frac{\exp(-\frac{1}{\lambda}S(\tau))\exp(\mathcal{D}(\tau, U))}{\mathbb{E}_{\mathbb{P}}\left[\exp(-\frac{1}{\lambda}S(\tau))\right]}\right)\right],$$

Next, using basic rules of logarithms and exponents we can re-write this as:

$$\mathbb{E}_{\mathbb{Q}^*}\left[-\frac{1}{\lambda}S(\tau) + \mathcal{D}(\tau, U) - \log\left(\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(\tau)\right)\right]\right)\right],$$

Since $S(\tau)$, does not depend[1] on the control input, we can remove the first and last terms from the minimization to get:

$$\underset{U}{\operatorname{argmin}}\,\mathbb{KL}\left(\mathbb{Q}^* \parallel \mathbb{Q}_{U,\Sigma}\right) = \underset{U}{\operatorname{argmin}}\,\mathbb{E}_{\mathbb{Q}^*}[\mathcal{D}(\tau, U)]. \tag{4.15}$$

[1]The *probability* of a trajectory does depend on the controls, but the state cost of the trajectory once it is given is not affected by U

The goal is to find the function U which minimizes (4.15). However, since we inevitably apply the control in discrete time it suffices to consider the class of step functions:

$$\mathbf{u}_t = \begin{cases} \mathbf{u}_0 & \text{if} \quad t < \Delta t \\ \vdots & \\ \mathbf{u}_j & \text{if} \; j\Delta t \le t < (j+1)\Delta t \\ \vdots & \\ \mathbf{u}_{N-1} & \text{if} \; (N-1)\Delta t \le t < N\Delta t \end{cases} \tag{4.16}$$

With $j = \{0, 1, \dots N-1\}$ and $T = N\Delta t$. We will slightly abuse notation and let U refer to both the continuous function $U(\cdot)$, and the sequence of control inputs $U = \{\mathbf{u}_0, \mathbf{u}_1, \dots \mathbf{u}_{N-1}\}$ which define the function.

Applying this parameterization to $\mathcal{D}(\tau, U)$ yields:

$$-\sum_{j=0}^{N} \left(\mathbf{u}_j^{\mathrm{T}} \int_{t_j}^{t_{j+1}} \boldsymbol{\mathcal{B}}(\mathbf{x}_t, t) \mathrm{d}\mathbf{w}^{(0)} + \frac{1}{2} \mathbf{u}_j^{\mathrm{T}} \int_{t_j}^{t_{j+1}} \boldsymbol{\mathcal{H}}(\mathbf{x}_t, t) \mathrm{d}t \, \mathbf{u}_j \right) \tag{4.17}$$

Where we have denoted:

i) $\boldsymbol{\mathcal{B}}(\mathbf{x}, t) = \mathbf{G}(\mathbf{x}, t)^{\mathrm{T}} \Sigma(\mathbf{x}, t)^{-1} \mathbf{B}(\mathbf{x}, t)$

ii) $\boldsymbol{\mathcal{H}}(\mathbf{x}, t) = \mathbf{G}(\mathbf{x}, t)^{\mathrm{T}} \Sigma(\mathbf{x}, t)^{-1} \mathbf{G}(\mathbf{x}, t)$

By noting that each $\mathbf{u}_j$ does not depend on the trajectory taken, when we apply the expectation operator to Eq. (4.17), we have $\mathbb{E}_{\mathbb{Q}^*}[\mathcal{D}(\tau, U)]$ equal to the following:

$$\mathbb{E}_{\mathbb{Q}^*}[\mathcal{D}(\tau, U)] = -\sum_{j=0}^{N} \mathbf{u}_j^{\mathrm{T}} \mathbb{E}_{\mathbb{Q}^*}\left[\int_{t_j}^{t_{j+1}} \boldsymbol{\mathcal{B}}(\mathbf{x}_t, t) \mathrm{d}\mathbf{w}^{(0)} \right] \\ + \sum_{j=0}^{N} \frac{1}{2} \mathbf{u}_j^{\mathrm{T}} \mathbb{E}_{\mathbb{Q}^*}\left[\int_{t_j}^{t_{j+1}} \boldsymbol{\mathcal{H}}(\mathbf{x}_t, t) \mathrm{d}t \right] \mathbf{u}_j. \tag{4.18}$$

Now it's easy to see that this is convex with respect to each $\mathbf{u}_j$, so to find $\mathbf{u}_j^*$ we can take the gradient of Eq. (4.18) with respect to $\mathbf{u}_j$, set it equal to zero and solve for $\mathbf{u}_j$. Doing this yields:

$$\mathbf{u}_j^* = \mathbb{E}_{\mathbb{Q}^*}\left[\int_{t_j}^{t_{j+1}} \boldsymbol{\mathcal{H}}(\mathbf{x}_t, t)\mathrm{d}t\right]^{-1} \mathbb{E}_{\mathbb{Q}^*}\left[\int_{t_j}^{t_{j+1}} \boldsymbol{\mathcal{B}}(\mathbf{x}_t, t)\mathrm{d}\mathbf{w}^{(0)}\right],$$

and for small Δt we can make the approximations that:

$$\int_{t_j}^{t_{j+1}} \boldsymbol{\mathcal{H}}(\mathbf{x}_t, t)\mathrm{d}t \approx \boldsymbol{\mathcal{H}}(\mathbf{x}_{t_j}, t_j)\Delta t,$$
$$\int_{t_j}^{t_{j+1}} \boldsymbol{\mathcal{B}}(\mathbf{x}_t, t)\mathrm{d}\mathbf{w}^{(0)} \approx \boldsymbol{\mathcal{B}}(\mathbf{x}_{t_j}, t_j)\int_{t_j}^{t_{j+1}} \mathrm{d}\mathbf{w}^{(0)},$$

which allows us to obtain:

$$\mathbf{u}_j^* = \frac{1}{\Delta t}\mathbb{E}_{\mathbb{Q}^*}\left[\boldsymbol{\mathcal{H}}(\mathbf{x}_{t_j}, t_j)\right]^{-1}\mathbb{E}_{\mathbb{Q}^*}\left[\boldsymbol{\mathcal{B}}(\mathbf{x}_{t_j}, t_j)\int_{t_j}^{t_{j+1}} \mathrm{d}\mathbf{w}^{(0)}\right]. \quad (4.19)$$

This sequence is optimal in the sense that it minimizes the KL-Divergence between the controlled and optimal distribution, which means that samples drawn from the the controlled distribution $\mathbb{Q}_{U,\Sigma}$ will have the same mean as samples drawn from the optimal distribution $\mathbb{Q}^*$. This is a powerful notion of optimality since if the optimal distribution is uni-modal samples drawn from $\mathbb{Q}_{U,\Sigma}$ will look just like samples drawn from $\mathbb{Q}^*$, which achieve lower cost (in expectation) than samples drawn from any other distribution.

We can also interpret this equation by relating it to path integral control. This is the same equation we obtained in the standard path integral case from chapter 3, with the exception that it is valid for $j = \{\mathbf{u}_0, \mathbf{u}_1, \dots \mathbf{u}_{N-1}\}$ as opposed to just $j = 0$. What this means is that the optimal control is the same as the mean of the optimal distribution (projected into control space), and the optimal control is valid even in the case that the optimal distribution is uni-modal. This provides a guarantee that symmetry will collapse. Additionally, it means that all we need to do to compute the optimal control is to maintain an estimate of the optimal distribution (through the KL-Divergence minimization framework) and then continuously execute the mean of the estimated optimal distribution.

Notice that for this derivation we have required the usage of $\Sigma(\mathbf{x})^{-1}$, which may not exist depending on if $\mathbf{B}(\mathbf{x})$ is degenerate or not. The presence of $\Sigma(\mathbf{x})^{-1}$ comes from the form of the Radon-Nikodym derivative of the system, and in certain cases it is possible to define the Radon-Nikodym derivative even if $\mathbf{B}(\mathbf{x})$ is degenerate. This derivation is shown in Appendix A. Once the Radon-Nikodym derivative is defined for degenerate systems it is relatively straight-forward to achieve an analogue of Eq. (4.19). If the system can be decomposed into directly and indirectly actuated components, as in chapter 3, then all that is necessary is to replace $\mathbf{B}(\mathbf{x})$ and $\mathbf{G}(\mathbf{x})$ with $\mathbf{B}_c(\mathbf{x})$ and $\mathbf{G}_c(\mathbf{x})$.

4.4 Application to Discrete Time Systems

Here we show how the information theoretic framework can be applied to discrete time systems. The result in the discrete time case is similar to continuous time, although there are some slight differences. Namely, the result can be applied even in the case of full non-linear dynamics, as opposed to just control-affine, and the projection operation from state to control space is not necessary since noise is inserted directly in control space.

Consider the discrete time stochastic dynamical system:

$$\mathbf{x}_{t+1} = \mathbf{F}(\mathbf{x}_t, \mathbf{v}_t). \tag{4.20}$$

The state vector is denoted $\mathbf{x}_t \in \mathbb{R}^n$, and $\mathbf{u}_t \in \mathbb{R}^m$ is the commanded control input to the system. We assume that if we apply an input $\mathbf{u}_t$ then the actual input will be

$$\mathbf{v}_t \sim \mathcal{N}(\mathbf{u}_t, \Sigma).$$

This is a reasonable noise assumption for many robotic systems where the commanded input has to pass through a lower level controller. A prototypical example is the steering and throttle inputs for a car which are then used as set-point targets for low level servomotor controllers. We define:

$$V = (\mathbf{v}_0, \mathbf{v}_1, \dots \mathbf{v}_{T-1}),$$

as a sequence of inputs over some number of timesteps T. This sequence is itself a random variable whose distribution can be changed by modifying the control input sequence:

$$U = (\mathbf{u}_0, \mathbf{u}_1 \dots \mathbf{u}_{T-1}).$$

Once again, we need to define $\mathbb{P}$, and $\mathbb{Q}_{U,\Sigma}$, the advantage that we have in the discrete time setting is that we can explicitly write down probability density functions[2] for $\mathbb{P}$ and $\mathbb{Q}_{U,\Sigma}$. The density function for $\mathbb{P}$ is denoted as $p(V)$ and takes the form:

$$p(V) = \prod_{t=0}^{T-1} \frac{1}{((2\pi)^m |\Sigma|)^{\frac{1}{2}}} \exp\left(-\frac{1}{2}\mathbf{v}_t^{\mathrm{T}} \Sigma^{-1} \mathbf{v}_t\right). \tag{4.21}$$

Whereas the density function for $\mathbb{Q}_{U,\Sigma}$ is denoted as $q(V|U,\Sigma)$ and takes the form:

$$q(V|U,\Sigma) = \prod_{t=0}^{T-1} \frac{1}{((2\pi)^m |\Sigma|)^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{v}_t - \mathbf{u}_t)^{\mathrm{T}} \Sigma^{-1} (\mathbf{v}_t - \mathbf{u}_t)\right). \tag{4.22}$$

We now have definitions for the controlled and base distributions. The next step, as in the continuous time case, is to define the cost-to-go function. Given an initial condition $\mathbf{x}_0$ and an input sequence V, we can uniquely determine the corresponding system trajectory by recursively applying $\mathbf{F}$. We thus have a mapping from inputs V to trajectories, we denote a trajectory sequence as τ. Now, let $\Omega_\tau \subset \mathbb{R}^n \times \{0, \dots T-1\}$ be the space of all possible trajectories and define:

$$\mathcal{G}_{\mathbf{x}_0} : \Omega_V \to \Omega_\tau,$$

[2]Note that probability density functions are Radon-Nikodym derivatives between a probability measure and the Lebesgue measure. The reason we cannot write down density functions in the continuous time case is because the infinite dimensional Lebesgue measure does not exist.

as the function which maps input sequences to trajectories for the given initial condition $\mathbf{x}_0$. Consider a state-dependent cost function for trajectories:

$$C(\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_T) = \phi(\mathbf{x}_T) + \sum_{t=1}^{T-1} q(\mathbf{x}_t),$$

where $\phi(\cdot)$ is a terminal cost and $q(\cdot)$ is an instantaneous state cost. We can use this to create a cost function over input sequences by defining $S : \Omega_V \to \mathbb{R}^+$ as the composition:

$$S = C \circ \mathcal{G}_{\mathbf{x}_0}. \tag{4.23}$$

Next, we have to compute the KL-Divergence between $\mathbb{Q}_{U,\Sigma}$ and $\mathbb{P}$. Unlike the continuous time case, where we had to rely on Girsanov's theorem, in the discrete time case we can simply compute:

$$\mathbb{KL}\left(\mathbb{Q}_{U,\Sigma} \parallel \mathbb{P}\right) = \mathbb{E}_{\mathbb{Q}_{U,\Sigma}}\left[\log\left(\frac{q(V|U,\Sigma)}{p(V)}\right)\right] = \frac{1}{2}\sum_{t=0}^{T-1} \mathbf{u}_t^{\mathrm{T}} \Sigma^{-1} \mathbf{u}_t,$$

and then the free-energy lower bound becomes:

$$\mathcal{F}\left(S, \mathbb{P}, \mathbf{x}_0, \lambda\right) \leq \mathbb{E}_{\mathbb{Q}_{U,\Sigma}}[S(V, \mathbf{x}_0)] + \frac{\lambda}{2}\sum_{t=0}^{T-1} \mathbf{u}_t^{\mathrm{T}} \Sigma^{-1} \mathbf{u}_t. \tag{4.24}$$

So, once again, the free-energy serves as a lower bound on the expected state-cost of a trajectory plus a quadratic control cost.

Our goal is to derive an expression the control sequence which pushes the controlled distribution as close as possible to the optimal distribution. Using the definition of KL divergence, we have $\mathbb{KL}\left(\mathbb{Q}^* \parallel \mathbb{Q}_{U,\Sigma}\right)$ equal to:

$$\begin{aligned}
&\int_{\Omega_V} q^*(V) \log\left(\frac{q^*(V)}{q(V|U,\Sigma)}\right) \mathrm{d}V, \\
=&\int_{\Omega_V} q^*(V) \log\left(\frac{q^*(V)}{p(V)} \frac{p(V)}{q(V|U,\Sigma)}\right) \mathrm{d}V, \\
=&\int_{\Omega_V} \underbrace{q^*(V) \log\left(\frac{q^*(V)}{p(V)}\right)}_{\text{Independent of } U} - q^*(V) \log\left(\frac{q(V|U,\Sigma)}{\mathbf{p}(V)}\right) \mathrm{d}V.
\end{aligned}$$

Neither the optimal distribution nor the distribution corresponding to the uncontrolled dynamics depends on the control input that we apply. Therefore, the left-most term does not depend on U and can be removed, which leaves us with:

$$U^* = \operatorname*{argmax}_U \int_{\Omega_V} q^*(V) \log\left(\frac{q(V|U,\Sigma)}{p(V)}\right) \mathrm{d}V \tag{4.25}$$

Note that we have flipped the sign and changed the minimization to a maximization. It is easy to show that:

$$\frac{q(V|U,\Sigma)}{p(V)} = \exp\left(\sum_{t=0}^{T-1} -\frac{1}{2}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t\right),$$

inserting this into Eq. (4.25) yields:

$$U^* = \operatorname*{argmax}_U \int q^*(V) \left(\sum_{t=0}^{T-1} -\frac{1}{2}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t\right) \mathrm{d}V. \tag{4.26}$$

Next, we can integrate out the probability in the first term. Doing this results in:

$$U^* = \operatorname*{argmax}_U \left[\sum_{t=0}^{T-1} \left(-\frac{1}{2}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + \mathbf{u}_t^{\mathrm{T}} \int q^*(V)\Sigma^{-1}\mathbf{v}_t \mathrm{d}V\right)\right].$$

This is concave with respect to each $\mathbf{u}_t$, so we can find the maximum with, respect to each

$\mathbf{u}_t$, by taking the gradient and setting it to zero. Doing this yields:

$$\mathbf{u}_t^* = \int q^*(V)\mathbf{v}_t \mathrm{d}V \tag{4.27}$$

Which states that the optimal control, in the sense of minimizing the KL-Divergence, is the mean of optimal distribution. As in the continuous time case, this has its own interpretation of optimality if the optimal distribution is uni-modal: executing the mean of the optimal distribution means that the disturbed input will look like it was generated by the optimal distribution, which has lower cost in expectation than any other distribution. This expression can be made a little more intuitive by re-writing $q^*(V) = \frac{q^*(V)}{p(V)}p(V)$. In that case we have:

$$\mathbf{u}_t^* = \int \frac{q^*(V)}{p(V)} p(V)\mathbf{v}_t \mathrm{d}V = \frac{1}{\eta}\int \exp\left(-\frac{1}{\lambda}S(V)\right) p(V)\mathbf{v}_t \mathrm{d}V, \tag{4.28}$$

with the normalizing term $\eta = \int \exp\left(-\frac{1}{\lambda}S(V)\right)\mathbf{p}(V)\mathrm{d}V$. This equation states that the optimal control is simply a cost weighted average over sampled control inputs. The weighting in this case resembles the output of a Bayes rule, with $\mathbf{p}(V)$ playing the role of the prior and $\exp(-\frac{1}{\lambda}S(V))$ acting as the observation.

Connection to Path Integral Control

Unlike in the continuous time case, there is not a direct analog to path integral control, which means there is not a direct relationship between this equation and the stochastic optimal control. However, if the discrete time system has been obtained via discretization of a continuous time system, then we can still make an approximate relationship. Consider the system:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{F}(\mathbf{x}_t, \mathbf{u}_t + \epsilon_t)\Delta t, \tag{4.29}$$

where $\epsilon \sim \mathcal{N}\left(0, \frac{\Sigma}{\Delta t}\right)$. Now, for a given control sequence $\bar{U} = \{\bar{\mathbf{u}}_0, \bar{\mathbf{u}}_1, \ldots \bar{\mathbf{u}}_{T-1}\}$, we can approximate this system as:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \left(\mathbf{F}(\mathbf{x}_t, \bar{\mathbf{u}}_t) + \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{x}_t, \bar{\mathbf{u}}_t)\delta \mathbf{u}_t\right)\Delta t + \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{x}_t, \bar{\mathbf{u}}_t)\epsilon_t \Delta t.$$

But, if instead of drawing $\epsilon \sim \mathcal{N}(0, \frac{\Sigma}{\Delta t})$, we draw $\mathbf{w} \sim \mathcal{N}(0, I)$, we can re-write this as:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \left(\mathbf{F}(\mathbf{x}_t, \bar{\mathbf{u}}_t) + \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{x}_t, \bar{\mathbf{u}}_t)\delta \mathbf{u}_t\right)\Delta t + \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{x}_t, \bar{\mathbf{u}}_t)\Lambda \frac{\mathbf{w}}{\sqrt{\Delta t}}\Delta t,$$

where $\Lambda = \sqrt{\Sigma}$. We then have:

$$\Delta \mathbf{x}_t = \left(\mathbf{F}(\mathbf{x}_t, \bar{\mathbf{u}}_t) + \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{x}_t, \bar{\mathbf{u}}_t)\delta \mathbf{u}\right)\Delta t + \left(\frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{x}_t, \bar{\mathbf{u}}_t)\Lambda\right)\mathbf{w}\sqrt{\Delta t}. \qquad (4.30)$$

Next, we can set $\mathbf{f}(\mathbf{x}_t, t) = \mathbf{F}(\mathbf{x}_t, \bar{\mathbf{u}}_t)$, $\mathbf{G}(\mathbf{x}_t, t) = \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{x}_t, \bar{\mathbf{u}}_t)$, and $\mathbf{B}(\mathbf{x}_t, t) = \frac{\partial \mathbf{F}}{\partial \mathbf{u}}(\mathbf{x}_t, \bar{\mathbf{u}}_t)\Lambda$, and taking the limit as $\Delta t \to 0$ puts it into a control-affine SDE form:

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

and the optimal $\delta \mathbf{u}$ for the continuous time system is obtained via Eq. (3.37). Recall that this takes the form:

$$\begin{aligned}
\mathbf{u}^* \mathrm{d}t &= \bar{\mathbf{u}}\mathrm{d}t + \Lambda \frac{\mathbb{E}_{\mathbb{Q}}\left[\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau)\right)\mathrm{d}\mathbf{w}\right]}{\mathbb{E}_{\mathbb{Q}}\left[\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau)\right)\right]}, \\
\tilde{S}(\tau) &= \phi(\mathbf{x}_T, T) + \int_0^T q(\mathbf{x}, t) + \frac{1}{2}\bar{\mathbf{u}}^{\mathrm{T}}\mathbf{R}^{-1}\bar{\mathbf{u}}^{\mathrm{T}}\mathrm{d}t + \int_0^T \bar{\mathbf{u}}^{\mathrm{T}}\mathbf{R}\Lambda \mathrm{d}\mathbf{w}, \\
\mathbf{R} &= \lambda \tilde{\mathbf{G}}^{\mathrm{T}}\left(\tilde{\mathbf{B}}\tilde{\mathbf{B}}^{\mathrm{T}}\right)^{-1}\tilde{\mathbf{G}} = \lambda\left(\Lambda\Lambda^{\mathrm{T}}\right)^{-1} = \lambda\Sigma^{-1}.
\end{aligned}$$

In reality, we must discretize these equations in order to implement them on a digital computer. So we will have:

$$\mathbf{u}^*\Delta t = \bar{\mathbf{u}}\Delta t + \Lambda \frac{\mathbb{E}_{\mathbb{Q}}\left[\exp\left(-\frac{1}{\lambda}\tilde{S}_{\Delta t}\right)\mathbf{w}\sqrt{\Delta t}\right]}{\mathbb{E}_{\mathbb{Q}}\left[\exp\left(-\frac{1}{\lambda}\tilde{S}_{\Delta t}\right)\right]},$$

$$\Rightarrow \mathbf{u}^* = \bar{\mathbf{u}} + \Lambda \frac{\mathbb{E}_{\mathbb{Q}}\left[\exp\left(-\frac{1}{\lambda}\tilde{S}_{\Delta t}\right)\frac{\mathbf{w}}{\sqrt{\Delta t}}\right]}{\mathbb{E}_{\mathbb{Q}}\left[\exp\left(-\frac{1}{\lambda}\tilde{S}_{\Delta t}\right)\right]}. \tag{4.31}$$

where $\tilde{S}_{\Delta t}$ is the discrete time approximation to $\tilde{S}(\tau)$.

$$\tilde{S}_{\Delta t} = \phi(\mathbf{x}_T, T) + \sum_{t=0}^{T-1}\left(q(\mathbf{x}, t) + \frac{\lambda}{2}\bar{\mathbf{u}}^{\mathrm{T}}\Sigma^{-1}\bar{\mathbf{u}}^{\mathrm{T}}\right)\Delta t + \lambda\sum_{t=0}^{T-1}\bar{\mathbf{u}}^{\mathrm{T}}\Sigma^{-1}\Lambda\mathbf{w}\sqrt{\Delta t}. \tag{4.32}$$

If we were to use the continuous time path integral equations to compute the optimal control, Eq. (4.31) is what we would have to implement. We want to know how this relates to the information theoretic optimal control (Eq. (4.28)) at time $t = 0$ (i.e. we want to compare the mean of the optimal distribution at $t = 0$ to the optimal control). Recall that this is:

$$\mathbf{u}_t^* = \frac{\int \exp\left(-\frac{1}{\lambda}S(V)\right)p(V)\mathbf{v}_t \mathrm{d}V}{\int \exp\left(-\frac{1}{\lambda}S(V)\right)p(V)\mathrm{d}V}, \tag{4.33}$$

$$S(V) = \phi(\mathbf{x}_T, T) + \sum_{t=0}^{T} q(\mathbf{x}_t)\Delta t.$$

Next, we define the discrete time analog of $\mathbb{Q}$ as:

$$q(V|\bar{U}, \Sigma) = \prod_{t=0}^{T-1}\frac{1}{((2\pi)^m|\Sigma|)^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\mathbf{v}_t - \bar{\mathbf{u}}_t)^{\mathrm{T}}\Sigma^{-1}(\mathbf{v}_t - \bar{\mathbf{u}}_t)\right),$$

and then using the Radon-Nikodym deriative, we can express Eq. (4.33) as:

$$\begin{aligned}
\mathbf{u}^* &= \frac{1}{\eta}\int \exp\left(-\frac{1}{\lambda}S(V)\right)\frac{p(V)}{q(V|U,\Sigma)}q(V|U,\Sigma)\mathbf{v}_t \mathrm{d}V, \\
&= \frac{1}{\eta}\mathbb{E}_q\left[\exp\left(-\frac{1}{\lambda}S(V)\right)\frac{p(V)}{q(V|U,\Sigma)}\right].
\end{aligned}$$

The term inside the expectation works out to:

$$\exp\left(-\frac{1}{\lambda}\left[\phi(\mathbf{x}_T,T)+\sum_{t=0}^{T-1}q(\mathbf{x}_t)\Delta t\right]\right)\exp\left(\sum_{t=0}^{T-1}\left[\frac{1}{2}\bar{\mathbf{u}}_t^{\mathrm{T}}\Sigma^{-1}\bar{\mathbf{u}}_t-\bar{\mathbf{u}}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t\right]\Delta t\right).$$

Now, we can replace[3] $V=\{\mathbf{v}_0,\mathbf{v}_1,\dots\mathbf{v}_{T-1}\}$ with $\bar{U}+\mathcal{E}=\{\bar{\mathbf{u}}_0+\epsilon_0,\bar{\mathbf{u}}_1+\epsilon_1,\dots\}$ where ϵ is zero mean noise with variance $\frac{\Sigma}{\Delta t}$. Doing this yields:

$$\exp\left(-\frac{1}{\lambda}\left[\phi(\mathbf{x}_T,T)+\sum_{t=0}^{T-1}q(\mathbf{x}_t)\Delta t\right]\right)\exp\left(\sum_{t=0}^{T-1}\left[-\frac{1}{2}\bar{\mathbf{u}}_t^{\mathrm{T}}\Sigma^{-1}\bar{\mathbf{u}}_t-\bar{\mathbf{u}}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right]\Delta t\right),$$

then combining the exponentials yields:

$$\begin{aligned}
&\exp\left(-\frac{1}{\lambda}\left[\phi(\mathbf{x}_T,T)+\sum_{t=0}^{T-1}\left(q(\mathbf{x}_t)+\frac{\lambda}{2}\bar{\mathbf{u}}_t^{\mathrm{T}}\Sigma^{-1}\bar{\mathbf{u}}_t+\lambda\bar{\mathbf{u}}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right)\Delta t\right]\right), \\
&=\exp\left(-\frac{1}{\lambda}\left[\phi(\mathbf{x}_T,T)+\sum_{t=0}^{T-1}\left(q(\mathbf{x}_t)+\frac{\lambda}{2}\bar{\mathbf{u}}_t^{\mathrm{T}}\Sigma^{-1}\bar{\mathbf{u}}_t\right)\Delta t+\lambda\sum_{t=0}^{T-1}\bar{\mathbf{u}}_t\Sigma^{-1}\Lambda\mathbf{w}\sqrt{\Delta t}\right]\right), \\
&=\exp\left(-\frac{1}{\lambda}\tilde{S}_{\Delta t}\right) \qquad (4.34)
\end{aligned}$$

So, the formula for the optimal control that we obtained in the information theoretic case

[3]Note that this is the discrete time analog of switching between $\mathrm{d}\mathbf{w}^{(0)}$ and $\mathrm{d}\mathbf{w}^{(1)}$ that we had to do frequently in the continuous time case.

is equivalent to:

$$\mathbf{u}_0^* = \frac{\mathbb{E}_q\left[\exp\left(-\frac{1}{\lambda}\tilde{S}_{\Delta t}\right)(\bar{\mathbf{u}}_0 + \epsilon_0)\right]}{\mathbb{E}_q\left[\exp\left(-\frac{1}{\lambda}\tilde{S}_{\Delta t}\right)\right]},$$
$$= \bar{\mathbf{u}}_0 + \frac{\mathbb{E}_q\left[\exp\left(-\frac{1}{\lambda}\tilde{S}_{\Delta t}\right)\epsilon_0\right]}{\mathbb{E}_q\left[\exp\left(-\frac{1}{\lambda}\tilde{S}_{\Delta t}\right)\right]}.$$

The final step, is to write ϵ_0 in terms of $\mathbf{w}$ (standard normal gaussian noise), which yields:

$$\mathbf{u}_0^* = \bar{\mathbf{u}} + \Lambda \frac{\mathbb{E}_q\left[\exp\left(-\frac{1}{\lambda}\tilde{S}_{\Delta t}\right)\frac{\mathbf{w}}{\sqrt{\Delta t}}\right]}{\mathbb{E}_q\left[\exp\left(-\frac{1}{\lambda}\tilde{S}_{\Delta t}\right)\right]}. \tag{4.35}$$

It should be clear that this is approximately equivalent to the continuous time optimal control, in the sense that if the continuous time path integral equations are discretized (Eq. (4.31)), then the result is nearly identical to the information theoretic optimal control obtained in Eq. (4.35). The only difference is that the expectation in the discretized path integral case is taken with the control-affine approximation (Eq. (4.30)) to the non-linear system, whereas the the information theoretic formula is computed with respect to the fully non-linear dynamics (Eq. (4.29). If the control-affine approximation is valid, we should expect that sampling from the control-affine approximation would yield similar results to sampling from the fully non-linear dynamics. We can therefore conclude, that the mean of the optimal distribution at the current time instance is approximately the optimal control for the best control-affine approximation to the non-linear dynamics.

4.5 Related Work on Sampling Based Control

In the previous sections we have developed a sampling based optimization framework based on free-energy and relative entropy inequalities, and detailed the relationship between this framework and path integral control. In this section we detail how the information theoretic framework compares with other sampling based optimization methods that are popular in robotics.

There are a number of alternative approaches that could be used to derive a similar update law to Eq. (4.27). The policy gradient theorem, reward weighted regression [56, 57], and black-box stochastic optimization methods [58] could be used, by choosing an appropriate control parameterization and applying an exponential cost transformation, to create a similar result. However, in those frameworks, applying an exponential transform to the cost would be a heuristic without a solid theoretical grounding. In our approach, the exponential transform appears naturally through the relationship between free-energy and the cost of a standard stochastic optimal control problem. Furthermore, defining the free energy using the exponential transform (as opposed to another monotonically increasing function), is the only way to get a lower bound on the cost of a stochastic optimal problem through Jensen's inequality, since it is necessary in order to obtain the KL-divergence from the likelihood ratio. Therefore, the exponential weighting of trajectories is not a heuristic guess in our framework, but rather a natural consequence given the form of the optimal distribution.

Our approach is also similar to Bayesian Inference approach to stochastic optimal control [59], however our approach differs significantly in both the theoretical framework and the algorithmic approach. In [59] it is shown that $\mathbb{KL}\left(\delta_U \parallel \hat{\mathbb{Q}}^*\right)$ is equivalent to the cost of a stochastic optimal control problem. In that case δ_U is the dirac-delta function (i.e. there is no noise in the control input), and $\hat{\mathbb{Q}}^*$ is the optimal distribution with a uniform prior (base distribution in our terminology). They then use this observation to develop an iterative scheme based on minimizing $\mathbb{KL}\left(\mathbb{Q}_{\pi,\Sigma} \parallel \hat{\mathbb{Q}}^*\right)$ which has non-increasing costs. However, in this case, there is not an interpretation of the "optimal distribution" other than the fact that plugging it into $\mathbb{KL}\left(\delta_U \parallel \hat{\mathbb{Q}}^*\right)$ reduces to a standard optimal control problem. This is less direct than our case, where the free energy lower bound provides the form of the optimal distribution and proof that it achieves a cost less than or equal to any other distribution. This motivates our goal of pushing the controlled distribution as close as possible to the optimal distribution, which we achieve by minimizing the KL-Divergence in the *opposite direction* as in [59], which leads to a significantly different sampling scheme.

The cross-entropy method for motion planning [60, 61, 62] is the previous work which is the closest, mathematically speaking, to our approach. As in our case, in the cross-entropy method the objective function has the form:

$$\theta^* = \operatorname*{argmin}_U \mathbb{KL}\left(\tilde{\mathbb{Q}}^* \parallel \mathbb{Q}\right),$$

where $\tilde{\mathbb{Q}}^*$ is a target distribution and $\mathbb{Q}$ is the distribution induced by the control parameters θ. However, instead of using the free-energy lower bound as we do, in the cross-entropy method the density of the optimal distribution is defined as follows:

$$\tilde{q}^*(V) = I\left(\{C(V) \leq \gamma\}\right),$$

where I is the indicator function, C is the cost-to-go function, and γ is a constant upper-bound on the trajectory cost that we would like to enforce. In order to optimize this objective, the following iterative procedure is proposed:

i) Sample parameters $\{\theta_1, \theta_2, \dots \theta_K\}$ from a given proposal distribution $P^i(\theta)$ (usually a Gaussian or a Gaussian mixture model).

ii) Determine the elite parameter set threshold: $\gamma_i = C(V, \theta_j)$ where j is the index of the L_{th} best trajectory sample. $L < K$.

iii) Compute the elite parameter set: $E_s = \{\theta_k | C(V; \theta_k) \leq \gamma_i\}$

iv) Update the parameters using expectation maximization over the elite set: $P^{i+1} \leftarrow$ EM(E_s)

v) If converged end, otherwise repeat.

In the case of optimizing the mean of a Gaussian distribution, the cross-entropy method described here is identical to the information-theoretic approach, except that the cross-entropy method takes an *un-weighted* average over the top L sampled parameters. In contrast, the information-theoretic approach takes a weighted average over all the parameter samples. This is an important difference: when planning trajectories the information theoretic approach has more discriminative power over rejecting (assigning very low weight) samples, whereas cross-entropy must assign the same weight to the top L samples, even if those samples have very different cost values. Later on in chapter 6, we compare a model predictive control version of our approach against an model predictive control version of cross-entropy.

CHAPTER 5

MODEL PREDICTIVE PATH INTEGRAL CONTROL

In the previous two chapters we have provided an overview of path integral control theory, developed the information theoretic control approach, and discussed the mathematical relationship between the two methods. In this chapter we show how these ideas can be used in a practical setting for controlling autonomous systems using on-the-fly optimization.

We label the algorithm that we describe as model predictive path integral control (MPPI). The key idea behind the algorithm is to randomly sample control sequences, compute the cost when each control sequence is applied to the system model, and then compute a cost-weighted average over the randomly sampled control sequences. In a typical MPC fashion, only the first element of the control sequence is executed. Then, the un-executed portion of the control sequence is used to warm-start the optimization at the next time-step, and the whole process is repeated.

Path integral control theory and the information theoretic control framework provide different interpretations for what a computing a cost-weighted average over trajectories achieves. In the information theoretic framework, the cost-weighted average over the controls is the best approximation of the optimal distribution parameterized by an open loop control sequence. In path integral control, the interpretation of the cost weighted average is that it is the optimal control for the current state and time. In developing a model predictive controller, it is possible to utilize either the stochastic optimal control interpretation or to purely utilize the information theoretic interpretation. A natural question to ask is: *which interpretation is most useful in this context, and is there a benefit to combining both interpretations?* Although, there is not necessarily a correct answer to this question, in our view, utilizing both interpretations provides the most complete picture. The advantage of utilizing both interpretations can be shown with an example of a highly counter-intuitive situation.

Consider the following problem: A noisy point mass robot is trying to reach a goal state, but, directly in between the goal state and the robot is an obstacle. We would like to compute the a current action to execute, and a control sequence which we can use to warm-start the optimization at the next time-step.

If we purely utilize the path integral control interpretation, then the cost-weighted average produces the optimal control, which, surprisingly, directs the robot to move directly towards the obstacle! The reason that this is the correct action is that, in a continuous time noisy system, the robot will immediately be pushed either to the left or to the right of the obstacle by system noise. After this push occurs, symmetry will collapse, and our robot will move to avoid the obstacle. However, before symmetry collapses it is best not to waste any energy, since it is unknown if moving left or right will turn out better. Therefore, the optimal control is to move towards the obstacle. This kind of symmetry breaking feature of path integral control was first explored by in [49].

Unfortunately, although the path integral control interpretation provides an explanation for the current control, it does not help to compute an entire sequence that we can use to warm-start the optimization at the next time-step. It is possible to apply the path integral control law to future time-steps in order to get a control sequence, but this is merely a heuristic in the path integral framework. Moreover, it appears that this heuristic fails in this case, since the result is an open loop control sequence that moves directly through the obstacle.

The information theoretic framework does provide a mechanism for computing the entire open-loop control sequence, in terms of minimizing the KL-Divergence. In this case it still results in a control sequence that moves the robot directly through the obstacle. However, this is actually correct from the perspective of matching the optimal distribution: sampling around a control sequence going directly through the obstacle will result in approximately half of the trajectories on appearing on the left and half of the trajectories appearing on the right, just like the optimal distribution. However, it is unclear how to generate a control from the KL-Divergence minimizing distribution. This is because the optimal distribution is multi-modal, so applying the mean does not necessarily make sense.

By utilizing both frameworks we have the following interpretation: computing a cost-weighted average over control sequences approximates the optimal distribution, and the optimal control is simply the mean of the optimal distribution at time $t = 0$, even if symmetries are present in the optimal distribution. After the first control input has been executed, we can then re-use the previous estimate of the optimal distribution in order to compute an updated estimate. In the case of a uni-model optimal distribution, executing the mean of the optimal distribution is reasonable without the usage of the path integral control interpretation. However, the path integral interpretation guarantees that symmetry breaking will occur in the case of a multi-modal optimal distribution.

Note that some care must be observed with relying on the symmetry breaking aspect of path integral control, since the information theoretic approach can be applied to more general systems than path integral control. In particular, the mean of the optimal distribution for a discrete time system is only approximately related to the continuous time optimal control law. Symmetry breaking *cannot* be guaranteed for discrete time systems, and if the time-step is large enough this could become a problem. However, in practice, this has not been observed, even for highly multi-modal cost landscapes [39].

It is also interesting to note that, under this interpretation, using the terminology model predictive control is a slight misnomer. In MPC, the control sequence that is optimized is usually assumed to be the optimal open-loop controls for a deterministic system. In our case, the “optimal sequence” may not actually be a set of controls that we want to execute in open-loop fashion, it is however, the set of controls which enable us to best approximate the stochastic optimal control. Thus, our method is better viewed as an implementation of a computationally expensive feedback control law than a traditional MPC method.

In the next few sections, we describe how the optimal distribution can be approximated using iterative importance sampling, and then dive into some practical issues that arise when trying to implement the algorithm. Lastly, we give a detailed summary of the algorithm at the end of the chapter. For the development of the MPPI algorithm we will assume that the system takes the form:

$$\begin{aligned} \mathbf{x}_{t+1} &= \mathbf{x}_t + \mathbf{F}(\mathbf{x}_t, \mathbf{v}_t)\Delta t, \\ \mathbf{v}_t &\sim \mathcal{N}(\mathbf{u}_t, \Sigma). \end{aligned} \tag{5.1}$$

This formulation handles most of the cases that we care about, since even continuous time systems must be approximated with discrete systems for Monte-Carlo samples to be computed. Note that we are not making any assumptions regarding the system model taking a control-affine form, since this is not necessary in the information theoretic framework and we can approximately relate Eq. (5.1) to a control-affine system through linearization.

5.1 Iterative Importance Sampling

We use the technique of importance sampling [63] to construct a set of samples that provide an unbiased estimate of the optimal control solution given a current control distribution. Recall that we are trying to estimate:

$$\begin{aligned} \mathbf{u}_t^* = \mathbb{E}_{\mathbb{Q}^*}[\mathbf{v}_t] &= \int q^*(V)\mathbf{v}_t \mathrm{d}V, \\ t &\in \{0, 1, \dots T-1\}, \end{aligned}$$

Where $\mathbb{Q}^*$ is the optimal distribution. Given an importance sampling control sequence, denoted by U, we can re-write this equation as:

$$\int q^*(V)\mathbf{v}_t \mathrm{d}V = \int \underbrace{\frac{q^*(V)}{q\left(V|U,\Sigma\right)}}_{w(V)} q\left(V|U,\Sigma\right)\mathbf{v}_t \mathrm{d}V,$$

This integral expression can be expressed as the following expectation:

$$\mathbb{E}_{\mathbb{Q}_{U,\Sigma}}[w(V)\mathbf{v}_t], \quad w(V) = \frac{q^*(V)}{q\left(V|U,\Sigma\right)}. \tag{5.2}$$

The weighting term, $w(V)$, is the importance sampling weight which allows us to compute expectations with respect to $\mathbb{Q}^*$ by sampling trajectories from the system with the importance sampling sequence applied (denoted $\mathbb{Q}_{U,\Sigma}$).

This weighting term in Eq. (5.2) can be split into two terms: one depending on the state cost of a trajectory, and the other is a likelihood ratio between the controlled and base distribution, which acts like a control cost. This split is achieved by using the base distribution $p(V)$ as follows:

$$\begin{aligned} w(V) &= \left(\frac{q^*(V)}{p(V)}\right)\left(\frac{p(V)}{q\left(V|U,\Sigma\right)}\right), \\ &= \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}S(V)\right)\left(\frac{p(V)}{q\left(V|U,\Sigma\right)}\right). \end{aligned} \tag{5.3}$$

In the case that the base distribution takes the form of the uncontrolled system dynamics, we have the following:

$$\frac{p(V)}{q\left(V|U,\Sigma\right)} = \frac{\exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}\mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t\right)}{\exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}(\mathbf{v}_t-\mathbf{u}_t)^{\mathrm{T}}\Sigma^{-1}(\mathbf{v}_t-\mathbf{u}_t)\right)},$$

Which expands out to:

$$= \exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}\mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t - \mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t + 2\mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t - \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t^{\mathrm{T}}\right),$$

And then simplifies to:

$$= \exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}2\mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t - \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t^{\mathrm{T}}\right),$$

Now, we can re-write $\mathbf{v}_t = \mathbf{u}_t + \epsilon_t$, doing this results in:

$$= \exp\left(-\frac{1}{2}\sum_{t=0}^{T-1} 2\epsilon_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t - \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t^{\mathrm{T}}\right),$$

which can be further simplified to:

$$= \exp\left(-\frac{1}{2}\sum_{t=0}^{T-1} \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\epsilon_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t\right),$$

Lastly, we re-combine the two importance sampling terms to get the following importance weighting and update rule:

$$w(V) = \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}\left(S(V) + \frac{\lambda}{2}\sum_{t=0}^{T-1}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right)\right), \tag{5.4}$$

$$\mathbf{u}_t' = \mathbb{E}_{\mathbb{Q}_{U,\Sigma}}[w(V)\mathbf{v}_t], \tag{5.5}$$

$$\mathbf{u}^* = \mathbf{u}_0'. \tag{5.6}$$

Equation (5.4) describes the importance sampling weight between the distribution induced by the current importance sampling sequence, and the optimal distribution. Equation (5.4) describes the update to the importance sampling sequence, and Eq. (5.6) is the first element of the importance sampling sequence which can be used as an approximation to the stochastic optimal control.

5.2 Covariance Variable Importance Sampling

The importance sampling that we described in the previous section involves randomly sampling perturbations around a nominal control sequence, which is iteratively updated according to Eq. (5.5). The variance of the random perturbations in the previous section is set to match $p(V)$, which, in principle, is the natural variance of the system. However, it can sometimes be beneficial to use a higher sampling variance than the natural variance in the system. This means that the likelihood ratio becomes:

$$\frac{p(V)}{q\left(V|U,\Sigma\right)} = \frac{\exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}\mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t\right)}{\exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}(\mathbf{v}_t-\mathbf{u}_t)^{\mathrm{T}}(\nu\Sigma)^{-1}(\mathbf{v}_t-\mathbf{u}_t)\right)},$$

Where $\nu \geq 1$ is a multiplier on the magnitude of the variance. It is possible to change the sampling variance in more complex ways than simply increasing the magnitude (as we do in [38]), but, increasing the magnitude is the simplest and most useful scenario.

As in the standard case, we can expand out and combine the numerator and denominator in order to get:

$$= \exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}\mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t - \mathbf{v}_t^{\mathrm{T}}(\nu\Sigma)^{-1}\mathbf{v}_t^{\mathrm{T}} + 2\mathbf{v}_t^{\mathrm{T}}(\nu\Sigma)^{-1}\mathbf{u}_t - \mathbf{u}_t^{\mathrm{T}}(\nu\Sigma)^{-1}\mathbf{u}_t\right),$$

Unlike in the standard case, the quadratic function depending purely on $\mathbf{v}$ does not cancel, instead we have:

$$\exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}(1-\nu^{-1})\mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t + 2\mathbf{v}_t^{\mathrm{T}}(\nu\Sigma)^{-1}\mathbf{u}_t - \mathbf{u}_t^{\mathrm{T}}(\nu\Sigma)^{-1}\mathbf{u}_t\right),$$

Now, let $\mathbf{v}_t = \mathbf{u}_t + \epsilon_t$, after some simplifications, the term inside of the summation then becomes:

$$(1-\nu^{-1})\left[\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t + \epsilon_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right] + \nu^{-1}\left[2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t + \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t\right],$$

When we combine these two terms we are left with:

$$\exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t + (1-\nu^{-1})\epsilon_t\Sigma^{-1}\epsilon_t\right),$$

The combined importance sampling weight in the case of increased covariance is then:

$$w(V) = \frac{1}{\eta} \exp\left(-\frac{1}{\lambda}\left(S(V) + \frac{\lambda}{2}\sum_{t=0}^{T-1} \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t + (1-\nu^{-1})\epsilon_t\Sigma^{-1}\epsilon_t\right)\right). \tag{5.7}$$

which is the same as the standard importance sampling weight, but with an extra term which penalizes large values of ϵ. Note that ν can also be a time varying parameter, which opens up the possibility of automatically adapting ν based on the given task. However, extreme care must be taken when adapting ν on the fly, since decreasing the covariance too much can lead to task failure in an MPC setting. For instance, in the past we have explored using the method from [64] to adapt the variance on-the-fly, but without success due to rapidly decreasing variance.

5.3 Practical Issues

The iterative importance sampling equations in (5.4) - (5.6), along with the equation for covariance variable importance sampling (Eq. (5.7)), form the basis of our sampling based control methodology. However, there are a few practical issues to address before describing the full algorithm. These are:

i) Shifting the range of the trajectory costs.

ii) Smoothing the solution.

iii) Decoupling the control cost and temperature.

iv) Sampling trajectories fast enough for online optimization.

In this subsection we explain effective solutions to these problems which keep the theoretical basis for the algorithm intact.

Shifting the range of the trajectory costs

The negative exponentiation required by the importance sampling weight is numerically sensitive to the range of the input values. If the costs are too high then the negative exponentiation results in values numerically equal to zero, and if the costs are not bounded from below then the negative exponentiation can lead to overflow errors. For this reason we shift the range of the costs so that the best trajectory sampled has a cost of 0. This simultaneously bounds the costs from below and ensures that at least one trajectory has an importance sampling weight which is not numerically zero. This is done as follows: first expand out the normalizing term η in (5.4) so that the importance sampling weight is:

$$= \frac{\exp\left(-\frac{1}{\lambda}\left(S(V) + \frac{\lambda}{2}\sum_{t=0}^{T-1} \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right)\right)}{\int \exp\left(-\frac{1}{\lambda}\left(S(V) + \frac{\lambda}{2}\sum_{t=0}^{T-1} \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right)\right) \mathrm{d}V},$$

Now define ρ as the minimum cost (in the Monte-Carlo approximation it is the minimum *sampled* cost). We then multiply by:

$$1 = \frac{\exp\left(\frac{1}{\lambda}\rho\right)}{\exp\left(\frac{1}{\lambda}\rho\right)},$$

which results in:

$$w(V) = \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}\left(S(V) - \rho + \frac{\lambda}{2}\sum_{t=0}^{T-1} \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right)\right), \tag{5.8}$$

$$\eta = \int \exp\left(-\frac{1}{\lambda}\left(S(V) - \rho + \frac{\lambda}{2}\sum_{t=0}^{T-1} \mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right)\right). \tag{5.9}$$

Since we have only multiplied by 1, this procedure does not change the optimality of the approach, however, it does prevent numerical overflow or underflow. Note that η is guaranteed to be between $[1, K]$ where K is the number of samples. This observation allows the value of η to be a useful tool for monitoring the health and debugging the MPPI algorithm.

If the value of η is close to 1 then the temperature is too high and all but the single best trajectory are being discarded. If the value of η is close to K than an un-weighted average is being taken, and the temperature or sampling variance may be to low. Although there is no single "correct" value for η, we have observed that a value between 1% and 10% of K generally indicates a healthy algorithm. Also note that the free-energy, which is related to the value function, can easily be computed from η, and is another good indicator of algorithmic performance.

Handling Control Constraints

Most interesting control systems, including autonomous vehicles, have actuator limits that the controller must take into account. A simple method for handling control constraints, which we utilize here, is to make the problem unconstrained by pushing the control constraints into the system dynamics:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{F}(\mathbf{x}_t, g(\mathbf{v}_t))\Delta t \tag{5.10}$$

where $g(\mathbf{v}_t)$ is a clamping function that restricts $\mathbf{v}_t$ to remain within an allowable input region. Since the sampling based update law does not require computing gradients or linearizing the dynamics, adding this additional non-linearity (and non-smooth) component into the dynamics is trivial to implement, and it works well in practice. Additionally, this step has no effect on the convergence of the importance sampling, since the clamping is realized as a change in the system dynamics, as opposed to a change in the algorithm itself.

Control Smoothing

The stochastic nature of the sampling procedure can lead to chattering in the resulting control, which can be removed by smoothing the output control sequence. One very effective method for smoothing is by fitting local polynomial approximations to the control sequence. Note that the objective in the information theoretic optimization framework (Eq. (4.26)) can be written as:

$$\mathbf{u}_t^* = \operatorname*{argmin}_{\mathbf{u}_t} \left(\mathbb{E}_{\mathbb{Q}^*} \left[\left(\mathbf{v}_t - \mathbf{u}_t \right)^{\mathrm{T}} \Sigma^{-1} \left(\mathbf{v}_t - \mathbf{u}_t \right) \right] \right),$$

And now consider fitting a local polynomial approximation (at every timestep) so that $\mathbf{u}_t = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \dots \mathbf{a}_k t^k = A\mathbf{t}$, where $A = (\mathbf{a}_0, \mathbf{a}_1, \dots \mathbf{a}_k)$ and $\mathbf{t} = (1, t, t^2, \dots t^k)$. Our goal is to then find the optimal set of coefficients at each timestep. The optimal coefficients, at timestep j, can be found through the following optimization:

$$A_j^* = \operatorname{argmin} \left(\mathbb{E}_{\mathbb{Q}^*} \left[\sum_{t=(j-k)}^{j+k} \left(\mathbf{v}_t - A\mathbf{t} \right)^{\mathrm{T}} \Sigma^{-1} \left(\mathbf{v}_t - A\mathbf{t} \right) \right] \right),$$

Note how the optimization now spans multiple timesteps into the past and future in order to compute a smoother control input. This optimization problem is equivalent to optimizing the objective:

$$\begin{aligned} &\mathbb{E}_{\mathbb{Q}^*} \left[\sum_{t=(j-k)}^{j+k} \mathbf{v}_t^{\mathrm{T}} \Sigma^{-1} A\mathbf{t} + \mathbf{t}^{\mathrm{T}} A^{\mathrm{T}} \Sigma^{-1} A\mathbf{t} \right], \\ &= \sum_{t=(j-k)}^{j+k} \mathbb{E}_{\mathbb{Q}^*}[\mathbf{v}_t]^{\mathrm{T}} \Sigma^{-1} A\mathbf{t} + \mathbf{t}^{\mathrm{T}} A^{\mathrm{T}} \Sigma^{-1} A\mathbf{t}, \end{aligned}$$

which is in turn is equivalent to the minimization:

$$A_t = \operatorname{argmin} \left(\sum_{t=j-k}^{j+k} \left(\mathbb{E}_{\mathbb{Q}^*}[\mathbf{v}_t] - A\mathbf{t} \right)^{\mathrm{T}} \Sigma^{-1} \left(\mathbb{E}_{\mathbb{Q}^*}[\mathbf{v}_t] - A\mathbf{t} \right) \right). \tag{5.11}$$

This is a convenient expression because it means that we can first compute the weighted average over trajectories, and then perform a local polynomial approximation in order to smooth the resulting control sequence. Note that this is equivalent to directly computing the optimal spline parameters with respect to minimizing the KL-Divergence between the optimal distribution and the normal distribution defined by the spline points, however, we do not have to handle any spline parameters inside the expectation.

The naive method for computing the controls is to then compute $\mathbb{E}_{\mathbb{Q}^*}[\mathbf{v}_t]$ using a Monte-Carlo approximation, solve for each A_t, and lastly compute the smoothed control inputs $\mathbf{u}_t$. However, a simpler method which achieves the same result is to use a Savitsky-Galoy filter [65] which implements local polynomial smoothing using a specific set of convolution coefficients. Using a Savitsky-Galoy filter, we simply compute $U' = \mathbb{E}_{\mathbb{Q}^*}[V]$ and then compute the smoothed control sequence, U, by passing U' through the convolutional filter.

Decoupling control cost and temperature

Consider the form of the importance sampling weight from Eq. (5.4) when we take the uncontrolled dynamics of the system as the base distribution[1]:

$$w(V) = \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}\left(S(V) + \frac{\lambda}{2}\sum_{t=0}^{T-1}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right)\right)$$

The challenge with this formulation is that changing the inverse temperature λ, also changes the relative control cost and vice versa. The inverse temperature determines how tightly peaked the optimal distribution is, as $\lambda \to 0$, the optimal distribution places all of its mass on a single trajectory, whereas as $\lambda \to \infty$ all points in the state space have equal weight. Figure 5.1 shows the probability weights corresponding to trajectory costs for varying values of λ. This coupling is sensible from a theoretical point of view: if we are allowed more control authority over the system, then we should be able to more tightly maintain a given trajectory. Unfortunately, raising the temperature too high results in numerical instability since most trajectories are rejected (have weight numerically equal to zero), at which point the importance sampling oscillates between solutions instead of converging.

Our solution is to change the base distribution which defines the control cost. Let U be

[1]This is a natural choice from an optimization perspective, since the minimum control cost is achieved with $U \equiv \mathbf{0}$.

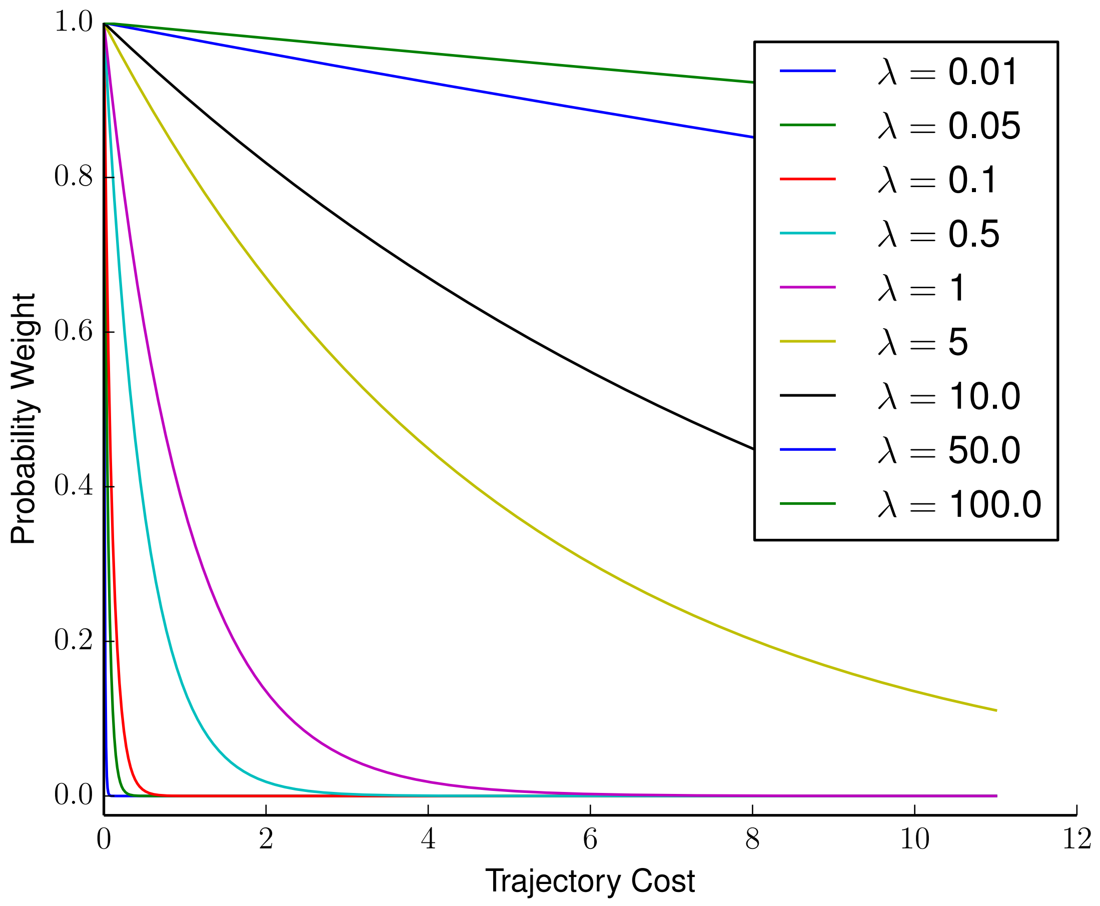


Figure 5.1: Effect of changing λ on the probability weight corresponding to a trajectory cost. Low values of λ result in many trajectories being rejected, high values of λ take close to an un-weighted average.

the current planned control sequence, and define the new base distribution as:

$$\tilde{p}(V) = p(V|\alpha U, \Sigma),$$

where $0 < \alpha < 1$. With $\alpha = 0$, the base distribution reverts back to the uncontrolled dynamics and pushes U to zero. And with $\alpha = 1$, the base distribution is the distribution corresponding to the current planned control law, which keeps U near the current distribution. This is useful for creating smooth control inputs since it prohibits $\mathbf{u}_0$ (the actual control input about to be applied) from moving to far from $\mathbf{u}_0$. To see why this helps create smooth motions, consider that the smoothing step at the previous iterations generated $\mathbf{u}_{-1}$ (the last control applied to the system), and $\hat{\mathbf{u}}_0$ as the first two elements in the (smoothed) sequence. Therefore, applying $\mathbf{u}_0$ would result in a smooth action. However, if the updated solution, $\mathbf{u}_0$, is far away from $\hat{\mathbf{u}}_0$ the resulting action will not be smooth. The solution is to therefore encourage $\mathbf{u}_0$ to stay close to $\hat{\mathbf{u}}_0$, a value of α in-between zero and one balances the two requirements of low energy and smoothness.

The construction of the optimal distribution and the corresponding control law are the same under this new base distribution. However, the control cost portion of the importance sampling weight now becomes:

$$\frac{\lambda}{2}(1-\alpha)\sum_{t=0}^{T-1}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t,$$

We then have $\gamma = \lambda(1-\alpha)$ as the new control cost parameter, resulting in

$$w(V) = \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}\left(S(V) + \frac{\gamma}{2}\sum_{t=0}^{T-1}\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right)\right),$$

as the new probability weighting for the algorithm. Note that in the case of using an amplified covariance, the multiplier on the exploration variance is still λ, so we have:

$$\frac{1}{\eta}\exp\left(-\frac{1}{\lambda}\left(S(V) + \frac{1}{2}\sum_{t=0}^{T-1}\gamma\left[\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right] + \lambda\left[(1-\nu)\epsilon_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right]\right)\right). \tag{5.12}$$

in the most general case.

GPU-Based Trajectory Sampling

The key requirement for applying MPPI is the ability to generate and evaluate a large number of samples in real time. This can be done by performing sampling in parallel on a graphics processing unit (GPU) with Nvidia's CUDA architecture. The CUDA implementation details differ significantly depending on the specific costs and system dynamics. The cost function is usually straightforward to implement in CUDA, but it can be difficult to implement the dynamics required to simulate the system forward fast enough to achieve real time performance. A challenging, but also highly effective, implementation that we cover in this thesis is using a neural network model of the AutoRally dynamics in order to simulate trajectory samples. An analysis of the different versions of that implementation is given in Appendix B. The best implementation from Appendix B can achieve control loops at 50 HZ using approximately 2,500 trajectory samples of 2.5 second long trajectories on an Nvidia GTX 1050 TI.

5.4 Algorithm Summary

With our iterative importance sampling update, as well as methods for handling control constraints, smoothing, and real-time sampling, we are now ready to describe the full model predictive path integral control (MPPI) algorithm. The algorithm (Alg. 1) starts by taking in the current state from an external state estimator, and then produces K trajectory samples in parallel on the GPU. Each sample is generated by randomly sampling a sequence of control perturbations, and then the dynamics are simulated forward and the cost is computed.

Once the costs for each perturbation sequence are computed, they are converted to probability weights. After the probability weights have been computed, the un-smoothed control update is computed via a probability weighted average over all the perturbation sequences. Lastly, this update is smoothed by passing it through a convolutional filter with the Savitsky-Galoy coefficients. The first control is then sent to the actuators, and the remaining sequence of length $T - 1$ is slid down and used to warm-start the optimization at the next time instance.

Algorithm 1: Model Predictive Path Integral Control (MPPI)

```
Given: $\mathbf{F}, g$: Dynamics and clamping function;
$K, T$: Number of samples and timesteps;
$U$: Initial control sequence;
$\Sigma, \nu, \lambda, \gamma, \phi, q$: Cost functions/parameters;
SGF: Savitsky-Galoy convolutional filter;
while task not completed do
    $\mathbf{x} \leftarrow Fn\_GetStateEstimate()$;
    /* Begin parallel block                                   */
    for $k \leftarrow 0$ to $K-1$ do
        $\tilde{S}_k \leftarrow 0$;
        Sample $\mathcal{E}^k = \left(\epsilon_0^k \dots \epsilon_{T-1}^k\right)$, $\epsilon_t^k \in \mathcal{N}(0, \nu\Sigma)$;
        for $t \leftarrow 0$ to $T-1$ do
            $\mathbf{v}_t = \mathbf{u}_t + \epsilon_t^k$;
            $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{F}(\mathbf{x}, g(\mathbf{v}_t))\Delta t$;
            // State cost and importance sampling weights
            $\tilde{S}_k \mathrel{+}= \mathbf{q}(\mathbf{x}) + \frac{1}{2}\left(\gamma\left[\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right] + \lambda\left[(1-\nu)\epsilon_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right]\right)$;
        $\tilde{S}_k \mathrel{+}= \phi(\mathbf{x})$;
    /* End parallel block                                     */
    // Compute trajectory weights
    $\rho \leftarrow \min\{\tilde{S}_0, \tilde{S}_1, \dots \tilde{S}_k\}$;
    $\eta \leftarrow \sum_{k=1}^{K} \exp\left(-\frac{1}{\lambda}(S_k - \rho)\right)$;
    for $k \leftarrow 1$ to $K$ do
        $w_k \leftarrow \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}(S_k - \rho)\right)$;
    // Control update with smoothing
    for $t \leftarrow 0$ to $T-1$ do
        $U \leftarrow \mathrm{SGF} * \left(U + \sum_{k=1}^{K} w_k \mathcal{E}^k\right)$;
    $Fn\_SendToActuators(\mathbf{u}_0)$;
    for $t \leftarrow 1$ to $T-1$ do
        $\mathbf{u}_{t-1} \leftarrow \mathbf{u}_t$;
    $\mathbf{u}_{T-1} \leftarrow \mathrm{Intialize}(\mathbf{u}_{T-1})$;
```

CHAPTER 6
APPLICATIONS OF MPPI TO AUTONOMOUS SYSTEMS

In this chapter we examine the application of MPPI to autonomous control systems. First, we apply the algorithm to several simulation problems that demonstrate some essential traits of the algorithm. Namely, we analyze the ability of the algorithm to handle non-linear dynamics, we verify that symmetry breaking does indeed occur, and we analyze the performance of the system on a high-dimensional multi-agent quadrotor system. After analyzing the method in simulation, we turn our attention to our motivating task of high-speed autonomous driving, and we analyze the strengths and weaknesses of MPPI when applied to the autonomous racing task.

6.1 Simulation Results

We use four simulated systems in order to test MPPI: a cart-pole, a simulated 1/5 scale miniature race car, a quadrotor attempting to navigate an obstacle filled environment, and lastly the same task but with multiple quadrotors. For the race car and quadrotor we used a model predictive control version of the differential dynamic programming (DDP) algorithm as a baseline comparison. In all of these experiments the controller operates at 50 Hz.

Cart-Pole

The cart-pole swing up task is a standard test for non-linear MPC controllers, and so it is important to verify that MPPI can competently handle this task. The swing-up tasks works as follows: a pendulum is attached to a cart and is initially in a downward position, the controller has the ability to laterally accelerate the cart, which imparts some angular velocity to the pendulum, the goal of the controller is to swing-up the pendulum to an upright facing position.

Mathematically, this task can be encoded using the following objective:

$$q(\mathbf{x}) = p^2 + 500(1 + \cos(\theta))^2 + \dot{\theta}^2 + \dot{p}^2 \tag{6.1}$$

Where p is the position of cart, $\dot{p}$ is the velocity and $\theta, \dot{\theta}$ are the angle and angular velocity of the pole. In this case $\theta = 0$ corresponds to the down pendulum position and $\theta = \pi$ corresponds to the upward position. The full equations of motion for the cart-pole system are given in Appendix C.

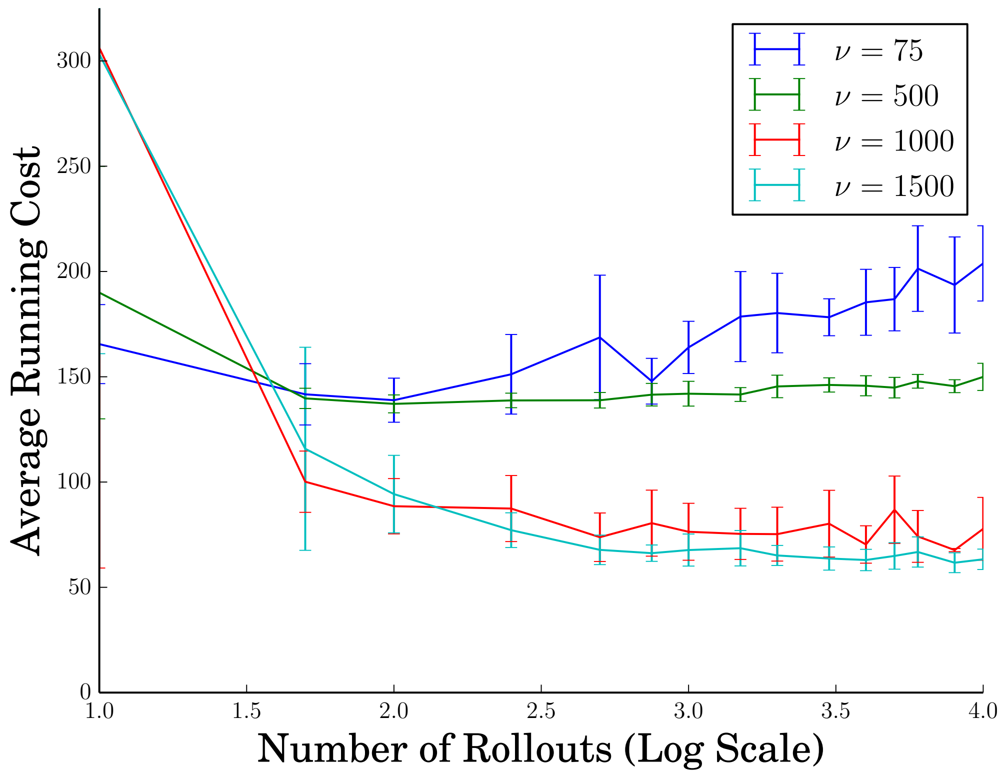


Figure 6.1: Performance of MPPI on the cart-pole swing up task. In all cases the algorithm is successful at the swing up task. The Y-axis is the average running cost achieved during the trial, and the X-axis is the number of samples used (base 10 log scale).

We set the natural system variance equal 0.1 and $\lambda = \gamma = 10$. The MPPI controller was given 10 seconds to swing up the cart-pole, and the optimization horizon was 1 second. After the initial swing up, controller has to keep the pendulum balanced in the upright position for the rest of the 10 second horizon. The variance magnitude parameter, ν, was varied between 1 and 1500. The MPPI controller is able to swing-up the pole faster with increasing exploration variance. Fig. 6.1 illustrates the performance of the MPPI controller as the variance magnitude and the number of samples are increased. These results demonstrate that MPPI performs competently at the cart-pole swing up task.

Race Car

The next simulated task tests MPPI's ability to control a simulated 1/5 scale vehicle similar to the real-world AutoRally vehicle, and compares its performance against an MPC implementation of an iterative linear quadratic gaussian controller (MPC-DDP). In this task the goal was to minimize the objective function:

$$\begin{aligned} q(\mathbf{x}) &= 100d^2 + (v_x - 7.0)^2, \\ d &= \left| \left(\frac{x}{13}\right)^2 + \left(\frac{y}{6}\right)^2 - 1 \right|. \end{aligned} \tag{6.2}$$

Where v_x is the forward (in body frame) velocity of the car. This cost ensures that the car to stays on an elliptical track while maintaining a forward speed of 7 meters/sec. The elliptical track is roughly the same size as the real-world Marietta street track. We use a non-linear vehicle model [66] which takes into account the (highly non-linear) interactions between tires and the ground.

Figure 6.2 illustrates the comparison of MPC-DDP (left) and MPPI (right) performing a cornering maneuver along the ellipsoid track. MPPI is able to make a much tighter turn than MPC-DDP while also carrying more speed in and out of the corner than MPC-DDP. MPPI is able to do this by sliding slightly into the corners, which shows that it can effectively handle the non-linearities in the vehicle model. Figure 6.3 illustrates the corresponding longitudinal and lateral velocities attained during the trial.

The MPPI controller is able to enter turns at close to the desired speed of 7 m/s and then

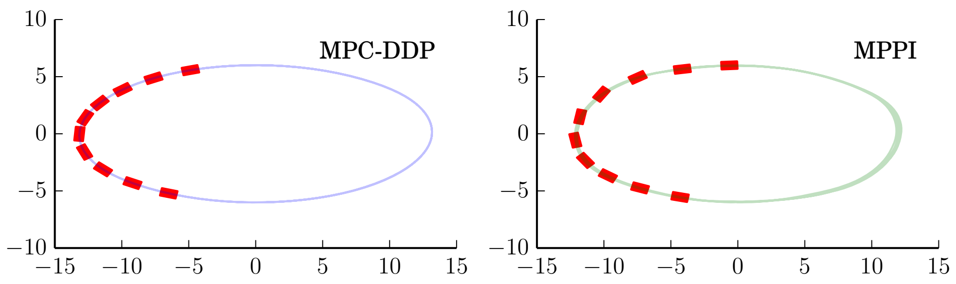


Figure 6.2: Comparison of MPC-DDP (left) and MPPI (right) performing a cornering maneuver. The direction of travel in these figures is counter-clockwise.

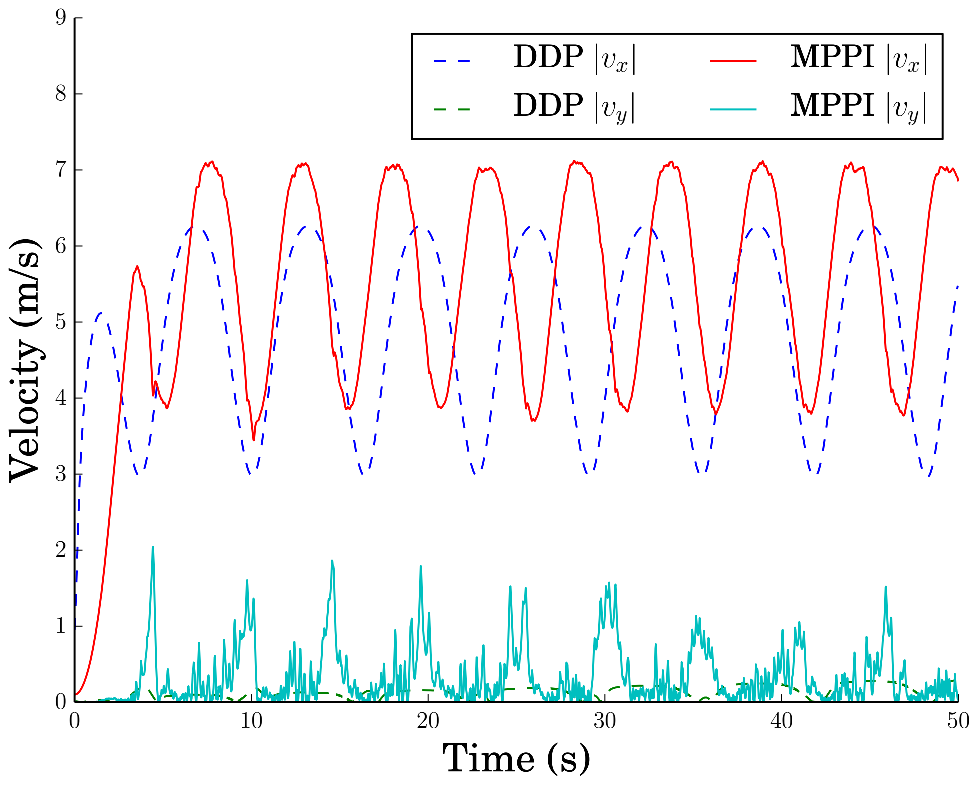


Figure 6.3: Velocity comparison of MPC-DDP (left) and MPPI (right) performing a cornering maneuver. MPPI is able to attain a faster top speed and carries more speed into and out of the corners than MPC-DDP, while also sliding slightly into the turns.

slide through the turn. The MPC-DDP solution does not attempt to slide and significantly reduces its forward velocity before entering the turn, this results in a higher average cost compared to the MPPI controller. Figure 6.4 shows the performance comparison in terms of average cost between MPPI and MPC-DDP as the exploration variance ν changes from

50 to 300 and the number of rollouts changes from 10 to 10000.

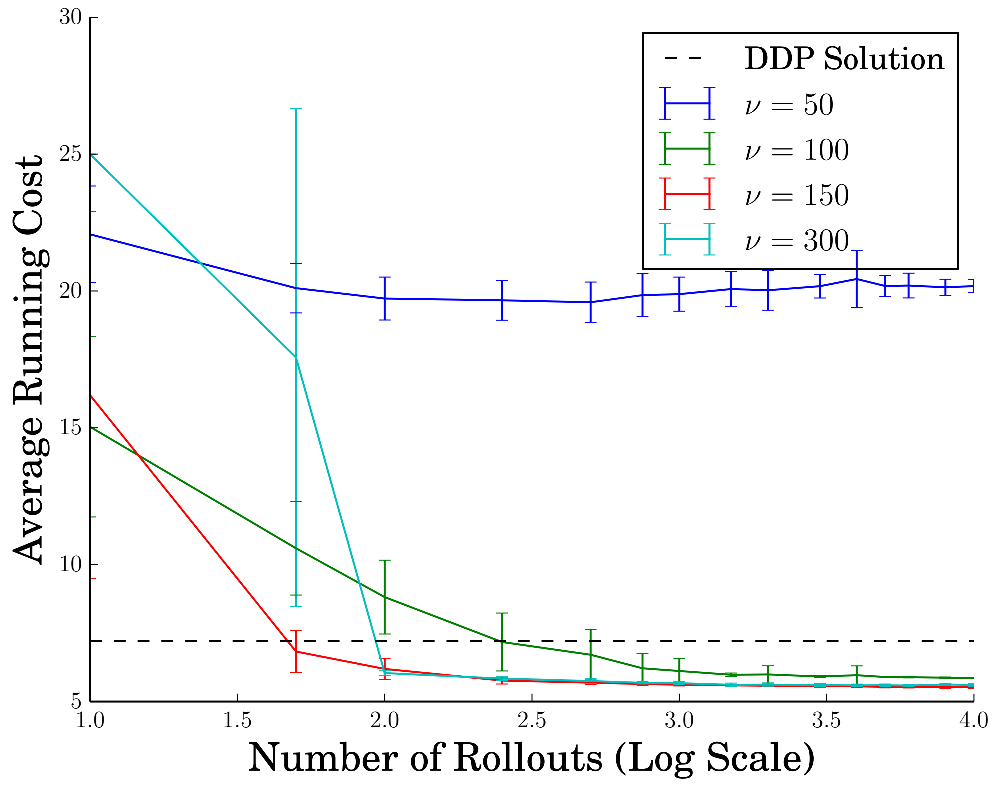


Figure 6.4: Performance comparison in terms of average cost between MPPI and MPC-DDP for the Race Car. With a high enough exploration variance and enough rollouts MPPI is able to outperform MPC-DDP.

Quadrotor

The quadrotor task was to fly through a field filled with cylindrical obstacles as fast as possible (see Fig. 6.6). We used the quadrotor dynamics model from [67], which is detailed in Appendix C. This is a non-linear model which includes position, velocity, euler angles, angular acceleration, and the rotor dynamics. We randomly generated three forests, one where obstacles are on average 3 meters apart, the second one 4 meters apart, and the third 5 meters apart. This task tests two key traits of the MPPI controller:

i) How well the MPPI controller can satisfy constraints (i.e. avoid obstacles).

ii) The effectiveness of symmetry breaking in a practical setting, since there are multiple paths around all of the obstacles.

The running cost function for MPPI was of the form:

$$2.5(p_x - p_x^{des})^2 + 2.5(p_y - p_y^{des})^2 + 150(p_z - p_z^{des})^2 + 50\psi^2 + \|v\|^2 + 350\exp(-\frac{d}{12}) + 1000C$$

Here (p_x, p_y, p_z) denotes the position of the vehicle, ψ denotes the yaw angle in radians, v is velocity, and d is the distance to the closest obstacle. The variable C indicates whether the vehicle has crashed into the ground or an obstacle, and it acts to enforce the task constraints.

We found that the crash indicator term is not useful for the MPC-DDP based controller, this is not surprising since the discontinuity it creates is difficult to approximate with a quadratic function. The term in the cost for avoiding obstacles in the MPC-DDP controller consists purely of a large exponential term: $2000\sum_{i=1}^{N}\exp(-\frac{1}{2}d_i^2)$, note that this sum is over all the obstacles in the proximity of the vehicle whereas the MPPI controller only has to consider the closest obstacle.

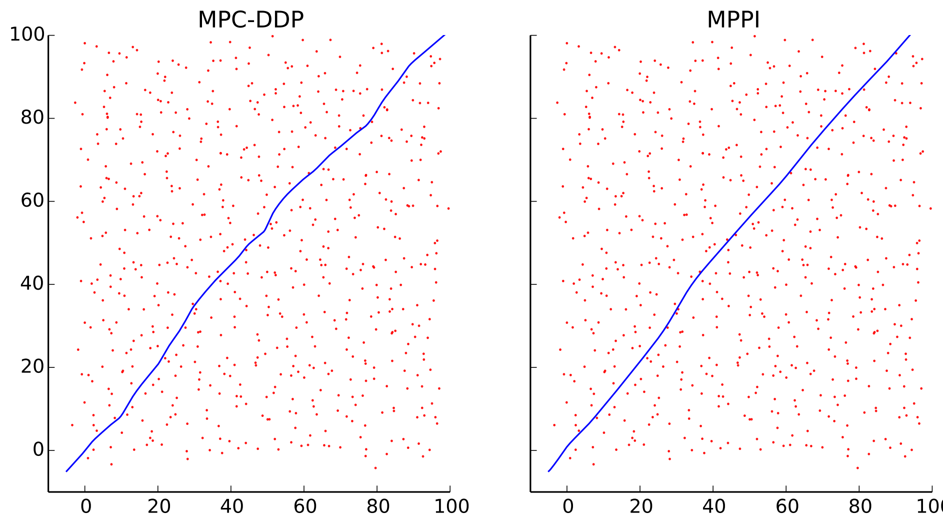


Figure 6.5: Sample trajectory through 4m obstacle field DDP (left) and MPPI (right). The MPPI controller is able to safely navigate the obstacle field, and goes faster than MPC-DDP.

Not only is the MPPI controller able to navigate the forst, but, since it can explicitly reason about crashing (as opposed to just staying away from obstacles), it is able to travel faster and closer to obstacles than the MPC-DDP controller. Figure 6.5 shows a trajectory taken by MPC-DDP and one of the MPPI runs on the forest with obstacles placed on average 4 meters away. Since the MPPI controller can directly reason about the shape of the obstacles, it is able to safely pass through the field by taking a much more direct route. Fig. 6.7 shows the performance difference between the two algorithms in terms of the time to navigate through the forest.

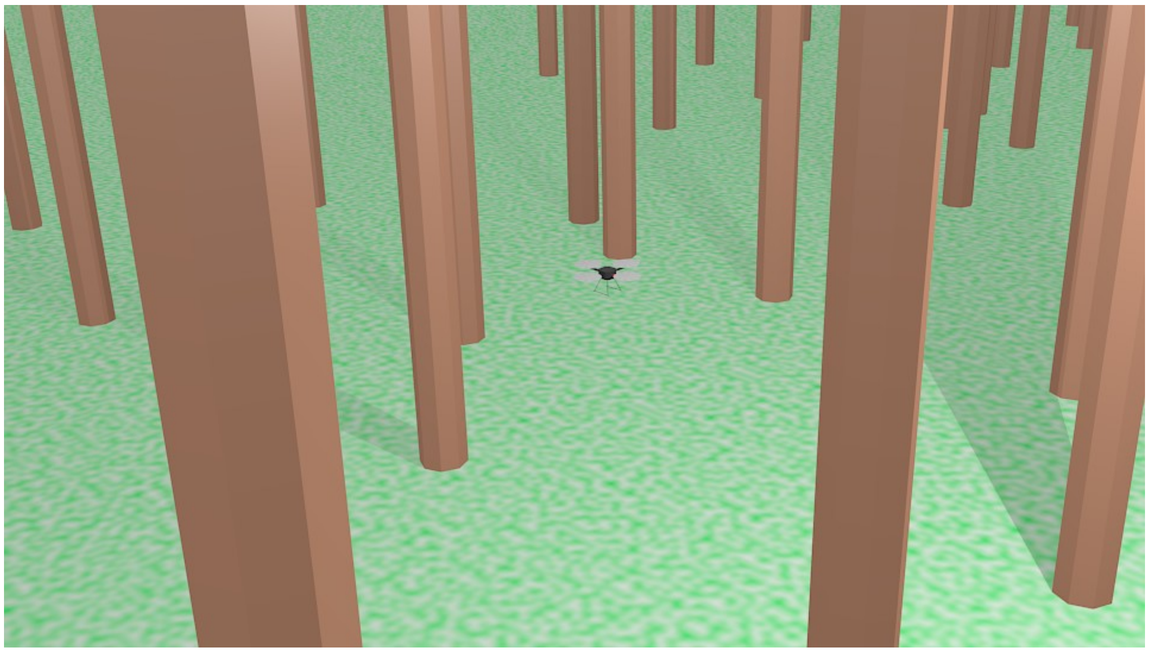


Figure 6.6: Simulated forest environment used in the quadrotor navigation task. There are multiple paths through the obstacle field, but symmetry breaking proves to be an effective mechanism with MPPI.

Multiple Quadrotors

We also tested MPPI's ability to simultaneously control multiple quadrotors operating in close proximity to each other, this was done by combining several quadrotors into one large system and then attempting the same obstacle navigation task. This results in an extremely high-dimensional system (for 9 quadrotors the system has 144 states and 36 control inputs). The algorithm was always able to successfully control 3 quadrotors at once moving through

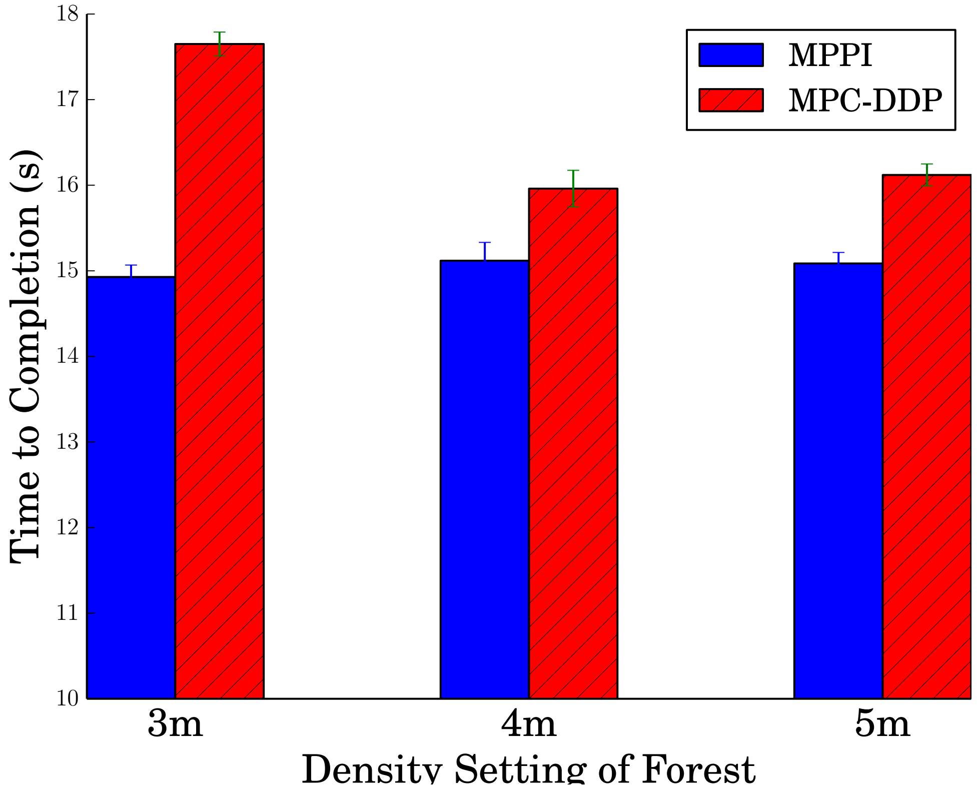


Figure 6.7: Time to navigate the obstacle field for MPC-DDP and MPPI.

the obstacle field, and most of the time it was able to control 9 quadrotors performing the same task. In order to further increase the task difficulty, we also set the obstacles to move at a rate of 3 m/s in a random direction.

Figure 6.8 illustrates trajectories of the 9 quadrotors moving through the cluttered environment with constant obstacles, and Fig. 6.9 illustrates the performance of the 9 quadrotors in terms of success rate as a function of the number of the samples for the case of moving and static obstacles.

Lastly, Fig. 6.10 includes the time for each iteration of the MPPI controller as a function of the number of the rollout for the case of 1, 3, and 9 quad rotors. The dashed line in Fig. 6.10 represents the real time requirement, which is in the order of 20ms. For the case of a single quadrotor, the optimization time crosses the real time for 4000 rollouts. In the case of 3 and 9 quadrotors the optimization time exceeds the real time for 1200 and 500 samples

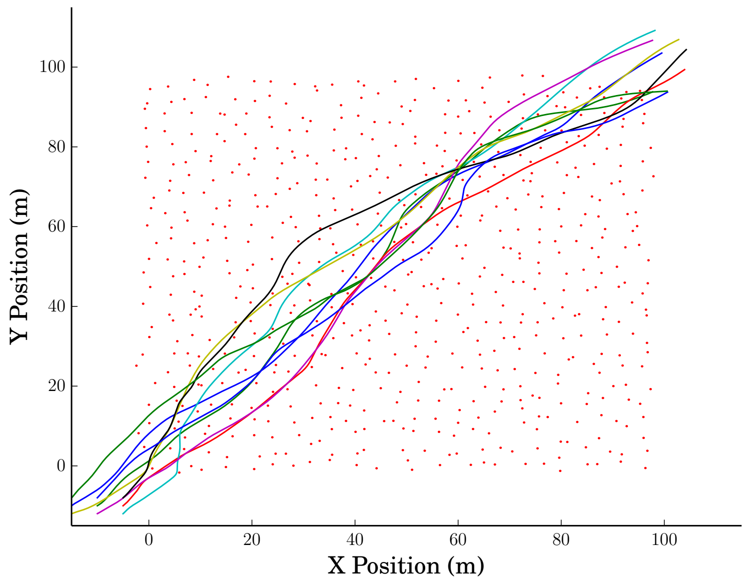


Figure 6.8: Trajectory trace of MPPI navigating the 9 quadrotor system through the obstacle field. MPPI considers all the quadrotors to be one, very large, system and then has to find controls which jointly guides all of the quadrotors through the field.

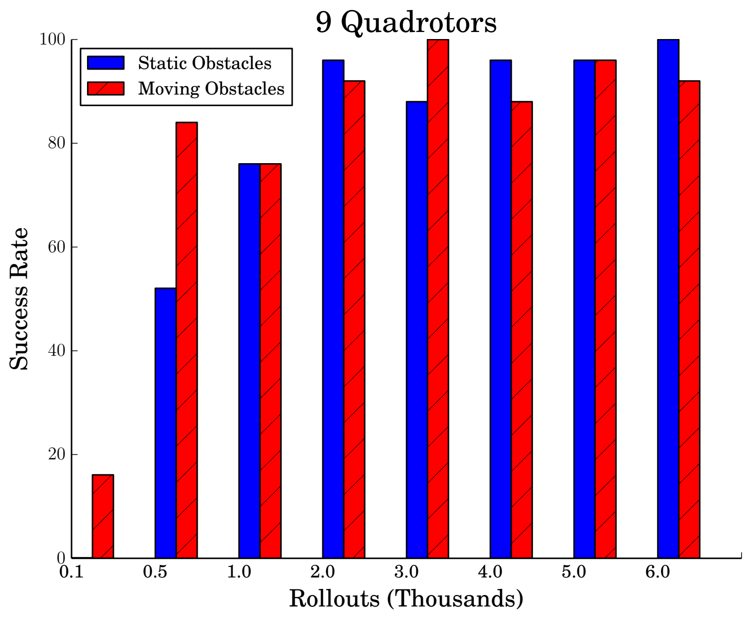


Figure 6.9: Success rate for MPPI guiding the 9 quadrotor system through the obstacle field, taken over 100 trials. With 6000 rollouts MPPI is always able to successfully guide the system through the obstacle field for static obstacles, but still occasionally fails when the obstacles are in motion.

respectively. 1200 samples is sufficient to control the three quadrotor system, however, for the 9 quadrotor system more than 500 rollouts are needed. In the case of the 9 quadrotor system, 6000 trajectories are required which runs in about 6x real-time. More importantly, the growth in computational time is approximately linear in the order of the state space. It therefore seems reasonable to believe that with near-term hardware improvements, MPPI with a system of the size of the 9 quadrotor system (144 state variables and 36 control inputs) will be possible in real-time.

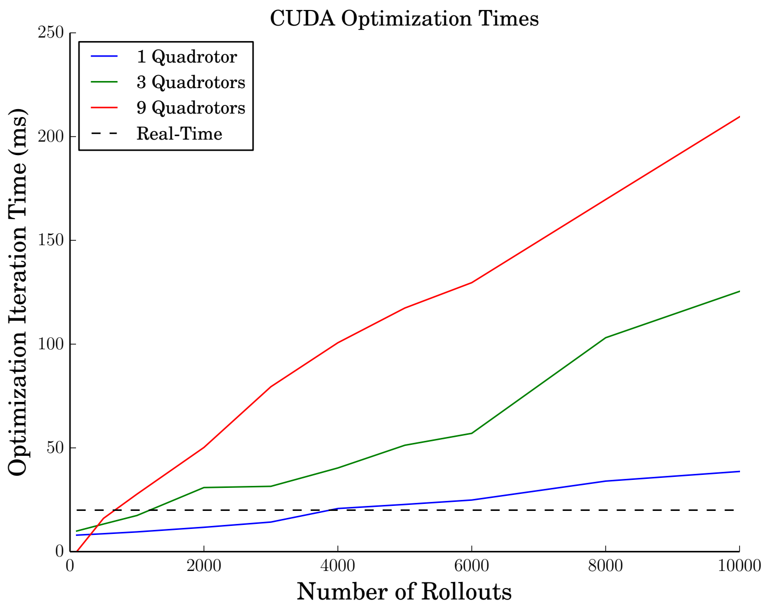


Figure 6.10: Time per iteration of the MPPI controller as a function of the number of sampled trajectories.

6.2 AutoRally Experimental Results

In this section we apply MPPI to our motivating research problem: autonomous racing on a dirt test track. Recall that there were three different criteria we want our controller to be able to satisfy: we want our controller to be able to perform at the limits of handling, satisfy constraints, and optimize for reasonably long time horizons. We conducted an extensive set of experiments with MPPI, and analyze the algorithms ability to satisfy these three requirements. In the previous section, we compared MPPI against DDP and showed the MPPI can attain comparable or superior performance. In this section, we compare MPPI against a competing sampling based controller: a model predictive control version of the cross-entropy method (CEM-MPC).

AutoRally Dynamics Models

In order to push the system to its performance limits, we require an accurate dynamics model which can capture where the limits are. In the case of the AutoRally vehicle, obtaining an accurate dynamics model is highly non-trivial. Within the autonomous driving literature there exist several types of models for full-scale vehicles [68, 69], as well as simplified "bicycle" vehicle models. However, there are a number of challenges in applying these models to the AutoRally system. Most notable among these are the dirt track which makes applying friction models meant for pavement difficult, and the significant roll dynamics of the vehicle which makes applying simplified models inaccurate.

The flexibility of MPPI to handle many different types of models is useful here, since we can use a machine learning approach to create a highly accurate data-driven as opposed to applying an inaccurate physics model. In this chapter we examine two different approaches: one hybrid-physics based approach, and a pure machine learning approach using a fully-connected feed-forward neural network.

Both models have the same state-space description of the AutoRally vehicle with seven state variables: x-position, y-position, heading, roll, longitudenal (body-frame forward) velocity, lateral (body-frame sideways) velocity, and heading rate. These are denoted as $\left(p_x, p_y, \theta, r, v_x, v_y, \dot{\theta}\right)$ respectively. The two control variables are the steering and throttle inputs, which are denoted by u_1 and u_2.

A certain subset of the equations of motion are kinematically trivial given the state space representation. We therefore partition the state space into kinematic state variables,

$\mathbf{x}_k$, and dynamic state variables, $\mathbf{x}_d$, such that:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}^k \\ \mathbf{x}^d \end{pmatrix}.$$

Let $\mathbf{x}^k = (p_x, p_y, \theta)^{\mathrm{T}}$, then we can write the equations of motion for the kinematic variables as:

$$\mathbf{x}^k_{t+1} = \mathbf{x}^k_t + \mathbf{k}(\mathbf{x})\Delta t,$$

where the function $\mathbf{k}(\mathbf{x})$ is defined as:

$$\mathbf{k}(\mathbf{x}) = \begin{pmatrix} \cos(\theta)v_x - \sin(\theta)v_y \\ \sin(\theta)v_x + \cos(\theta)v_y \\ \dot{\theta} \end{pmatrix}.$$

Given these kinematic updates, the dynamics model only has to determine the update equations for the dynamic state variables:

$$\mathbf{x}^d = \left(r, v_x, v_y, \dot{\theta}\right)^{\mathrm{T}}.$$

The dynamics of these variables do not depend on the global coordinate frame, and therefore are not functions of the kinematic state variables. Therefore the equations of motions for the dynamic state variables can be written as:

$$\mathbf{x}^d_{t+1} = \mathbf{x}^d_t + \mathbf{f}(\mathbf{x}^d_t, \mathbf{v}_t)\Delta t,$$

Where $\mathbf{v}_t = (u_1(t) + \epsilon_1(t), u_2(t) + \epsilon(t))$ is the randomly perturbed control input. The full equations of motion are then:

$$\mathbf{x}_{t+1} = \begin{pmatrix} \mathbf{x}_k(t) \\ \mathbf{x}_d(t) \end{pmatrix} + \begin{pmatrix} \mathbf{k}(\mathbf{x}(t)) \\ \mathbf{f}(\mathbf{x}_d(t), \mathbf{v}(t)) \end{pmatrix} \Delta t.$$

Given these equations, the challenge is to determine the function $\mathbf{f}$. Both methods fit their parameters using a system identification dataset collected by a human pilot executing a series of choreographed maneuvers:

i) Slow driving (3 - 6 m/s) around the track.

ii) Zig-Zag maneuvers at slow speeds (3 - 6 m/s).

iii) High acceleration maneuvers by applying full throttle at the beginning of a straight and applying full brake before entering the next turn.

iv) Sliding maneuvers where the pilot attempts to slide as much as possible.

v) High speed driving where the pilot simply attempts to drive around the track as fast as possible.

Each maneuver was executed for 3 minutes going counter-clockwise and 3 minutes going clockwise for a total of 30 minutes worth of driving data.

Basis Function Model

The basis function model has the form:

$$\mathbf{f}(\mathbf{x}_d) = \Theta^{\mathrm{T}} \phi(\mathbf{x}_d),$$

where $\Theta \in \mathbb{R}^{b \times 4}$ and $\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots \phi_b(\mathbf{x}))^{\mathrm{T}} \in \mathbb{R}^b$ is a matrix of coefficients and a vector of non-linear basis functions respectively. The term b denotes the number of basis functions in the model. Given this model form, there are two challenges: determining an appropriate set of basis functions, and computing the coefficient matrix Θ. For determining an appropriate set of basis function we analyzed the non-linear bicycle model of vehicle dynamics from [66], and extracted out all of the non-linear functions that appeared in the algebraic equations. This led to a set of 21 basis functions, and then, using trial and error, we added 4 more basis functions to account for the roll dynamics and the non-linear throttle calibration. The vehicle model and the basis functions extracted from it are described in Appendix C.

Given a set of basis functions and some data collected from the system, determining the coefficient matrix Θ is an unconstrained linear regression problem which is easy to solve. We used linear regression with Tikhonov regularization to solve for Θ given the basis functions and the system identification dataset. Even though we are interested in simulating entire trajectories forward in time, we train the model to minimize the one-step prediction error (i.e. given $(\mathbf{x}_t^d$ and $\mathbf{u}_t)$ predict $\mathbf{x}_{t+1}^d)$ as opposed to the multi-step prediction error. Minimizing multi-step prediction error is technically the correct objective, but considerably more difficult as the problem becomes non-convex [70]. An important detail to note is that performing standard linear regression (without Tikhonov regularization) does not work for multi-step prediction as the weights tend to be very large which results in unstable forward simulation, even if the one-step prediction error is lower than the regularized method.

Neural Network Model

The second model that we trained to approximate the dynamics function was a multi-layer neural network model. We use a two hidden layer, fully connected model with hyperbolic tangent non-linearities. Each hidden layer had 32 neurons for a total of 1412 parameters. The neural network model is trained using the same 30 minute system identification dataset as the basis function model, and again we minimize one-step prediction error. The model is trained with mini-batch gradient descent using the RMSProp optimizer [71] and L2 regularization.

Table 6.1: Errors for Basis Function and Neural Network

	Basis Function	Neural Network
R2 Score	.68	.78
Mean Squared Error	2.07	1.39
Mean Absolute Error	.93	.76

The neural network significantly outperformed the basis function model on the validation dataset. The coefficient of determination, mean squared error, and mean absolute error for the two models on the validation set are shown in Table 6.1. Despite the inferiority of the basis function model on these testing metrics, we still tested both models in order to determine whether the physics based features provided superior generality to the purely black-box neural network. Both models suffer from inaccuracies due to the effect of hidden variables, like track condition and battery voltage, that affect the dynamics but are not represented in the state space representation of the system.

Cost Function and Algorithm Parameters

There are a number of free parameters in the MPPI and CEM-MPC algorithms. We used simulation experiments to initially determine these parameters, and then used a small number of real-world experiments in order to fine tune them. The same cost function and algorithmic parameters were used across all the experimental settings (except for the speed target which modulates how fast the vehicle goes). Table 6.2 lists the parameter values used during the experiments.

Table 6.2: MPPI and CEM-MPC Parameters

Parameter	Value
Control Frequency	40 Hz
Time Horizon	2 seconds
λ	12.5
γ	0.1
ν	1.0
Σ	Diag$(0.0306, 0.0506)$
Initialize$(\mathbf{x_{T-1}})$	$(0,0)$
Eliteness threshold (Cross-Entropy only)	> 0.8 percentile

Since the MPPI and CEM-MPC are both sampling based methods, we did not design separate cost functions for the two algorithms. This would not be the case in comparing with a gradient based method, where smoothness would have to be enforced. The state-dependent cost function that we used was of the form:

$$\alpha_1 \text{Track}(\mathbf{x}) + \alpha_2 \text{Speed}(\mathbf{x}) + \alpha_3 \text{Stabilizing}(\mathbf{x}) \tag{6.3}$$

The three components of the cost function are as follows:

Track Cost

For the track cost we require a map representation of the track which gives an indication of how close to the edge the vehicles position is. There are a variety of ways to create such a map, our approach was to take a GPS survey of the boundaries of the track, then a cubic 2-dimensional spline was used to regress a cost map with points on the outer boundary set to 1 and points on the inner boundary set to -1. The absolute value of this map was taken to produce the overall cost map. Lastly, the total cost was capped at 2.5 in order to avoid regression artifacts far away from the track. The cost-map is stored in CUDA texture memory which enables fast lookups for data exhibiting 2-d locality, it also automatically interpolates the grid so that look-ups with continuous positions are efficient. Letting $h(p_x, p_y)$ denote the value returned by the cost map, the overall track cost is then:

$$\text{Track}(\mathbf{x}) = h(p_x, p_y) + .9^t \left(10000 I(\{h(p_x, p_y) > .99\})\right).$$

In the second term t is the timestep and I is an indicator function. This is a time-decaying impulse penalty for being located outside the track boundaries. It is necessary to include the time-decay because of disturbances and errors in the dynamics. Not using a time-decaying penalty is effective in simulation with perfect dynamics, but fails on the actual system. This is because a strong disturbance can push the importance sampling trajectory far off the track, which results in most samples receiving the impulse cost and being rejected, this destabilizes the optimization. Including the time-decay term enables trajectories which stay on the track until the very end of the horizon to play a role in the optimization, while still enforcing a hard constraint like objective by rejecting trajectories that are immediately about to exit the track.

Speed Cost

The speed cost is a simple quadratic cost for achieving a desired forward speed:

$$\text{Speed}(\mathbf{x}) = (v_x - v_{\text{des}})^2 ,$$

where v_x is the longitudinal velocity in body-frame.

Stabilizing Cost

The stabilizing cost penalizes samples which exhibit extreme maneuvers that are known to result in undesirable behaviors (e.g. rollovers and spin-outs). This cost follows the track cost pattern where there is both a soft and hard cost. The stabilizing cost is:

$$\begin{gathered} \text{Stabilizing}(\mathbf{x}) = \zeta^2 + 10000 I\left(\{|\zeta| > .75\}\right) \\ \zeta = -\arctan\left(\frac{v_y}{\|v_x\|}\right), \end{gathered} \tag{6.4}$$

the term ζ is known as the side slip angle of the vehicle and measures the difference between the velocity vector of the vehicle and heading angle. Under normal driving conditions the side slip angle of the vehicle is zero. The stabilizing cost function provides a quadratic penalty for slip angles up to .75 radians (approximately 42 degrees), and then rejects any trajectories with a slip angle greater than 0.75 radians.

Results

All experiments were conducted at the Georgia Tech's Marietta Street Autonomous Racing facility. The facility consists of a roughly elliptical dirt track which is 30 meters across at its widest point. An image of the track with the robot is shown in Fig. 6.11. A ground station is set up in the center of the track which consists of an operating control system (OCS) laptop, runstop, and a base station GPS module to provide RTK corrections to the GPS module on-board the robot. The OCS laptop is used to remotely communicate with the robot and monitor its status over WiFi. However, all of the software required for autonomous operation runs on the vehicle's on-board computer. We want to emphasize that *all computations used for driving were performed on-board*. In our experiments, we tested 3 different speed targets (6 m/s, 8.5 m/s, and 11 m/s) for each of the two control methods with each of the two different dynamics models. Each setting was tested by maneuvering the vehicle clockwise and counter-clockwise around the track for 100 laps. Out of the 24 different scenarios, we were able to successfully collect 100 laps for 17 of the test scenarios, for a total of over 1700 laps around the track. This is equivalent to over 100 kilometers of driving data[1]. The other 7 settings resulted in controllers that were too reckless or unstable, so we were unable to complete those trials in their entirety.

Each lap was classified as either a success, a failure, or invalid if the cause of a failure was external to the controller. The controller is not the only part of the system that can cause a failure, much more common are state estimator errors due to loss of the GPS signal. In addition, the first lap in each batch of data was discarded. This is because the starting lap has slightly different statistics than the other laps, due to the vehicle accelerating up from zero velocity.

[1]The data from these experiments is publicly available at the AutoRally project page: https://autorally.github.io

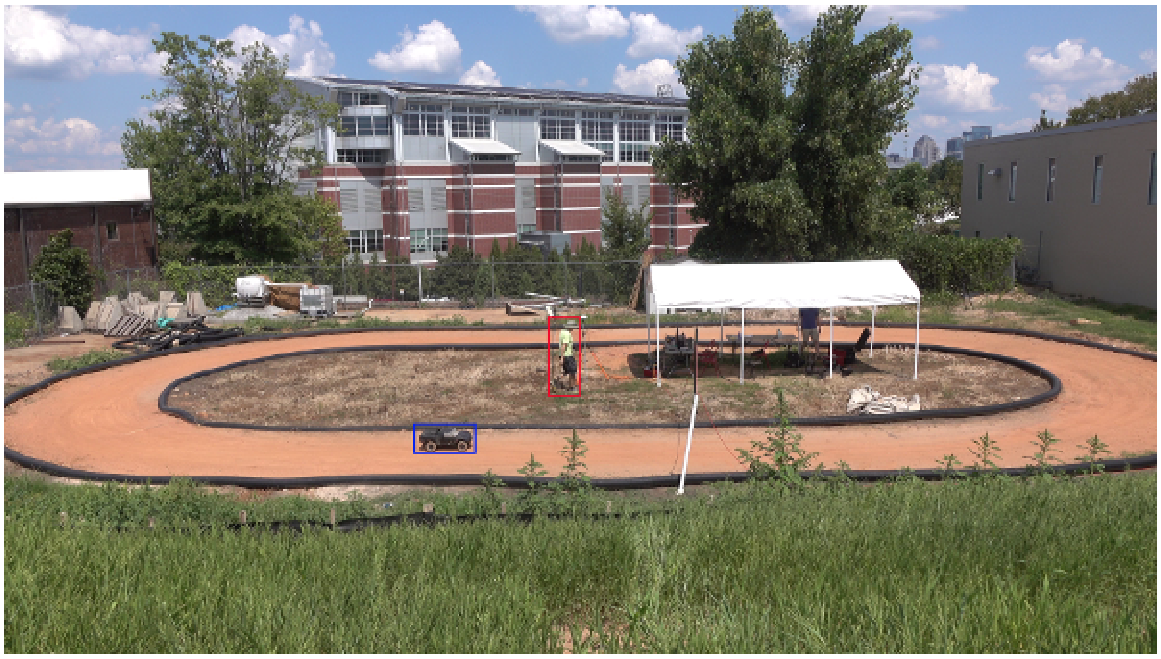


Figure 6.11: Experimental setup at the Georgia Tech Autonomous Racing Facility. All of the state estimation and control software is run on-board the robot itself, making it fully-autonomous and self-contained.

Overall Performance

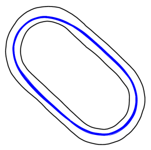
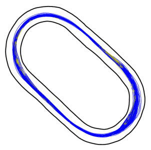
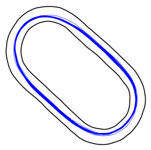
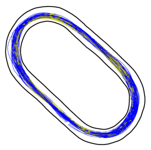
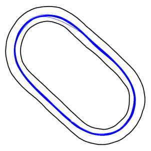
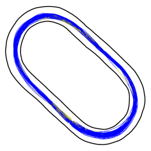
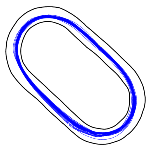
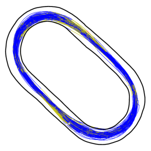
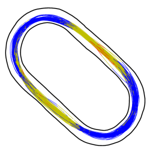
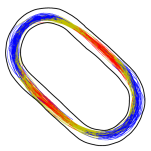
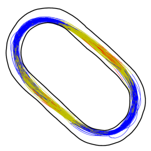
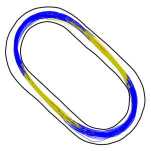
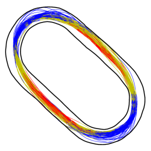
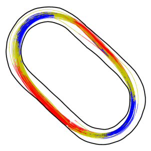
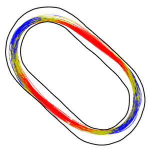
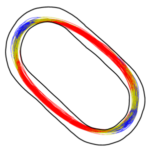
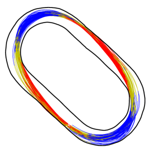
Table 6.3 shows lap time, success rate, and speed statistics for each of the tested settings, and Table 6.4 shows the raw trajectory traces overlayed onto the track for all of the runs at each setting. The vehicle's behavior differed significantly depending upon the choice of algorithm (MPPI or CEM-MPC), the dynamics model (basis function or neural network), and the speed target (6m/s, 8.5m/s, or 11m/s).

At the 6 meter per second target, the MPPI controllers all perform very consistently, albeit conservatively. Using both the basis function and neural network model the controller navigates the vehicles around the track at speeds varying from just over 1 m/s to a maximum of 5.77 m/s. This keeps the vehicle below the friction limits of the track and vehicle system, which means the car does not slide. The performance of MPPI with the neural network is remarkably consistent, especially from a stochastic controller, as the 100 laps in both counter-clockwise and clockwise have extremely low variance from lap to

Table 6.3: MPPI and CEM-MPC Performance Statistics

Method	Lap Success Rate	Average Lap Time (s)	Speed Range m/s
MPPI-NN 6 m/s CC	100%	$16.98 \pm .32$	$1.99 - 4.96$
CEM-NN 6 m/s CC	100%	$12.61 \pm .26$	$2.86 - 6.63$
MPPI-BF 6 m/s CC	100%	$18.42 \pm .21$	$1.28 - 5.65$
CEM-BF 6 m/s CC	83.16%	$11.82 \pm .43$	$0.39 - 6.90$
MPPI-NN 6 m/s C	100%	$16.03 \pm .22$	$2.58 - 4.78$
CEM-NN 6 m/s C	100%	$12.58 \pm .25$	$2.33 - 6.28$
MPPI-BF 6 m/s C	100%	$16.00 \pm .37$	$1.97 - 5.77$
CEM-BF 6 m/s C	97.30%	$12.14 \pm .31$	$1.85 - 7.17$
MPPI-NN 8.5 m/s CC	100%	$11.78 \pm .26$	$1.84 - 7.5$
CEM-NN 8.5 m/s CC	91.20%	$10.74 \pm .41$	$1.60 - 8.39$
MPPI-BF 8.5 m/s CC	89.10%	$11.30 \pm .70$	$1.16 - 7.71$
CEM-BF 8.5 m/s CC	<50%	N/A	N/A
MPPI-NN 8.5 m/s C	100%	$12.16 \pm .33$	$2.09 - 7.46$
CEM-NN 8.5 m/s C	85.42%	$10.83 \pm .55$	N/A
MPPI-BF 8.5 m/s C	89.00%	$9.81 \pm .31$	$4.22 - 9.72$
CEM-BF 8.5 m/s C	<50%	N/A	N/A
MPPI-NN 11 m/s CC	100%	$9.27 \pm .30$	$3.46 - 9.06$
CEM-NN 11 m/s CC	66.32%	$8.42 \pm .23$	$4.00 - 10.01$
MPPI-NN 11 m/s C	76.00%	$10.09 \pm .35$	$1.47 - 9.37$
CEM-NN 11 m/s C	<50%	N/A	N/A

Table 6.4: Trajectory traces of MPPI and CEM-MPC during testing runs.

	MPPI-NN	CEM-NN	MPPI-BF	CEM-BF
CC-6ms				
C-6ms				
CC-8.5ms				
C-8.5ms				
CC 11ms				
C 11ms				

🟥 Over 7.0 m/s
🟨 5.5 - 7.0 m/s
🟦 Under 5.5 m/s

lap. Figure 6.12 shows the 100 laps collected at the 6 m/s target with the MPPI algorithm and neural network model traveling counter-clockwise. The cross-entropy method using the neural network model performs perfectly at this settings as well, and actually achieves significantly faster speeds than the MPPI algorithm. However, the cross-entropy method cannot be as discriminative as the MPPI controller, since MPPI can discard, by assigning a low weight, any trajectories that leave the track. In contrast, the cross-entropy method must accept the top 20% of trajectories into its solution. Even at the slow setting of 6 m/s the cross-entropy method has a failure with the basis function model, and only achieves an 83.16% (79/95 successful laps) success rate going clockwise around the track using the basis function model. The trajectory traces for each of the different settings at the 6 m/s target are displayed in the first two rows of Table 6.4.

At the 8.5 meter per second target, differences between the algorithms and models become more apparent. MPPI using the neural network model is the only method which performs flawlessly going both clockwise and counter-clockwise at this setting. MPPI is still more cautious than the cross-entropy method, and achieves maximum speeds about 1 m/s slower than the target velocity. This is consistent with the performance at the 6 m/s target. The speed ranges also start to become dramatic at this setting, for instance MPPI with the neural network model (in the counter-clockwise direction) had speed ranges between 1.84 m/s and 7.5 m/s during the approximately 100 laps collected at that setting. The 1.84 m/s speed at this setting was not typical, but was the result of the vehicle encountering a large disturbance (due to a bump in the track), and it demonstrates the controller's ability to make drastic mode shifts in order to react to disturbances. Also, note that the MPPI controller no longer maintains the extremely tight variance that it did at the 6 m/s target, as the speed cost at 8.5 m/s reduces the relative importance of staying near the center of the track.

The cross-entropy method has significant difficulty at this setting. At the 8.5 m/s setting using the basis function model, the algorithm was unable to complete the trials at a

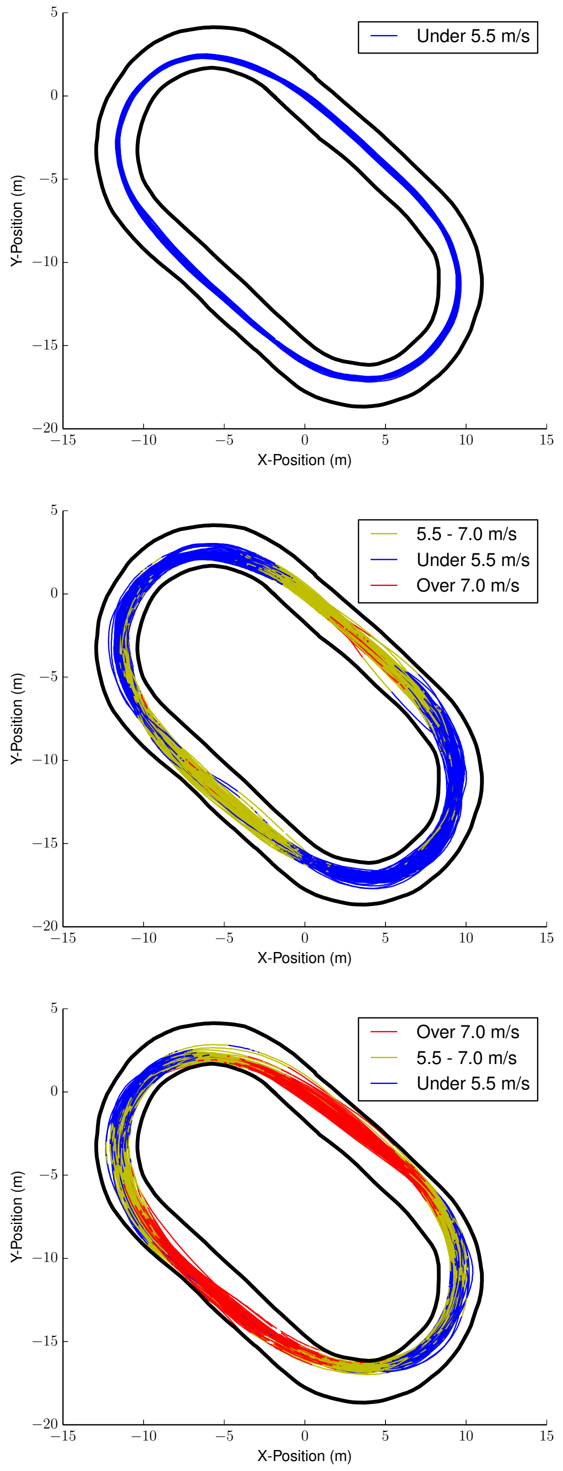


Figure 6.12: Top: Trajectory traces of MPPI controller using the neural network model at the 6 m/s (Top), 8.5 m/s (Middle), and 11 m/s (Bottom) target velocity. Each figure represents 100 laps, or approximately 6 kilometers of driving. Direction of travel is counter-clockwise.

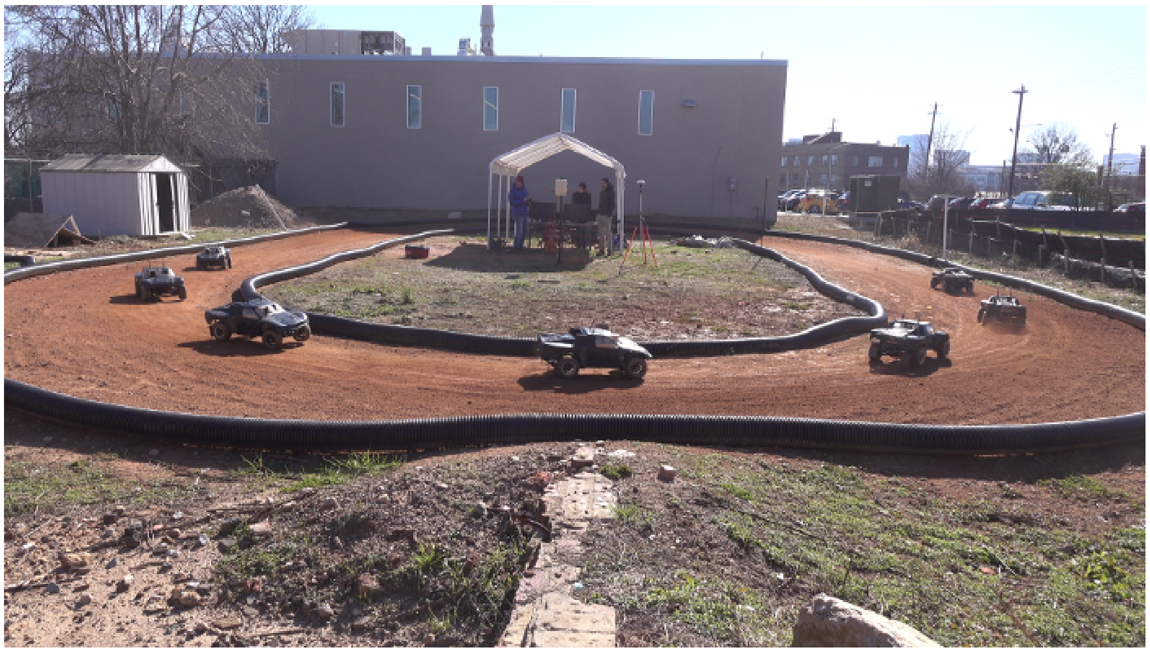


Figure 6.13: Time-lapse image of the vehicle making a cornering maneuver. Notice how the front left wheel is off the ground as the vehicle enters the turn.

satisfactory rate, and was generally unsafe to run. The issue was that it disregarded the track boundaries, and collided with either the inside or outside track barrier on over 50% of the trials. The cross-entropy method still maintained a high success rate using the neural network model.

At the fast speed target of 11 m/s only the MPPI controller traveling in the counter-clockwise direction is able to complete all 100 laps without a significant violation of the track boundaries. Note that actually achieving the 11 m/s target is very difficult on this track (a skilled human driver is not able to consistently achieve a top speed of 11 m/s). The top speed achieved by MPPI at this setting is 9.06 m/s, or approximately 20 miles per hour. The cross-entropy method is still faster than MPPI, however the success rate of the algorithm for actually completing laps is very low (only 66.32%). Figure 6.12 shows the trajectory traces for the MPPI controller with the neural network traveling counter-clockwise. The trajectory traces come extremely close to the barrier, but do not collide with it.

Cornering Maneuvers

The most difficult part of aggressive driving, from a control perspective, is cornering. Successful cornering requires significantly reducing speed, and then applying the throttle as the vehicle exits the turn. Failing to reduce speed or applying the throttle too soon can result in spin-outs (uncontrolled high heading rate). Using the neural network model, the MPPI controller decreases speed by performing a small slide into turns. This is a delicate maneuver that often results in the left front inside wheel momentarily lifting off the ground. Once the vehicle straightens out, the controller hits the throttle and resumes sliding slightly as it exits the turn and enters the straight. Figure 6.13 shows a time lapse image of the vehicle entering the turn at the 11 m/s target, and fig. 6.15 shows the same maneuver from an overhead perspective.

Another common behavior of the controller is counter-steering (steering right to turn left) when exiting turns. This is a behavior which requires taking advantage of the non-linear dynamics of vehicle, and is only effective at high speeds. Figure 6.14 shows this behavior as the car exits a turn on one of the 11 m/s trials.

Robustness to Model Error

In order to navigate the vehicle around the track, the controller has to be robust to modeling error (Table 6.1). Figure 6.16 shows how the predicted model differs from reality around a typical turn at the 11 m/s target with the neural network model. Going counter-clockwise the model is able to accurately predict out to the 2 second time horizon. However, in the clockwise direction the model incorrectly predicts over-steer when in fact the vehicle under-steers.

This behavior is likely due to a mismatch in track conditions between the clockwise training data and the conditions of the track during the tests: the track was much drier (see Fig. 6.11) during the system identification data collection than during the testing runs (see Fig. 6.13). A drier track means a lower coefficient of friction, which makes it easier to turn

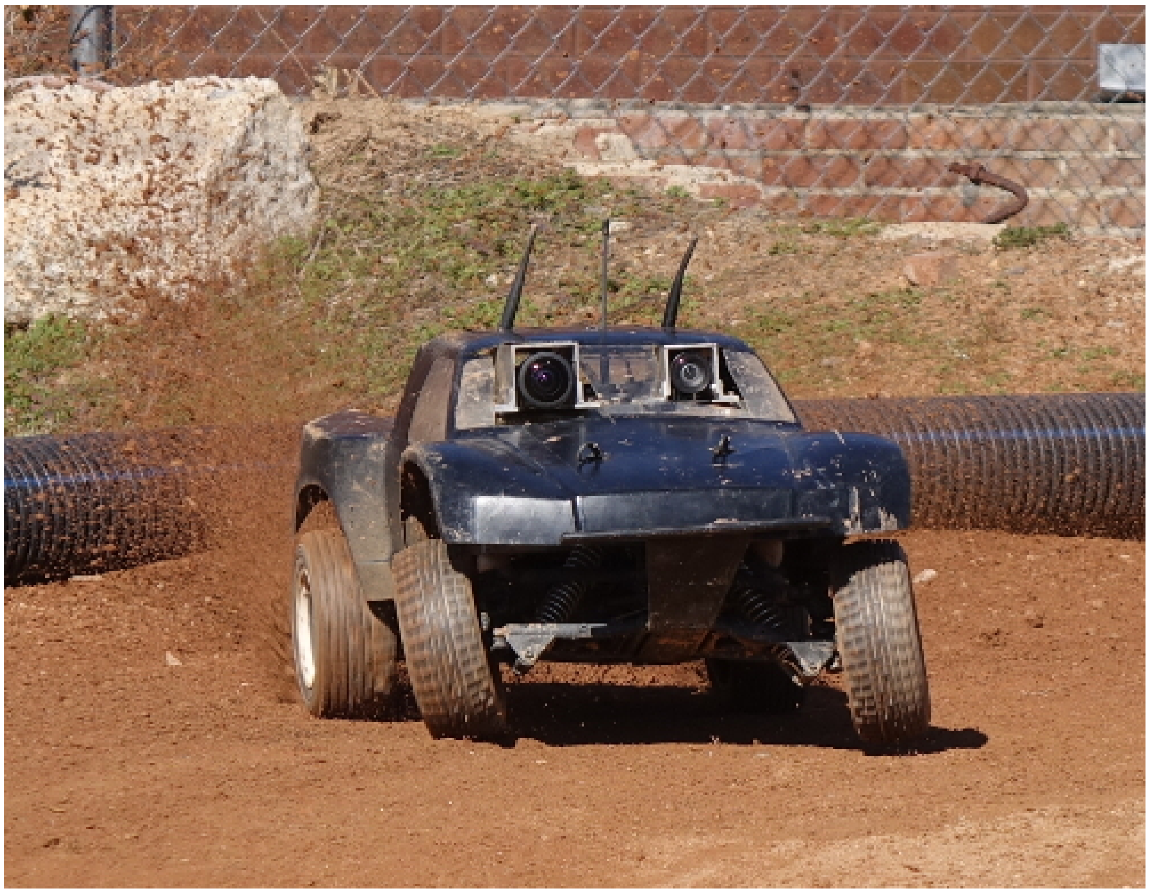


Figure 6.14: Example of the AutoRally executing a controlled powerslide while cornering while under the control of MPPI.

by sliding the back end of the vehicle. However, with a wet track, the coefficient of friction is higher, so the back end of the vehicle is harder to break free and attempting a similar maneuver results in under-steer. It is unclear why the counter-clockwise direction was not similarly effected due to this mismatch.

Disturbance Recovery

In addition to systemic modeling error, the dirt track provides a source of strong disturbances which cannot be modeled using our state representation. This includes environment effects like holes and lose patches of dirt. This became especially difficult during the 8.5 m/s test runs when dry weather and hundreds of consecutive laps around the track made it very difficult to drive. Despite these effects, the neural network model with MPPI was

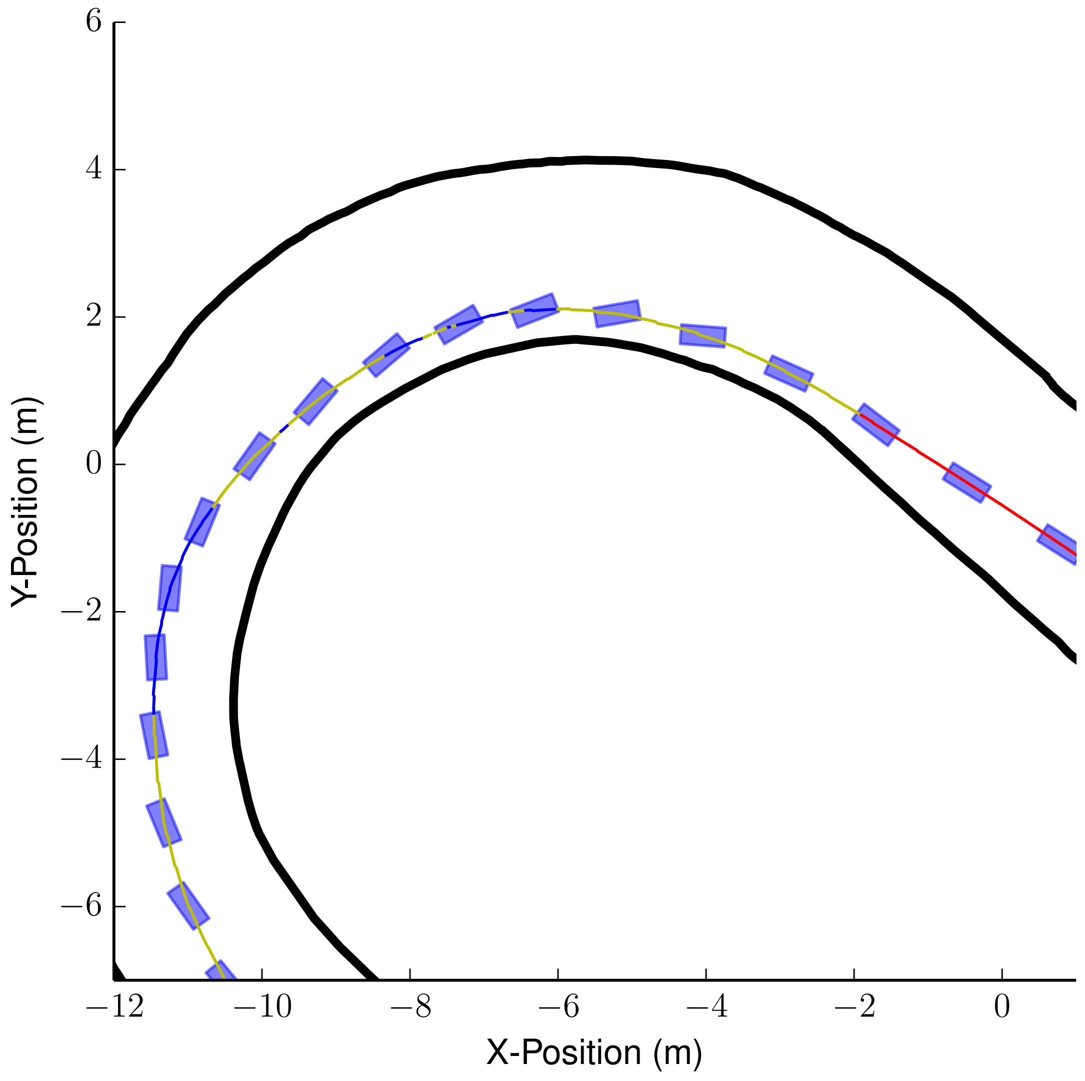


Figure 6.15: Trajectory and heading trace of cornering maneuver. The direction of travel is counter-clockwise, heading indicator is not to scale.

able to successfully complete all 100 laps. Figure 6.17 shows a series of images which demonstrate the effect of these disturbances on the vehicle.

Failure Modes

Although the neural network model generally outperformed the basis function model, and MPPI generally outperformed the CEM-MPC in terms of success rate. All of the methods suffered some failures. With MPPI and the neural network, the only failures came from

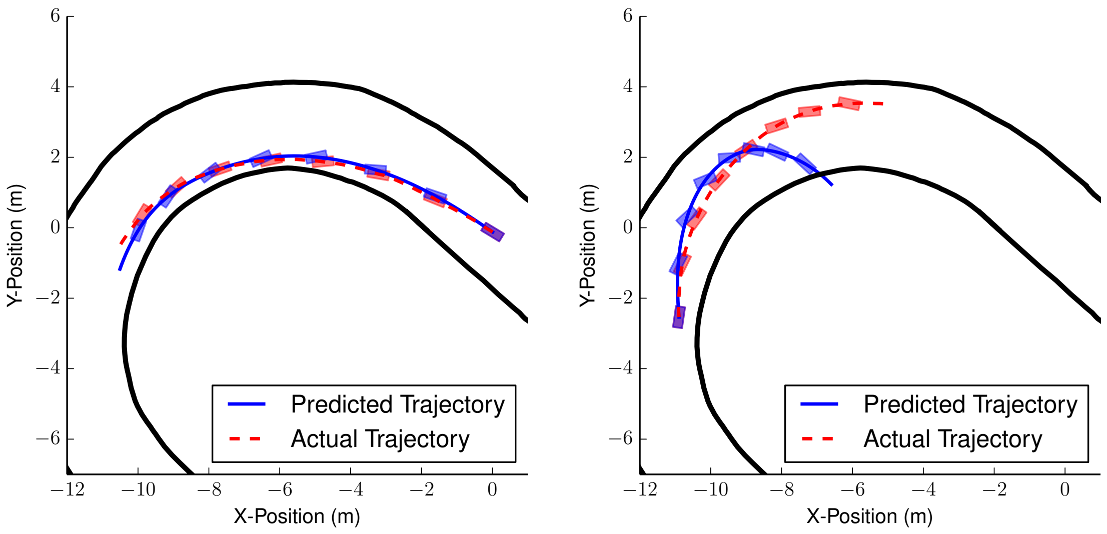


Figure 6.16: Neural network modeling error at 11 m/s target. Going counter-clockwise the model prediction is accurate, but clockwise the model predicts severe over-steer when it should have predicted under-steer. The predicted trajectory is generated by taking the applied input sequence from the data recording and running it through the neural net model starting from the same initial condition.

attempting to navigate the track clockwise at the 11 m/s target. The problem in this case was systematic modeling error which caused the vehicle to under-steer around the corners. Figure 6.18 shows all of the trajectories generated by the MPPI controller at this setting which failed. Notice that all of the trajectories fail in a similar manner, note that the track boundaries were pushed out a little bit so that we could continue collecting data even when the vehicle violated the boundary. In the case of cross-entropy, the failure come from not respecting the track boundary. Even when going clockwise with the neural network model, which is very accurate, the cross-entropy method consistently violates the track boundary. This is due to the sampling method used by cross-entropy, which allows trajectories into the sampling even if they violate the track constraint.

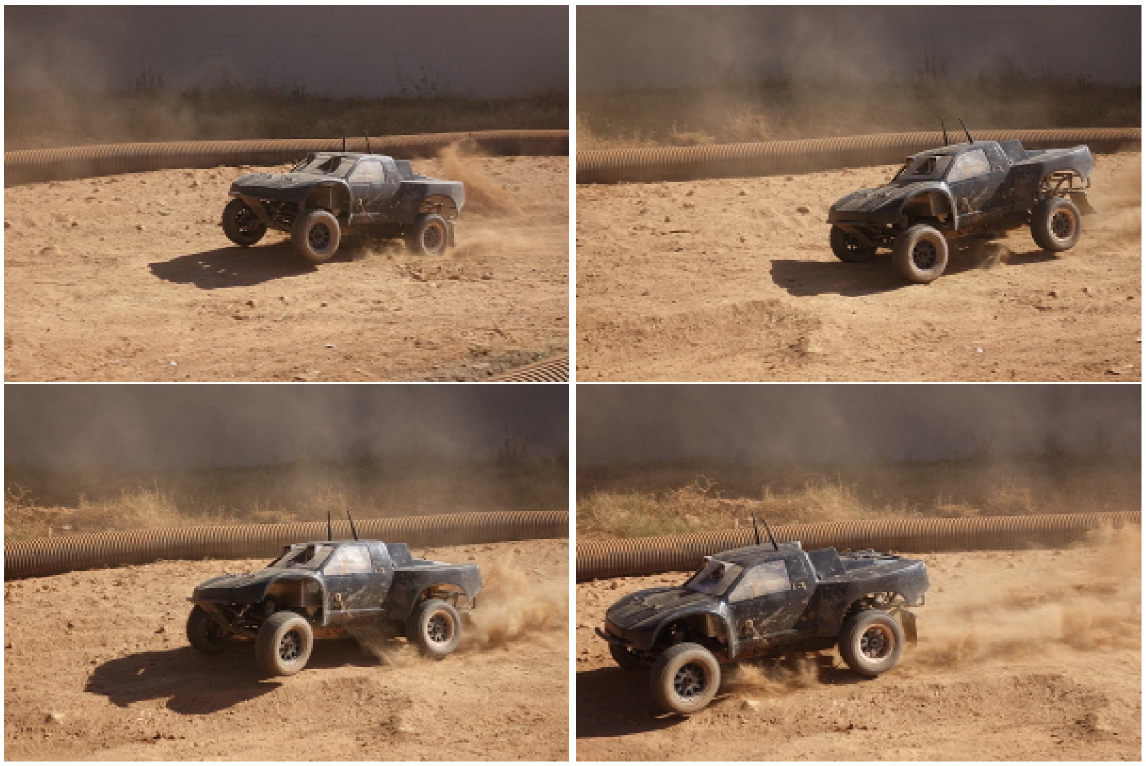


Figure 6.17: Disturbance rejection by the MPPI controller. The car hits a large hole in the track and the front and rear wheels leave the ground in alternating fashion while the vehicle is attempting to steer around the corner.

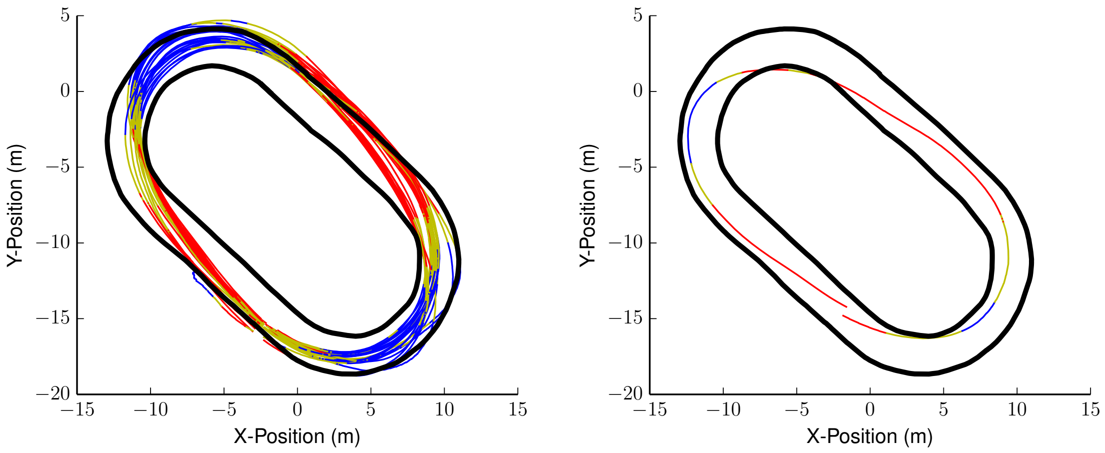


Figure 6.18: Failure mode of the MPPI algorithm. Speed setting is 11 m/s and the direction of travel is clockwise. The model under-estimates the amount of steering input required, which results in under-steer and collision with the barrier.

6.3 Discussion

In this chapter we have examined the performance of MPPI on a variety of simulation experiments and on the AutoRally platform. The simulation experiments demonstrate that MPPI can handle high-dimensional non-linear systems, and they demonstrate some of the advantages that MPPI has over gradient based methods when it comes to handling constraints (avoiding obstacles in the quadrotor case). Since MPPI can use costs with impulse like penalties it can discriminate very finely between actions that do or do not violate constraints, which results in increased performance. This contrasts with MPC-DDP which needs finely tuned gradients in order to satisfy constraints.

In the real-world AutoRally experiments, the ability to satisfy constraints (in this case staying on the track) was once again the differentiating factor between MPPI and CEM-MPC. In this case, the difference in performance is not due to the difference between a gradient based vs. gradient free method, but due to the way MPPI and CEM-MPC computes the averaged update. CEM-MPC takes an unweighted average over the elite trajectories, which may include trajectories that violate constraints. MPPI, on the other hand, takes a weighted average which means that it can virtually eliminate trajectories that violate constraints from the optimization - so long as at least one constraint free trajectory is sampled.

If we return to the desired features of our controller: performance at the limits of handling, constraint satisfaction, and real-time computation. We see that MPPI can satisfy these three criteria, and it has an advantage over both the cross-entropy method and MPC-DDP when it comes to satisfying constraints. However, if we dig a little deeper into MPPI's performance it becomes apparent that MPPI's ability to satisfy task constraints still leaves something to be desired, especially considering its performance in the real-world on the AutoRally system.

In the simulation experiments the term in the cost function which was responsible for obstacle avoidance was of the form:

$$350\exp(-\frac{d}{12}) + 1000C.$$

Here d is the distance to the nearest obstacle, and C a indicator variable denoting if a crash occurred or not. The first term encourages the quadrotor to keep a cushion between itself and the obstacle, which is preferable since there is additional state-dependent noise in the system that the MPPI controller is not aware of. If the cushioning term is removed the overall performance remains largely the same, except the overall success rate drops slightly since the quadrotor starts traveling close enough to obstacles that an unfortunate state disturbance can cause a collision. If the constraint term is removed, the success rate for the system decreases dramatically. Also, as opposed to the soft cushioning term, we could have instead simply enlarged the radius of the obstacle and used purely an indicator function to achieve the same effect. This is how constraint satisfaction in MPPI is supposed to work: a very sharp (in this case discontinuous) function is the main source of constraint information, and an additional cushioning function can optionally be added in order to provide some insurance.

In the AutoRally experiments we initially tried to use a similar cost function for the quadrotor task:

$$h(p_x, p_y) + 10000I(\{h(p_x, p_y) > .99\}).$$

Using this type of cost function in simulations of the AutoRally (simulations where the model used by the controller is the model used in the simulation) works well. However, in the real world the controller occasionally makes inappropriate actions that result in a failure when using this type of cost function. The occurrence of these failures can sometimes be hard to observe because they only happen once every several laps when the vehicle is directed to drive fast. Nonetheless, they are a major problem. The underlying cause of these failures is that disturbances in the real world can cause almost the entire "spray" of trajectories to move off the track, at which point the sampling starts rapidly jumping between solutions instead of converging. Sometimes, especially at lower speeds, the controller is able to recover but occasionally this behavior in the underlying optimization will result in a series of uncoordinated actions being executed, causing a failure.

Our solution was to use a modified cost function:

$$\text{Track}(\mathbf{x}) = h(p_x, p_y) + .9^t \left(10000 I(\{h(p_x, p_y) > .99\})\right).$$

with a time decay on the penalty term for leaving the track. Although this is effective at enabling the vehicle to drive, it turns the paradigm that was so effective in the quadrotor case on its head: now the soft cost is being utilized as the main source of information which keeps the vehicle on the track, and the indicator type function exists in order to make last second aggressive maneuvers to avoid the boundary.

Using this combination of a soft cost and then a time-decaying hard penalty does provide a benefit. For instance, MPPI was much better at CEM-MPC at staying on the track. However, it nullifies the primary advantage that MPPI had against MPC-DDP in our earlier experiments: which was the ability to use a simpler cost function for satisfying constraints while simultaneously achieving better performance. Since this is the major benefit of MPPI over competing methods, it is essential that we find a way to overcome the issues which prevent us from using the simpler cost functions that we utilized earlier. In the subsequent chapters, we explore solutions to the underlying causes of this problem: the numerical instability of the sampling-based optimization, and mitigating modeling error.

CHAPTER 7
ROBUSTNESS IN SAMPLING BASED CONTROL ARCHITECTURES

The results in the previous chapter demonstrate the applicability of MPPI to solve challenging control problems. However, in order to get results with MPPI on a real-world system, we needed to create and tune soft constraint terms in the cost function. Although it is possible to make this work for certain situations, from a general constraint satisfaction perspective it is unsatisfactory. Tuning soft constraints is a tricky task, and it is generally difficult or impossible to systematically tune soft constraints and determine when or how they might fail. Particularly troubling is the fact that changing the relative values of the speed cost and track cost can have unpredictable results. For instance, if the track cost is set too low relative to the speed cost the car tends to intentionally drive off the track. Moreover, if extensive tuning of the cost function is still required for constraint satisfaction, it is unclear what advantage MPPI has over current methods. For instance, an MPC-DDP controller can perform the same task of driving fast around the nearly elliptical Marietta street track [72], using a carefully tuned cost function.

Ideally, instead of tuning soft constraints, we could directly specify constraints or use cost terms that easily specify constraint violations. In the information theoretic and stochastic HJB-PDE frameworks, directly encoding hard state constraints is difficult. However, what worked effectively with MPPI, in simulation, was encoding constraints using weighted indicator type functions:

$$\sum_{i=1}^{N} w_i \mathbb{1}_{C_i}(\mathbf{x}). \tag{7.1}$$

Where the value of w_i is set very high so that if a cost above w_i is ever achieved we know that a constraint has been violated. Later, it will become clear that utilizing these types of costs is not ideal, because cost terms of this form are not Lipshitz continuous, and therefore not amenable to analysis which makes providing any sort of guarantee about performance difficult. However, looking at some of the benefits and issues of using these types of cost terms is instructive. Therefore, in this chapter we will consider cost terms consisting of these weighted indicator functions for encoding constraints.

MPPI can, in principle, handle cost functions of the form of Eq. (7.1). Unfortunately, in practice, when using cost functions with such sparse objective information, sampling-based methods like MPPI are brittle and prone to failure in the face of unexpected disturbances. The fundamental problem is that sampling-based methods, while gradient free, are still iterative local search methods. This is simply because it is intractable to fully sample high dimensional state spaces. As a result, it is still possible for MPPI to become stuck in local minima, and the local minima it finds is highly dependent on its initialization.

In an MPC setting, initialization takes the form of a warm start. If $(\mathbf{u}_0, \mathbf{u}_1, \dots \mathbf{u}_{T-1})$ is the current control solution, then $(\mathbf{u}_1, \mathbf{u}_2, \dots)$ will be used to initialize the next iteration. Implicit in this procedure is the assumption that the actual next state is close to the predicted next state. In the presence of disturbances, that assumption may fail. Figure 7.1 gives an example. In the worst case, a disturbance can move push the solution towards a bad (high cost) local minima. In the case of optimizing with weighted indicator functions, bad

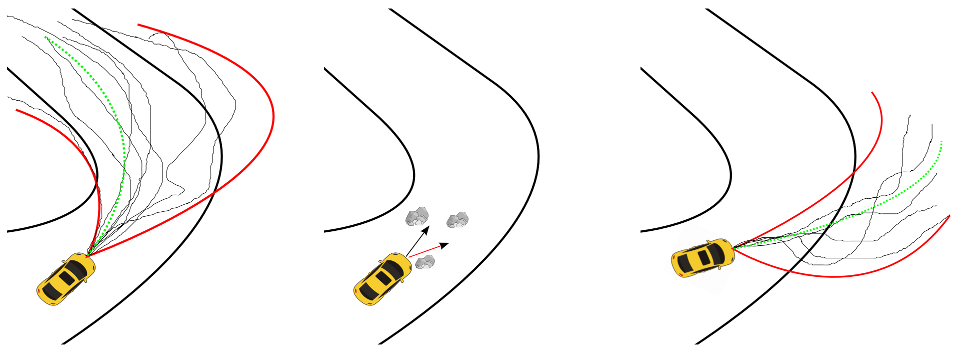


Figure 7.1: Effect of disturbances on sampling based MPC. In (a), an autonomous vehicle has a good sampling distribution. In (b) the vehicle executes the control, but hits a disturbance, resulting in (c) the sampling distribution leads to high cost.

local minima are common since they have zero gradient everywhere, which can cause the algorithm to fail.

In this chapter, we investigate a solution to this robustness problem. The key idea is to augment a sampling based MPC method with an ancillary controller for disturbance rejection by utilizing Tube-MPC. Tube-MPC was originally developed as a way to guarantee robustness for constrained linear systems [73] in the presence of disturbances and was later extended to non-linear systems [74]. The original version of non-linear Tube-MPC, which we utilize in this work, consists of two model predictive controllers. The first controller, termed the nominal controller, attempts to solve the primary optimal control problem for an idealized nominal state, and the second controller, called the ancillary controller, has the goal of rejecting disturbances in order to keep the actual system state close to the nominal state. We also make use of some of the suggestions from [75], in order to improve the performance of the method. This is only one variant of Tube-MPC, and there are other non-linear versions which utilize a separate feedback controller instead of another MPC controller [76, 77] in order to devise an ancillary controller.

7.1 Tube-Based MPPI

Once again, we consider general discrete time non-linear systems, but this time with an extra disturbance on the state:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{F}(\mathbf{x}_t, \mathbf{u}_t + \epsilon_t)\Delta t + \mathbf{w}_t, \tag{7.2}$$

where $\mathbf{x} \in \mathbb{R}^N$ is the state, $\mathbf{u} \in \mathbb{R}^M$ is the control input, and the term $\epsilon \in \mathcal{N}(0, \Sigma)$ is a disturbance directly on the control input. The term $\mathbf{w}$ is an external disturbance, which exists due to a combination of modeling error and purely stochastic or unobserved environmental effects. The goal is to optimize systems with running costs in the standard path integral form:

$$\mathcal{L}(\mathbf{x}, \mathbf{u}) = q(\mathbf{x}) + \lambda \mathbf{u}^{\mathrm{T}} \Sigma^{-1} \mathbf{u}, \tag{7.3}$$

$$q(\mathbf{x}) = c(\mathbf{x}) + \sum_{i=1}^{N} w_i \mathbb{1}_{C_i}(\mathbf{x}). \tag{7.4}$$

The state dependent portion of the cost is composed of $c(\mathbf{x})$, and the weighted indicator function terms. The goal of the first portion of the cost is to encode some overarching directive to the robot (e.g. go a certain speed), and the second portion of the cost acts to encode constraints into the system.

The linear version of Tube-MPC utilizes a nominal controller, a nominal state, an ancillary controller, and the real system state. The nominal controller is able to select the initial nominal state (subject to it being nearby the actual system state) and the nominal solution, both of which are readily available via the solution of a quadratic program. The ancillary controller then takes the form of a simple linear feedback gain, which maintains the actual state of the system in a tube around the nominal state solution. In non-linear Tube-MPC, which is the basis for our approach, much of the convenience of the solution for the linear case is lost. However, the algorithmic structure and the end result remain the same.

We consider the augmented system:

$$\mathbf{x}^*_{t+1} = \mathbf{x}^*_t + \mathbf{F}(\mathbf{x}^*_t, \mathbf{u}^*_t)\Delta t, \tag{7.5}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{F}(\mathbf{x}_t, \mathbf{u}_t + \epsilon_t)\Delta t + \mathbf{w}_t. \tag{7.6}$$

These systems are identical, except that one is disturbed via noise, and the other is disturbance free. The nominal state and control are denoted by $\mathbf{x}^*$ and $\mathbf{u}^*$ respectively. The nominal controller is a non-linear model predictive controller, which can consider general costs and constraints, and it computes a solution $\{(\mathbf{x}^*_0, \mathbf{x}^*_1, \dots \mathbf{x}^*_{T-1}, \mathbf{x}^*_T), (\mathbf{u}^*_0, \mathbf{u}^*_1, \dots \mathbf{u}^*_{T-1})\}$. The nominal system is allowed to ignore system disturbances, so we can have $\mathbf{x}^*_0 \neq \mathbf{x}_0$, where $\mathbf{x}_0$ is the real state of the system. The role of the ancillary controller is to then track the nominal system state. We implement the ancillary controller using an MPC-DDP/iLQG controller, which solves a standard tracking problem.

Unlike the linear Tube-MPC case, in the non-linear case the nominal controller does not consider the initial nominal state as an input variable, however in certain instances, the nominal state can be reset back to the actual state. There are then 3 components of the Tube-MPC algorithm that we need:

i) A nominal controller.

ii) A method for setting the nominal state.

iii) An ancillary controller.

We will hereon refer to our method as Tube-MPPI.

Nominal Controller - Model Predictive Path Integral Control

For the nominal controller we use the MPPI controller developed in chapter 5. Besides the fact that the initial condition utilized may not be the real system state, there is another minor variation that needs to be done in order for MPPI to be suitable for usage as a nominal controller. The difference is that instead of directly returning the optimal control, the algorithm returns a control and state sequence that the ancillary controller attempts to track. This means that we are using the cost-weighted average as an optimal control sequence, which is potentially problematic given our previous observations about the difference between the optimal importance sampler and an optimal control sequence. This issue will be addressed in the next chapter, but for this chapter we will treat the optimal importance sampler as an optimal control sequence. An iteration of the modified algorithm is shown in Alg. 2.

```
Algorithm 2: Nominal Controller (MPPI)
Given: $\mathbf{F}, g$: Dynamics and clamping function;
$K, T$: Number of samples and timesteps;
$U$: Initial control sequence;
$\Sigma, \nu, \lambda, \gamma, \phi, q$: Cost functions/parameters;
SGF: Savitsky-Galoy convolutional filter;
while task not completed do
    // Nominal state instead of real state
    $\mathbf{x} \leftarrow Fn\_GetNominalState()$;
    /* Begin parallel block                                */
    for $k \leftarrow 0$ to $K-1$ do
        $\tilde{S}_k \leftarrow 0$;
        Sample $\mathcal{E}^k = \left(\epsilon_0^k \dots \epsilon_{T-1}^k\right)$, $\epsilon_t^k \in \mathcal{N}(0, \nu\Sigma)$;
        for $t \leftarrow 0$ to $T-1$ do
            $\mathbf{v}_t = \mathbf{u}_t + \epsilon_t^k$;
            $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{F}(\mathbf{x}, g(\mathbf{v}_t))\Delta t$;
            $\tilde{S}_k \mathrel{+}= \mathbf{q}(\mathbf{x}) + \frac{1}{2}\left(\gamma\left[\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right] + \lambda\left[(1-\nu)\epsilon_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right]\right)$;
        $\tilde{S}_k \mathrel{+}= \phi(\mathbf{x})$;
    /* End parallel block                                  */
    $\rho \leftarrow \min\{\tilde{S}_0, \tilde{S}_1, \dots \tilde{S}_k\}$;
    $\eta \leftarrow \sum_{k=1}^{K} \exp\left(-\frac{1}{\lambda}(S_k - \rho)\right)$;
    for $k \leftarrow 1$ to $K$ do
        $w_k \leftarrow \frac{1}{\eta}\exp\left(-\frac{1}{\lambda}(S_k - \rho)\right)$;
    for $t \leftarrow 0$ to $T-1$ do
        $U \leftarrow \mathrm{SGF} * \left(U + \sum_{k=1}^{K} w_k \mathcal{E}^k\right)$;
    // Simulate state sequence
    $X \leftarrow Fn\_Simulate(\mathbf{x}_0, U)$;
    return $(U, X)$;
```

Selecting the Nominal State

The key feature of Tube-MPPI differentiating it from standard MPPI is the usage of a nominal state as the initial condition. The role of the nominal state is to prevent MPPI from jumping into high-cost local minima, and the mechanism for selecting the nominal state is essential to the Tube-MPPI algorithm. In the original description of non-linear Tube-MPC, the nominal state is initially set equal to the actual state, and then it is simulated forward without ever receiving feedback from the actual system. This scheme has two primary drawbacks:

i) Using pure forward simulation means that the algorithm is completely reliant on the tracking ability of the ancillary controller. In cases where the ancillary controller fails, the nominal state and real state will quickly diverge resulting in a failure of the overall control scheme. In real world experiments on the AutoRally system, purely relying on forward simulation and the ancillary controller to keep the ancillary and actual states close together never succeeded in completing an entire lap.

ii) Most disturbances are not catastrophic to the nominal controller, and in those cases, it is preferable to let feedback enter the nominal controller in order to re-plan from the actual system state. In some cases, disturbances can even be beneficial. If the fortunate situation occurs where a disturbance improves the current solution, then that disturbance should be taken advantage of, not rejected.

In [75] a modification to Tube-MPC is suggested whereby two copies of the nominal controller are run, one from the nominal state and one from the real system state. If the nominal controller finds a better solution using the real state of the system, then the superior solution is used, and the nominal state is reset back to the real system state before moving onto the next time-step. We propose a similar, albeit more relaxed version of this mechanism, where we accept the solution from the nominal controller using the real system state if the cost is less than the nominal state solution plus some threshold. The threshold is set as follows: let $\{C_{i_0}, C_{i_1}, \dots C_{i_M}\}$ denote the sets of constraints that are considered safety critical, then the minimum of $\{w_{i_0}, w_{i_1} \dots w_{i_M}\}$ is set as the threshold. This mechanism ensures that a disturbance can never push the solution of the nominal controller into a constraint region.

Ancillary Controller - MPC-DDP

The last component of the Tube-MPPI controller is an ancillary controller which solves a tracking problem in order to keep the actual system state within a tube centered about the nominal state. This is a standard tracking problem, where there is a small initial error and a quadratic cost, and there are numerous effective solutions. We elected to use an MPC version of differential dynamic programming (the iterative linear quadratic gaussian control (iLQG) variant), as in [21, 25]) as the ancillary controller, and we found that it provided good performance at a mild computational cost.

This algorithm works by iteratively linearizing the system dynamics, and quadratizing the cost function to create a linear-quadratic approximation of the control system. Then the approximate system is optimized for using standard tools from linear systems theory. After the optimal correction for the linear system is found, an optimization step is taken, and the state and control sequences are updated before re-approximating the system and taking another step. Usually the most difficult part of DDP is creating a cost function that is well conditioned for quadratization, however, for the tracking problem this is trivial since we have a purely quadratic cost to begin with. Although, we still need to define running state, terminal state, and control cost matrices (Q, P, and R respectively), which define how important the states are relative to each other.

Algorithm Summary

Now that we have descriptions of the three main components of the Tube-MPPI algorithm, we can give its full description. The steps in the algorithm are given in psuedo-code in Alg. 3. Initially, the nominal and real state are set equal to each other. Entering into the main loop, solution sequences for both the nominal and real state are computed. If the real state is below the given threshold, then the real state is set equal to the nominal state and the real solution is set equal to the nominal solution. Next, the tracking problem is solved, note that this is trivial if the “IF” condition immediately above was triggered. Lastly, state feedback is received and the control sequences are slid down one step in order to warm-start the next iteration.

Algorithm 3: Tube-MPPI

```
Given: $\mathbf{F}, g$: System dynamics;
$S$: State-sequence cost function;
$P, Q, R$: Terminal state, running state, and control cost matrices;
$\mathcal{T}$: Threshold for accepting solution from real state;
Input: $\mathbf{x}, \mathbf{x}^*$: Initial real and nominal state;
$U_{init}$: Solution initialization;
// Initialization
$U \leftarrow U_{init}$;
$U^* \leftarrow U_{init}$;
$\mathbf{x} \leftarrow Fn\_StateEstimator()$;
$\mathbf{x}^* \leftarrow \mathbf{x}$;
while Task Not Finished do
    // Compute solution from nominal and real state
    $U^*, X^* \leftarrow \text{MPPI}(U^*, \mathbf{x}^*)$;
    $U, X \leftarrow \text{MPPI}(U, \mathbf{x})$;
    // Condition for accepting real state
    if $S(U, \mathbf{x}) \leq S(U^*, \mathbf{x}^*) + \mathcal{T}$ then
        $\mathbf{x}^* \leftarrow \mathbf{x}$ ;
        $U^*, X^* \leftarrow U, X$;
    /* Asynchronous Block: Run ancillary controller    */
    $\mathbf{u} \leftarrow Fn\_ILQG(\mathbf{x}, P, Q, R, U^*, X^*)$;
    $Fn\_SendToActuators(\mathbf{u})$;
    /* End Asynchronous Block                          */
    $\mathbf{x} \leftarrow Fn\_StateEstimator()$;
    $\mathbf{x}^* \leftarrow \mathbf{x}^* + \mathbf{F}(\mathbf{x}^*, g(\mathbf{u}_0^*))\Delta t$;
    // Slide control sequences in order to warm-start
    for $t \leftarrow 0$ to T-2 do
        $\mathbf{u}_t \leftarrow \mathbf{u}_{t+1}$;
        $\mathbf{u}_t^* \leftarrow \mathbf{u}_{t+1}^*$;
    $\mathbf{u}_{T-1} \leftarrow \mathbf{u}_{init}$;
    $\mathbf{u}_{T-1}^* \leftarrow \mathbf{u}_{init}$;
```

In our real-time implementation, which we used for the AutoRally, the two MPPI iterations run simultaneously on the GPU. Each instantiation of MPPI samples 1,200, 2 second long trajectories with a control frequency of 50 Hz. The nominal controller publishes solutions at a rate of 50 Hz. The ancillary controller runs asynchronously on a separate CPU thread, and performs optimization for the latest solution published by the nominal controller. The ancillary controller optimizes for a shorter time horizon (1 second), but runs at a faster frequency (100Hz).

7.2 Experimental Results

We tested the Tube-MPPI algorithm on a simulated linear point mass system, a simulated helicopter landing task, and both a simulated and real-world autonomous racing task. These experiments were chosen in order to highlight both the advantages and disadvantages of the proposed method. Through-out these experiments we refer to 3 different experimental conditions for MPPI:

i) **Baseline-MPPI** refers to MPPI operating on a system where there is no additional disturbance beyond the control dependent noise assumed in the MPPI framework.

ii) **Disturbance-MPPI** refers to the normal MPPI algorithm operating on a system with additional disturbances besides what has been assumed by the MPPI algorithm. Depending on the system, this additional noise takes the form of extra noisy control inputs or non-control dependent noise.

iii) **Tube-MPPI** refers to Alg. 3 operating on the same extra-noisy system as disturbance Disturbance-MPPI.

Note that the Baseline-MPPI method is impossible to implement in the real-world, since it requires a perfect description of the systems dynamics and noise distribution. We include it in the experiments in order to highlight the fundamental role that disturbances which violate MPPIs underlying noise assumptions have on the algorithm.

Illustrative Example: Point Mass System

This illustrative example concretely demonstrates the susceptibility of standard MPPI to catastrophic failure in the case of large disturbances and shows how Tube-MPPI can be an effective remedy. Consider the simple 2-D double integrator system:

$$\mathbf{x}_{t+1} = \begin{pmatrix} I_2 & I_2\Delta t \\ 0 & I_2 \end{pmatrix} \mathbf{x}_t + \begin{pmatrix} 0 \\ I_2\Delta t \end{pmatrix} (\mathbf{u}_t + \epsilon_t). \tag{7.7}$$

The goal is to move this system at a constant velocity while staying within a ring centered about the origin, this can be interpreted mathematically as[1]:

$$q(\mathbf{x}_t) = \left(\sqrt{v_x^2 + v_y^2} - v_{des}\right)^2 + 1000\left(\mathbb{1}_C(\mathbf{x})\right), \tag{7.8}$$

$$C' = \{\mathbf{x} \mid 1.875 < \sqrt{x^2 + y^2} < 2.125\}. \tag{7.9}$$

The level of noise that MPPI assumes present is $\epsilon \in \mathcal{N}(0, \Sigma)$ with $\Sigma = I$. For Disturbance-MPPI and Tube-MPPI the actual noise present in the system is set ten times higher at $\tilde{\Sigma} = 10I$.

Figure 7.2 shows the accumulation of the warm-start trajectories for each condition. These trajectories are obtained by simulating the control sequence used to warm-start MPPI at each iteration from the new initial state of the system. This trajectory defines the mean of the sampling distribution, so it is essential that it lies in a good region of the state-space. Baseline-MPPI performs perfectly, and the system state is always kept outside of C. With Disturbance-MPPI the increased noise in the system results in the state consistently entering C, and eventually diverging. The reason for this failure is that the system disturbances push the warm-start trajectories into poor regions of the state space, since sampling takes place locally around the warm-start trajectory, it then becomes likely that no trajectory that stays outside of C is sampled. With Tube-MPPI, the nominal control plan is prevented from entering the constraint set due to the condition for selecting the nominal state, this results in the importance sampling behaving similarly to the Baseline-MPPI condition, even with the increased system noise.

[1]Note that we have defined C through its complement

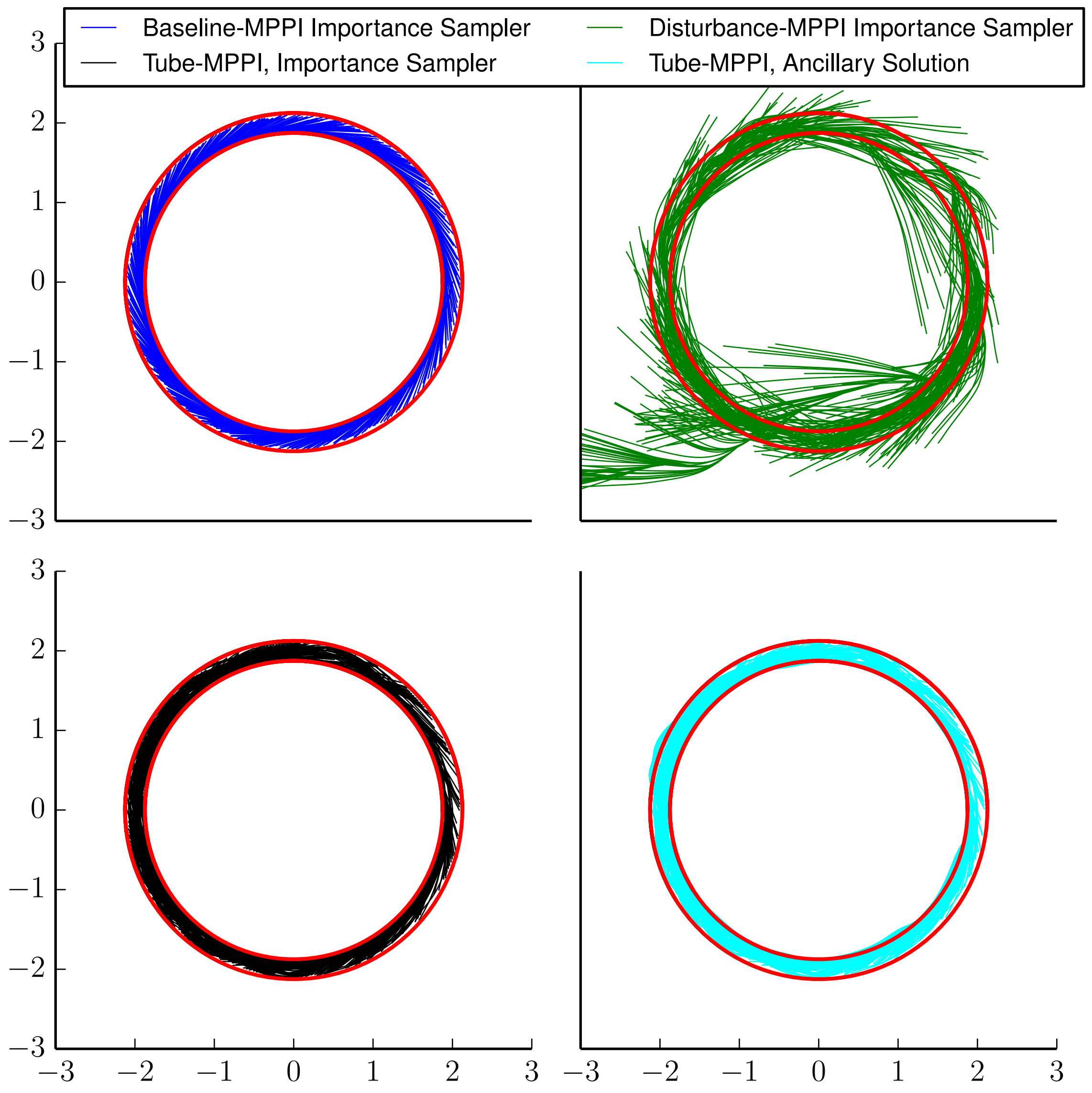


Figure 7.2: Point mass system results for Baseline-MPPI (top-left), Disturbance-MPPI (top-right), Tube-MPPI's nominal controller's importance sampling (bottom left), and the ancillary controller's solution (bottom right).

Simulated Helicopter Landing

In this simulated example we consider the task of landing a helicopter on a circular pad subject to Gaussian disturbances. This example demonstrates a few important aspects of the proposed approach. First, it shows how the proposed approach can effectively be applied to a system with non-trivial dynamics performing a task with many different constraints.

However, it also highlights one of the main downsides of non-linear Tube-MPC: the optimal control is not being directly computed. Instead of directly computing the optimal control, the control is a combination of the optimal control for the nominal system and the ancillary controller attempting to track that system. Even if the nominal system is operating perfectly, if the error between the nominal and real system is large then the system could still violate constraints. Therefore, in order for this type of approach to be effective a method for approximating the tube-size would be required.

For helicopter dynamics we use the non-linear model described in [23]. In this model the state space for this helicopter is position (x, y, z), orientation (ϕ, θ, ψ), body frame velocity (v_x, v_y, v_z), and body frame angular velocity (p, q, r). The control inputs are collective thrust u_τ, roll rate u_p, pitch rate u_q, and yaw rate u_r. The cost function for the landing task then takes the form:

$$
\begin{aligned}
&q(\mathbf{x}) = \mathbf{x}^{\mathrm{T}} Q \mathbf{x} + \sum_{i=1}^{8} w_i \mathbb{1}_{C_i}, \\
&C_1 = \{\mathbf{x} \mid \; (|\phi| > .15 \vee |\theta| > .1) \wedge z < -9.5\}, \\
&C_2 = \{\mathbf{x} \mid \; (\|(v_x, v_y, v_z)\| > 5 \vee v_z > 2.5) \wedge z > -8\}, \\
&C_3 = \{\mathbf{x} \mid \|(x, y)\| > 1.0 \wedge z > -8\}, \\
&C_4 = \{\mathbf{x} \mid x < -1.0\}, \;\; C_5 = \{\|(v_x, v_y, v_z\| > 12\}, \\
&C_6 = \{\mathbf{x} \mid z > \max(-.5\|(x, y)\| - 7.5, -50) \wedge \|(x, y\| > 1\}, \\
&C_7 = \{\mathbf{x} \mid |\phi| + |\theta| > .33\}, \\
&C_8 = \{\mathbf{x} \mid z > -7.5 \wedge x \notin C_1 \wedge x \notin C_2 \wedge x \notin C_3\}, \\
&w_1 = w_2 = w_3 = w_4 = 10000, \;\; w_5 = w_6 = 1000 \\
&w_7 = 100, \;\; w_8 = -10000.
\end{aligned}
$$

The first three terms direct the helicopter to land in the proper area with limits on the orientation and speed. The fourth term disallows the helicopter from over-shooting the landing area, the fifth and sixth terms prevent the helicopter from going too fast or using too aggressive of a combined roll and pitch angle, the seventh term directs the vehicle to stay above a certain glide-path, and the last term is a reward for successfully meeting all the landing criteria. The constraints are tightened to allow for some error in the final landing criteria.

Note that creating a cost function with a smooth gradient for this task, with either soft or hard constraints, would be *extremely challenging*! Many of the conditions have non-differentiable components (e.g. the $\max$ and norm operators) and composing a cost with eight different non-linear terms could easily result in local minima being created. In this case, specifying the cost function is easy and intuitive, and results in predictable behavior.

MPPI assumes that there is noise in the control inputs with:

$$\Sigma_{\mathbf{u}} = \text{Diag}(0.75, 0.125, 0.125, 0.125),$$

for Disturbance-MPPI and Tube-MPPI we inject additional noise into the system by increasing $\Sigma_{\mathbf{u}}$ and adding the additional disturbances for the velocities and orientation according to:

$$\Sigma_{\mathbf{u}} = (1.25)I_4, \;\; \Sigma_{v_x,v_y,v_z} = (1.25)I_3, \;\; \Sigma_{\phi,\theta,\psi} = (0.0125)I_3.$$

Figure 7.3 shows the results over 100 randomized trials for Baseline, Disturbance, and Tube-MPPI. Baseline-MPPI performs perfectly, and never violates any constraints while landing the helicopter. Disturbance-MPPI ends most trials with a satisfying landing. However, there are several large outliers that significantly miss the target region, this would be catastrophic on an actual helicopter system. The distribution for Tube-MPPI closely mirrors that for Baseline-MPPI, but with a higher covariance.

Figure 7.4 shows the resulting orientations for the trials with the highest pitch magnitude, in the case of Tube-MPPI the worst case pitch is still within an acceptable landing

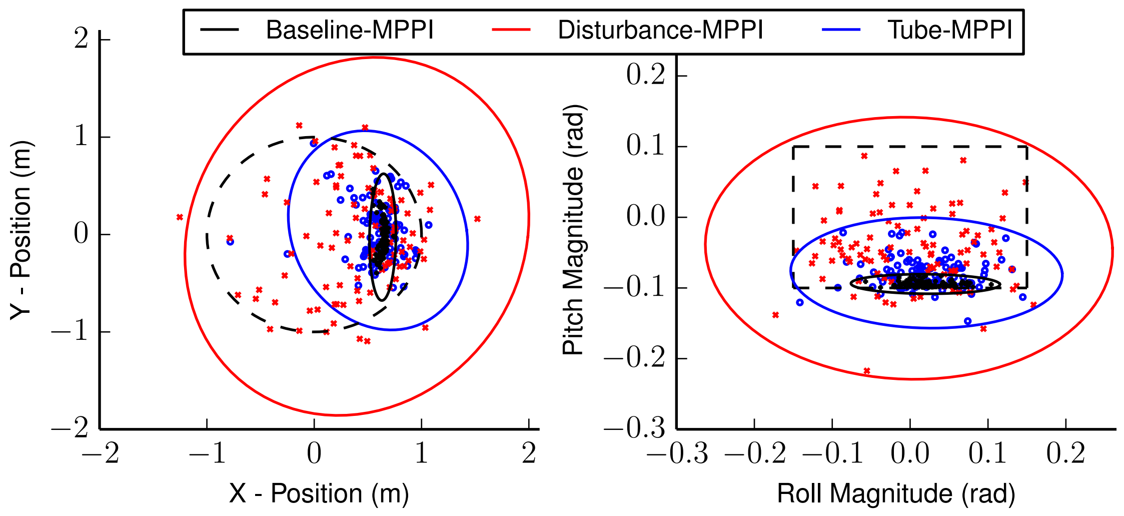


Figure 7.3: Helicopter landing positions (left) and orientations (right) for 100 random trials of Baseline-MPPI, Disturbance-MPPI, and Tube-MPPI with large noise. Dashed lines are specified landing area, colored line indicate 3-sigma bounds for a Gaussian distribution fitted to the 100 trials.

envelope, whereas with Disturbance-MPPI the result would be the tail contacting the platform before the wheels touched down.

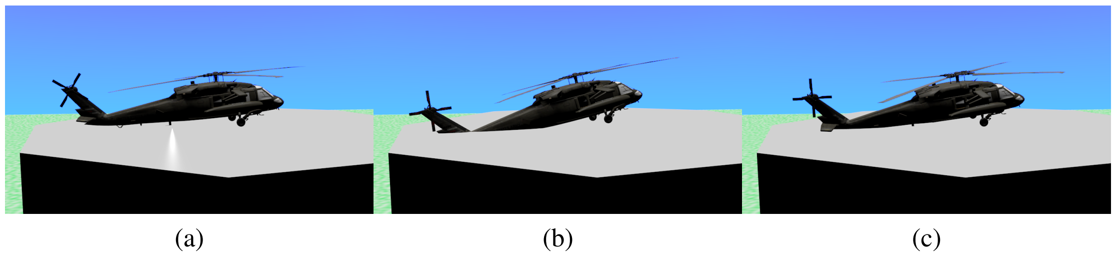


Figure 7.4: Results of helicopter landing experiment for ((a)) Baseline-MPPI, ((b)) Disturbance-MPPI (worst trial by pitch magnitude), ((c)) Tube-MPPI (worst trial by pitch magnitude).

Table 7.1 shows the mean, standard deviation, and worst case over the 100 trials for total distance from origin, roll angle, and pitch angle at touch-down. Although, Tube-MPPI does improve the overall system performance, in a real application it could still be problematic: since we do not have a precise bound for how large the tube can be, we cannot systematically tighten the constraints to ensure that the task is completed successfully.

Table 7.1: Helicopter landing statistics with Tube-MPPI

	Distance	Roll	Pitch
MPPI - Small Noise	0.66 +/- 0.025	0.02 +/- 0.03	-0.09 +/- 0.00
MPPI - Large Noise	0.77 +/- 0.25	0.0 +/- 0.08	-0.04 +/- 0.05
Tube - MPPI	0.69 +/- 0.15	0.02 +/- 0.05	-0.07 +/- 0.02

Simulated Autonomous Racing

In this simulation experiment, we used a Gazebo simulation of the AutoRally vehicle operating on a roughly elliptical track shown in Fig. 7.5. In this simulation environment, we do

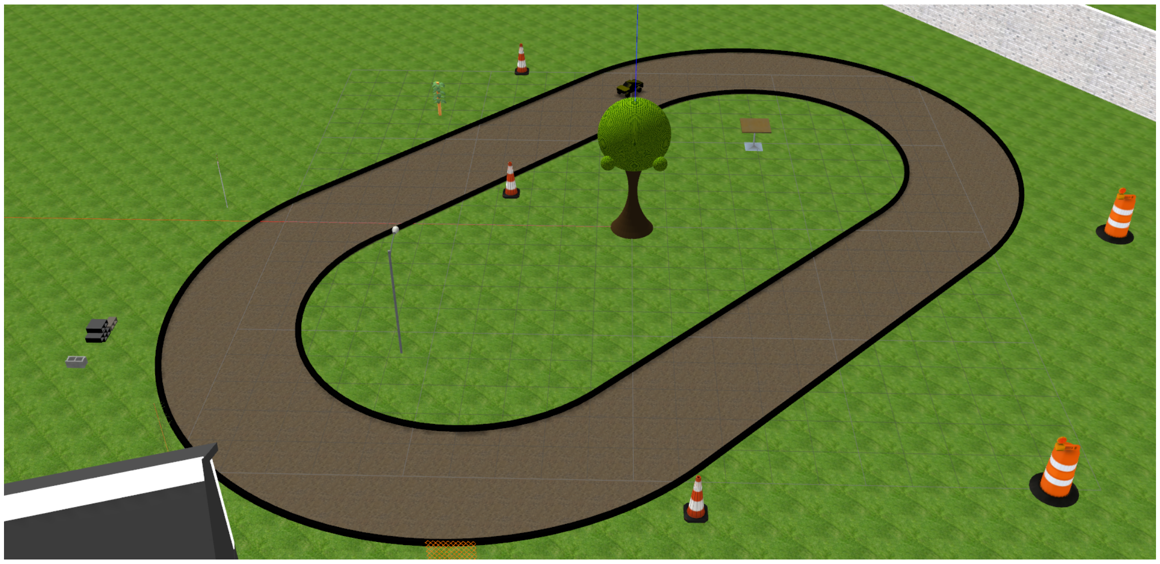


Figure 7.5: Gazebo simulation environment used for comparing Tuned-MPPI and Tube-MPPI.

not have access to the underlying model, so we fit one using a hybrid physics-neural networks approach. The state space of the vehicle is $\mathbf{x} = (x, y, \theta, r, v_x, v_y, \dot{\theta})$, and the model has the form: $\mathbf{F}(\mathbf{x}, \mathbf{u}) = \mathbf{x}_t + \left(W^{\mathrm{T}}\phi(\mathbf{x}, \mathbf{u}) + N(\mathbf{x}, \mathbf{u}; \theta)\right)\Delta t$ where W is a linear weight matrix, and $N(\mathbf{x}, \mathbf{u}; \theta)$ represents a neural network. This model is fit via a combination of linear regression and stochastic gradient descent, and there is a significant error between the learned model and the actual system dynamics. This error is the source of disturbances in this experiment.

Learning a model means that we cannot apply the Baseline-MPPI condition (since we cannot remove the extra disturbances), so instead we compare Tube-MPPI to a version of MPPI that has been extensively tuned with a cost function for this track. We refer to this as **Tuned-MPPI**. This is an important comparison, as it quantifies our ability to use an intuitive indicator function-based cost structure to approach a level of performance only achievable previously through extensive hand-tuning. The cost function for Tube-MPPI was set as:

$$\begin{aligned}
q(\mathbf{x}) &= \|v_x - v_x^{des}\|^2 + w_1 \mathbb{1}_{C_{\text{track}}}(\mathbf{x}) + w_2 \mathbb{1}_{C_{\text{slip}}}(\mathbf{x}), \\
C_{track} &= \left\{ \mathbf{x} \, \middle| (p_x, p_y) \notin \text{Points contained in Track} \right\}, \\
C_{slip} &= \left\{ \mathbf{x} \, \middle| \, \left\| \arctan^{-1}\left(\frac{v_y}{|v_x|}\right) \right\| > 1.25 \right\}, \\
w_1 &= w_2 = 10000.
\end{aligned}$$

The first component of the cost tells the vehicle to try and achieve a desired velocity, the second component tells the vehicle to stay on the track, and the last term tells the vehicle to keep the slip angle below 1.25 radians (70 degrees). In the case of Tuned-MPPI the cost function takes the same form as in chapter 6:

$$\begin{aligned}
q(\mathbf{x}) &= w_1 M(x,y) + w_2 \|v_x - v_x^{des}\|^2 + w_3 \tan^{-1}\left(\frac{v_y}{v_x}\right)^2 + \beta^t w_4 \mathbb{1}_{C_{\text{track}}}(\mathbf{x}) + w_5 \mathbb{1}_{C_{\text{slip}}}(\mathbf{x}), \\
& w_1 = 100,\ w_2 = 4.25,\ w_3 = 250,\ w_4 = 10000, \\
& w_5 = 10000,\ \beta = 0.9.
\end{aligned}$$

The first term $M(x, y)$ is a signed distance function for the set C_{track}. This term helps push the sampling distribution back towards the track if large disturbances are found. Note the presence of the time-decay on the constraint, which prevents the optimization within vanilla MPPI from becoming unstable and diverging.

For each experimental condition, the target speed was gradually increased by 1 m/s starting from 5 m/s until the algorithm could no longer consistently complete 100 laps

Table 7.2: Gazebo racing statistics for Tube-MPPI and Tuned-MPPI

	Avg. Lap Time	Max Speed	Max Slip
Disturbance -MPPI	11.87 +/- .47	5.22 +/- 0.06	0.04 +/- 0.04
Tuned - MPPI	8.33 +/- 1.05	7.53 +/- 0.04	0.09 +/- 0.15
Tube - MPPI	9.39 +/- 0.76	7.51 +/- 0.18	0.12 +/- 0.10

while staying on the track. For Tuned-MPPI the maximum target speed was 8 m/s, and for Tube-MPPI this was 9 m/s. For Disturbance-MPPI there was a massive performance drop-off, with the maximum target speed only reaching 5 m/s.

The performance statistics for each of the three trial conditions is shown in Table 7.2. Both Tube-MPPI and Tuned-MPPI achieve top velocities slightly over 7.5m/s, and sub 10 second lap times. However, since Tube-MPPI optimizes with slightly tightened boundaries it takes a longer overall path around the track, which results in longer lap times than Tuned-MPPI

AutoRally Experimental Results

Here we examine the real-world performance of Tube-MPPI on the AutoRally platform. This was the first set of autonomous vehicle experiments at the larger Autonomous racing facility at Cobb County Research Facility (CCRF), see Fig. 7.6. This track features a variety of different radius turns, and a long straight-away. An important detail of this track is that there are several areas where the boundaries for different segments of the track either touch or are very close to each other. This makes designing a smooth cost or constraint function based on a signed distance function difficult, since such a function would have local minima that would encourage the vehicle to drive over the track boundaries. However, using only weighted sums of indicator functions we obtain a very simple cost design based on a grid of binary values that represent the set of points on the track. The cost function for this task is the same as for the Gazebo simulation environment, where there is a term for speed, a term for staying on the track, and a term for avoiding excessive slip angle. The desired speed was set to 9 m/s, and we collected 12 laps around the test track, which is

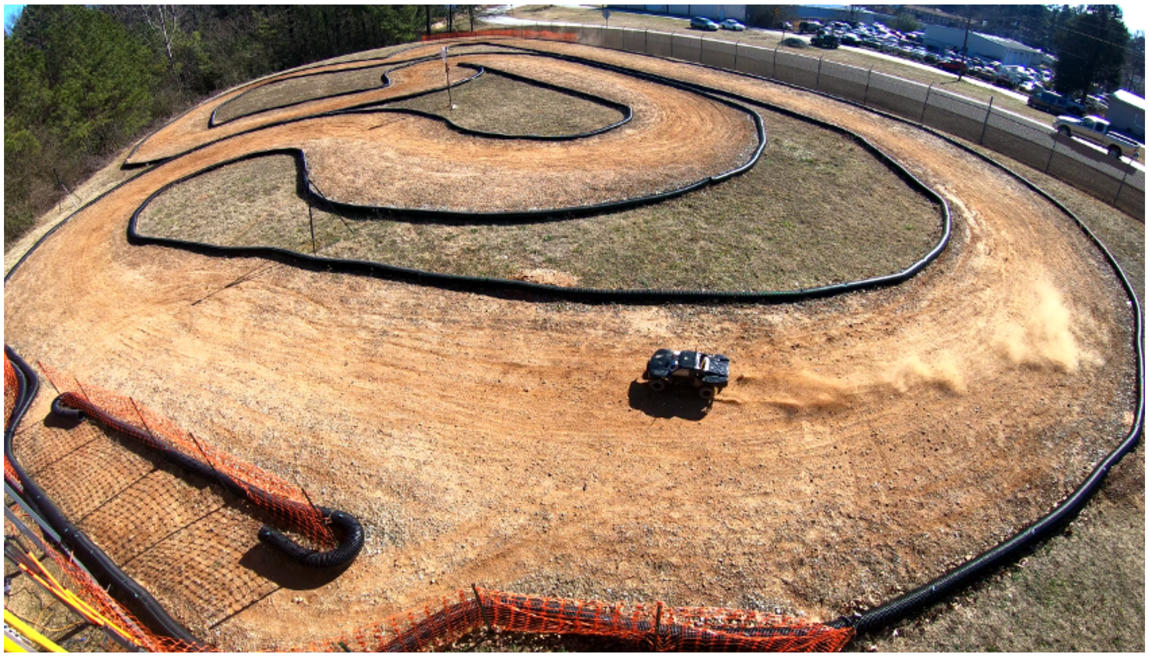


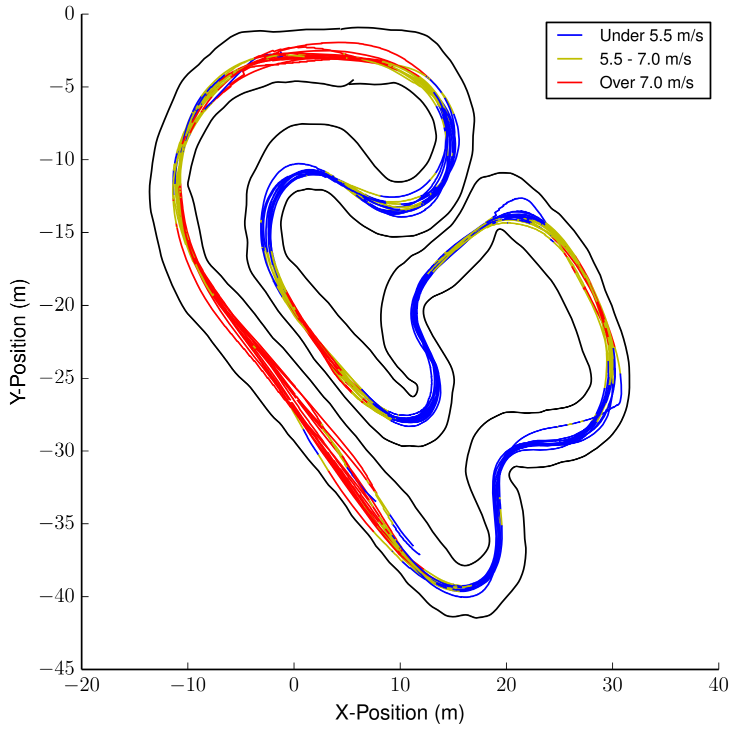
Figure 7.6: Test track for the AutoRally vehicle at the Cobb County Research Facility (CCRF).

Table 7.3: Racing Experiment Statistics

	Avg. Lap Time	Max Speed	Max Slip
Tube - MPPI	32.02 +/- 7.27	8.52 +/- -0.26	0.88 +/- 0.48

approximately 2 kilometers worth of driving data.

Figure 7.7 depicts the trajectory traces of the 12 trial laps around the track. This figure identifies one of the main benefits of using sparse indicator cost functions: since the vehicle is only penalized for leaving the track, it is free to use the entire track surface in order to achieve its primary goal of going fast. As a result, the position of the vehicle on the track does not follow the center line, but significantly varies depending on the upcoming track geometry (note that the overall direction of travel is clockwise). Also depicted in Fig. 7.7 is the state divergence as the vehicle navigates the track over the course of one lap, which indicates how well the ancillary control is performing. Overall, the magnitude of the positional state divergence stays relatively small compared to the overall track width. The mean state divergence for the lap shown is 14 centimeters, and the maximum is 47 centimeters.



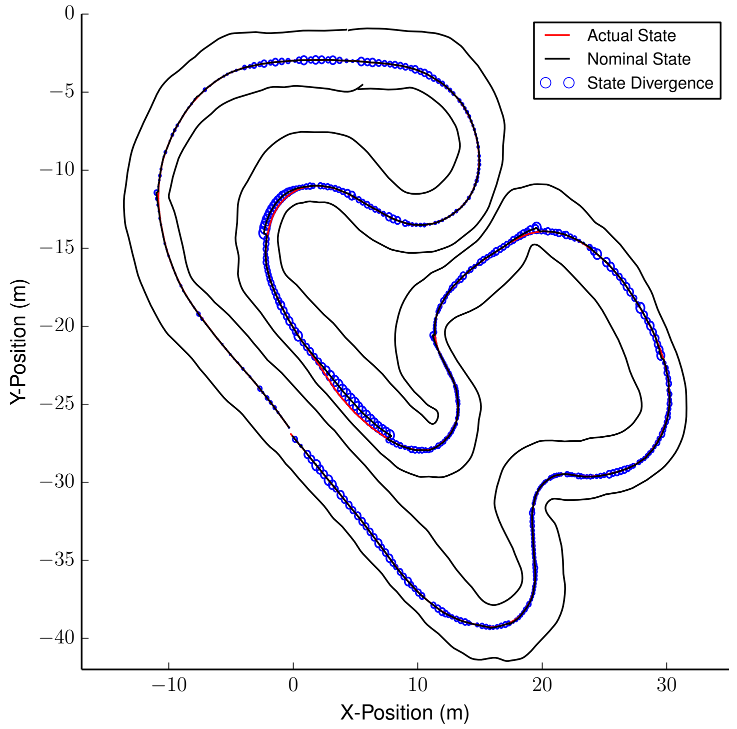


Figure 7.7: Top: Trajectory traces of test run on the CCRF track using Tube-MPPI. The desired speed during this run is 9 m/s. Direction of travel is clockwise. Bottom: Magnitude of the state divergence (distance between the nominal and actual state) over the course of a test lap at CCRF.

7.3 Discussion

The focus of this chapter has been on demonstrating some of the problems that arise when trying to utilize cost functions mimicking hard constraints in MPPI, and to show the pathway to a potential solution method in Tube-MPPI. By default, MPPI is brittle and prone to failure when operating in noisy real-world environments unless the cost function possesses a strong gradient signal. This nullifies one of the main advantages we observed when using MPPI in idealized simulations: which was its ability to use simple cost functions with very sharp slopes (or discontinuities) in order to act as constraints.

Tube-MPPI offers the pathway to a potential solution. The main idea with Tube-MPPI is to protect the importance sampling by using a noise free nominal state to compute a solution, and to then use a tracking controller to compute the actual control input. This enables the usage of simple functions for constraint satisfaction, but it introduces a couple of new problems. The first issue is that, for Tube-MPPI to be practically useful, a bound on the size of the tube has be estimated. Once this bound is estimated, the constraints need to be tightened in order to account for the potential divergence between the idealized and nominal state. While this is technically feasible, it introduces a couple of very complex operations (tube-size estimation and constraint tightening) that go against our goal of using simple cost functions for constraints. The other significant issue with the Tube-MPPI approach we have outlined here, is that it is not clear how compatible the underlying mathematics of MPPI are with the Tube-MPC framework. This goes back to the issue of treating the optimal importance sampling distribution as the optimal control sequence, although this did not seem to be an issue in any of the experiments here, this treatment is not always theoretically justified. Nevertheless, Tube-MPPI does serve to be a useful guide in the development of a far superior optimization scheme, which we describe in the next chapter.

CHAPTER 8
ROBUST MODEL PREDICTIVE PATH INTEGRAL CONTROL

In the previous chapter we saw the types of difficulties sampling based controllers like MPPI encounter when trying to utilize cost functions that incorporate terms acting as hard constraints. The fundamental issue is that, for a warm-started model predictive controller, a system disturbance can change the initial condition so that the seed trajectory lies in an undesirable local minima. If this occurs, then there is little hope that the optimization will recover - the system will immediately execute a control computed from the bad estimate bringing it even closer to failure. Tube-MPPI offers a potential solution method by optimizing with a nominal state that is protected from catastrophic disturbances, however it comes with some significant theoretical and practical drawbacks.

In this chapter we examine another solution technique, which we term augmented importance sampling. Augmented importance sampling is related to Tube-MPPI in that it also utilizes a fictitious nominal state in order to stabilize the importance sampling distribution. However, unlike Tube-MPPI the output of the augmented importance sampling method is an unbiased estimate of the optimal control. This means that we do not need to explicitly worry about bounds on the tube size in order to tighten constraints, which simplifies the overall algorithm design. Additionally, it is possible to obtain a theoretical bound on how much the solution can degrade from iteration to iteration when using the augmented importance sampler, which prevents sudden jumps into bad local minima. In general, we believe the augmented importance sampling method to be superior to the Tube-MPPI method in every area.

One might wonder, why present the Tube-MPPI method at all, given that the augmented importance sampler solves the same problem better. The reason that we have presented Tube-MPPI first is that it is a straightforward implementation of an existing technique, Tube-MPC, but simply utilizing MPPI as the nominal controller. In other words, the novel part of Tube-MPPI is the usage of MPPI as the nominal controller, as opposed to the method itself. This makes the approach easy to understand for those already familiar with the Tube-MPC literature. However, the augmented importance sampling technique is fundamentally new. Even though we borrow some ideas and terminology from Tube-MPC, it would not be accurate to characterize it as a Tube-MPC method. However, it will be helpful to have a firm grasp of Tube-MPPI in order to understand some of the mechanisms and design choices made in the augmented importance sampler.

Up until now, we have been rather informal about precisely what a "jump into a bad local minima" means and what observable effect it has. In plain language, what it means is that previously when we solved the problem, we had a good solution, and then after receiving state feedback and resolving the problem we had a much worse solution. Mathematically, this means that a jump into a bad local minima is characterized numerically by a large increase in the estimated value function. Recalling the relationship between the value function and free-energy established in chapter 4, we use the free-energy in this chapter as a surrogate for the value function. Therefore, we want to prevent large increases in the estimate of the free-energy, which means our goal is to provide a bound on how much the estimate of the free-energy can increase from time-step to time-step. In order to do this, we need to distinguish between two possible underlying causes of unbounded free-energy growth.

The first possible cause, which is our main concern, is that the true free-energy has not changed significantly, but the numerical estimate provides an incorrect result showing a high free-energy. The reason this type of numerical instability occurs is because the sampling-based optimization is reliant on the importance sampling distribution to compute a good estimate. Since the importance sampling distribution is defined by an open-loop control sequence, there is no guarantee that the importance sampling distribution will be useful from iteration to iteration since the initial condition is subject to disturbances encountered by the system.

The second possible cause is that the underlying free-energy really has significantly increased. For general stochastic non-linear dynamics and costs, there is no way to ensure that free-energy growth is bounded. In order to make the problem approachable we therefore must assume that both the costs and dynamics are Lipshitz continuous. This rules out the usage of the weighted indicator type functions that we utilized in the previous chapter, however we can still construct similar functions that are Liptshitz continuous and quickly ramp up to the maximum value as opposed to immediately jumping to one.

8.1 Augmented Importance Sampling

In this section, we describe an idealized scheme, which provides a bound on the growth of the free-energy function. The scheme we describe here is impractical to deploy, however we will use it as a guide towards implementing a practical approach later on. The key idea in this approach is to augment the importance sampling with an idealized nominal state, whose evolution we control. This is similar to Tube-MPC approaches, however, it differs in that nominal system only indirectly affects the solution through the importance sampler. This means that we are still directly computing the optimal control, which is *not* the case in Tube-MPC.

Nominal System

The nominal system is denoted $\mathbf{x}^*$, and has dynamics equivalent to the real system:

$$\mathbf{x}^*_{t+1} = \mathbf{x}^*_t + \mathbf{F}(\mathbf{x}^*_t, \mathbf{u}_t + \epsilon_t)\Delta t. \tag{8.1}$$

These are the dynamics used for the purposes of estimating the free energy, whose value depends on the cost function, probability induced by the system dynamics, initial condition, and inverse temperature. As before, the free energy is denoted as $\mathcal{F}(S, \mathbb{P}, \mathbf{x}^*, \lambda)$. We will also want to distinguish between the true free-energy, and the free-energy obtained through a Monte-Carlo estimate. Monte-Carlo estimates of the free-energy are actually random variables, which we can describe as:

$$\mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{x}^*, \lambda) = \mathcal{F}(S, \mathbb{P}, \mathbf{x}^*, \lambda) + e_M^B + \epsilon_M^V. \tag{8.2}$$

As the value returned by our Monte-Carlo estimator. Here e_M^B is the bias of the estimate, and ϵ_M^V is a random variable that depends on the noise distribution used to compute the estimate. The presence of the bias term may seem initially strange, since we use an unbiased importance sampler. Using an unbiased importance sampler means that as the number of samples goes to infinity the estimate converges to the correct estimate. However, when the value computed is a non-linear function of the sampled mean, it does not imply that repeatedly calling an estimator which uses a finite number of samples will result in a distribution of values whose mean is equal to the true value. We will assume that e_M^B is deterministic, but possibly large, and that ϵ_M^B is a small random number. This matches up well with empirical observations about fluctuations of the free-energy when making repeated estimates with the same importance sampler.

For the nominal system, we perform the same standard type of importance sampling as described in chapter 5, where the importance sampler is defined by an open loop control sequence:

$$U = \{\mathbf{u}_0, \mathbf{u}_1, \dots \mathbf{u}_{T-1}\},$$

but, instead of updating the nominal system state using a state estimate, we control the evolution of the system through the following update rule:

$$\mathbf{x}_{t+1}^* = \underset{\mathbf{z}}{\operatorname{argmin}} \, \|\mathbf{z} - \mathbf{x}_{t+1}\|, \quad \text{s.t} \;\; (\mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{z}, \lambda) < \alpha) \vee (\mathbf{z} = \mathbf{x}_t^*) \tag{8.3}$$

Where $\mathbf{x}_{t+1}$ is the state of the real system given by the state estimator. This update rule tries to move the nominal state as close as possible to the real system state, while ensuring that the estimate of the free energy from the nominal state stays below a threshold, α, which we set.

The condition on the right-hand side of the logical "OR" says that the current nominal state is always considered a valid nominal state. The rationale behind this is that (assuming we initialized the nominal state such that $\mathcal{F}_{MC} < \alpha$) the current nominal state is close to the desired threshold since a previously estimated free-energy value estimated it to be beneath the threshold. This ensures that there is always a feasible solution to (8.3).

Tracking Controller

Now, consider a trajectory generated by sampling from the system dynamics with the initial condition set as the nominal state. We would like to generate a corresponding trajectory sample from the real system state, which closely tracks the nominal sample. In order to do this we assume that there exists a feedback controller which, for any given sequence of control and disturbance inputs, drives the real system state to the nominal state exponentially fast. We thus have:

$$\mathbf{x}_{t+1}^* = \mathbf{x}_t^* + \mathbf{F}(\mathbf{x}_t^*, \mathbf{u}_t + \epsilon_t)\Delta t, \tag{8.4}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{F}\left(\mathbf{x}_t, \mathbf{u}_t + \epsilon_t + k(\mathbf{x}_t, \mathbf{x}_t^*)\right)\Delta t, \tag{8.5}$$

$$\|\mathbf{x}_{t+1} - \mathbf{x}_{t+1}^*\| \leq \gamma^t \|\mathbf{x}_0 - \mathbf{x}_0^*\|, \quad 0 < \gamma < 1. \tag{8.6}$$

The disturbance, ϵ, is shared between the nominal and real system, so it can be treated as an additional deterministic control input by the feedback controller. This is only possible because we are only going to use the feedback controller to generate samples in simulation.

For non-linear systems it can be very difficult to design a global stabilizing tracking controller. However, for a completely controllable linear system one such option is to use a controller of the form:

$$k(\mathbf{x}, \mathbf{x}^*) = K(\mathbf{x} - \mathbf{x}^*) \tag{8.7}$$

where K is chosen in order to make the closed loop system asymptotically stable. In practice, we will utilize locally linear approximations of the system dynamics in order to get a feedback controller which is effective in a local region of the state space.

Importance Sampler

With the definition of the nominal system, and the tracking controller, we can now define the augmented importance sampler. The idea behind the augmented importance sampler is to use the system augmented by the nominal state ((8.4) - (8.5)) to generate trajectory samples. Recall that an importance sampler biases the distribution in order to achieve a more efficient estimate of the expectation. In order to compute an unbiased estimate it is necessary to compute an importance sampling weight (Radon-Nikodym derivative). Therefore, in order to define a valid importance sequence we need:

i) A method for generating a random variable

ii) A formula for computing the importance sampling weights

In the augmented importance sampling framework, samples are generated by simulating the combined system forward:

$$\begin{aligned}
\mathbf{x}_{t+1}^* &= \mathbf{x}_t^* + \mathbf{F}(\mathbf{x}_t^*, \mathbf{u}_t + \epsilon_t)\Delta t, \\
\mathbf{x}_{t+1} &= \mathbf{x}_t + \mathbf{F}\left(\mathbf{x}_t, \mathbf{u}_t + \epsilon + k(\mathbf{x}_t, \mathbf{x}_t^*)\right)\Delta t.
\end{aligned}$$

Where the feedforward control, $U = \{\mathbf{u}_0, \mathbf{u}_1, \dots \mathbf{u}_{T-1}\}$, comes from the importance sampler for the nominal system. Although we simulate both the nominal and real system forward, *we only utilize the real system trajectories for computing the optimal control*. The nominal system trajectories are just a tool which provide input to the feedback controller in the evolution of the real system.

Next we need to derive a formula for the importance sampling weights. This is relatively straightforward. Since ϵ appears linearly in the combined control input, $\mathbf{u} + \epsilon + k(\mathbf{x}_t, \mathbf{x}_t^*)$, we can view the combined control input as a Gaussian random variable with mean: $\mathbf{u} + k(\mathbf{x}_t, \mathbf{x}_t^*)$.

Lemma 2. *Let $\mathbb{Q}_A$ denote the probability distribution defined by sampling from the augmented system dynamics, and let $\mathbb{P}$ be the distribution defined by the uncontrolled system. Then the importance sampling weight (Radon-Nikodym derivative) takes the form:*

$$\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}_A} = \exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}\left(\mathbf{u}_t + k(\mathbf{x}_t, \mathbf{x}_t^*)\right)^{\mathrm{T}}\Sigma^{-1}\left(\mathbf{u}_t + k(\mathbf{x}_t, \mathbf{x}_t^*)\right) + \epsilon_t^{\mathrm{T}}\Sigma^{-1}\left(\mathbf{u}_t + k(\mathbf{x}_t, \mathbf{x}_t^*)\right)\right).$$

Proof. We start by writing the weight as the ratio of two Gaussian distributions:

$$\underbrace{\frac{\mathrm{d}\mathbb{Q}_A}{\mathrm{d}\mathbb{P}}}_{=\star} = \frac{\exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}\left(\mathbf{v}_t - \mathbf{u}_t + k(\mathbf{x}_t, \mathbf{x}_t*)\right)^{\mathrm{T}}\Sigma^{-1}\left(\mathbf{v}_t - \mathbf{u}_t + k(\mathbf{x}_t, \mathbf{x}_t^*)\right)\right)}{\exp\left(-\frac{1}{2}\mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{v}_t\right)},$$

We can simplify this by combining the exponential terms and canceling:

$$\star = \exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}\left(\mathbf{u}_t + k(\mathbf{x}_t, \mathbf{x}_t^*)\right)^{\mathrm{T}}\Sigma^{-1}\left(\mathbf{u}_t + k(\mathbf{x}_t, \mathbf{x}_t^*)\right) + \mathbf{v}_t^{\mathrm{T}}\Sigma^{-1}\left(\mathbf{u}_t + k(\mathbf{x}_t, \mathbf{x}_t^*)\right)\right),$$

but, we can re-write $\mathbf{v}_t$ in terms of zero-mean noise ϵ, so $\mathbf{v}_t = \mathbf{u} + k(\mathbf{x}_t, \mathbf{x}_t^*) + \epsilon$. Incorporating this into the importance sampling weight yields:

$$\star = \exp\left(\frac{1}{2}\sum_{t=0}^{T-1}\left(\mathbf{u}_t + k(\mathbf{x}_t, \mathbf{x}_t^*)\right)^{\mathrm{T}}\Sigma^{-1}\left(\mathbf{u}_t + k(\mathbf{x}_t, \mathbf{x}_t^*)\right) + \epsilon_t^{\mathrm{T}}\Sigma^{-1}\left(\mathbf{u}_t + k(\mathbf{x}_t, \mathbf{x}_t^*)\right)\right).$$

Notice that the sign of the first term has been flipped. For un-biasing the expectation estimated with samples from $\mathbb{Q}_A$, we need the inverse of this derivative $\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}_A}$, this is computed simply by flipping the sign inside the exponential. So we have:

$$\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}_A} = \exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}\left(\mathbf{u}_t + k(\mathbf{x}_t,\mathbf{x}_t^*)\right)^{\mathrm{T}}\Sigma^{-1}\left(\mathbf{u}_t + k(\mathbf{x}_t,\mathbf{x}_t^*)\right) + \epsilon_t^{\mathrm{T}}\Sigma^{-1}\left(\mathbf{u}_t + k(\mathbf{x}_t,\mathbf{x}_t^*)\right)\right)$$

Which is the desired result. $\square$

Bounding Free-Energy Growth

We now have a method for generating samples using the augmented importance sampling system, and a formula for the importance sampling weight. Next, we can examine the consequences of utilizing it to compute the free-energy, and show how it can be used to bound the growth in free-energy. Recall the definition of free-energy:

$$\mathcal{F}(S,\mathbb{P},\mathbf{x}_0,\lambda) = -\lambda\log\left(\mathbb{E}_{\mathbb{P}}\left[\exp\left(-\frac{1}{\lambda}S(V,\mathbf{x}_0)\right)\right]\right),$$

by using the standard importance sampling trick we can write this as:

$$\mathcal{F}(S,\mathbb{P},\mathbf{x}_0,\lambda) = -\lambda\log\left(\mathbb{E}_{\mathbb{Q}_A}\left[\exp\left(-\frac{1}{\lambda}S(V,\mathbf{x}_0)\right)\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}_A}\right]\right).$$

The cost function $S(V,\mathbf{x}_0)$ is the cost of the real system's trajectory for a given disturbance input. This cost takes the form:

$$S(V,\mathbf{x}_0) = \phi(\mathbf{x}_T) + \sum_{t=0}^{T-1} q(\mathbf{x}_t),$$

but we can re-write this in terms of the nominal system state as:

$$S(V,\mathbf{x}_0) = \phi(\mathbf{x}_T^* + e_T) + \sum_{t=0}^{T-1} q(\mathbf{x}_t^* + e_t),$$

where $e_t = \mathbf{x}_t - \mathbf{x}_t^*$. Now, if we assume that both q and ϕ are Lipshitz continuous, with Lipshitz constants L_ϕ and L_q respectively, we have:

$$S(V) = \phi(\mathbf{x}_T^* + e_T) + \sum_{t=0}^{T-1} q(\mathbf{x}_t^* + e_t) \leq \phi(\mathbf{x}_T^*) + \sum_{t=0}^{T-1} q(\mathbf{x}_t^*) + L_\phi \|e_T\| + \sum_{t=0}^{T-1} L_q \|e_t\|. \quad (8.8)$$

Using Eq. (8.6), along with standard results from the geometric series, we have:

$$S(V, \mathbf{x}_0) \leq S(V, \mathbf{x}_0^*) + \left(L_\phi \gamma^T + L_q \frac{1-\gamma^T}{1-\gamma} \right) \|\mathbf{x}_0 - \mathbf{x}_0^*\|. \quad (8.9)$$

Since both the exponential and logarithm functions are monotonically increasing, we can insert this inequality into the expectation to get:

$$\mathcal{F} \leq -\lambda \log \left(\mathbb{E}_{\mathbb{Q}_A} \left[\exp\left(-\frac{1}{\lambda} S(V, \mathbf{x}_0^*) - \frac{1}{\lambda} \left(L_\phi \gamma^T + L_q \frac{1-\gamma^T}{1-\gamma} \right) \|\mathbf{x}_0 - \mathbf{x}_0^*\| \right) \frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}_A} \right] \right),$$

and then we can pull the constant terms outside of the expectation, so that we have:

$$\mathcal{F} \leq -\lambda \log \left(\mathbb{E}_{\mathbb{Q}_A} \left[\exp\left(-\frac{1}{\lambda} S(V, \mathbf{x}_0^*) \right) \frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}_A} \right] \right) + \left(L_\phi \gamma^T + L_q \frac{1-\gamma^T}{1-\gamma} \right) \|\mathbf{x}_0 - \mathbf{x}_0^*\|.$$

This implies the following relationship between the free-energy of the nominal and real system:

$$\mathcal{F}(S, \mathbb{P}, \mathbf{x}_0, \lambda) \leq \mathcal{F}(S, \mathbb{P}, \mathbf{x}_0^*, \lambda) + \left(L_\phi \gamma^T + L_q \frac{1-\gamma^T}{1-\gamma} \right) \|\mathbf{x}_0 - \mathbf{x}_0^*\|. \quad (8.10)$$

This shows that the value function is bounded by the value function for the nominal system, plus a term depending on the steepness of the cost function, the distance between the nominal and real state, and the rate of convergence achieved by the tracking controller. This inequality also holds if we replace the true free-energy values by the estimated free-energy values (as long as the same noise profiles are used to generate both estimates), since the expectations can be replaced with summations over a finite number of samples and the derivation will still hold. Therefore, we can write:

$$\mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{x}_0, \lambda) \leq \mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{x}_0^*, \lambda) + \left(L_\phi \gamma^T + L_q \frac{1-\gamma^T}{1-\gamma} \right) \|\mathbf{x}_0 - \mathbf{x}_0^*\|. \tag{8.11}$$

Now let us assume that, at some point, our Monte-Carlo estimator had returned an estimate below the threshold for the current nominal state: $\mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{x}_0^*, \lambda) < \alpha$. We thus have:

$$\mathcal{F}(S, \mathbb{P}, \mathbf{x}_0, \lambda) + e_M^B + \epsilon_M^V < \alpha,$$

for some value of ϵ_M^V. Recalling our assumption that ϵ_M^V is small, we can bound it by some value E_M^V, and we then have that any new estimate must satisfy:

$$\mathcal{F}_{MC} < \alpha + 2E_M^V. \tag{8.12}$$

So, the worst that the nominal free-energy will be is α plus twice the expected amount fluctuation in the estimator. We thus have the bound:

$$\mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{x}_0, \lambda) \leq \alpha + \left(L_\phi \gamma^T + L_q \frac{1-\gamma^T}{1-\gamma} \right) \|\mathbf{x}_0 - \mathbf{x}_0^*\| + 2E_M^V. \tag{8.13}$$

This gives an upper bound on the estimate of the free-energy from the current real system state.

Now, given the current estimate of the free-energies at the nominal and real states (denoted $\mathbf{x}_0$, and $\mathbf{x}_0^*$), we would like to provide a bound for what the free-energy estimate at the next real state will be (denoted as $\mathbf{x}_1$). Using Eq. (8.13), we know that:

$$\mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{x}_1, \lambda) \leq \alpha + \left(L_\phi \gamma^T + L_q \frac{1-\gamma^T}{1-\gamma} \right) \|\mathbf{x}_1 - \mathbf{x}_1^*\| + 2E_M^V \tag{8.14}$$

Now, we need to examine $\|\mathbf{x}_1 - \mathbf{x}_1^*\|$. In the worst case, $\mathbf{x}_1^*$ will be the same as $\mathbf{x}_0^*$, so we have:

$$\|\mathbf{x}_1 - \mathbf{x}_1^*\| \leq \|\mathbf{x}_1 - \mathbf{x}_0^*\|,$$

but the real system state evolves according to:

$$\mathbf{x}_1 = \mathbf{F}(\mathbf{x}_0, \mathbf{u}) + \mathbf{w},$$

where $\mathbf{w}$ incorporates both the control dependent and any additional state-dependent noise. In the worst case, $\mathbf{w}$ could be unbounded, however it is usually possible to at least provide a probabilistic bound for $\mathbf{w}$. For now, we assume that $\|\mathbf{w}\| \leq D$. So we have:

$$\|\mathbf{x}_1 - \mathbf{x}_0\| \leq \|\mathbf{F}(\mathbf{x}_0, \mathbf{u})\| + D.$$

We have that $\|\mathbf{x}_1 - \mathbf{x}_0^*\| = \|\mathbf{x}_1 - \mathbf{x}_0 - (\mathbf{x}_0^* - \mathbf{x}_0)\|$, and then, using the triangle inequality, we have:

$$\|\mathbf{x}_1 - \mathbf{x}_0 - (\mathbf{x}_0^* - \mathbf{x}_0)\| \leq \|\mathbf{x}_1 - \mathbf{x}_0\| + \|\mathbf{x}_1^* - \mathbf{x}_0\| = \|\mathbf{F}(\mathbf{x}_0, \mathbf{u}) - \mathbf{x}_0\| + \|\mathbf{x}_0^* - \mathbf{x}_0\| + D,$$

denoting $\|\mathbf{F}(\mathbf{x}_0, \mathbf{u}) - \mathbf{x}_0\| + \|\mathbf{x}_0^* - \mathbf{x}_0\|$ as $D_{\mathbf{F}}(\mathbf{x}_0, \mathbf{x}_0^*, \mathbf{u})$, we then have:

$$\mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{x}_1, \lambda) \leq \alpha + \left(L_\phi \gamma^T + L_q \frac{1-\gamma^T}{1-\gamma}\right) D_{\mathbf{F}}(\mathbf{x}_0, \mathbf{x}_0^*, \mathbf{u}) + 2E_M^V.$$

Lastly, we can subtract $\mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{x}_0, \lambda)$ from both sides to get:

$$\Delta\mathcal{F}_{MC} \leq (\alpha - \mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{x}_0, \lambda)) + \left(L_\phi \gamma^T + L_q \frac{1-\gamma^T}{1-\gamma}\right) D_{\mathbf{F}}(\mathbf{x}_0, \mathbf{x}_0^*, \mathbf{u}) + 2E_M^V. \quad (8.15)$$

Note that this is a *numerical* result, which are typically difficult to obtain in stochastic optimal control. The maximum increase in the estimate of the free energy is determined by three terms:

i) The first term is defined by a threshold which we set, below which it is assumed that the system is satisfying all task related constraints. The augmented importance sampling scheme places no restriction on the free-energies ability to jump between values below α. This term is only large if the current free-energy is well below α, which indicates that a large change in the free-energy is allowable without endangering any task related constraints.

ii) The second term depends on three major components: how steep the cost function is, how fast the dynamics/disturbances are, and the rate of convergence of the tracking controller. This provides a useful guide for designing cost functions: if we have slow dynamics and a very good tracking controller then this bound will be small even with a steep cost. However, with fast dynamics we may need to tune the cost in order to reduce the Lipshitz constants.

iii) The last term encodes the amount of fluctuation we can expect to see from the Monte-Carlo estimator working on the nominal system, and is a function of both the number of samples used and the exploration variance.

The key point of this analysis is that, when utilizing augmented important sampling, the growth in the free-energy estimate cannot suddenly increase far above the performance threshold set by α, which is the cause of most catastrophic failure. Instead, the algorithm can only fail gradually if the nominal and real state drift apart. This is the same type of failure mode that Tube-MPC have, but in this case, we are still directly computing an estimate of the optimal control.

8.2 Practical Algorithm

The idealized scheme from the previous section provides the mathematical motivation for pursuing an augmented importance sampling scheme, however the algorithm described in the previous section is impractical for two reasons. First the evolution of the nominal system involves solving a constrained optimization problem which is intractable, and second finding a tracking controller with the desired properties is difficult for non-linear systems. In this section we describe practical solutions to these problems, which results in an algorithm that is easy to implement and deploy on a real system.

Nominal System Propagation

Recall that the update rule for the nominal system was defined as:

$$\mathbf{x}^*_{t+1} = \underset{\mathbf{z}}{\operatorname{argmin}} \|\mathbf{z} - \mathbf{x}_{t+1}\|, \quad \text{s.t} \ \ (\mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{z}, \lambda) < \alpha) \vee (\mathbf{z} = \mathbf{x}^*_t)\,. \tag{8.16}$$

Obviously, we cannot search over every state and compute an estimate of the free-energy. Even computing one estimate of the free-energy is equivalent to a path integral iteration, which for the AutoRally vehicle takes between 5-15 milliseconds depending on parameter settings and system load. So we have to be able to approximately solve Eq. (8.16).

This first step we can take is to reduce the computation time required for a free-energy approximation. This can be done simply by reducing the number of samples used when computing the free-energy, although this results in a lower quality estimate, it is acceptable in this scenario since the estimate is not being directly used to compute the control law. Note that the reduced sample estimate is only used for propagating the nominal state, not in the estimation of the optimal control. In practice, on the AutoRally system we can utilize as few as 64 samples to get a quick estimate free-energy, whereas 1200 samples are typically used to compute the optimal control.

By reducing the number of samples used to estimate the free-energy, we are able to evaluate around 10 points in real-time. One possibility is to do a line search from the current nominal state, and the new real state, and pick the point closest to the new real state that satisfies that free-energy threshold. The issue with the aforementioned approach is that it can easily result in the nominal state getting “stuck”, and then the nominal and real state diverge. Another possibility is to copy the approach suggested in non-linear Tube-MPC from the previous chapter. In that approach, the nominal state is propagated according to the undisturbed dynamics. If the disturbances are small compared to the dynamics, then the newly propagated nominal state will be close to the new real state. The problem with this approach is that, while there is a high probability that the new nominal state has a low free-energy, there is no guarantee that it does.

The approach we take is a hybrid between the line search approach and the non-linear Tube-MPC approach. First the nominal state is propagated according to the mean of the importance sampler, this gives us three base points which we use to create a two-step line search. The first line goes from the current nominal state to the propagated nominal state, and the second line goes from the propagated nominal state to the current real state, this is visualized in Fig. 8.1. There are 3 intermediate points on each line, which yields a total of

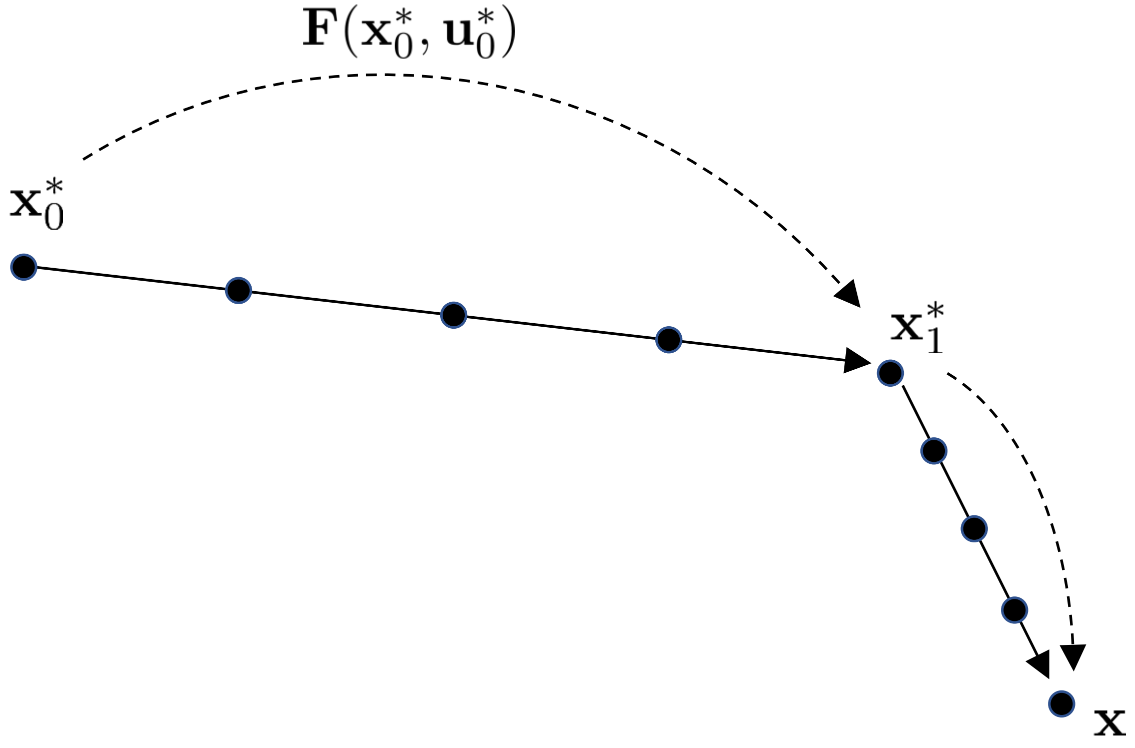


Figure 8.1: Nominal state propagation. The three base points are the current nominal state, the nominal state simulated forward, and the real state which differs by a disturbance $\mathbf{w}$ from the forward simulated state.

9 search points. From each candidate point, a small number of trajectories are sampled in order to preview the free energy approximation from that point. Then the point which is closest to the real system state is selected. Mathematically, this operation is described by the following constrained optimization problem:

$$\mathbf{x}^* = \underset{\mathbf{p}_i}{\operatorname{argmin}} \|\mathbf{p}_i - \mathbf{x}\| \tag{8.17}$$

$$s.t \;\; \mathcal{F}_{MC}(S, \mathbb{P}, \mathbf{p}_i, \lambda) \leq \alpha \tag{8.18}$$

$$i \in \{0, 1, \dots 8\} \tag{8.19}$$

$$\mathbf{p}_0 = \mathbf{x}_0^*, \;\; \mathbf{p}_4 = \mathbf{x}_1^*, \;\; \mathbf{p}_8 = \mathbf{x} \tag{8.20}$$

$$\mathbf{p}_i = \frac{1}{4}\left(j\mathbf{p}_4 + (4-j)\mathbf{p}_0\right), \;\; 0 < j < 4, \;\; 0 < i < 4 \tag{8.21}$$

$$\mathbf{p}_i = \frac{1}{4}\left(j\mathbf{p}_8 + (4-j)\mathbf{p}_4\right), \;\; 0 < j < 4, \;\; 4 < i < 8 \tag{8.22}$$

Algorithm 4 summarizes the steps in selecting the nominal state. Note that in addition to selecting the nominal state, this algorithm also slides down the nominal control if the nominal state is selected has not been frozen at the old nominal state.

In practice, the real state is usually selected as the nominal state, in which case the augmented importance sampling scheme reverts back to standard importance sampling. Less often, one of the points in between the forward propagated nominal state and real state is chosen. This indicates that the previous solution was good, but a disturbance knocked it off course. Note that even though this rarely happens, the unprotected importance sampling scheme would fail in these cases, so the presence of the augmented importance sampling is critical. In the AutoRally experiments, it was never observed that one of the points in between the old nominal and forward propagated nominal state was chosen (such a selection could indicate a loss of recursive feasibility).

Algorithm 4: Nominal State Propagation

```
Given: $\mathbf{F}$, $U_{init}$: System Dynamics ;
$\Sigma$, $\phi$, $q$, $T$, $N$, $\gamma$, $\nu$: Cost function and sampling parameters;
$\lambda$, $\alpha$: Temperature and cost thresholds;
Input : $U$: Current control sequence;
$\mathbf{x}^*$: Old nominal state;
$\mathbf{x}$: Current real state;
$\{\mathbf{p}_0, \dots \mathbf{p}_8\} \leftarrow Fn\_GenerateCandidates(\mathbf{x}, \mathbf{x}^*, \mathbf{F}, U_0)$;
for $i \leftarrow 0$ to $8$ do
    $\mathbf{x}^* \leftarrow \mathbf{p}_i$;
    if $i > 0$ then
        for $t \leftarrow 1$ to $T-1$ do
            $\mathbf{u}^i_{t-1} = \mathbf{u}_t$;
        $\mathbf{u}^i_{T-1} = 0$;
    else
        $U^i = U$;
    for $n \leftarrow 1$ to $N$ do
        Sample $\mathcal{E}^n = \left(\epsilon^n_0 \dots \epsilon^n_{T-1}\right)$ , $\epsilon^n_t \in \mathcal{N}(0, \Sigma)$;
        for $t \leftarrow 0$ to $T-1$ do
            $\mathbf{x}^* = \mathbf{x}^* + \mathbf{F}(\mathbf{x}^*, \mathbf{u}^i_t + \epsilon^n_t)\Delta t$;
            $S_n \mathrel{+}= q(\mathbf{x}^*) + \frac{\lambda}{2}\left(\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\mathbf{u}_t + 2\mathbf{u}_t^{\mathrm{T}}\Sigma^{-1}\epsilon_t\right)$;
        $S_n \mathrel{+}= \phi(\mathbf{x}^*)$;
    for $n \leftarrow 1$ to $N$ do
        $\eta \mathrel{+}= \exp\left(-\frac{1}{\lambda}S_n\right)$;
    $\mathcal{F}_i = -\lambda \log(\eta)$;
// Select best nominal state
$a = \text{argmin}_i \left(\|\mathbf{p}_i - \mathbf{x}\|\right)$ ,  $s.t$  $\mathcal{F}_i \leq \alpha$ ;
if $a$ = NULL then
    a = 0 // Select old nominal state if no solution found
return $\mathbf{p}_a$, $U_a$ ;
```

Nominal System Importance Sampler

In addition to the forward propagation step, we need a method for estimating the importance sampler associated with the nominal system. This is just a standard open loop sequence of control inputs ($U = \{\mathbf{u}_0, \mathbf{u}_1, \dots \mathbf{u}_{T-1}\}$). Since the entire goal of introducing the augmented importance sampler is to keep the importance sampling distribution of the real system close to the importance sampling distribution of the nominal system, it is essential that this distribution remains in a good region of the state space.

The most obvious choice for computing the importance sampling estimate is to treat the nominal system independently of the real system, and perform the standard importance sampling as done in chapter 5. This finds the optimal importance sampler for the system and state cost:

$$\mathbf{x}^*_{t+1} = \mathbf{x}^*_t + \mathbf{F}(\mathbf{x}^*_t, \mathbf{v}_t)\Delta t, \;\; \mathbf{v}_t = \mathbf{u}_t + \epsilon_t,$$
$$S(V, \mathbf{x}^*_0) = \phi(\mathbf{x}^*_T) + \sum_{t=0}^{T-1} \mathbf{q}(\mathbf{x}^*_t).$$

However, performing importance sampling for the nominal system in this manner misses a key opportunity. Since the goal of the nominal system is to help the optimization of the real system, it makes sense to include the real system cost in the objective for the nominal system. To do this, we can have the nominal importance sampler consider the dynamics of the full augmented system:

$$\mathbf{x}^*_{t+1} = \mathbf{x}^*_t + \mathbf{F}(\mathbf{x}^*_t, \mathbf{v}_t)\Delta t,$$
$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{F}(\mathbf{x}_t, \mathbf{v}_t + k(\mathbf{x}_t, \mathbf{x}^*_t))\Delta t,$$

and then optimize a combined cost between the two systems.

When creating a combined cost, some care must be taken. A first consideration is that we need to be sure that the free-energy evaluated from the nominal state always remains below α. Let $\tilde{S}(V, \mathbf{x}_0, \mathbf{x}^*_0)$ be the combined cost function, what must be satisfied is that the free-energy evaluated with $\tilde{S}(V, \mathbf{x}_0, \mathbf{x}^*_0)$ is less than α if and only if the free-energy evaluated with $S(V, \mathbf{x}^*_0)$ is less than α. This rules out the simplest possible strategy, which would be taking a convex combination of the two costs. A second issue is that the mean of the importance sampler here is $\mathbf{u}$, not $\mathbf{u} + k(\mathbf{x}, \mathbf{x}^*)$, which means that the control cost on $k(\mathbf{x}, \mathbf{x}^*)$ is left out. It can be beneficial, therefore, to reintroduce the control cost that appears as the importance sampling weight in evaluating the free-energy from the real state. Taking both of these considerations together, the final combined cost, denoted $\tilde{S}$, that we use is:

$$\tilde{S}(V, \mathbf{x}_0, \mathbf{x}_0^*) = \frac{1}{2}S(V, \mathbf{x}_0^*) + \frac{1}{2}\max\left(\min\left(\hat{S}(V, \mathbf{x}_0, \mathbf{x}_0^*), \alpha\right), S(V, \mathbf{x}_0^*)\right), \tag{8.23}$$

$$\hat{S}(V, \mathbf{x}_0, \mathbf{x}_0^*) = S(V, \mathbf{x}_0) + \frac{\lambda}{2}\sum_{t=0}^{T-1} k(\mathbf{x}_0, \mathbf{x}_0^*)^{\mathrm{T}}\Sigma^{-1}k(\mathbf{x}_0, \mathbf{x}_0^*). \tag{8.24}$$

Note the introduction of the quadratic cost penalizing the activation of the tracking controller on the cost for the real part of the system, this satisfies the second requirement. The form of Eq. (8.23) is chosen so that the first requirement is satisfied, as we show in the following lemma.

Lemma 3. $S(V, \mathbf{x}_0^*) \leq \alpha$ *if and only if* $\tilde{S}(V, \mathbf{x}_0, \mathbf{x}_0^*) \leq \alpha$.

Proof. We start with the forward direction. Suppose that $S(V, \mathbf{x}_0^*) \leq \alpha$, we know that $\min(\hat{S}(V, \mathbf{x}_0, \mathbf{x}_0^*), \alpha) \leq \alpha$. So we have:

$$\tilde{S}(V, \mathbf{x}_0, \mathbf{x}_0^*) \leq \frac{1}{2}S(V, \mathbf{x}_0^*) + \frac{1}{2}\max\left(\alpha, S(V, \mathbf{x}_0^*)\right) \tag{8.25}$$

But, since $S(V, \mathbf{x}_0^*) \leq \alpha$, we have: $\tilde{S}(V, \mathbf{x}_0, \mathbf{x}_0^*) \leq \frac{1}{2}S(V, \mathbf{x}_0^*) + \frac{1}{2}\alpha \leq \alpha$, which demonstrates the forward direction.

Next, suppose that

$$\tilde{S}(V, \mathbf{x}_0, \mathbf{x}_0^*) \leq \alpha \tag{8.26}$$

We have two cases to consider, the first case is when: $\min\left(\hat{S}(V, \mathbf{x}_0, \mathbf{x}_0^*), \alpha\right) > S(V, \mathbf{x}_0^*)$. In this case we have:

$$\tilde{S}(V, \mathbf{x}_0, \mathbf{x}_0^*) = \frac{1}{2}S(V, \mathbf{x}_0^*) + \frac{1}{2}\min\left(\hat{S}(V, \mathbf{x}_0, \mathbf{x}_0^*), \alpha\right) \tag{8.27}$$

Which means we have:

$$\tilde{S}(V, \mathbf{x}_0, \mathbf{x}_0^*) - \frac{1}{2}\min\left(\hat{S}(V, \mathbf{x}_0, \mathbf{x}_0^*), \alpha\right) = \frac{1}{2}S(V, \mathbf{x}_0^*) \tag{8.28}$$

But, since $\frac{1}{2}\min\left(\hat{S}(V, \mathbf{x}_0, \mathbf{x}_0^*), \alpha\right) > \frac{1}{2}S(V, \mathbf{x}_0^*)$, we have:

$$\underbrace{\tilde{S}(V, \mathbf{x}_0, \mathbf{x}_0^*) - \frac{1}{2}\min\left(\hat{S}(V, \mathbf{x}_0, \mathbf{x}_0^*), \alpha\right)}_{=\frac{1}{2}S(V, \mathbf{x}_0^*)} \leq \frac{1}{2}\tilde{S}(V, \mathbf{x}_0, \mathbf{x}_0^*) \leq \frac{1}{2}\alpha \tag{8.29}$$

Which which shows that $S(V, \mathbf{x}_0^*) \leq \alpha$.

We now only need to worry about the case where $S(V, \mathbf{x}_0^*) > \hat{S}(V, \mathbf{x}_0, \mathbf{x}_0^*)$. In that case we have:

$$\max\left(\min\left(\hat{S}(V, \mathbf{x}_0, \mathbf{x}_0^*), \alpha\right), S(V, \mathbf{x}_0^*)\right) = S(V, \mathbf{x}_0^*) \tag{8.30}$$

Which implies that $S(V, \mathbf{x}_0^*) = \tilde{S}(V, \mathbf{x}_0, \mathbf{x}_0^*) \leq \alpha$, which proves the result. $\square$

What we have shown is that the cost function for $\tilde{S}$ is equivalent to the standard nominal cost in the sense that either both cost functions are above the threshold α, or neither of them are. Assuming that α corresponds to violating constraints, this means that both cost functions agree if constraints have been violated. This is useful because optimizing using the combined cost takes into account both the nominal and actual system state, which leads to improved performance. However, it does not enable the system to violate constraints any more easily than the using the standard nominal cost.

Algorithm 5 summarizes the steps in generating and evaluating samples in for the nominal system with the combined cost. The first step is to generate samples from the augmented system, this is done by simulating the system forward using the system dynamics and initial conditions for the nominal and real state. Next, the cost is computed for the resulting state and controls pairs. There are three costs that are then computed.

i) The state cost for the real system augmented with the penalty on applying control

Algorithm 5: Augmented Importance Sampler (AIS)

```
Given: $\mathbf{F}, k$: Transition model and feedback controller;
$\Sigma, \phi, q$: Cost function/sampling parameters;
$T, N$: Time horizon and number of samples;
$\lambda, \alpha$: Temperature and cost threshold
Input: $\mathbf{x}_0, \mathbf{x}_0^*$: Real and nominal state;
$U$: Current importance sampling sequence;
for $n \leftarrow 1$ to $N$ do
    $\mathbf{x} \leftarrow \mathbf{x}_0$;
    $\mathbf{x}^* \leftarrow \mathbf{x}_0^*$;
    $S, \hat{S}, S^{real} \leftarrow 0$;
    Sample $\mathcal{E}^n = \left(\epsilon_0^n \dots \epsilon_{T-1}^n\right), \ \epsilon_t^n \in \mathcal{N}(0, \Sigma)$;
    for $t \leftarrow 0$ to $T-1$ do
        $\mathbf{x} \leftarrow \mathbf{F}\left(\mathbf{x}, \mathbf{u}_t + \epsilon_t^n + k(\mathbf{x}, \mathbf{x}^*)\right)$;
        $\mathbf{x}^* \leftarrow \mathbf{F}\left(\mathbf{x}^*, \mathbf{u}_t + \epsilon_t^n\right)$;
        $\hat{S} \mathrel{+}= q(\mathbf{x}) + \frac{\gamma}{2} k(\mathbf{x}, \mathbf{x}^*)^{\mathrm{T}} \Sigma^{-1} k(\mathbf{x}, \mathbf{x}^*)$;
        $S \mathrel{+}= q(\mathbf{x}^*)$ ;
        $S^{real} \mathrel{+}= q(\mathbf{x}) + \frac{\lambda}{2}\left(\mathbf{u} + k(\mathbf{x}, \mathbf{x}^*)\right)^{\mathrm{T}} \Sigma^{-1}\left(\mathbf{u} + \epsilon + k(\mathbf{x}, \mathbf{x}^*)\right)$;
    $\hat{S} \mathrel{+}= \phi(\mathbf{x})$ ;
    $S \mathrel{+}= \phi(\mathbf{x}^*)$ ;
    $S^{real} \mathrel{+}= \phi(\mathbf{x})$ ;
    $S^{nom} = \frac{1}{2} S + \frac{1}{2} \max\left(\min\left(\hat{S}, \alpha\right), S\right)$;
    for $t \leftarrow 0$ to $T-1$ do
        $S^{nom} \mathrel{+}= \frac{\lambda}{2} \sum_{t=0}^{T-1}\left(\mathbf{u}^{\mathrm{T}} \Sigma^{-1} \mathbf{u}_t + 2\mathbf{u}^{\mathrm{T}} \Sigma^{-1} \epsilon_t^n\right)$;
return $\{S_1^{nom}, S_2^{nom}, \dots S_N^{nom}\}$, $\{S_1^{real}, S_2^{real}, \dots S_N^{real}\}$ $\{\mathcal{E}^1, \mathcal{E}^2, \dots \mathcal{E}^N\}$;
```

action to drive the real system state to the nominal state.

ii) The state cost for the nominal state.

iii) The total (state and control) cost for the real system.

The first two costs are used for updating the importance sampling sequence, and propagating the nominal state forward, and the last cost is used for computing the optimal control. After the costs for the entire state and control sequence is computed the first two costs are combined according to Eq. (8.23), and then the control cost is added to the combined cost.

Tracking Controller

The last issue that needs to be addressed in order to make a practical algorithm is that a feedback controller, which drives samples generated from the real system to the corresponding sample from the nominal system, must be designed. In order to obtain the theoretical guarantee, we need the tracking controller to do this exponentially fast. However, the AutoRally dynamics are described by a neural network, and as such are non-linear and hard to work with from an analysis perspective. In particular, parameters usually required for applying tracking controller designed for autonomous vehicles are not available when utilizing learned dynamics.

As opposed to a hand-designed a tracking controller, we can linearize the system and use a linear quadratic regulator (LQR) to obtain time-varying linear feedback gains. This means that $k(\mathbf{x}, \mathbf{x}^*, t)$ takes the form:

$$k(\mathbf{x}_t, \mathbf{x}_t^*) = K_t \left(\mathbf{x}_t - \mathbf{x}_t^*\right) \tag{8.31}$$

Note that by doing this we lose the the theoretical guarantee, since we are using a linear approximation of a non-linear system. However, LQR is very general, easy to implement, and works well in practice which justifies its usage.

In order to apply LQR, we need to linearize the system about a nominal trajectory. Ideally, we could generate a set of optimal feedback gains for each nominal sample. However, this would involve computing thousands of LQR gains every iteration, which is intractable with current hardware. Instead, we can linearize about the current nominal solution, and then compute LQR gains around that single trajectory. These gains are then re-used for all of the generated samples. This is sub-optimal, however it is effective in practice. One reason for its effectiveness is that, initially, all the samples are tightly clustered together close to the nominal solution. This means that the corresponding feedback gains are close to optimal, and drives the real state to the nominal state before the trajectory samples spread apart significantly. Once the nominal and real states are close together, the feedback controllers contribution is negligible. It should also be noted that the feedback controller is only used in *simulation*, where there is no uncertainty. In this setting, the LQR controller is highly effective.

Algorithm Summary

With tractable methods for updating the nominal state, updating the nominal importance sampling sequence, and computing a feedback control law, we can now describe the full implementation of MPPI with augmented importance sampling. We refer to this algorithm as robust model predictive path integral control (R-MPPI). The psuedocode for the algorithm is given in Alg. 6.

In the first initialization step, the nominal and actual states are set equal to each other, and the nominal control sequence is specified. In the case of a ground vehicle U_{init} can be randomly initialized or set to zero. Then the algorithm enters into the main MPC loop. The first step in the MPC loop is to simulate the system forward from the nominal state using the current importance sampling sequence. This generates a state trajectory, for which the system is linearized around and feedback gains are computed. Note that this step requires the specification of quadratic state and control costs for which the optimal feedback gains are computed with respect to. These costs must make a trade-off between smoothness and performance. Even though the samples generated by the augmented importance sampler are unbiased, in practice the limited number of samples available makes the selection of these tracking costs important. If the tracking costs are very low then samples generated by the augmented importance sampler can fail to track the corresponding nominal state sequence, which can result in constraint violation. If the tracking costs are too high, than the resulting gains will be high, and result in jerky control inputs.

After the feedback gains are computed, samples are generated from the augmented importance sampler (Alg. 5) utilizing the feedback control law specified by the gains returned

Algorithm 6: Robust Model Predictive Path Integral Control (R-MPPI)

```
Input: F, $U_{init}$: Dynamics and initial control sequence ;
$\Sigma$, $\phi$, $q$, $T$, $N$: Cost function and sampling parameters;
$\lambda$, $\alpha$: Temperature and cost thresholds;
// Initialization
$U \leftarrow U_{init}$;
$\mathbf{x} \leftarrow Fn\_StateEstimator()$;
$\mathbf{x}^* \leftarrow \mathbf{x}$;
// Main Loop
while Task Not Finished do
    $\{K_0, K_1, \dots K_{T-1}\} \leftarrow Fn\_LQR(\mathbf{x}^*, U)$;
    $\{S_1^{nom}, \dots S_N^{nom}\}$, $\{S_1^{\text{real}}, \dots S_N^{\text{real}}\}$, $\{\mathcal{E}^1, \dots \mathcal{E}^N\} \leftarrow Fn\_AIS(\mathbf{x}, \mathbf{x}*, U)$;
    $\mathbf{u}_{opt} \leftarrow U_0 + K_0(\mathbf{x} - \mathbf{x}^*) + \sum_{n=1}^N \frac{\exp\left(-\frac{1}{\lambda}S_n^{real}\right)\epsilon_0^n}{\sum_{n=1}^N \exp\left(-\frac{1}{\lambda}S_n^{real}\right)}$ // Optimal control
    $U \leftarrow U + \sum_{n=1}^N \frac{\exp\left(-\frac{1}{\lambda}S_n^{nom}\right)\mathcal{E}^n}{\sum_{n=1}^N \exp\left(-\frac{1}{\lambda}S_n^{nom}\right)}$ // Update importance sampler
    $Fn\_SendToActuators(\mathbf{u}_{opt})$;
    $\mathbf{x} \leftarrow Fn\_StateEstimator()$;
    $\mathbf{x}^*, U \leftarrow Fn\_NominalStateUpdate(\mathbf{x}^*, U)$;
```

by LQR. The augmented importance sampler returns disturbance sequences along with two different cost weightings for each sample. One cost is the combined nominal and real cost given by Eq. (8.23), and the other is purely the cost function of the real system.

After samples are generated and cost weightings computed, the optimal control and updated importance sampler can be computed. The optimal control is the combination of the importance sampling input computed using the current feed-forward control and feedback control: $U_0 + K_0(\mathbf{x} - \mathbf{x}^*)$, plus the optimal disturbance computed according to the real cost. The optimal importance sampler is updated using the optimal disturbance sequence computed according to the combined cost plus the current importance sampling sequence.

Lastly, the optimal control is executed, state feedback is received, and then the nominal state is updated (Alg. 4). Note that the nominal control sequence is slid down during the nominal state update according to where the nominal state is selected. This whole process is executed in a fast control loop (40 - 50 Hz for the AutoRally). In practice, the state estimator and actuator interface execute in separate threads, so $Fn\_SendToActuators$ and $Fn\_StateEstimator$ only require reading and writing results from the perspective of the thread running R-MPPI.

8.3 Experimental Results

We implemented R-MPPI on the AutoRally vehicle and tested it on the autonomous racing task at the CCRF research facility. The track cost here consists of a piece-wise linear function of normalized distance from the centerline. The distance is normalized so that $(-1, 1)$ denotes the track boundaries. In between $(-0.75, 0.75)$ the track cost is simply $3.3\|d\|$ where d is the normalized distance. Then, between 0.75 and 1 (or -0.75 and -1) the cost linearly ramps up to 1250. After reaching its maximum value at the track boundaries, the cost is completely flat afterwards. A cross section of the cost function is shown in Fig. 8.2.

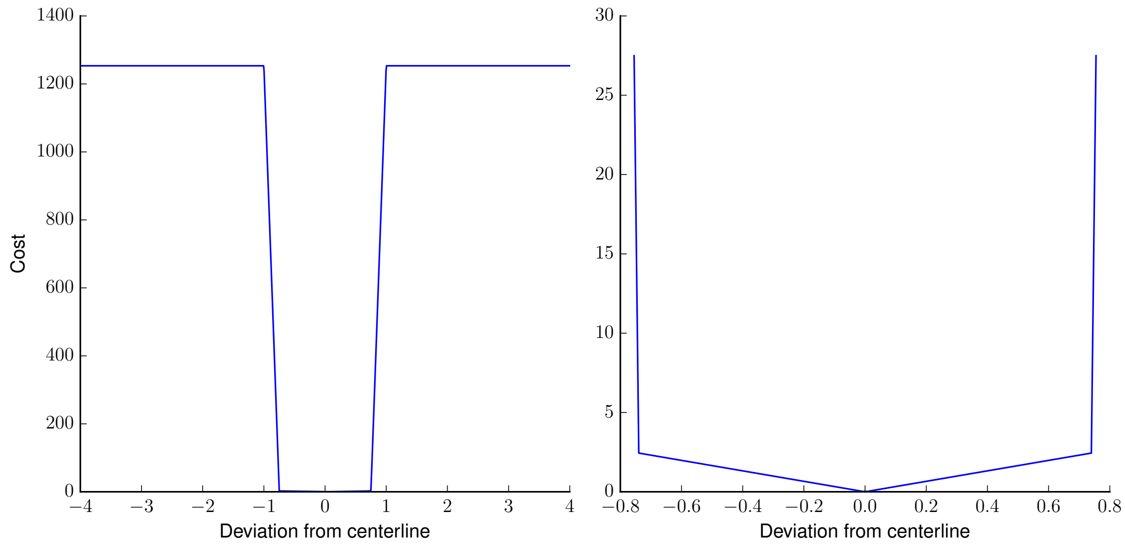


Figure 8.2: Cross section of the track cost function used by R-MPPI. There's a small cost on staying near the centerline, and then the cost quickly ramps up to indicate the presence of a constraint (the barrier). On the right is a zoomed in version of the graph on the left which contrasts the steepness of the performance cost (staying near the centerline), with the cost enforcing the constraint.

This type of cost can be more formally described as a function of distance from some desired operating point (denoted d) using the following function:

$$B(d;\beta,\kappa,\mathcal{T},M) = \begin{cases} \kappa \cdot d & \text{if} \quad d < \mathcal{T} \\ \kappa \cdot d + \beta \cdot \frac{d-\mathcal{T}}{M-\mathcal{T}} & \text{if}\ \mathcal{T} \leq d < M \\ \kappa \cdot M + \beta & \text{if} \quad M \leq d \end{cases} \tag{8.32}$$

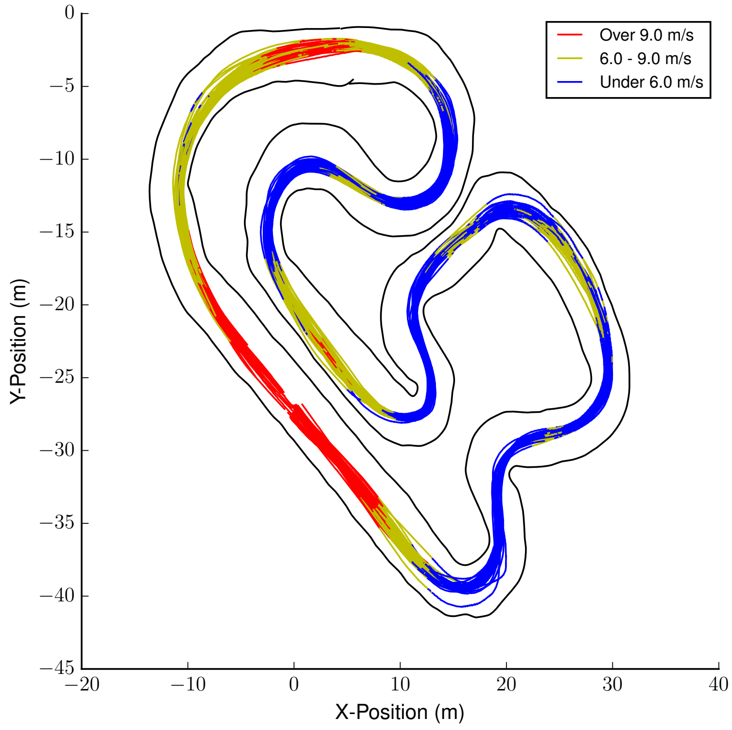
which is parametrized by 4 values: the small slope (3.3 in this case), the penalty threshold (0.75), the steep slope (1250 here), and the maximum violation threshold (1.0). The total running cost we used for R-MPPI was then:

$$q(\mathbf{x}) = B(d_{track}; 1250, 3.3, 0.75, 1) + B(d_{slip}; 1250, 0, 0.75, 0.9) + 25 \cdot |v_x - 25|. \tag{8.33}$$

This cost consists of two constraint type costs, and then the cost for going fast. Notice that this is now an absolute value instead of a squared cost, using an absolute value instead of a squared cost. We have observed that using an absolute value cost is helpful for inducing racing type behavior, since it can be set far above what the robot is reasonably capable of achieving without creating unstable behavior (the AutoRally physically cannot go 25 m/s on the CCRF facility). For the dynamics model we used a neural network with the same structure as in chapter 6, however we improve on the model by adapting it online, the method for online adaptation with a neural network is described in the next section.

High-Speed Driving Results

We conducted approximately 50 laps of driving both counter-clockwise and clockwise around the CCRF track, after discarding the starting lap from each trial (the starting lap has slightly different statistics than the other laps), we ended up with 45 counter-clockwise laps and 47 clockwise laps. The trajectory traces of these runs are shown in Fig. 8.3. Note that the amount of driving recorded here amounts to approximately 17 kilometers (10 miles) of driving.



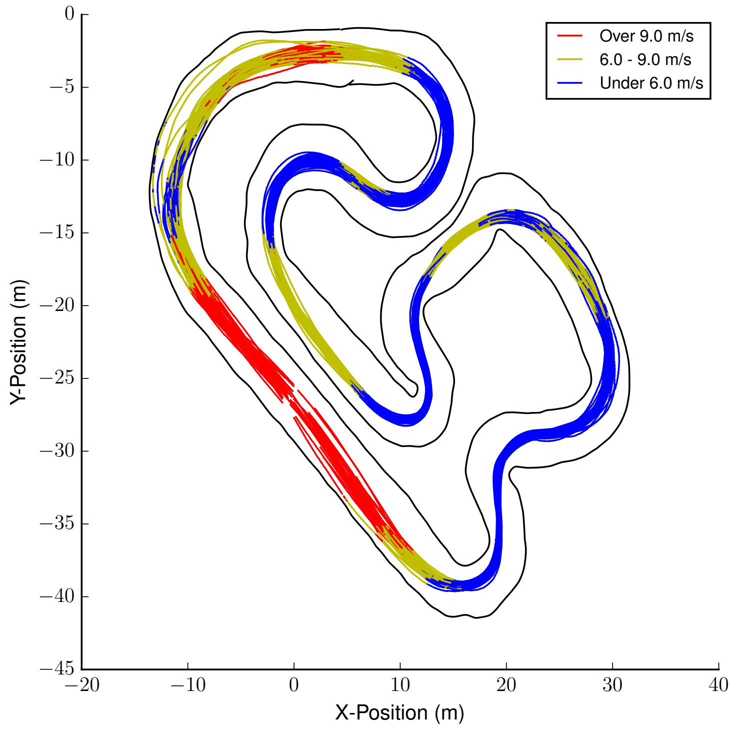


Figure 8.3: Trajectory traces of test runs with R-MPPI at CCRF autonomous racing facility. The top image shows the counter-clockwise laps and the bottom image shows the clockwise laps. The top speed on the straight can exceed 50 kph.

Table 8.1 shows the performance statistics achieved over these tests. The top speed achieved over these trials is 14.11 m/s, which is slightly over 50 kilometers per hour. This is a significant performance increase from the Tube-MPPI method (Table 7.3). Besides the

Table 8.1: R-MPPI High-Speed Driving Performance at CCRF

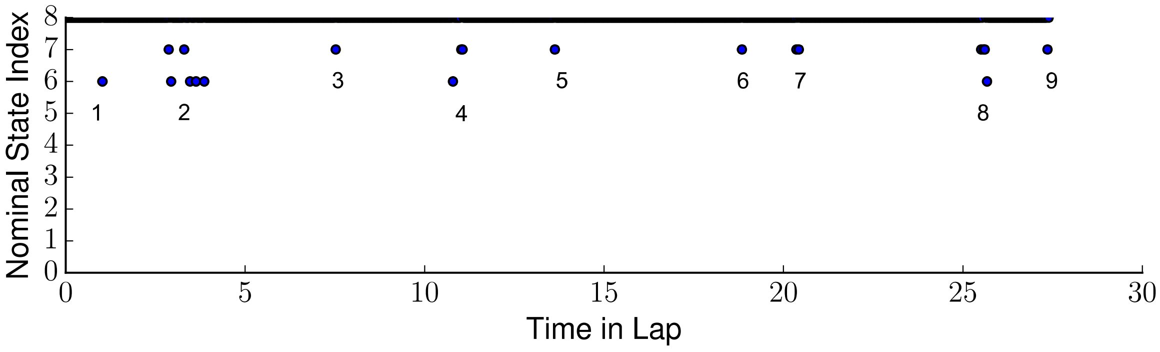
	Avg. Lap (s)	Best Lap (s)	Max Speed (m/s)	Max Slip (Rad)
Counter-Clockwise	28.40 +/- 0.95	27.13	14.11	0.90
Clockwise	28.76 +/- 0.83	27.42	13.93	0.91

lap times and top speeds, the maximum slip angle recorded during the runs is a positive sign that the method is working well. The maximum slip angle that we set was 0.9 radians, and this constraints is always satisfied in the counter-clockwise case, and in the clockwise case it is violated just slightly. This is even with the vehicle driving near its performance limits in a highly stochastic environment.

Augmented Importance Sampling Statistics

It is clear that the R-MPPI algorithm achieves good, robust performance based on the results from the previous section. Here, we take a closer look at exactly how this is achieved through the augmented importance sampling procedure. There are a few variables internal to the augmented importance sampling procedure that are of interest: what index the nominal state is at, the free-energy of the nominal system, and the free-energy of the real system. It is easiest to understand the role of the augmented importance sampler if we examine a specific lap, since we can see how different maneuvers correspond to changes in the free-energy and nominal state.

First we can see how often and precisely where the nominal and real state differ. The nominal state differing from the real state indicates that the estimated free-energy of the open-loop importance sampler violates our constraint threshold, this is when the augmented importance sampling kicks in and starts making a difference. Figure 8.4 shows a plot of the nominal state index over the course of a lap, and the corresponding lap with annotations denoting the locations where the nominal state index drops below 8. The vast majority of the time, the nominal state index is 8, in which case augmented importance sampling is the same as standard importance sampling. However, there are a few cases where the nominal state index drops below 8 and the augmented importance sampler is activated. This usually occurs during hard accelerations into and out of turns.



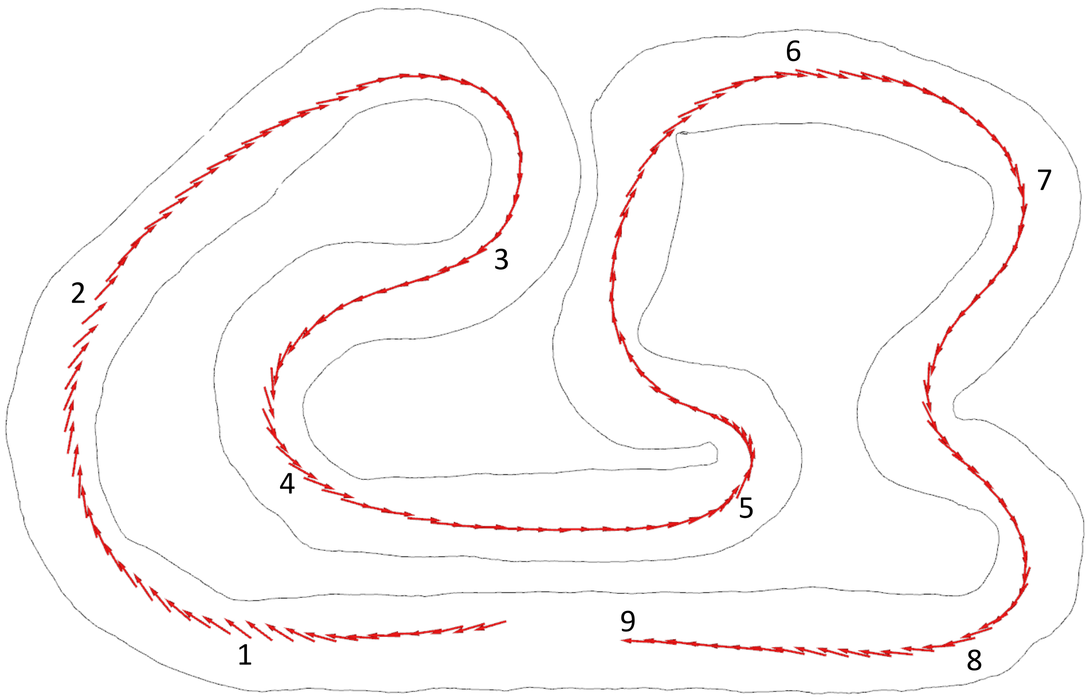


Figure 8.4: Top: Nominal state index of the augmented importance sampler during a fast lap. Their are 9 events where the nominal state index is below 8 (which means that the nominal and real state differ). This indicates times where the estimate of the free-energy using the open-loop importance sampler from the nominal state violates the α. Bottom: Pose history as the vehicle completes a fast clockwise lap at CCRF. The numbers indicate areas where the nominal state differed from the real state.

When the augmented importance sampler is active, the free energy estimates from the nominal and real states differ. However, if the stabilizing feedback controller is performing its job, the free-energy estimate from the real state (using the augmented importance sampler) should be close to the free-energy estimate from the real state. Figure 8.5 shows the plot of the free-energy over the course of the same lap as in Fig. 8.4. During the high-

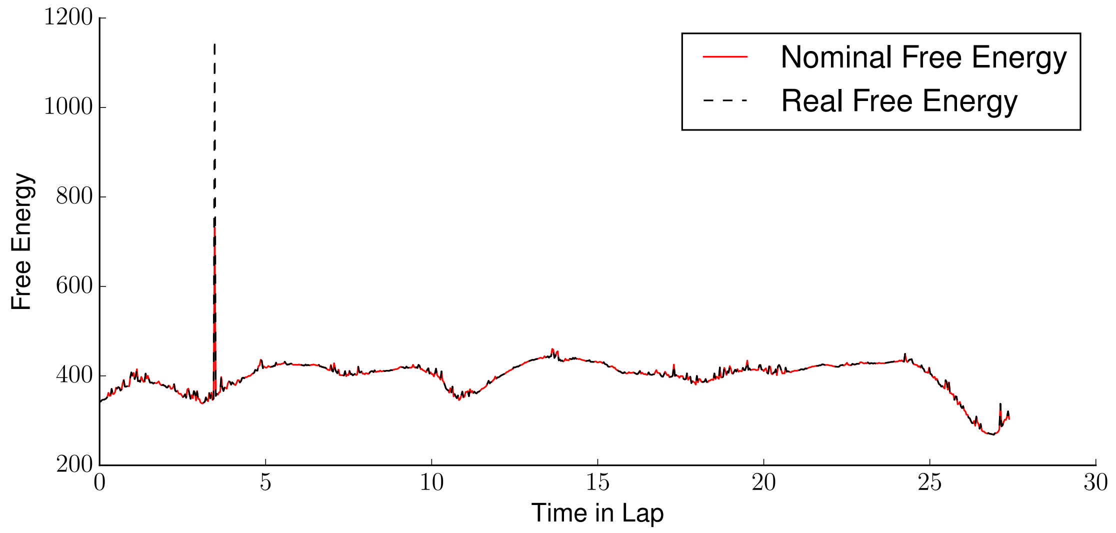


Figure 8.5: Free energy estimate calculated from the nominal and real state. Note that α is set to 1000 here, and 1250 indicates a predicted constraint violation.

speed run, there is only one instance where the real and nominal free energy significantly diverge. This shows that the tracker (sub-optimal LQR) is largely successful at keeping the trajectories generated from the real state close to the trajectories generated starting from the idealized nominal state. Note that without the augmented importance sampler, every time the nominal and real state differ, the free-energy calculated from the real state would be at least 1000, and possibly much higher.

Extreme Maneuvering Example

With the level of noise in the environment, and the speeds that the AutoRally is oprating at, there are times when the vehicle can get itself into trouble. One of the major benefits of R-MPPI is that it knows very precisely where the track boundaries are (based on the shape of the cost function), and it is capable of executing extremely aggressive maneuvers in order to avoid them. Figure 8.6 illustrates one such maneuver.

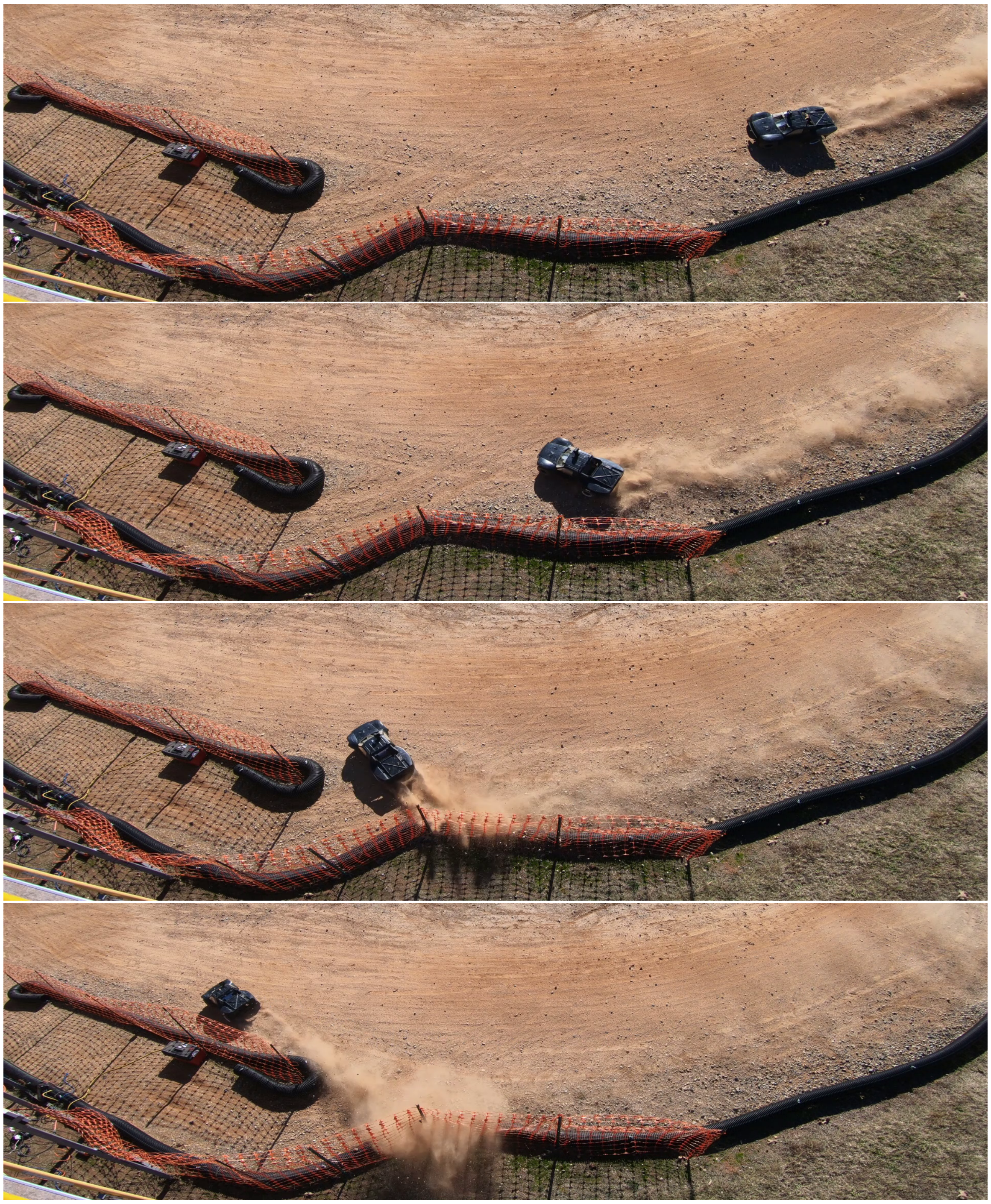


Figure 8.6: Aggressive maneuver by R-MPPI in order to avoid the barrier. Note that the augmented importance sampling is active here.

The setup for this maneuver is that the vehicle did not turn sharply enough (due to modeling error) on the previous turn. The result is that the vehicle ends up heading straight towards the barrier at nearly 30 mph with only a few meters distance to maneuver away from the barrier. In order to avoid the barrier, the vehicle initiates a spin, and then counter-steers in order to stabilize itself. This gets the vehicle pointed towards the center of the track, once the vehicle is pointed this way, it applies full throttle in order to accelerate away from barrier, and then continues. In order to execute this maneuver, the controller must be operating in a fast control loop, be aware of precisely where the barrier is, and understand the non-linear dynamics enough to spin and counter-steer.

8.4 Discussion

In this chapter we presented the augmented importance sampling approach, and its resulting application to model predictive path integral control, which we call robust model predictive path integral control. Robust MPPI significantly improves on MPPI, as it can work with very simple cost terms in order to represent constraints, and it is also a major improvement on Tube-MPPI. As opposed to Tube-MPPI, which places the MPPI algorithm into the existing Tube-MPC literature, the augmented important sampling approach brings ideas from the Tube-MPC literature into the importance sampling framework that is native to MPPI. Since the nominal state indirectly affects the stat through the importance sampling, we can compute an unbiased estimate of the optimal control like in standard MPPI, but the nominal state keeps the importance sampling distribution stable from iteration to iteration.

The Robust MPPI algorithm achieves all of the objectives that we initially outlined as goals for our control algorithm all the way back in chapter 1. Control is performed in real-time completely on-the-fly, the controller is clearly able to handle the non-linear dynamics and perform at the limits of handling, and we can handle constraints by using very simple cost formulations.

In addition to a highly practical algorithm, augmented importance sampling provides a potential route towards providing performance guarantees, and we have shown how the growth in the free-energy can be bounded for the idealized scheme. We did not end up computing this bound for the AutoRally, since finding a tracking controller with the desired properties was not feasible. However, for other systems such controllers do exist, and computing the bound may be possible.

CHAPTER 9
MODEL LEARNING AND ADAPTATION

The application of model predictive control on physical systems requires highly accurate dynamics models. Even methods like R-MPPI which are robust to stochastic disturbances need a good model of the system dynamics in order to plan intelligently. One of the most promising approaches to obtaining accurate dynamics models is to learn them by utilizing function approximation methods (e.g. neural networks). We initially explored the combination of neural networks with MPPI for autonomous driving in chapter 6. In that case, there was nothing particularly unique about the way the neural network was trained: some system identification data was collected, and then the model is trained using mini-batch stochastic gradient descent. After the model is trained, it is used just like any other machine learning or physics based model would be.

Although this approach proved to be effective, it ignores a major issue in that the system dynamics are constantly changing. Changes in the dynamics can occur either due to environmental factors (e.g. the road surface condition), or due to internal factors that are hard to measure (e.g. weight distribution, suspension stiffness, steering slop). Therefore, we would like to be able to adapt the neural network online in order to fit the current environment. Unfortunately, adapting neural networks online is a notoriously difficult problem. The key issue that must be overcome is catastrophic forgetting [78, 79], which is the tendency for neural networks to forget old data when fed new data from a different distribution. This is especially problematic in the case of autonomous control, where catastrophic forgetting in the system model can lead to a (catastrophic) controller failure.

In this chapter we propose a method to overcome the catastrophic forgetting problem in neural networks. In order to do this we make use of a class of modeling techniques which are naturally immune to catastrophic forgetting: locally weighted linear regression

[80]. Locally weighted linear regression methods work by building a global model up from a set of many small local linear models. Each model is equipped with a receptive field, which determines how much the model should respond to a given input. The output of the global model is then computed as a weighted average of all the local model outputs [81, 82]. Since a given training pair only affects a highly localized region of the state space, locally weighted regression methods can safely make incremental updates. One of the most mature local modelling methods is locally weighted projection regression (LWPR) [82]. LWPR has proven successful in modeling robot dynamics, even for high-dimensional systems in online learning scenarios.

Given the success of LWPR at the task of learning robot dynamics, and the ability for it to be used in an incremental online setting, one may wonder: why not use LWPR instead of neural networks for learning system dynamics? The issue with utilizing locally linear methods in model predictive control is the computational cost. In the past, locally weighted regression methods have been limited to inverse control, which only requires a single prediction per timestep, or offline trajectory/policy optimization [80, 83]. In cases where they have been used in MPC [84], the model had to be severely restrained in order to control the number of local models generated. The issue is that, for local linear methods to achieve high accuracy, usually thousands or even tens of thousands of local models are required to be effective. This requires an order of magnitude more floating point operations than a neural network to get comparable prediction accuracy. Therefore, utilizing locally linear methods instead of neural networks inside of an MPC method would come at a significant opportunity cost, if it were even possible to run in real-time.

Instead of directly utilizing LWPR in the MPC controller, our approach will be to maintain both a neural network model and an LWPR model. The neural network model is used by an MPC controller and it is updated online using data recently collected from the system, but it is also regularized using pseudo-samples generated by an LWPR model. The LWPR model is updated in an incremental online manner, in order to ensure that the artificially generated data matches the current target distribution.

This is a similar approach to some of the earliest methods proposed for mitigating catastrophic forgetting in neural networks: rehearsal and pseudo-rehearsal [79, 85, 86, 87, 88]. In rehearsal methods, the original training data is retained and used alongside the new data in order to update the model. Pseudo-rehearsal methods do not retain old data, but instead they randomly create input vectors (pseudo-inputs) that are then fed through the current network in order to produce a corresponding output point (pseudo-output). The resulting artificially generated sample (pseudo-sample) can then be used for training the network alongside newly received data. The idea is that, by using the pseudo-samples alongside real data, the network can be encouraged to learn the new data without forgetting the current mapping. Recently, there has been success using rehearsal [89] and pseudo-rehearsal based methods for vision tasks [90, 91, 92, 93]. In these methods the primary challenge that must be overcome is either storing previous data samples (in rehearsal methods) or randomly generating realistic inputs (for pseudo-rehearsal methods).

In the case of learning vehicle dynamics, generating pseudo-inputs is relatively easy due to the low dimensional state-space representation of a vehicle. Instead, there is another challenge that must be overcome that cannot be handled by the usual rehearsal or pseudo-rehearsal techniques: *both* the input and target distributions are non-stationary. This means that sometimes we need to learn new data while retaining old data, which is the case when encountering a novel region of the state space. But, other times, we need to learn new data which overwrites old data, this is the case if we are in a familiar region of the state-space but the target distribution has changed. The approach we develop to handle this problem is best thought of as a type of pseudo-rehearsal method, with the key innovation being the use of an incrementally updated LWPR model to produce the pseudo-outputs. We also make use of a constrained gradient descent update rule in order to prevent large errors on new training data from overwhelming the training signal on previously seen data.

Besides rehearsal and pseudo-rehearsal methods, there are a variety of methods for updating neural networks which mitigate catastrophic forgetting by controlling how far the parameters of the model can move away from the current model. For instance, this is the approach taken in [94, 95, 96]. However, as in the case of rehearsal and pseudo-rehearsal, it is not clear how controlling changes in the weights works when the target distribution is variable, since in that case the network weights corresponding to previously learned data will need to be changed as well. Another promising approach to online adaptation for neural networks that has recently been explored is meta-learning [97]. However, the meta-learning approach does not have an explicit mechanism to combat catastrophic forgetting, and it is currently unclear how to perform the meta-training in order to ensure that catastrophic forgetting cannot occur.

9.1 Online Learning Problem Formulation

Consider an autonomous vehicle operating at some task, while performing the task the vehicle encounters system states, which we denote by $\mathbf{z}$ in this case, and executes controls, denoted $\mathbf{u}$. Our goal is to update the model of the vehicle dynamics, which is defined using the discrete time dynamical system:

$$\mathbf{z}_{t+1} = \mathbf{z}_t + \mathbf{F}(\mathbf{z}_t, \mathbf{u}_t; \theta)\Delta t \tag{9.1}$$

Where θ denotes the parameters of the model, in our case these are the weights of a neural network. The system states for the vehicle are position, heading, roll, velocities, and heading rate, and the control inputs are steering and throttle commands. In the case of a ground vehicle, the position and heading updates are kinematically trivial, so we can re-write the dynamics as:

$$\mathbf{z}_{t+1}^k = \mathbf{z}_t^k + k(\mathbf{z}_t)\Delta t \tag{9.2}$$

$$\mathbf{z}_{t+1}^d = \mathbf{z}_t^d + \mathbf{f}(\mathbf{z}_t^d, \mathbf{u}_t; \theta)\Delta t \tag{9.3}$$

Here $\mathbf{z}^k$ denotes kinematic states, which are position and heading, and $\mathbf{z}^d$ denotes dynamics states, which are roll, body-frame longitudinal and lateral velocity, and heading rate. The motion update for the kinematic states is trivial, so we need only focus on learning $\mathbf{f}(\mathbf{z}_t^d, \mathbf{u}_t; \theta)$. Now, we define the following variables:

$$\mathbf{x} = \begin{pmatrix} \mathbf{z}_t^d \\ \mathbf{u}_t \end{pmatrix}, \quad \mathbf{y} = \frac{\mathbf{z}_{t+1}^d - \mathbf{z}_t^d}{\Delta t} \tag{9.4}$$

as the inputs and targets for our learning algorithm. Now, as the vehicle moves about in the world, it encounters states and controls according to some probability distribution:

$$\text{Local Operating Distribution : } \quad \mathbf{x} \sim \mathcal{P}_L(\mathcal{X}). \tag{9.5}$$

The distribution $\mathcal{P}_L$ is called the local operating distribution, and it is highly task dependent. In addition to the local operating distribution, we assume that there is a system identification dataset, which contains data consisting of all the various maneuvers that the vehicle needs to learn in order drive competently. The system identification dataset consists of samples drawn from another distribution:

$$\text{System Identification Distribution : } \quad \mathbf{x} \sim \mathcal{P}_{ID}(\mathcal{X}). \tag{9.6}$$

which we denote as $\mathcal{P}_{ID}$, and refer to as the system identification distribution. Note that this distribution is constant, but the mapping which takes input points drawn from this distribution to the corresponding dynamics output is not. Our goal is to incrementally update a neural network describing the system dynamics. However, we must be sure that by updating the model we do not forget any of the system modes contained in $\mathcal{P}_{ID}$.

First consider a simple approach to performing online model adaptation based on standard stochastic gradient descent (SGD). Suppose that we have access to streaming data, and that we maintain a set of recently encountered input and output pairs. By randomly drawing pairs from this set, we can get independent and identically distributed (I.I.D.) samples from the local operating distribution. The standard SGD rule updates the parameters as follows:

$$\theta_{i+1} = \theta_i - \gamma \nabla_{\theta_i} \|\mathbf{y} - \mathbf{f}(\mathbf{x}; \theta_i)\|^2 \tag{9.7}$$

since training pairs are drawn from the local operating distribution, this update will improve the model's performance for the inputs drawn from that distribution. Mathematically, this means that we are optimizing for the objective:

$$\mathbb{E}_{\mathbf{x} \sim \mathcal{P}_L} \left[\|\mathbf{y} - \mathbf{f}(\mathbf{x}; \theta)\|^2 \right] \tag{9.8}$$

This is *not* what we want. The issue with this is that the local operating distribution may not contain all the maneuvers that the vehicle needs to operate effectively. A typical example in the case of autonomous driving is highway driving: a vehicle operating on a highway only needs to maintain a constant velocity and make slight turns the vast majority of the time, if the model is updated with inputs purely drawn from a highway driving dataset, there is no guarantee that the model will remember the basic maneuvers necessary for other types of driving. This problem, known as *catastrophic forgetting*, is a well known deficiency of neural networks, and it is especially problematic when the model being updated is being used to control the vehicle.

If we had access to I.I.D. samples from the system identification dataset, we could instead use an SGD update law that jointly learns the target mapping for inputs drawn both from the system identification dataset and the local operating distribution. For instance, the following update law achieves this:

$$\theta_{i+1} = \theta_i - \gamma\left(G_L(\theta_i) + G_{ID}(\theta_i)\right) \tag{9.9}$$

$$G_L = \nabla_{\theta_i}\|\mathbf{y}_L - \mathbf{f}(\mathbf{x}_L;\theta_i)\|^2 \tag{9.10}$$

$$G_{ID} = \nabla_{\theta_i}\|\mathbf{y}_{ID} - \mathbf{f}(\mathbf{x}_{ID};\theta_i)\|^2 \tag{9.11}$$

$$(\mathbf{x}_L, \mathbf{y}_L) \sim \mathcal{P}_L, \quad (\mathbf{x}_{ID}, \mathbf{y}_{ID}) \sim \mathcal{P}_{ID} \tag{9.12}$$

This update balances optimizing the model on the local operating distribution and the system identification dataset, and it is the basic idea behind traditional rehearsal and pseudo-rehearsal techniques. In the case of classification, this type of update is effective at preventing catastrophic forgetting. But, in the regression setting, even this type of update could be problematic since the magnitude of the error incurred by the network can vary greatly depending on the region of state-space the system is in. If the error incurred by the local operating distribution is very high it can overwhelm the error signal from the system identification part of the data, which can still lead to the vehicle forgetting basic maneuvers.

Instead, we want to ensure that the model cannot forget the system identification dataset, in this context "forgetting" means intentionally degrading the performance of the model on input data drawn from $\mathcal{P}_{ID}$. One way to enforce that is to ensure that update steps always move in the direction of the local minima for input data drawn from the system identification distribution. This constraint can be enforced by ensuring that the cosine of the angle between the update direction and the gradient computed from the system identification data is always positive, and it can be achieved with the following update law:

$$\theta_{i+1} = \theta_i - \gamma\left(\alpha G_L(\theta_i) + G_{ID}(\theta_i)\right) \tag{9.13}$$

$$\alpha = \max_{a\in[0,1]} \quad s.t \ \langle aG_L(\theta_i) + G_{ID}(\theta_i), G_{ID}(\theta_i)\rangle \geq 0 \tag{9.14}$$

This update law still balances the objective of simultaneously optimizing for the local operating distribution and system identification distribution. However, it constrains the combined gradient to always point in the same direction as the gradient computed from system identification data.

The problem with implementing the update rule defined by Eq. (9.13) is that, in an online setting, we only have access to data generated from the local operating distribution. Additionally, since the target mapping is changing, we cannot simply re-draw samples from the original system identification dataset or generate pseudo-outputs by running random inputs through the current model like in standard rehearsal and pseudo-rehearsal methods.

9.2 Locally Weighted Projection Regression Pseudo-Rehearsal

Our goal is to approximately implement the constrained gradient update defined in Eq. (9.13). Our strategy will be to use artificially generated pseudo-training points to enforce the constraint, with the additional requirement that the pseudo-training points must somehow match the changing target distribution. This means that artificially generating training points requires two steps:

i) A method for generating artificial input points that are I.I.D. samples from $\mathcal{P}_{ID}(\mathcal{X})$.

ii) A method for computing the corresponding target, $\mathbf{y}$, for an artificially generated input point. This should be a function approximator that is capable of online adaptation, since the target mapping $\mathbf{y}$ is actively changing.

Given these requirements, it appears as though we cannot go anywhere, since the preceding discussion can roughly be summarized as: in order to perform online adaptation we first need a method that can do online adaptation in order to constrain the stochastic gradient descent update. The reason why this is not the case, is that the speed requirements on the model used for artificial data generation are much less strenuous than the speed requirements for a model used in model predictive control.

Suppose that we are receiving inputs points at a rate of 40 Hz, a reasonable number for a robotic system, then in order to produce an equal number of artificial points for regularization our model needs to be able to produce 40 predictions/second. In contrast, for our sampling based MPC controller we require the model to produce on the order of millions of predictions per second. Even less computationally intensive MPC algorithms (iLQG for instance) require on the order of tens of thousands of predictions (plus derivative computations) per second. So, even for relatively inexpensive MPC algorithms the computational demands on the model used for artificial data generation are 3 orders of magnitude less than the model used for MPC, and 5 orders of magnitude less than for our sampling based controller. This means that we can use more computationally demanding models that are specifically suited to online adaptation in order to generate the artificial data, such as LWPR.

Algorithm Description

Our approach will be to train an LWPR model, which will be updated online, in order to compute the target mapping for artificially generated input points. For the generation of the input points, a gaussian mixture model (GMM) is used. The GMM is trained offline and kept static, which reflects the fact that the input distribution defined by the system identification dataset does not change. The artificial input/output pairs are then used to compute a synthetic gradient, which is used to regularize the online stochastic gradient descent. The algorithm consists of four sub-modules, which we now describe in detail. The overall flow of the algorithm is shown in Fig. 9.1.

Gaussian Mixture Model

The purpose of the GMM is to generate synthetic input points consistent with the system identification dataset. We denote a mini-batch of synthetic input points as X_s. The GMM uses diagonal covariance matrices, and is trained using standard expectation maximization.

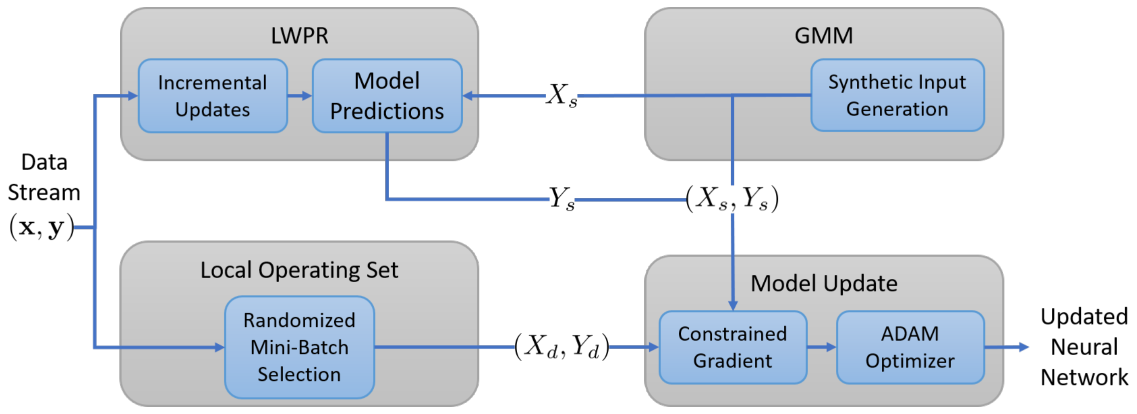


Figure 9.1: LW-PR^2 Algorithm. The GMM produces synthetic input points which are combined with predictions from LWPR to create synthetic training pairs. These are combined with randomized mini-batches created from recently collected data in order to compute the constrained gradient update.

We use the Bayesian Information Criterion (BIC) in order select the number of gaussian models used. After the initial training the GMM is not modified again. This is because the input distribution for the system identification dataset should be carefully curated in order to ensure that it contains a balance of all necessary maneuvers.

LWPR Module

The LWPR module takes in the synthetic input points generated by the Gaussian mixture model, and then runs those input points forward through the LWPR model in order to produce synthetic output points. If we let $\mathbf{y}_i$ and $\mathbf{c}_i$ be the mean and center of the i_{th} local model and let D_i be the distance metric which defines the receptive field for the i_{th} model, then LWPR computes the global prediction as:

$$\mathbf{y}(\mathbf{x}) = \sum_{i=1}^{L} w_i \cdot \mathbf{y}_i(\mathbf{x} - \mathbf{c}_i) \tag{9.15}$$

Where the weight governing the response of each local model is:

$$w_i = \frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{c}_i)^{\mathrm{T}} D_i (\mathbf{x} - \mathbf{c}_i)\right)}{\sum_{j=1}^{L} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{c}_j)^{\mathrm{T}} D_j (\mathbf{x} - \mathbf{c}_j)\right)} \tag{9.16}$$

Since the response of a given model to an input decays exponentially fast, model updates have only a negligible impact on models with centers far from the current input point. This is the feature that makes LWPR largely immune to catastrophic forgetting, and it ensures that LWPR can be safely updated online.

The mini-batch output of the LWPR module is denoted Y_s. The LWPR model is initially trained over several epochs on the system identification dataset using the standard LWPR update rule. Online, the local linear models making up the LWPR model are continually updated. We train one LWPR model for each different output dimension (roll rate, longitudenal acceleration, lateral acceleration, and heading acceleration). The LWPR module is implemented using [98].

Local Operating Set

The local operating set consists of the last several seconds of training points received from the stream of data generated by the system. In our implementation this set contains between 500 and 1000 (10 - 20 seconds) of data. Out of this set of data, randomized mini-batches are created (denoted as (X_d, Y_d)) and then fed into the model updater.

Model Update

The synthetic mini-batch and the mini-batch generated by the local operating set are fed into the model updater module which computes the gradient. The constrained gradient is computed via equations (9.9) - (9.13), with the synthetic data acting as the data drawn from the system ID dataset. After the gradient is computed we use the ADAM optimizer [99] to perform the update step.

Neural Network Initialization

Before the neural network can be updated, an initial model needs to be trained on the original system identification dataset. One option is to initialize the model using standard stochastic gradient descent (i.e. without taking into account the other modules), but we have found it is more effective to jointly train the initial model with the LWPR model and GMM model. This means that the actual system identification dataset takes the place of the local operating set, but synthetic data is still generated by the GMM and LWPR modules, which is used to compute the constrained gradient. We have observed that training the initial model in this manner has a negligible effect on the initial performance of the trained model, but helps with the adaptation.

9.3 Experimental Results

We tested our locally weighted projection regression pseudo-rehearsal (LW-PR^2) approach using four different sets of experiments in simulation and on the AutoRally platform. Our first experiment tests the algorithm's ability to prevent catastrophic forgetting, using a dataset designed specifically to induce catastrophic forgetting on naive adaptation methods. The second experiment tests the method's ability to adapt to drastic changes in the system dynamics using a driving dataset collected on a muddy surface. The third experiment tests how effective the updated model is when used as part of an MPC algorithm. Lastly, the fourth experiment consists of running the model adaptation during a full day of testing with the AutoRally, and measures how well the algorithm works in a practical setting. Throughout these experiments there are 3 different types of experimental modes that are run:

i) An **offline** test is a test where the model is not allowed to adapt during the experiment.

ii) An **online** test is a test where the model is allowed to adapt during the experiment, but the adapted model is not used to control the vehicle.

iii) An **active** test is a test where the model is allowed to adapt during the experiment, and the adapted model is used to control the vehicle.

Recall that in an online training scenario, there is not an explicit training, validation, and test set. Instead, as each training pair is received, we compute the current error on that training pair, and we then record the result. After the error has been computed and recorded, the training pair is fed into the model updater. We compare our method against the base model (no adaptation), and the base model adapted with standard stochastic gradient descent. We also record the performance of the LWPR model used to generate synthetic training inputs.

It is important to realize that the LWPR model we use for generating synthetic data would not be feasible for use in a real-time control loop. Tables 9.1 and 9.2 detail the computational requirements of the neural network and LWPR respectively. For these calculations, we assume that a dot product operation takes $2N - 1$ floating point operations (FLOPs) for vectors of dimension N, and that a matrix-vector multiplication takes $2MN - M$ FLOPs where the matrix has dimension $M \times N$. We also assume that any non-linear function $(\exp, \tanh, (\cdot)^2)$ takes a FLOP. For LWPR it can be difficult to predict the throughput required since the number of active local models can vary greatly, so we compute a lower bound based only on how many local model activations must be computed. Computing a local model activation requires first subtracting the mean for the local model from the current input point (6 FLOPs), then individually squaring each result (6 FLOPs), then computing the dot product between the result and the receptive field weight ($2 \cdot 6 - 1$ FLOPs), and then computing the negative exponential of the result (2 FLOPs). This results in a total of 25 floating point operations for each local model, plus the additional computations required to actually compute the weighted average. The neural network simply consists of matrix-vector multiplies, the additions of the bias, and $\tanh$ non-linearities. Note also, that LWPR works best when a separate model is used for each output dimension, whereas only one neural network is required.

The key takeaway from Tables 9.1 and 9.2 is that making predictions with the neural network is two orders of magnitude cheaper than making predictions with LWPR models. Since MPC controllers need to make tens of thousands or millions of predictions per

Table 9.1: LWPR Computational Requirements

Output Variable	Receptive Fields	FLOPs/Pred
Roll Rate	162	$> 4,050$
Longitudinal Acc.	$1,409$	$> 35,225$
Lateral Acc.	$1,738$	$> 43,450$
Heading Acc.	$2,336$	$> 58,400$
Total	$5,645$	$> 141,125$

Table 9.2: Neural Network Computational Requirements

Layer Transition	Input - Output Neurons	FLOPs/Pred
Input - Hidden 1	6 - 32	416
Hidden 1 - Hidden 2	32- 32	2,080
Hidden 2 - Output	32 - 4	256
Total	6-32-32-4	2,688

second, this is important. For instance, our MPC controller performs 6 million dynamics predictions every second. If we used the LWPR model that we have trained we would need to achieve a throughput of at least 847 GFLOP/S to run in real time. Although this number is technically achievable for modern graphics cards on dense matrix multiplication benchmarks (the 1050 Ti in AutoRally has a peak measured performance of 1.8 TFLOPS), it is not currently possible for algorithms with more complicated memory usage, control flow and synchronization requirements - such as forward propagating an LWPR model. In contrast, the synthetic data generation only requires on the order of tens or hundreds of predictions per second, which is easily manageable on any reasonably capable modern processor.

Catastrophic Interference Test

In this experiment we test the ability of LW-PR^2 to improve online modeling performance while simultaneously “remembering” other parts of the system identification dataset. These experiments utilize two datasets, which we selected from the publicly available dataset accompanying [40]. The first dataset we call the *online training dataset* and the second is called the *offline validation dataset*. These datasets were collected on the same day, so the environmental differences are minimal[1]. The online training dataset consists of 100 laps (approximately 27 minutes) of slow speed driving around a roughly elliptical track in the clockwise direction. This is an extremely monotonous dataset, as the robot mostly follows the same line over the 100 laps.

Given the monotonous nature of the online training dataset, it is easy for the online adaptation to overfit to the local operating distribution and forget parts of the system identification dataset. In order to test how well the model adaptation is able to remember other system modes, we utilize the offline validation dataset. The offline validation dataset consists of the same type of low speed monotonous driving in the *opposite* (counter-clockwise) direction of the online training dataset. If the model adaptation algorithm is successful at remembering the system identification dataset, then we should be able to run the adaptation on the online training dataset and see minimal degradation when testing the final adapted model on the offline validation dataset.

Table 9.3: Online Training Dataset Errors

	Base	SGD	LW-PR^2	LWPR
Roll Rate $(rad./s)$	0.01	0.01	0.01	0.01
Longitudinal Acc. (m/s^2)	0.35	0.33	**0.32**	0.33
Lateral Acc. (m/s^2)	0.69	0.63	**0.61**	0.65
Heading Acc. $(rad./s^2)$	2.11	**0.46**	0.49	0.53
Total MSE	0.79	**0.36**	**0.36**	0.38

Table 9.4: Offline Validation Dataset Errors

	Base	SGD	LW-PR^2	LWPR
Roll Rate $(rad./s)$	0.01	0.01	0.01	**0.00**
Longitudinal Acc. (m/s^2)	0.20	0.22	0.20	**0.17**
Lateral Acc. (m/s^2)	0.99	1.56	1.10	**0.83**
Heading Acc. $(rad./s^2)$	1.47	2.11	0.83	**0.42**
Total MSE	0.67	0.97	0.53	**0.36**

The testing procedure works as follows: we first test the online performance of each of the methods using the online training dataset. Then, after the online test is finished, the

[1]The offline validation dataset was collected approximately 30 minutes before the online training dataset

final adapted model is taken and an offline test (no adaptation allowed) is performed on the offline validation dataset. The results of these tests are shown in Tables 9.3 and 9.4.

For the online training dataset all of the adaptive methods perform similarly, and they all significantly decrease the total mean-squared error of the model predictions compared with the base neural network model. However, when using the final adapted model from the online training dataset on the offline validation dataset, the differences between the methods become apparent. The standard SGD methods suffers the characteristic catastrophic forgetting, particularly in the heading acceleration which makes sense given the difference in the direction of travel between the two datasets. As expected, LWPR is unaffected by the change in local operating distribution, and performs better than the base network. Our LW-PR^2 method performs only slightly worse than LWPR and outperforms the base network. This shows that the method is not only capable of preventing catastrophic forgetting, but that it is actually able to generalize knowledge gained in one dynamics regime to another related dynamics regime.

Modified Dynamics Test

In this experiment we test the ability of the algorithm to adapt to highly modified vehicle dynamics. The dynamics are modified by running the vehicle on a muddy track surface. The mud changes the dynamics because it clings to the tires and reduces the friction between the vehicle and the ground. Additionally, the mud clinging to the body of the vehicle adds over 10 kg of extra weight (the normal weight is 21 kg), which has a significant effect on the vehicle's dynamics. All of the system identification data was collected on a dry surface with a mud free robot, so this is a completely novel dynamics regime for the system.

The vehicle is driven by an expert 1/5 scale RC car driver in the muddy conditions. Despite the poor driving conditions, the driver is still able to attain speeds over 50 kph and slip angles in excess of 60 degrees. This means that the dataset is challenging not only

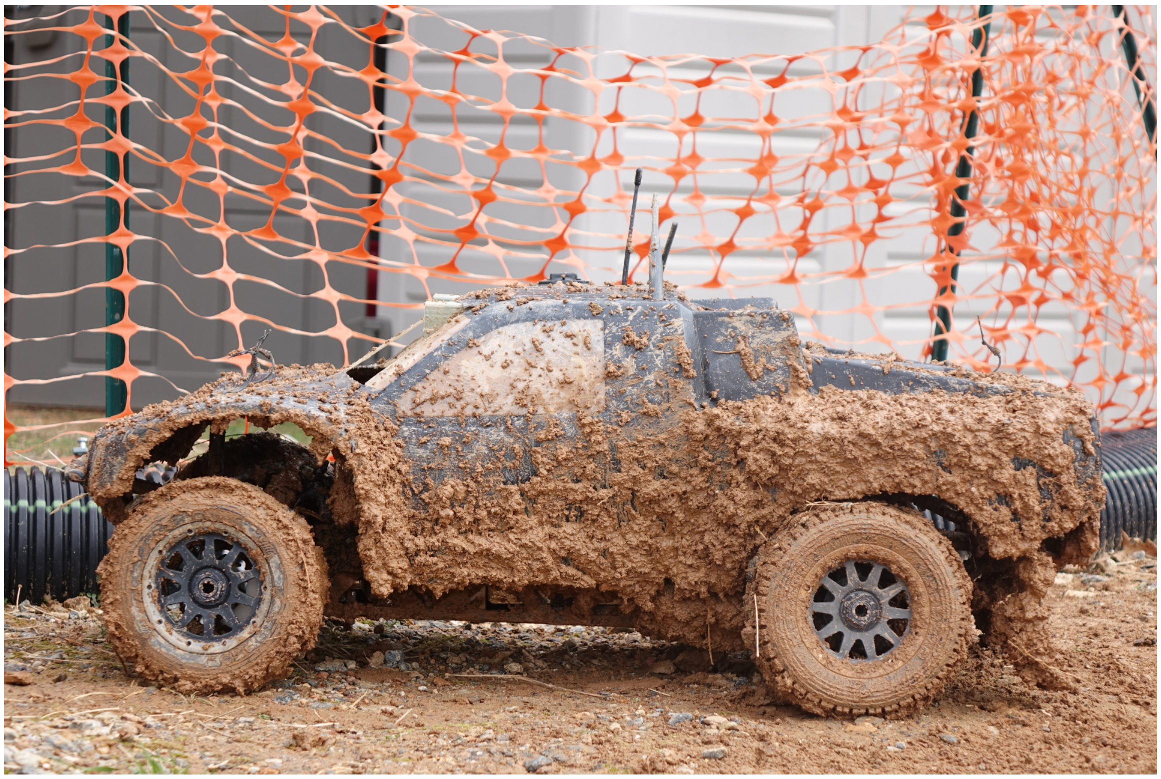


Figure 9.2: AutoRally vehicle after running on muddy surface. Notice the depressed rear suspension, which is caused by the extra weight from the mud accumulating on the chassis and body.

because of the changing conditions, but also because of the highly dynamic regime that the vehicle operates in. This dataset consists of slightly more than 2.5 minutes of data, which is 5 laps around our test track. The results are given in Table 9.5.

Table 9.5: Modified Dynamics Dataset Errors

	Base	SGD	LW-PR^2	LWPR
Roll Rate $(rad./s)$	0.02	0.02	0.02	0.02
Longitudinal Acc. (m/s^2)	2.42	**1.52**	1.60	1.70
Lateral Acc. (m/s^2)	1.04	1.02	**0.96**	0.98
Heading Acc. $(rad./s^2)$	4.96	**2.64**	2.74	3.10
Total MSE	2.11	**1.29**	1.33	1.45

All of the incremental methods achieve a better total MSE than the un-modified base network. The LW-PR^2 method actually outperforms LWPR by a significant margin. The standard SGD method performs best, as it did in the catastrophic forgetting test on the online training dataset. The weakness of SGD is not that it does not fit the local operating distribution, but that it can easily forget the system identification dataset.

Simulated Autonomous Driving Tests

The previous two experiments tested our method on datasets where the model being produced by the model adaptation was not being used to control the vehicle. In this section, we test the algorithm in an active setting where a model predictive controller uses the updated model to control the vehicle. We use the same open source Gazebo simulation of the AutoRally vehicle from [1] for these experiments. The system identification dataset that we use to train the base model is the same as in the previous sections (i.e. it is based on real world data). Note that the simulation dynamics are significantly different from the real-world AutoRally dynamics, so the starting base model is highly inaccurate.

We ran three different model adaptation settings: standard SGD, LW-PR^2, and no adaptation. For each setting we performed trials running 10 laps around the track, and we collected 5 trials for each different setting. The vehicle is driven using the model predictive path integral controller from [40], with a desired speed set[2] at $8m/s$. In order to emphasize the impact of the model adaptation, we set the maximum slip angle the vehicle is allowed to achieve to be relatively low (≤ 13 degrees). Since the vehicle slides less easily in the gazebo simulation than in the real world, this results in a very conservative controller when using the base model (which has only seen real-world data). As the model adapts, the controller should realize that it can increase speed without slipping, leading to improved performance.

The results of all of the trials are shown in Table 9.6. Using either the base model or the LW-PR^2 adapted model, the controller is able to successfully navigate around the track. However, when using standard SGD to update the model the controller consistently fails

[2]For speeds higher than $8m/s$ the accelerations reported by the simulator exhibited high frequency oscillations indicating numerical instability of the simulator, this prevented us from trying faster speeds using the model adaptation.

after completing 1 lap, typically the controller tries to take a turn too fast, which results in the vehicle rolling over.

Table 9.6: Gazebo Simulation Results

	Base Network	SGD	LW-PR^2
Avg. Laps Completed	10	1	10
Avg. Trial MSE	1.84	2.49	**0.65**
Avg. Lap Time	34.78	N/A	**32.04**

Trajectory traces for the base model and LW-PR^2 are shown in Fig. 9.3. The base model performs adequately, and the controller is able to consistently drive the vehicle around the track using the base model. However, the controller with the updated LW-PR^2 model is able to attain a higher average velocity around the track. Figure 9.4 shows the progression of lap

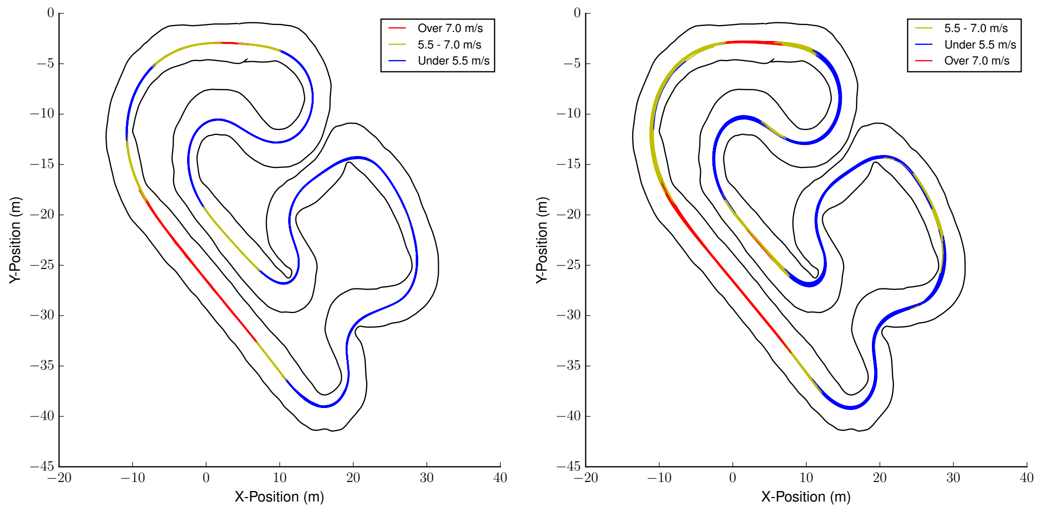


Figure 9.3: Trajectory traces for simulated autonomous driving runs with model adaptation. Top: Base Model, Bottom: Updated model with LW-PR^2. Notice the increased areas of high speed for the LW-PR^2 setting compared with the base model.

times and total MSE per lap as each trial progresses. On average, it takes less than one lap for the MPC controller to start benefiting from the model adaptation: as the model adapts it realizes it can go faster without slipping in the simulation than it can in the real world and the result is a performance increase. The performance on the second lap is significantly better with the LW-PR^2 adapted model than with the base model. After the second lap, the model continues to make small improvements in the per lap MSE.

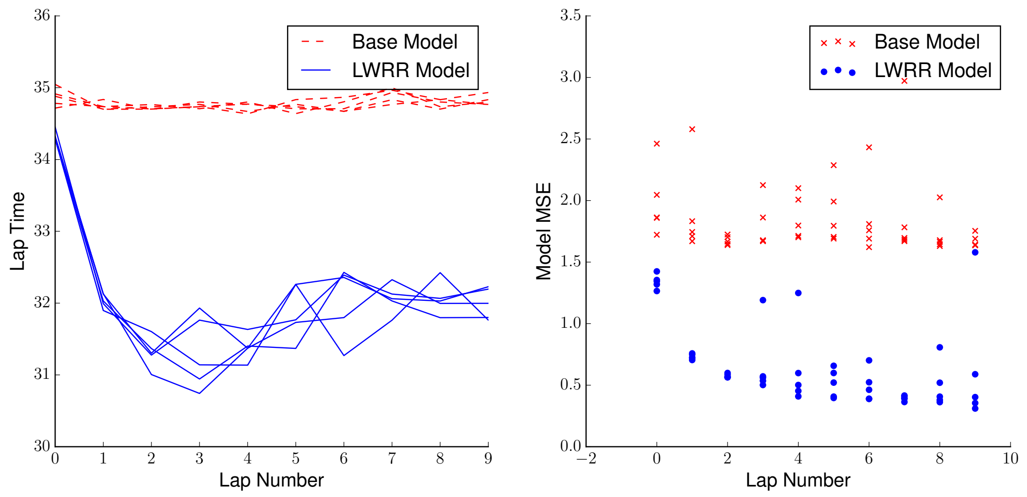


Figure 9.4: Improvement in lap time and total MSE accumulated per lap for the updated LW-PR^2 model versus the base model.

AutoRally Experimental Results

In the previous sections we tested our model adaptation approach in three essential areas: robustness to catastrophic interference, ability to adapt to drastic changes in the dynamics, and effectiveness when utilized by an MPC controller. These experiments were conducted in a controlled manner using either simulation or specially collected datasets. In this section, we examine how the model adaptation scheme works in a more natural environment - the model adaptation is turned on at the beginning of a day of testing and allowed to run uninterrupted[3] for the entire day. The resulting dataset consists of over 1 hour of autonomous data collected over a period of 4.5 hours. The 1 hour of autonomous data consists of approximately 18 kilometers of driving data with speeds up to 50 kph. Note that this dataset contains a significant amount of natural variation: the early morning runs are with a fresh damp track, whereas test runs in the afternoon are with a drier track that has less grip. Many of the tests start when the vehicle has a fully charged battery, and then end with a dead battery, and then are continued with another fully charged battery, this means that the adaptation has to constantly re-learn similar parts of the dynamics over again. The tires

[3]Technically the actual program running the model adaptation is interrupted when there are long pauses in testing. However it saves the current parameters, and reloads them when testing resumes.

also become gradually worn out, which has a significant effect on the friction available.

For this dataset we record the same performance metrics as in the earlier online dataset experiments for each of the four adaptation strategies. Additionally, we have available the active performance of LW-PR^2 data since the model produced by LW-PR^2 was being used to drive the vehicle autonomously. For the active version of LW-PR^2 we used a slightly more conservative learning rate, which explains the performance difference between the active and non-active LW-PR^2-SGD algorithms. Note that the errors produced by the vehicle running autonomously in these experiments are on average higher than the errors reported in the previous sections. This is due to a combination of speed and control style: moving faster produces higher accelerations which lead to higher errors than in the catastrophic forgetting test, and the autonomous control system does not produce as smooth of control inputs as the expert human which can also lead to high accelerations relative to the expert.

The results over the full day of testing are given in Table 9.7. Once again, all of the incremental method significantly improve the performance from the base model. The SGD and LW-PR^2 methods perform nearly identically on the online test, but the method updated with LW-PR^2 is able to be safely utilized by an MPC controller.

Table 9.7: Autonomous Field Test

	Base	SGD	LW-PR^2	LWPR
Roll Rate $(rad./s)$	0.01	0.01	0.01	0.01
Longitudinal Acc. (m/s^2)	2.73	2.28	2.30	**2.06**
Lateral Acc. (m/s^2)	1.71	1.29	**1.24**	1.28
Heading Acc. $(rad./s^2)$	8.28	**4.48**	4.87	4.54
Total MSE	3.18	2.10	2.11	**1.97**
Active Performance (MSE)	N/A	N/A	2.54	N/A

9.4 Discussion

The key contributions from this chapter are:

i) Using an incrementally updated LWPR model in order to create artificial training pairs. The use of incrementally updated LWPR enables pseudo-rehearsal to be applied to systems where both the input and target distributions are non-stationary.

ii) Using a constrained gradient update, which ensures that the adaptation cannot move away from the system identification dataset, no matter how large the errors it encounters in other regions of the state-space are.

In order to test our method we created a series of datasets and simulation tests that stressed essential requirements for an online adaptation scheme: the ability to prevent catastrophic forgetting, adapt to drastic changes in the dynamics, and the ability to produce models usable by an MPC controller. These experiments demonstrated the capability of our approach and also highlighted some of the nuances involved in validating online adaptation algorithms: in all of the online experiments the standard SGD method performed at the level of the LWPR and LW-PR^2 algorithms, it was not until the SGD method was tested on a specifically collected validation dataset or tried to be combined with an MPC controller, that the deficiencies of the SGD method became apparent. We have also demonstrated the practicality of the approach by performing an extended test of the method, where it is required to run continually for hours at a time while producing models that are capable of high speed driving.

CHAPTER 10
CONCLUSIONS AND FUTURE DIRECTIONS

We started this thesis by outlining the autonomous racing problem, and defined three key criteria that we wanted an autonomous control system to be able to achieve: performance at the limits of handling, constraint satisfaction, and real-time performance. The method which finally achieves these criteria is the Robust-MPPI algorithm developed in chapter 8, along with online model adaptation from chapter 9.

In order to get to that point, we first developed new theory based on an information theoretic interpretation of stochastic optimal control. The information theoretic framework relies on a fundamental inequality between free-energy and KL-Divergence and gives rise to an optimal distribution. The free-energy of the system can be related to the value function, and the mean of the optimal distribution can be related to the path integral form of the optimal control.

The information theoretic framework, with path integral control theory, and advances in GPU computing, led to the MPPI algorithm in chapter 5. Extensive experiments on the real-world AutoRally system highlighted the strengths and deficiencies of MPPI in chapter 6, and further analysis identified key issues and potential solutions in chapter 7. In total, the experiments in chapter 6 consists of over 100 kilometers of autonomous driving. Finally, chapter 8 brings all of the theoretical results and experimental findings together to create an algorithm that satisfies our criteria, and is capable of consistently driving the AutoRally at speeds over 50 kilometers per hour.

10.1 Future Directions

There are several exciting research directions that are natural extensions of the work contained in this thesis. One of the most obvious research directions is applying the algorithms developed in this work to higher dimensional robotic systems. Legged locomotion tasks are an especially interesting direction where the algorithms applied in this work could be highly effective: differentiating through contact is one of the most challenging aspects for locomotion control, and in our sampling-based frameworks this step can be bypassed altogether. The primary challenge with applying sampling-based algorithms in the context of locomotion is the cost of evaluating the system dynamics, which usually involves solving linear complementarity problems, and can be very expensive. However, with continual increases in parallel architectures it is looking increasingly likely that sufficient computational power could exist in the next few years to apply sampling-based methods to locomotion problems.

Another important area that could be further developed is the theoretical treatment of robust MPPI in chapter 8. The theoretical work in that sections illustrates how useful performance guarantees could reasonably be obtained in a sampling-based architecture. However, the treatment there is preliminary, and used more of a guide towards developing a practical algorithm that performs well. In fact, in the final algorithm we end using LQR, which forgoes the theoretical guarantees that the idealized algorithm could achieve. Although this led to an effective algorithm, there is still significant room for improvement. The theoretical analysis is especially important in applications to aerospace systems and applying sampling-based methods to systems such as hypersonic vehicles is a promising direction given the challenging hybrid dynamics that such systems have.

The most exciting future direction for this research, however, can be found by returning to the AutoRally system. In this thesis we considered the case of a single vehicle driving fast, but we now have the capability of running multiple AutoRally platforms (see Fig. 10.1, which creates new challenges and opportunities related to autonomous racing. Interactive racing against an adversarial opponent requires solving an extremely difficult prediction problem in a fast, dynamic environment, and it can serve as a motivating proxy problem for multi-agent interaction, much as we have considered the single vehicle case in this thesis. In fact, we have already taken some preliminary steps in this direction in the *cooperative case*:

best-response MPC. This initial research is highlighted in the next section. It should be emphasized that this work is highly preliminary and should be thought as a useful potential direction for future research as opposed to completed work.

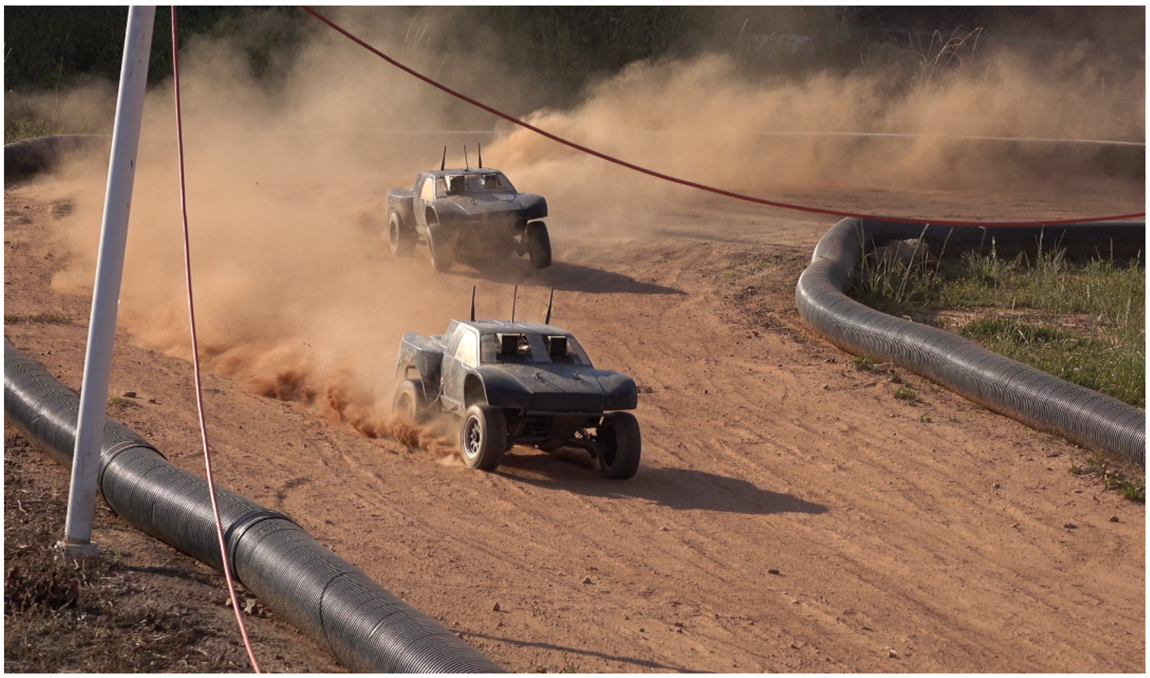


Figure 10.1: Two AutoRally vehicles operating on the dirt test track at the Georgia Tech Autonomous Racing Facility.

10.2 Head-to-Head Autonomous Racing

Consider a scenario where two vehicles are operating in close proximity to each other and cannot directly communicate. The alternative is to develop methods which do not require explicit coordination. The main problem which we must overcome in the multi-vehicle, non-communicative setting is that the robot must be able to "guess" at the other vehicles' current plans and predict how the other vehicles might react to changes it makes in its own policy. The usual methods for predicting the behavior of other vehicles are to treat them as obstacles moving at a constant speed subject to stochastic disturbances [100, 101], or to use predefined behavior models [102, 103]. Although these methods can be effective, they both have significant drawbacks: treating other intelligent vehicles as unintelligent obstacles is inaccurate in many cases, and predefined behavior models are computationally expensive and require careful data collection and analysis to train [104].

As opposed to pre-specifying a behavior model, we can attempt to formulate the problem as a differential game, and then use optimization to approximately solve the problem. The game-theoretic approach treats each vehicle as an independent agent attempting to optimize an objective, subject to their internal dynamic constraints, and it can be viewed as the generalization of optimal control to multiple, potentially competing, agents. This approach can generate realistic predictions of the behavior of other vehicles, while only requiring knowledge of the other vehicles' dynamics, objectives, and current state. This is advantageous because, in many cases, it is fundamentally easier to infer intent than to predict behavior. For example, during highway driving, activating a turn signal almost always means that a vehicle would like to change lanes. However, the precise behavior that results from that intent is dependent on the density of traffic, geometry of the road, etc. In the game theoretic approach, the result of this interaction between intent and environment can be determined immediately by the optimization, whereas in a behavioral model approach it would be necessary to collect data and verify the model for each environmental sub-case.

The drawback of the game-theoretic approach is that the resulting optimization problem is very difficult to solve. For example, it is not practical to develop even a general numerical scheme which converges to a solution (a Nash Equilibrium) in the case of non-linear stochastic differential games. Instead, we propose to use a simple numerical method, known as *best-response iteration*, which is an iterative system of equations with the solutions to the differential game as fixed points. We then combine best-response iteration with MPPI, and we demonstrate the ability of the resulting best response model predictive control algorithm to independently control two one-fifth scale autonomous ground vehicles operating in close proximity. As a baseline method for comparison, we utilize the method of treating the other vehicle as an obstacle moving in a straight line with constant velocity and compare its performance to that of the best response controller.

Best-response iteration is one of the oldest methods in game theory [105] and can be applied to any game. Although the idea of best response iteration is old, its use in the control of autonomous robotic systems fairly new, although a couple of recent papers have appeared recently promoting the use of the method [106, 107], including our work in [42].

Problem Setup

We begin by describing the differential[1] game problem formulation for a general two-player system (the generalization to more than two is straight-forward). Let F_a and F_b be the dynamics of player A and player B respectively. We assume that the equations of motions for the players are stochastic, discrete time equations of the form:

$$\mathbf{x}_i^{t+1} = \mathbf{f}_i(\mathbf{x}_i^t, \mathbf{v}_i^t) \tag{10.1}$$

$$\mathbf{v}_i^{t+1} \sim \mathcal{N}(\mathbf{u}_i^t, \Sigma_i) \tag{10.2}$$

$$i \in \{a, b\} \tag{10.3}$$

where $\mathbf{u}_i^t$ is the commanded input for player i at time t, and $\mathbf{v}_i^t$ is the input perturbed by a Gaussian disturbance. This is the same type of control-dependent noise we have assumed throughout this text. Next $\mathbf{x}_a^0 \in \mathbb{R}^{n_a}$ and $\mathbf{x}_b^0 \in \mathbb{R}^{n_b}$ denote the initial conditions of the two players, $\mathcal{U}_a$ and $\mathcal{U}_b$ denotes the set of admissible control inputs for the systems, and C_a and C_b are the cost functions for the two players. It is assumed that both cost functions take the form:

$$C_i = \mathbb{E}_{F_a, F_b}\left[\phi_i(\mathbf{x}_a^T, \mathbf{x}_b^T) + \sum_{t=0}^{T-1} \mathcal{L}_i(\mathbf{x}_a^t, \mathbf{x}_b^t, \mathbf{u}_a^t, \mathbf{u}_b^t)\right] \tag{10.4}$$

$$\mathcal{L}_i = c_i(\mathbf{x}_a^t, \mathbf{x}_b^t) + \lambda(\mathbf{u}_i^t)^{\mathrm{T}}\Sigma_i^{-1}\mathbf{u}_i^t \tag{10.5}$$

[1]Technically a difference game, since we will consider discrete time dynamics.

where $t \in \{0, 1, \dots T\}$ and $i \in \{a, b\}$. Note that although both players cost functions have the same general structure, the two cost functions need *not* be the same. The resulting differential game can then be described by the tuple:

$$\mathcal{G} = \{F_a, F_b, \mathbf{x}_a^0, \mathbf{x}_b^0, C_a, C_b, \mathcal{U}_a, \mathcal{U}_b\} \tag{10.6}$$

Each players' objective is to minimize their cost function subject to their own dynamical constraints. However, since the costs are functions of both players, and each players' costs may not be aligned, it may be impossible to find a set of control which minimizes the cost functions for both players simultaneously. This creates the need for an alternative solution concept to functional minimization. The most appropriate notion of solution in this setting is the Nash equilibrium. Let $\mathcal{X}$ denote the concatenated states of both players $\mathcal{X}_t = (\mathbf{x}_a^t \; \mathbf{x}_b^t)^{\mathrm{T}}$, and let $\pi_a(\mathcal{X}_t)$ and $\pi_b(\mathcal{X}_t)$ denote policies for players A and B respectively, we then have the following definition:

Definition 10.2.1. *A set of policies $\pi_a(\mathcal{X})$ and $\pi_b(\mathcal{X})$ are said to be in a Nash Equilibrium, for the game $\mathcal{G}$, if:*

$$\forall i \in \{a, b\}, \; C_i(\pi_a, \pi_b) = \min_{\pi} \left[C_i\left(\pi, \pi^{(a,b)\setminus i}\right)\right]$$

That is, each players policy is optimal given that the policy of the other player is fixed.

If the players' policies are in Nash equilibrium then neither player has an incentive to unilaterally change their policy, thus the Nash equilibrium acts as a natural solution concept for multi-player games. Our goal will therefore be to find a Nash equilibrium for our game. We use open-loop control laws $\pi_i = \{\mathbf{u}_i^0, \mathbf{u}_i^1, \dots \mathbf{u}_i^{T-1}\}$ as the policy parameterization, although it is theoretically possible to work with more powerful policy parameterizations.

Semi-Stochastic Game

In our initial problem formulation, we assumed that both of the systems under consideration have stochastic dynamics. This is a realistic assumption, however it creates a very difficult objective function from an optimization standpoint. When both systems have stochastic dynamics, the objective takes the form of an expectation over the joint distribution of F_a and F_b, which then needs to be estimated in order to approximate a solution.

A more tractable approach is to treat the problem as semi-stochastic, where each player assumes the other player acts in a noise-free manner, but treats their own dynamics as stochastic. The underlying assumption behind this approach is that the other player will be able to effectively correct for any stochastic disturbances that they encounter, with minimal changes to their trajectory. This semi-stochastic game set-up can be described using dynamics and costs for each player that take the form:

$$F_a(\mathbf{x}_a^t, \mathbf{v}_a^t, \mathbf{u}_a^t) = \begin{pmatrix} \mathbf{f}_a(\mathbf{x}_{a,n}^t, \mathbf{v}_a^t) \\ \mathbf{f}_a(\mathbf{x}_{a,d}^t, \mathbf{u}_a^t) \end{pmatrix}, \quad \mathbf{v}_a^t \sim \mathcal{N}(\mathbf{u}_a^t, \Sigma_a) \tag{10.7}$$

$$F_b(\mathbf{x}_b^t, \mathbf{v}_b^t, \mathbf{u}_b^t) = \begin{pmatrix} \mathbf{f}_b(\mathbf{x}_{b,n}^t, \mathbf{v}_b^t) \\ \mathbf{f}_b(\mathbf{x}_{b,d}^t, \mathbf{u}_b^t) \end{pmatrix}, \quad \mathbf{v}_b^t \sim \mathcal{N}(\mathbf{u}_b^t, \Sigma_b) \tag{10.8}$$

The state of each player consists of a noisy copy of the state, $\mathbf{x}_{i,n}^t$ and a deterministic copy $\mathbf{x}_{i,d}^t$, with $i \in \{a, b\}$. The objective functions then take the form:

$$C_a = \mathbb{E}_{\mathbf{f}_a}\left[\phi(\mathbf{x}_{a,n}^T, \mathbf{x}_{b,d}^T) + \sum_{t=0}^{T-1} \mathcal{L}(\mathbf{x}_{a,n}^t, \mathbf{x}_{b,d}^t, \mathbf{u}_a^t, \mathbf{u}_b^t)\right] \tag{10.9}$$

$$C_b = \mathbb{E}_{\mathbf{f}_b}\left[\phi(\mathbf{x}_{b,n}^T, \mathbf{x}_{a,d}^T) + \sum_{t=0}^{T-1} \mathcal{L}(\mathbf{x}_{b,n}^t, \mathbf{x}_{a,d}^t, \mathbf{u}_a^t, \mathbf{u}_b^t)\right] \tag{10.10}$$

Note how each players' objective function only considers the stochastic copy of their own state, and the deterministic copy of the other player's state. This eliminates the need to compute the expectation over the joint distribution, but still enables some stochasticity to enter into the problem through the dynamics.

Best Response Model Predictive Control

Our problem formulation results in a differential game with non-linear stochastic dynamics, and a potentially non-convex cost function. This type of problem generality is required for controlling ground vehicles in agile close-quarters maneuvers, however it makes finding a solution in an online optimization framework extremely difficult. Here, we propose a simple iterative solution method, best response iteration, which has Nash equilibrium as fixed points. The fundamental object that we consider in iterated best response is the *best response set*:

Definition 10.2.2. *Assume that player B follows the policy π_b, then the best response set for player A is the set:*

$$\left\{\pi_a \mid C_a(\pi_a, \pi_b) = \min_{\pi}\left[C_a(\pi, \pi_b)\right]\right\} \tag{10.11}$$

A similar definition applies for best response set for player B. Observe how if both players are playing policies that are best responses to each other, then the game is in a Nash equilibrium. Now let $\mathcal{H}_a(\mathcal{X}, \pi_b)$ and $\mathcal{H}_b(\mathcal{X}, \pi_a)$ be functions which take the current state of the game and the opponents strategy, and returns a strategy from the best-response set. The iterative best response system is then defined by the dynamical system:

$$\pi_a^{k+1} = \mathcal{H}_a(\mathcal{X}, \pi_b^k) \tag{10.12}$$

$$\pi_b^{k+1} = \mathcal{H}_b(\mathcal{X}, \pi_a^k) \tag{10.13}$$

At each iteration of this system, each player responds by playing their best-response to the other players current policy, and a point is a fixed point in the system if and only if it is a Nash equilibrium. For certain classes of games exhibiting cooperative properties (e.g. potential games), the iterated best response system converges to a Nash Equilibrium. However, in general the system may not converge, but instead cycle between policies. Our key assumption is that the dynamic constraints of the system combined with the stochastic nature of the environment are enough to either prevent cycling, or quickly break it if it does occur. This is similar to symmetry breaking in standard stochastic optimal control theory [49].

In order to utilize iterated best response in an online optimization scheme, we need a method for rapidly approximating a best-response. In order to perform this computation, we use the MPPI controller (Alg. 1). Note that we are using the standard MPPI algorithm here as opposed to the robust version, this is because these experiments pre-dated the work on robust MPPI. In fact, difficulties with tuning a cost function that could enable safe agile interaction between two vehicles was one of the motivating factors in developing a more robust version of the MPPI algorithm.

The best response model predictive control algorithm combines the concept of iterated best response with the information theoretic optimization procedure from chapter 4. The algorithm starts with an initial guess of the other vehicle's control plan, and then estimates the current state of both vehicles. Then, it uses the information theoretic optimization procedure to simultaneously compute an estimate of the control sequences for both itself and the other agent. Next, it executes the first element in the control sequence, and lastly it slides down the remaining elements in the sequence by one timestep and uses that sequence to start the optimization on the next round.

A key detail in the best response MPC algorithm is that the best responses for both players are computed simultaneously. This is important, because it enables calling the two MPPI optimizers in parallel, which significantly reduces the run time of a single best response iteration. The best response MPC (BR-MPC) algorithm is given in Alg. 7. Also, notice how the semi-stochastic game formulation enables the MPPI algorithm to run with only a single sampling loop, if the game were formulated as fully stochastic, there would need to be a sampling loop for both players. This would either significantly increase the number of samples required, or drastically increase the variance of the stochastic optimization.

Algorithm 7: BR-MPC for Player i

```
Given: $\mathcal{G}$: Differential game description;
MPPI-OPT: MPPI optimizer;
$U_a, U_b$: Initial control sequences;
$\theta_{\text{MPPI}}$: MPPI hyper-parameters;
while task not completed do
    $\mathcal{X} \leftarrow$ GameStateEstimator();
    $U'_a =$ MPPI$(U_a, U_b, \mathcal{X}, C_a, \mathbf{f}_a, \mathbf{f}_b, \theta_{\text{MPPI}})$;
    $U'_b =$ MPPI$(U_b, U_a, \mathcal{X}, C_b, \mathbf{f}_b, \mathbf{f}_a, \theta_{\text{MPPI}})$;
    $U_a = U'_a$;
    $U_b = U'_b$;
    Execute$(\mathbf{u}_i^0)$;
    for $t \leftarrow 0$ to $T - 2$ do
        $U_a^t \leftarrow U_a^{t+1}$ ;
        $U_b^t \leftarrow U_b^{t+1}$ ;
    $U_a^{T-1} = 0$;
    $U_b^{T-1} = 0$;
```

We want to emphasize that the algorithm described here (Alg. 7) is meant for a *single vehicle*, and that the two vehicles are not iteratively transmitting any internal planning information to each other. Even though in the BR-MPC algorithm two control plans are computed, one is simply an informed guess at the other vehicles motion, and only one of the control plans is actually used to control a vehicle.

Implementation Details

The goal in our experiments was to have two AutoRally vehicles autonomously operate in close proximity to each other, and to test the limits of the best response MPC algorithm as the target speed was increased. In order to implement the BR-MPC controller for this task, we require two key components: a cost function encoding the task, and vehicle dynamics models.

Cost Function Design

The state-dependent cost for the task that we are trying to achieve has three main components: stay on the track, go close to the set target speed, and stay 1 meter away from the other vehicle, but do not hit the other vehicle. This last condition is particularly challenging since at 1 meter distance between center of masses, the vehicles are nearly touching, so the algorithm has to balance this objective with the stochastic dynamics of the system. We describe the cost from player A's perspective, the cost for player B is the same, but with the a and b subscripts swapped. Recall that the state of the game is $\mathcal{X}_t = (\mathbf{x}_a^t \ \mathbf{x}_b^t)$. For player A, the cost components encoding the first two instructions are only concerned with the $\mathbf{x}_a$ portion of the state. For the individual portion of the cost we use Eq. (6.3).

In addition to the individual portion of the cost function, the two vehicles have an interaction cost which forces them to stay close to each other:

$$C_a^{mix} = w_1 \left(\|(x_a^{pos}, y_a^{pos}) - (x_b^{pos}, y_b^{pos})\| - D \right)^2 \tag{10.14}$$

where D is the target distance (set to 1 meter in all of our experiments). Lastly, a crashing cost is added which is activated if the vehicle either collides with the other:

$$C_a^{crash} = w_2 \beta^t I \tag{10.15}$$

here β is a time-decay rate, and I is an indicator variable which is 1 is the vehicle crashed and 0 otherwise.

Vehicle Dynamics Models

Pushing the vehicles to their limits while operating nearby each other requires precise agile maneuvering. This means that we need a high-fidelity, non-linear model capable of capturing sliding dynamics, and therefore rules out using simple kinematic models. Additionally, the nature of the dirt track that we perform experiments on makes traditional system identification difficult, so we use a data-driven approach and model the dynamics of the vehicle with a multi-layer neural network. This is the same network used in 4.

Although the neural network that we trained is highly accurate, a drawback is that it is computationally expensive, and running two simultaneous MPPI optimizations with the neural network is too slow to operate in real-time. As a solution we use two dynamics models, the more accurate neural network is used for the optimization of the vehicle being controlled, and a faster, less accurate model is used to predict the motion of the other vehicle. The faster model is the non-linear basis function model from chapter 4. The idea behind this double model approach is that, since we do not actually have to control the other vehicle, a less accurate model can be effective if it captures the dynamics constraints of the other vehicle.

Experiments

Experimental data was collected using a pair of AutoRally robots at the Georgia Tech Marietta Street Autonomous Racing Facility. Each AutoRally robot ran the BR-MPC algorithm on-board in order to control the vehicle and estimate the future motion of the other vehicle. In principle, no vehicle to vehicle communication is required for BR-MPC where the requisite information can be inferred from onboard sensor information. However, in order to simplify the experiments we chose to enable the AutoRally robots to share their pose estimates over low-bandwidth XBee Radios. This removes the requirement of estimating the other vehicles pose using on-board sensors, which is a difficult perception problem by itself.

Each robot runs a standalone state estimator to produce an accurate state estimate at 200 Hz. The high rates are necessary for high speed, real time control, but would saturate the XBee network with even two vehicles within communication range. For that reason,

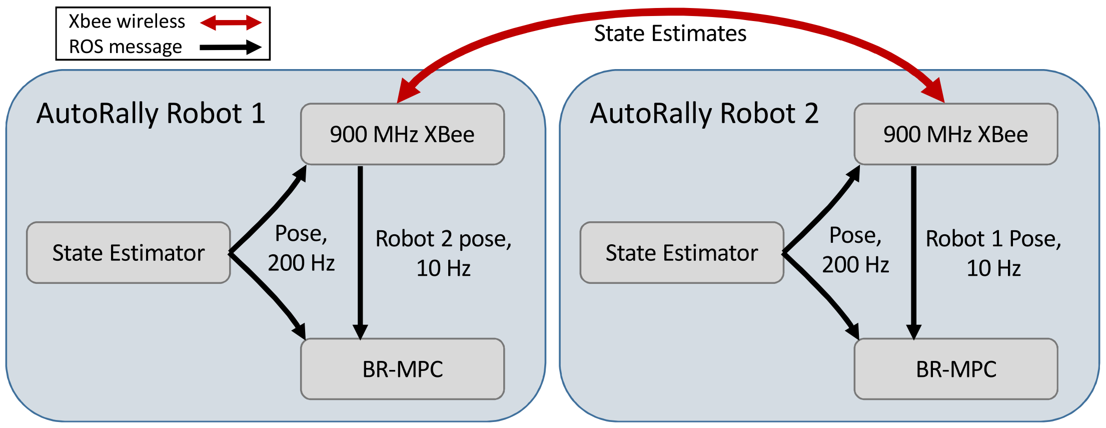


Figure 10.2: Vehicle to vehicle system for broadcasting state estimates at 10 Hz from one robot to all other robots within communications range.

we implemented a configurable rate, currently set at 10Hz, to down sample pose estimates before transmission over the XBee network. Figure 10.2 shows the wireless pose communication system for two vehicles and the routing of signals within each robot between the state estimator, XBee interface software, and each robots BR-MPC controller.

As a baseline comparison, we implemented a version of MPPI that treated the other vehicle as a "dumb" obstacle with a constant velocity. All of the other implementation details are the same, but instead of using a second MPPI optimization to simulate the motion of the other vehicle, it is simulated according to the linear extrapolation:

$$x_i^{pos}(t) = x_i^{pos}(0) + t\left(\mathrm{d}x_i^{pos}(0)\right) \tag{10.16}$$

$$y_i^{pos}(t) = y_i^{pos}(0) + t\left(\mathrm{d}y_i^{pos}(0)\right) \tag{10.17}$$

This type of model is reasonably accurate on the straights, and on the corners at slow speeds. However, it quickly starts to become inaccurate around corners as the vehicle speed is increased. We refer to this method as Velocity-Obstacle Model Predictive Control (VO-MPC for short).

We tested both the BR-MPC algorithm and the VO-MPC algorithm at the task of maneuvering two AutoRally vehicles around an elliptical dirt track. The desired distance

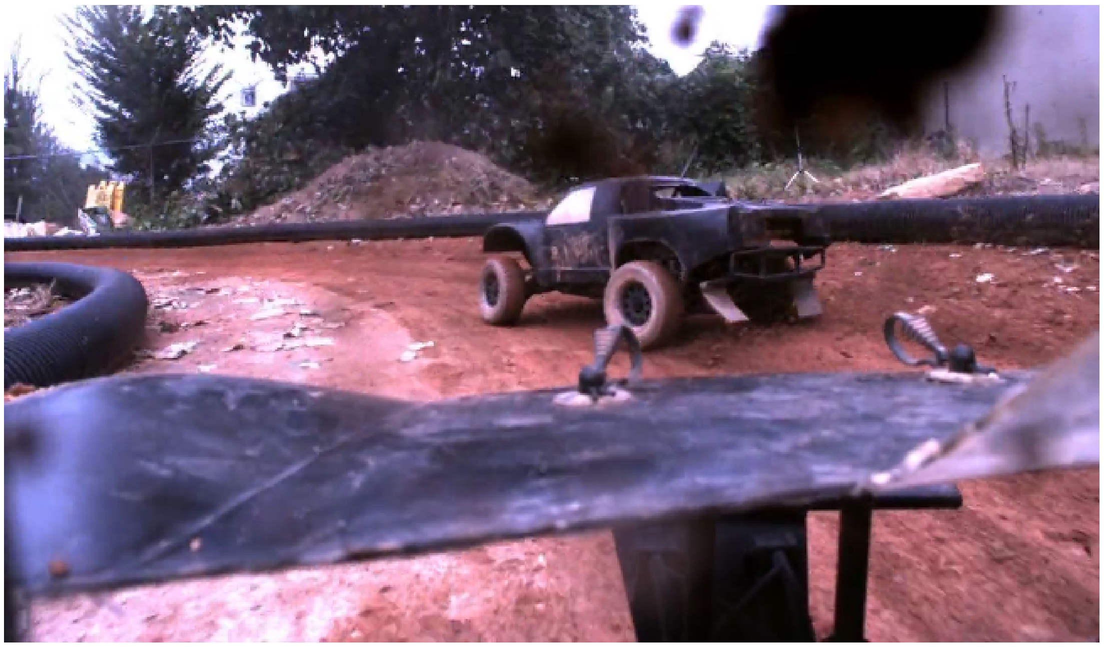


Figure 10.3: View from the rear vehicle exiting a turn with BR-MPC at the 10 m/s target.

between the two vehicles center of mass was one meter, and the target speed was varied between 5, 6, 7, 8, 9, and 10 m/s. Each speed setting was tested for 10 laps around the test track, except for the 7 m/s VO-MPC setting which was only run for four laps due to safety concerns. This amounts to a total of 84 laps, which is roughly 3.5 miles worth of driving data for each robot. Figure 10.4 and Table 10.1 show the change in distance between the two vehicles as the desired speed is increased from 5 to 10 m/s.

At the slowest speed setting of 5 m/s, there is minimal performance difference between the simple baseline method, and the BR-MPC method. This is expected, since the distances the AutoRally vehicles travel during the two second time horizon are very short and can roughly be approximated with straight lines. They both maintain approximately 2.5 m between the two vehicles, even though the desired distance is 1 m, this extra cushion is automatically included due to the collision penalty and the stochastic dynamics. As the desired speed is increased, the VO-MPC method quickly degrades. At the 6 m/s target, VO-MPC results in 4 distinct collision events during the 10-lap trial run, with one collision requiring a manual take-over of the system. Then, at the 7 m/s target, the two vehicles

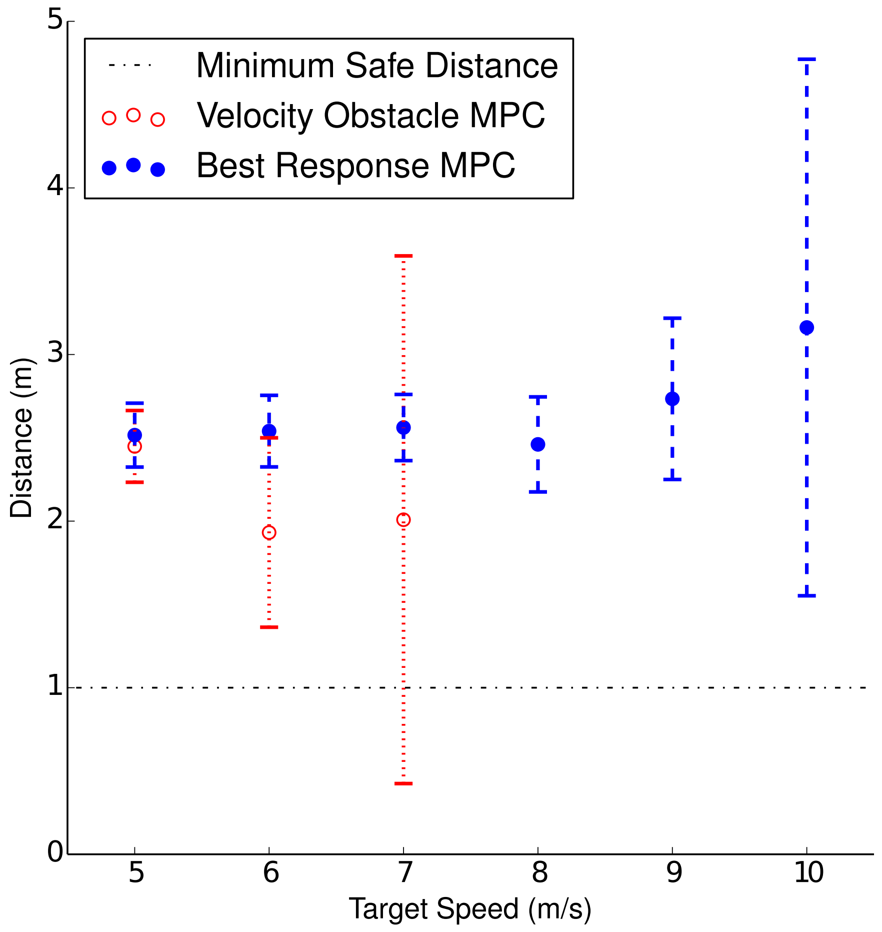


Figure 10.4: Following distance and standard deviation for best response dynamics and velocity obstacles. Distance is measured from the center of mass, so a distance of less than one meter indicates a collision if the two vehicles are oriented end-to-end.

Table 10.1: Following Distance Performance

Method	Target	Min Dist.(m)	Max Dist.(m)	Avg. Dist (m)
BR-MPC	5 m/s	2.04	3.40	2.52
BR-MPC	6 m/s	1.92	3.22	2.54
BR-MPC	7 m/s	2.03	3.09	2.56
BR-MPC	8 m/s	1.80	3.80	2.46
BR-MPC	9 m/s	1.83	4.91	2.73
BR-MPC	10 m/s	1.59	10.65	3.16
VO-MPC	5 m/s	1.79	3.25	2.45
VO-MPC	6 m/s	0.58	5.09	1.93
VO-MPC	7 m/s	0.22	7.03	2.01

consistently collide with each other when controlled using VO-MPC.

At the 6 m/s and 7 m/s targets, the BR-MPC method performs nearly identically, in

Table 10.2: BR-MPC Performance Statistics (Lead / Trail)

Target	Avg. Lap Time (s)	Max Speed (m/s)	Max Slip (Deg.)
5 m/s	20.85 / 20.84	3.8 / 4.3	5.6 / 8.8
6 m/s	15.53 / 15.51,	5.1 / 5.4	9.0 / 10.3
7 m/s	12.82 / 12.79	6.0 / 6.2	11.6 / 15.2
8 m/s	11.06 / 11.07	7.2 / 7.3	18.6 / 20.4
9 m/s	9.96 / 9.94	8.0 / 7.9	19.0 / 25.6
10 m/s	9.66 / 9.65	8.2 / 8.1	23.9 / 26.0

terms of following distance, as the 5 m/s target. As the desired speed is further increased from 7 m/s to 10 m/s, the mean and variance of the following distance both increases. However, the two vehicles avoid ever colliding with each other during the trial runs. Table 10.2 shows the lap statistics for the lead and trail vehicles with the BR-MPC method during the trial runs. At the highest speed target, the vehicles obtain maximum speeds over 8 m/s, while maintaining an average distance of 3.16 meters between their center of mass (this is 2.16 meters from bumper-to-bumper). Additionally, at the highest speed target both vehicles attain a significant side-slip angle, which is the difference between the heading angle and the vehicle's velocity vector, indicating highly dynamic maneuvers. One of the key benefits of the stochastic optimization approach is that only the high-level objective needs to be specified, and the precise method for achieving that objective is left to the autonomy system. This is illustrated by Fig. 10.5, at the 5 m/s target the two vehicles follow nearly the same track with the trail vehicle directly behind the lead vehicle. However, at the 10 m/s target, the behavior looks considerably different, with the two vehicles entering the turns in a staggered formation. This formation enables there to be more room for error in the estimate of the other vehicles longitudinal direction, which is critical for operating at high speeds.

These experiments show that the best response model predictive control strategy can control fast ground vehicles operating near each other. This is one of the first approaches to autonomous racing which takes a game-theoretic optimization approach to the problem, and which can anticipate and react to the other vehicles in the environment only using

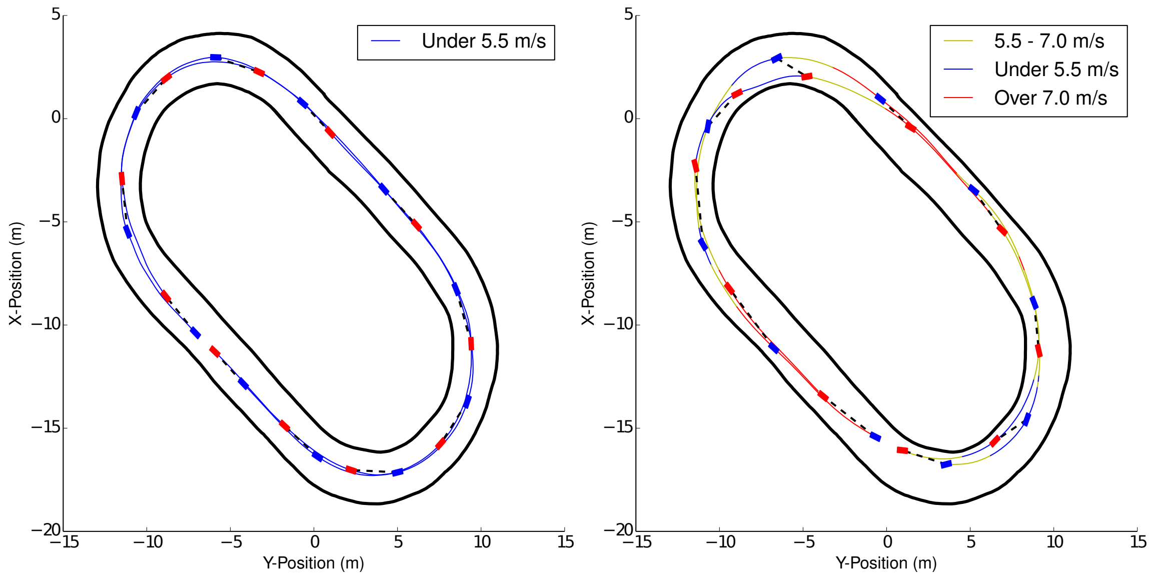


Figure 10.5: Top: Following behavior at the 5 m/s target. Bottom: Following behavior at 10 m/s target. The blue marker indicates the lead vehicle, and the red the trail vehicle, with the dashed line showing which markers are synced in time. Colors on the trajectory traces indicate vehicle speed range.

knowledge of the other vehicles pose, dynamics, and objective.

This is only the beginning of research into what is an extremely challenging problem. We avoided attempts at performing passing maneuvers or setting up the game in an explicitly adversarial method. Part of the issue was the difficulty in tuning MPPI for these experiments, which is a challenge that has mostly been overcome with robust MPPI and applying robust MPPI in this framework could lead to better results. The more fundamental issue though, is that the best response mechanism we have proposed is only useful for obtaining coarse grained information about other agents actions. This is because the we have restricted the strategies to open-loop feedback policies, using a more powerful policy representation is possible, but complicates the problem significantly. Despite these issues, exploring generalization of best response MPC is a good initial step at achieving head-to-head autonomous racing, and it moves us closer to the goal of out-performing an expert human driver in a head-to-head race.

Appendices

APPENDIX A
DEGENERATE DIFFUSION RADON-NIKODYM DERIVATIVES

We consider two stochastic diffusion processes of the form:

$$\mathrm{d}\mathbf{x}_1 = \mathbf{F}(\mathbf{x}, \mathbf{u})\mathrm{d}t + \mathbf{B}(\mathbf{x})\mathrm{d}\mathbf{w}_1 \tag{A.1}$$

$$\mathrm{d}\mathbf{x}_2 = \mathbf{F}(\mathbf{x}, \mathbf{v})\mathrm{d}t + \mathbf{B}(\mathbf{x})\mathrm{d}\mathbf{w}_2 \tag{A.2}$$

Where $\mathbf{x} \in \mathbb{R}^n, \mathbf{u} \in \mathbb{R}^m, \mathbf{v} \in \mathbb{R}^m, \mathrm{d}\mathbf{w} \in \mathbb{R}^p$ and the diffusion matrix $\mathbf{B}(\mathbf{x}) \in \mathbb{R}^{n\times p}$ with (possibly state-dependent) rank $k(\mathbf{x}) > 0$. Our goal is to find the Radon-Nikodym derivative between the distributions $\mathbb{P}, \mathbb{Q}$ induced by (A.1) and (A.2) respectively. This result is well known in the case of a square invertible diffusion matrix $\mathbf{B}(\mathbf{x})$, and comes as a result of Girsanov's theorem. In the following we will derive a form for the Radon-Nikodym derivative under the relaxed assumption:

Condition 2. *Consider the two diffusion processes (A.1) and (A.2), and suppose that* $\forall \mathbf{x} \in \mathbb{R}^n$*,* $\forall \mathbf{u} \in \mathbb{R}^m$*, and* $\forall \mathbf{v} \in \mathbb{R}^m$*,* $\exists \mathbf{y} \in \mathbb{R}^p$ *such that:*

$$\mathbf{B}(\mathbf{x})\mathbf{y} = \mathbf{F}(\mathbf{x}, \mathbf{v}) - \mathbf{F}(\mathbf{x}, \mathbf{u}) \tag{A.3}$$

This assumption ensures that any change in the drift can equivalently be achieved by random noise, and it ensures the absolute continuity of the two measures. It is a generalization of the usual requirement that $\mathbf{B}(\mathbf{x})\mathbf{B}(\mathbf{x})^{\mathrm{T}}$ be invertible, since in that case the system of linear equations described by (A.3) has at least one solution. However, it also holds in situations where $\mathbf{B}(\mathbf{x})\mathbf{B}(\mathbf{x})^{\mathrm{T}}$ is not invertible. For instance suppose that the dynamics are control affine, with the diffusion and control matrices identical. Then we have: $\mathbf{G}(\mathbf{x})\mathbf{y} = \mathbf{G}(\mathbf{x})(\mathbf{v} - \mathbf{u})$, which is clearly satisfied for any non-zero $\mathbf{G}(\mathbf{x})$ and any choice of $\mathbf{u}, \mathbf{v}$.

The strategy for deriving the radon-nikodym derivative is as follows: first we discretize the system and split it into a non-degenerate system and a degenerate system, next we derive the expression for a one-step transition probability of the discretized system which takes the form of a gaussian and an impulse (dirac-delta function), then we show that the impulse function cancels out in the ratio. Once this is done we're left with a product of gaussians, which when we take the limit as $\Delta t \to 0$ produces the desired result.

A.1 Discretization and Non-Degenerate Subsystem

The first step in the derivation of the ratio is to make a discrete time approximation of the dynamics:

$$\Delta \mathbf{x}_1 = \mathbf{F}(\mathbf{x}, \mathbf{u})\Delta t + \mathbf{B}(\mathbf{x})\epsilon_1 \sqrt{\Delta t} \tag{A.4}$$

$$\Delta \mathbf{x}_2 = \mathbf{F}(\mathbf{x}, \mathbf{v})\Delta t + \mathbf{B}(\mathbf{x})\epsilon_2 \sqrt{\Delta t} \tag{A.5}$$

Here ϵ_1 and ϵ_2 are p-dimensional standard normal random variables. We can then approximately describe the distributions $\mathbb{P}$ and $\mathbb{Q}$ in terms of the probability density functions:

$$\mathbb{P} \approx \prod_{t=0}^{T} p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) \tag{A.6}$$

$$\mathbb{Q} \approx \prod_{t=0}^{T} q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) \tag{A.7}$$

Now, let $0 < k \leq p$ be the rank of $\mathbf{B}(\mathbf{x})$. Then, we can select k rows from $\mathbf{B}(\mathbf{x})$ such that they are all linearly independent. We denote the set of indices of these rows as ζ so that $\mathbf{B}_{\zeta_1}(\mathbf{x})$ is the first row selected, $\mathbf{B}_{\zeta_2}(\mathbf{x})$ is the second, etc. Then the dynamics of the ζ

sub-system of $\mathbb{P}$ are described as:

$$\begin{aligned}
\Delta \mathbf{x}_{\zeta_1} &= \mathbf{F}_{\zeta_1}(\mathbf{x}, \mathbf{u})\Delta t + \mathbf{B}_{\zeta_1}(\mathbf{x})\epsilon_1\sqrt{\Delta t} \\
\Delta \mathbf{x}_{\zeta_2} &= \mathbf{F}_{\zeta_2}(\mathbf{x}, \mathbf{u})\Delta t + \mathbf{B}_{\zeta_2}(\mathbf{x})\epsilon_1\sqrt{\Delta t} \\
&\vdots \\
\Delta \mathbf{x}_{\zeta_k} &= \mathbf{F}_{\zeta_k}(\mathbf{x}, \mathbf{u})\Delta t + \mathbf{B}_{\zeta_k}(\mathbf{x})\epsilon_1\sqrt{\Delta t}
\end{aligned}$$

And we can denote this system of equations more compactly as:

$$\Delta \tilde{\mathbf{x}}_1 = \tilde{\mathbf{F}}(\mathbf{x}, \mathbf{u})\Delta t + \tilde{\mathbf{B}}(\mathbf{x})\epsilon_1\sqrt{\Delta t} \tag{A.8}$$

This sub-system describes a non-degenerate diffusion process, since the diffusion matrix, $\tilde{\mathbf{B}}(\mathbf{x})$, has rank k (it was explicitly constructed from k linearly independent vectors). This implies that $\tilde{\Sigma}(\mathbf{x}) = \tilde{\mathbf{B}}(\mathbf{x})\tilde{\mathbf{B}}(\mathbf{x})^{\mathrm{T}}$ has rank k as well. Thus $\tilde{\Sigma}(\mathbf{x})$ is invertible, and the one-step transition distribution is described by a multi-variate Gaussian:

$$p(\Delta \tilde{\mathbf{x}}_t | \mathbf{x}_t, \mathbf{u}_t) = Z^{-1} \exp(-\frac{1}{2\Delta t}\left(\Delta \tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x}, \mathbf{u})\Delta t\right)^{\mathrm{T}} \tilde{\Sigma}(\mathbf{x})^{-1} \left(\Delta \tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x}, \mathbf{u})\Delta t\right) \tag{A.9}$$

Now, since $\tilde{\mathbf{B}}(\mathbf{x})$ was constructed using all of the linearly independent rows of $\mathbf{B}(\mathbf{x})$. The rest of the rows must be linearly dependent on $\tilde{\mathbf{B}}$. This implies that we can write the remaining random variables as deterministic functions of $\tilde{\mathbf{x}}$. To see this more clearly, let s be the index of a row of $\mathbf{B}(\mathbf{x})$ where s is not in ζ. Then:

$$\Delta \mathbf{x}_s = \mathbf{F}_s(\mathbf{x}, \mathbf{u})\Delta t + \mathbf{B}_s(\mathbf{x})\epsilon_1\sqrt{\Delta t} \tag{A.10}$$

Using the linear dependence relationship we can write $\mathbf{B}_s(\mathbf{x})$ as:

$$\mathbf{B}_s(\mathbf{x}) = \sum_{i=1}^{k} \alpha_i \tilde{\mathbf{B}}_{\zeta_i}(\mathbf{x}) \tag{A.11}$$

So we have:

$$\Delta\mathbf{x}_s = \mathbf{F}_s(\mathbf{x},\mathbf{u})\Delta t + \left(\sum_{i=1}^{k} \alpha_i \tilde{\mathbf{B}}_{\zeta_i}(\mathbf{x})\right)\epsilon_1\sqrt{\Delta t}$$
$$\Delta\mathbf{x}_s = \mathbf{F}_s(\mathbf{x},\mathbf{u})\Delta t + \sum_{i=1}^{k}\left(\alpha_i \tilde{\mathbf{B}}_{\zeta_i}(\mathbf{x})\epsilon_1\sqrt{\Delta t}\right)$$

And each $\mathbf{B}_{\zeta_i}(\mathbf{x})\epsilon$ can be expressed in terms of $\Delta\mathbf{x}_{\zeta_i}$:

$$\mathbf{B}_{\zeta_i}(\mathbf{x})\epsilon_1 = -\left(\mathbf{F}_{\zeta_i}(\mathbf{x},\mathbf{u})\Delta t - \Delta\mathbf{x}_{\zeta_i}\right) \tag{A.12}$$

So the final result is that:

$$\Delta\mathbf{x}_s = \mathbf{F}_s(\mathbf{x},\mathbf{u})\Delta t - \sum_{i=0}^{k} \alpha_i\left(\mathbf{F}_{\zeta_i}(\mathbf{x},\mathbf{u})\Delta t - \Delta\mathbf{x}_{\zeta_i}\right) \tag{A.13}$$

Therefore, knowing $\tilde{\mathbf{x}}_t$ completely determines the rest of the variable transitions at time t. Denoting the variables not included in $\tilde{\mathbf{x}}$ as $\hat{\mathbf{x}}$. We have the following:

$$p(\Delta\hat{\mathbf{x}}|\Delta\tilde{\mathbf{x}},\mathbf{x},\mathbf{u}) = \delta\left(\Delta\hat{\mathbf{x}} - h(\Delta\tilde{\mathbf{x}})\right) \tag{A.14}$$

With the function $h(\Delta\tilde{\mathbf{x}})$ defined as:

$$h_s(\Delta\tilde{\mathbf{x}}_s) = \mathbf{F}_s(\mathbf{x},\mathbf{u})\Delta t - \sum_{i=0}^{k} \alpha_i\left(\mathbf{F}_{\zeta_i}(\mathbf{x},\mathbf{u})\Delta t - \Delta\mathbf{x}_{\zeta_i}\right) \tag{A.15}$$

and δ is the dirac delta function which is zero everywhere except when its argument is zero and is then a unit impulse.

A.2 Ratio of One-Step Dynamics

Using the results from the previous section, it is possible to describe the full one-step transition probabilities as:

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t) = p(\Delta\mathbf{x}_t|\mathbf{x}_t,\mathbf{u}_t) = p(\Delta\tilde{\mathbf{x}}_t,\Delta\hat{\mathbf{x}}_t|\mathbf{x}_t,\mathbf{u}_t) \tag{A.16}$$

and using the chain rule this is:

$$= p(\Delta\tilde{\mathbf{x}}_t|\mathbf{x}_t,\mathbf{u}_t)p(\Delta\hat{\mathbf{x}}_t|\Delta\tilde{\mathbf{x}}_t,\mathbf{x}_t,\mathbf{u}_t) \tag{A.17}$$

The first term is the multi-variate Gaussian describing the transition probability for the non-degenerate sub-system, and the second term is the dirac-delta prescribing the result of the remaining variable changes given the changes in the non-degenerate subsystem. Mathematically this is expressed as:

$$= \left(Z^{-1}\exp(-\frac{1}{2\Delta t}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t\right)^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t\right)\right)\delta\left(\Delta\hat{\mathbf{x}} - h(\Delta\tilde{\mathbf{x}})\right) \tag{A.18}$$

Now an identical procedure can be done for $q(\mathbf{x}_{t+1}|\mathbf{x}_t,\mathbf{u}_t)$, so that we have:

$$q = \left(Z^{-1}\exp(-\frac{1}{2}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\Delta t\right)^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\Delta t\right)\right)\delta\left(\Delta\hat{\mathbf{x}} - g(\Delta\tilde{\mathbf{x}})\right) \tag{A.19}$$

where:

$$g_s(\Delta\tilde{\mathbf{x}}) = \mathbf{F}_s(\mathbf{x},\mathbf{v})\Delta t - \sum_{i=0}^{k}\alpha_i\left(\mathbf{F}_{\zeta_i}(\mathbf{x},\mathbf{v})\Delta t - \Delta\mathbf{x}_{\zeta_i}\right) \tag{A.20}$$

Note that the α_i terms are the same for both p, and q. This comes from the fact that the α_i terms are chosen based on the relationship between $\tilde{\mathbf{B}}(\mathbf{x})$ and $\hat{\mathbf{B}}(\mathbf{x})$ and nothing else.

Dividing these two terms yields the ratio between the one-step probabilities as:

$$\frac{p}{q} = \frac{\left(Z^{-1}\exp(-\frac{1}{2\Delta t}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t\right)^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t\right)\right)\delta\left(\Delta\hat{\mathbf{x}} - h(\Delta\tilde{\mathbf{x}})\right)}{\left(Z^{-1}\exp(-\frac{1}{2\Delta t}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\Delta t\right)^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\Delta t\right)\right)\delta\left(\Delta\hat{\mathbf{x}} - g(\Delta\tilde{\mathbf{x}})\right)} \tag{A.21}$$

In order to proceed it is necessary to remove the dirac delta functions. This is possible if we can show that the two conditions are actually identical. In other words we need to show that $h(\Delta\tilde{\mathbf{x}}) = g(\Delta\tilde{\mathbf{x}})$. If this is the case, then the two delta functions will always be in agreement, and therefore cancel out. Consider:

$$h_s(\Delta\tilde{\mathbf{x}}) - g_s(\Delta\tilde{\mathbf{x}}) \tag{A.22}$$

$$= \mathbf{F}_s(\mathbf{x},\mathbf{u})\Delta t - \sum_{i=0}^{k}\alpha_i\left(\mathbf{F}_{\zeta_i}(\mathbf{x},\mathbf{u})\Delta t - \Delta\mathbf{x}_{\zeta_i}\right) - \left(\mathbf{F}_s(\mathbf{x},\mathbf{v})\Delta t - \sum_{i=0}^{k}\alpha_i\left(\mathbf{F}_{\zeta_i}(\mathbf{x},\mathbf{v})\Delta t - \Delta\mathbf{x}_{\zeta_i}\right)\right) \tag{A.23}$$

$$= \left(\mathbf{F}_s(\mathbf{x},\mathbf{u}) - \mathbf{F}_s(\mathbf{x},\mathbf{v})\right)\Delta t - \sum_{i=0}^{k}\alpha_i\left(\mathbf{F}_{\zeta_i}(\mathbf{x},\mathbf{u}) - \mathbf{F}_{\zeta_i}(\mathbf{x},\mathbf{v})\right) \tag{A.24}$$

Now recall the assumption that $\forall\mathbf{x},\mathbf{u},\mathbf{v},\exists\mathbf{y}$ such that $\mathbf{B}(\mathbf{x})\mathbf{y} = \mathbf{F}(\mathbf{x},\mathbf{u}) - \mathbf{F}(\mathbf{x},\mathbf{v})$. Let us pick some $\mathbf{y}$ which satisfies this condition, we can then substitute:

$$\left(\mathbf{F}_s(\mathbf{x},\mathbf{u}) - \mathbf{F}_s(\mathbf{x},\mathbf{v})\right)\Delta t - \sum_{i=0}^{k}\alpha_i\left(\mathbf{F}_{\zeta_i}(\mathbf{x},\mathbf{u}) - \mathbf{F}_{\zeta_i}(\mathbf{x},\mathbf{v})\right) = \mathbf{B}_s(\mathbf{x})\mathbf{y} - \sum_{i=0}^{k}\alpha_i\mathbf{B}_{\zeta_i}\mathbf{y} \tag{A.25}$$

But, recall that $\sum_{i=0}^{k}\alpha_i\mathbf{B}_{\zeta_i} = \mathbf{B}_s(\mathbf{x})$. So we're left with:

$$\mathbf{B}_s(\mathbf{x})\mathbf{y} - \mathbf{B}_s(\mathbf{x})\mathbf{y} = 0 \tag{A.26}$$

Therefore, the delta functions are identical, and cancel out of the equation leaving us with:

$$\frac{p}{q} = \frac{\left(\exp(-\frac{1}{2\Delta t}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t\right)^{\mathrm{T}} \tilde{\Sigma}(\mathbf{x})^{-1}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t\right)\right)}{\left(\exp(-\frac{1}{2\Delta t}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\Delta t\right)^{\mathrm{T}} \tilde{\Sigma}(\mathbf{x})^{-1}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\Delta t\right)\right)} \tag{A.27}$$

A.3 Taking the limit

The last part of the derivation is to simplify and take the limit as $\Delta t \to 0$ of p/q. Expanding out A.27 yields:

$$\begin{aligned} = \exp\big(-\frac{1}{2\Delta t}&\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t\right)^{\mathrm{T}} \tilde{\Sigma}(\mathbf{x})^{-1}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t\right) + \\ &\frac{1}{2\Delta t}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\Delta t\right)^{\mathrm{T}} \tilde{\Sigma}(\mathbf{x})^{-1}\left(\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\Delta t\right)\big) \end{aligned}$$

And then expanding the term inside the exponent further yields:

$$\begin{aligned} &-\frac{1}{2\Delta t}\Delta\tilde{\mathbf{x}}^{\mathrm{T}}\tilde{\Sigma}^{-1}(\mathbf{x})\Delta\tilde{\mathbf{x}} + \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\tilde{\Sigma}^{-1}\Delta\tilde{\mathbf{x}} - \frac{1}{2}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t + \\ &\frac{1}{2\Delta t}\Delta\tilde{\mathbf{x}}^{\mathrm{T}}\tilde{\Sigma}^{-1}(\mathbf{x})\Delta\tilde{\mathbf{x}} - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\tilde{\Sigma}^{-1}\Delta\tilde{\mathbf{x}} + \frac{1}{2}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\Delta t \end{aligned}$$

Which simplifies to:

$$-\left(\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v}) - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\right)\tilde{\Sigma}^{-1}\Delta\tilde{\mathbf{x}} - \frac{1}{2}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t + \frac{1}{2}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\Delta t$$

Now recall that $\Delta\tilde{\mathbf{x}} = \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t + \tilde{\mathbf{B}}(\mathbf{x})\epsilon_1\sqrt{\Delta t}$, and equivalently. So we have:

$$\begin{aligned} &-\left(\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v}) - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\right)\tilde{\Sigma}^{-1}\Delta\left(\tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t + \tilde{\mathbf{B}}(\mathbf{x})\epsilon_1\sqrt{\Delta t}\right) \\ &-\frac{1}{2}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})\Delta t + \frac{1}{2}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v})\Delta t \end{aligned}$$

which simplifies to:

$$\frac{p}{q} = \exp\left(-\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\tilde{\mathbf{B}}(\mathbf{x})\epsilon_1\sqrt{\Delta t} + \frac{1}{2}\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\Delta t\right) \tag{A.28}$$

Where $\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u}) = \tilde{\mathbf{F}}(\mathbf{x},\mathbf{v}) - \tilde{\mathbf{F}}(\mathbf{x},\mathbf{u})$. Lastly we have:

$$\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} = \lim_{\Delta t\to 0}\prod_{t=0}^{T}\frac{p}{q}$$

$$\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} = \lim_{\Delta t\to 0}\prod_{t=0}^{T}\exp\left(-\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\tilde{\mathbf{B}}(\mathbf{x})\epsilon_1\sqrt{\Delta t} + \frac{1}{2}\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\Delta t\right)$$

$$\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} = \lim_{\Delta t\to 0}\exp\left(\sum_{t=0}^{T}-\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\tilde{\mathbf{B}}(\mathbf{x})\epsilon_1\sqrt{\Delta t} + \frac{1}{2}\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\Delta t\right)$$

$$\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}} = \exp\left(\int_0^T -\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\tilde{\mathbf{B}}(\mathbf{x})\mathrm{d}\mathbf{w}_1 + \frac{1}{2}\int_0^T \Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\tilde{\Sigma}(\mathbf{x})^{-1}\Delta\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})^{\mathrm{T}}\mathrm{d}t\right) \tag{A.29}$$

Which is the desired result. This implies that in order to compute the Radon-Nikodym derivative between two diffusion processes it is necessary to (1) Ensure that the assumption $\mathbf{B}(\mathbf{x})\mathbf{y} = \mathbf{F}(\mathbf{x},\mathbf{u}) - \mathbf{F}(\mathbf{x},\mathbf{v})$ is satisfied, and (2) Pick out a non-degenerate sub-system which has as many equations as the rank of the diffusion matrix. Then the radon-nikodym derivative of the two diffusions is given by $A.29$.

Note the subscript of $\mathrm{d}\mathbf{w}_1$. This notation means that $\mathrm{d}\mathbf{w}_1$ is a brownian motion with respect to the first process: $\mathbb{E}_{\mathbb{P}}[\mathrm{d}\mathbf{w}_1] = 0$. However, it is generally not the case that it is a brownian motion with respect to the second one, i.e. $\mathbb{E}_{\mathbb{Q}}[\mathrm{d}\mathbf{w}_1] \neq 0$. We could have just as easily used $\mathrm{d}\mathbf{w}_2$ in place of $\mathrm{d}\mathbf{w}_1$, the two quantities are related via the transformation:

$$\mathrm{d}\mathbf{w}_1 = \tilde{\mathbf{B}}(\mathbf{x})^{-1}\tilde{\mathbf{F}}(\mathbf{x},\mathbf{v},\mathbf{u})\mathrm{d}t + \mathrm{d}\mathbf{w}_2 \tag{A.30}$$

So it is straight-forward to replace one with the other. If expectations are being taken, it is good practice to align the subscript with the distribution the expectation is taken over. Otherwise, the results can seem counter-intuitive.

APPENDIX B
GRAPHICS PROCESSING UNIT IMPLEMENTATION

In this appendix chapter we examine a few different approaches for sampling trajectories in parallel on a GPU, in the case that the system dynamics are described by a neural network. We suppose that we are given a single network, and K input sequences, and that we would like to quickly compute all of the K output sequences using a GPU. Our focus is solely on how to efficiently perform this computation utilizing CUDA with the single thread multiple instruction (SIMT) execution model. In particular, we want to understand how the mapping from thread blocks to warps occurs in our computation, and what consequences this has for the overall performance of our algorithm. Note that this section assumes some basic familiarity with GPU programming (CUDA specifically), and we do not attempt to give an introduction to GPU programming. A good introduction to CUDA is [108].

Recall that a dynamical system described by a neural network can be written as follows:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{F}(\mathbf{x}_t, \mathbf{u}_t; \theta)\Delta t \tag{B.1}$$

Where θ is some set of parameters: $\theta = \{W_1, b_1, W_2, b_2, \dots W_M, b_M\}$, and then the function $\mathbf{F}$ is computed via the recursive formula:

$$\mathbf{z}_0 = (\mathbf{x}_t, \mathbf{u}_t) \tag{B.2}$$

$$\mathbf{z}_i = \sigma(W_i \mathbf{z}_{i-1} + b_i) \tag{B.3}$$

$$\mathbf{F} = W_M \mathbf{z}_{M-1} + b_M \tag{B.4}$$

The function σ is some element-wise non-linearity. To simplify the problem, we consider only one particular neural network, which is the neural network used to represent the AutoRally dynamics. In this case the (dynamic) state dimension is 4 (roll, longitudenal velocity, lateral velocity, and heading rate) and the control dimension is 2 (steering and throttle). We consider the case where we want to simulate 100 timesteps, and the number of trajectories varies between 1200 and 10800. The neural network for the AutoRally has two hidden layers with 32 hidden units each, this corresponds to about 1500 parameters.

B.1 Kernel Implementations

There are five different kernel implementations that we examine here, and three different types of mappings from blocks to warps. The only significant difference between the kernels is in how the matrix-vector multiplication is performed. Although initially it may seem that this is a standard matrix-vector multiplication, there is a key difference in that these matrix-vector multiplications are relatively small (the largest matrix we have in the computational graph is 32×32), but there are many, many repetitions of this procedure in both time and space.

Vanilla Implementation

The first kernel we implement is called the "vanilla" kernel. With this kernel, each CUDA thread computes an entire RNN sequence by itself. This means that the parallelism only exists because multiple sequences are being computed. This is the easiest kernel to implement, and also the least efficient. Figure B.1 demonstrates the layout of the matrix-vector multiplication for this kernel. Matrix elements are stored in global memory, and vector elements are stored in shared memory. Note that its not feasible to store all of the matrix parameters in shared memory without running into space limitations since the intermediate activations take up most of the shared memory space.

Baseline Implementation

In the baseline implementation, each thread computes a single dot product in the matrix vector multiplication. In this implementation the x-indices of the threads runs across the

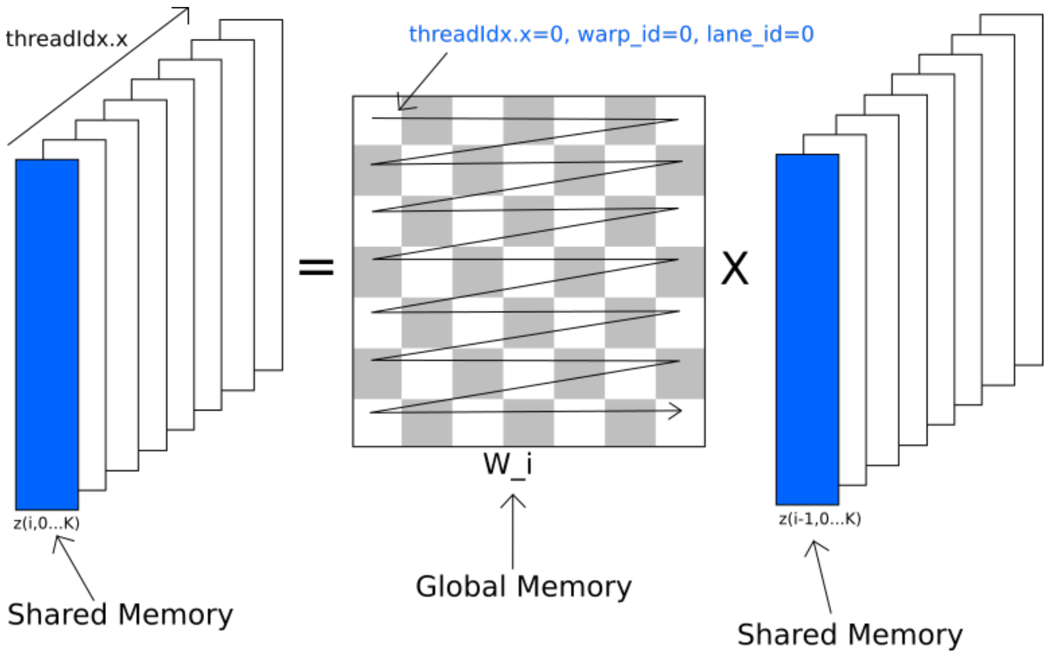


Figure B.1: Diagram of neural network computation for the vanilla kernel. Each thread performs a full matrix-vector multiplication. Shown is the traversal pattern for threadIdx.x=0, which has a warp and lane id of zero, blue denotes the vector elements that are worked on by threadIdx.x=0. Each shared memory block represents the intermediate layer computations of a single sample.

samples, which means that threads next to each other in a warp compute same element, but for different samples. This is demonstrated in Fig. B.2, notice how the lane indices increase along the x-axis. In the actual implementation the block dimension in the x-axis is 16, which means every thread in a half-warp performs the same portion of the matrix-vector multiplication. This enables fast access to global memory, but it also requires that a synchronization across all the threads in a block before the computation can move on to computing the next layer or output in the sequence.

Optimized Baseline Implementation

The optimized baseline implementation is the same as the baseline implementation except for two key differences:

i) The parameters of the neural network are stored in constant memory, as opposed to normal global memory.

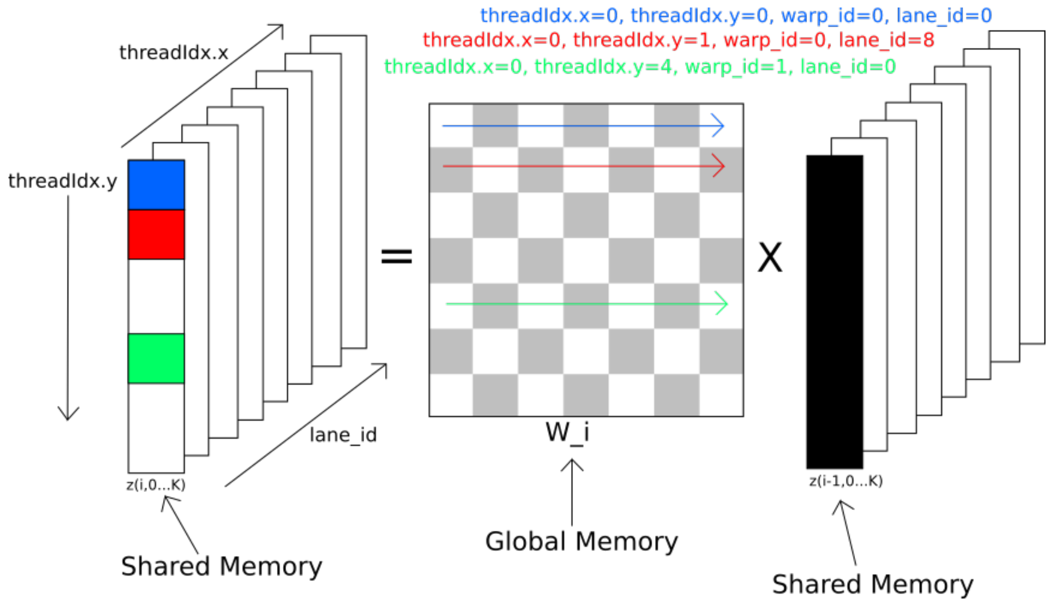

threadIdx.x=0, threadIdx.y=0, warp_id=0, lane_id=0
threadIdx.x=0, threadIdx.y=1, warp_id=0, lane_id=8
threadIdx.x=0, threadIdx.y=4, warp_id=1, lane_id=0
threadIdx.x
threadIdx.y
lane_id
z(i,0...K)
=
W_i
X
z(i-1,0...K)
Shared Memory
Global Memory
Shared Memory

Figure B.2: Diagram of neural network computation for the baseline kernel with a blockDim=(8,8,1). Each thread performs a single dot product in the matrix-vector multiplication. Shown is the traversal pattern for threadIdx.x, threadIdx.y = (0,0), (0,1), and (0,4).

ii) The shared memory array which holds the intermediate computations are padded in order to remove shared memory conflicts.

These two changes significantly improve the performance of the method. Although, they don't fundamentally change the computational structure of the kernel.

Warp-Synchronous Implementation

In the warp-synchronous implementation, the x and y axis in from the baseline implementation are flipped so that each warp is responsible for computing a single sample (as opposed to computing the same element across different samples). This is illustrated in Fig. B.3. Note that the traversal pattern for threads remains the same, however the warp lane ID now runs across the output vector indices, not the sample index. This means that all of the computations for a single sample are contained within a single warp. Since warps are executed in lockstep (and the __warpsync() command can be used to ensure this this), this alleviates the need to perform a synchronization step across all the threads in a block.

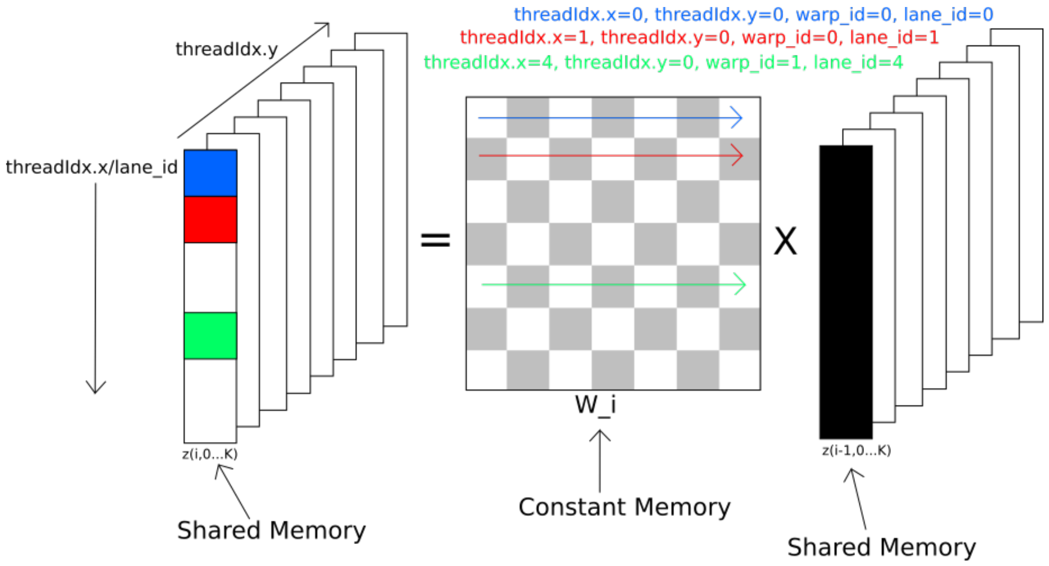

threadIdx.x=0, threadIdx.y=0, warp_id=0, lane_id=0
threadIdx.x=1, threadIdx.y=0, warp_id=0, lane_id=1
threadIdx.x=4, threadIdx.y=0, warp_id=1, lane_id=4
threadIdx.y
threadIdx.x/lane_id
=
X
W_i
z(i,0...K)
z(i-1,0...K)
Shared Memory
Constant Memory
Shared Memory

Figure B.3: Diagram of neural network computation for the warp synchronous kernel with a blockDim=(8,8,1). Warps are organized so that each sample is computed by threads in a single warp.

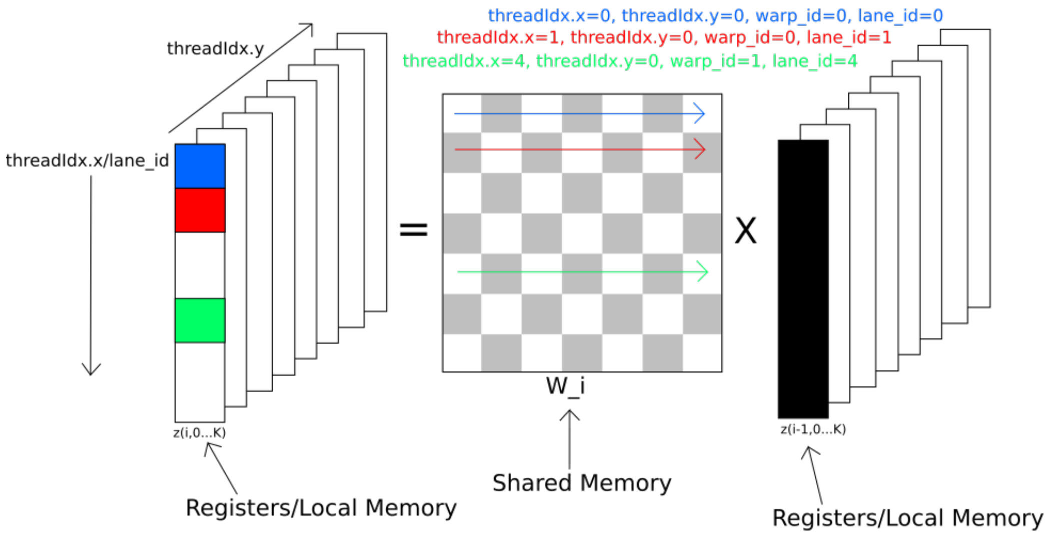

threadIdx.x=0, threadIdx.y=0, warp_id=0, lane_id=0
threadIdx.x=1, threadIdx.y=0, warp_id=0, lane_id=1
threadIdx.x=4, threadIdx.y=0, warp_id=1, lane_id=4
threadIdx.y
threadIdx.x/lane_id
=
X
W_i
z(i,0...K)
z(i-1,0...K)
Registers/Local Memory
Shared Memory
Registers/Local Memory

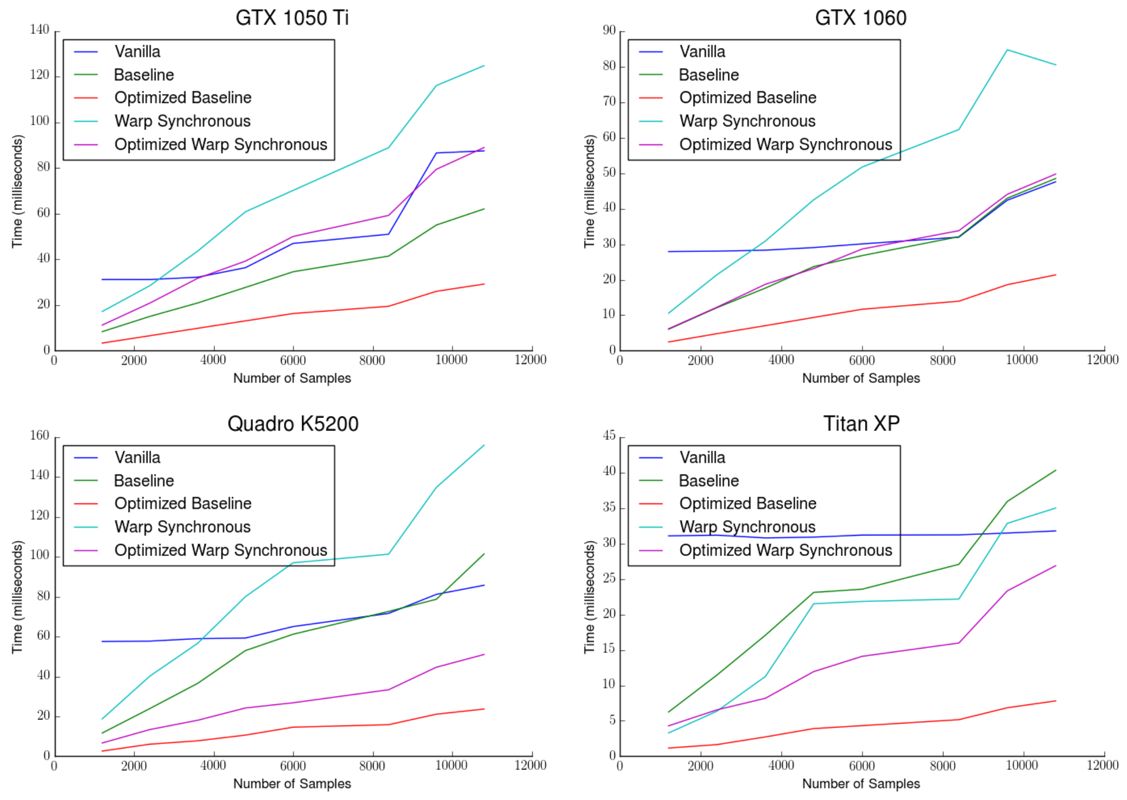
Figure B.4: Optimized warp-synchronous kernel diagram.

Optimized Warp Synchronous Implementation

The fact that all of the computation for a sample is contained in a warp can be further taken advantage of to create an optimized warp synchronous kernel. This kernel is the same as the warp synchronous kernel, but it has the following differences: (1) instead of storing intermediate computations in shared memory, intermediate computations are stored in registers. This is possible because all the threads in a warp can access the registers of the other threads in the warp, (2) storing intermediate computations in registers frees up the shared memory that was being used to store intermediate computations, which enables the parameters of the neural network to be stored in shared memory. The computational diagram with these changes is shown in Fig. B.4.

B.2 Timing Results

Here we report timing results for the five kernels for performing the sampling operation where the number of samples ranges from 1200 to 10800. We report results for four different GPUs: (1) GTX 1050 TI, (2) GTX 1060, (3) Quadro K5200, (4) Titan XP.

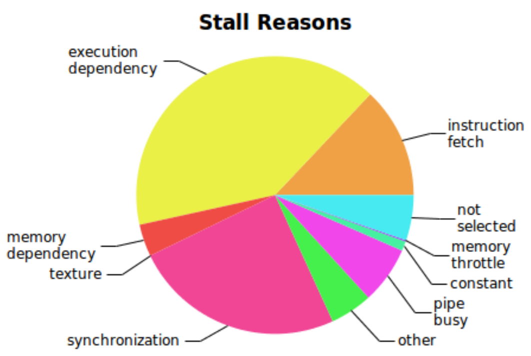


On every GPU, the optimized baseline kernel performs the best. On 2 out of the four cards the optimized warp synchronous kernel performs second best, and is competitive with the second best implementation, which was the normal baseline kernel. The vanilla kernel always performs poorly, although it starts to become more competitive as the number of samples increases towards 10,000. In terms of overall performance, the optimized baseline kernel on the Titan XP performs extremely well, and is able to compute over 10,000 samples (1 million forward propagations through the neural network) in under 10 milliseconds. Out of the three different strategies implemented (vanilla, baseline, and warp synchronous), the timing ordering based on the results can roughly be summarized as:

$$\text{vanilla} > \text{warp synchronous} > \text{baseline}.$$

Using the NVIDIA profiler, it is easy to see why the vanilla implementation performs so poorly: the kernel launches to few threads to achieve full occupancy. The achieved occupancy for the vanilla kernel is only $.06$ for 1200 samples, as the number of samples increases to 10800 this becomes $.44$. In contrast, the baseline kernels always achieve $>$ $.99$ occupancy, and the warp synchronous kernels $.75$. So, although the vanilla kernel has some beneficial properties (easy to implement, good memory access pattern, and no synchronization required), the low occupancy severely limits the performance when fewer than 10,000 samples are computed.

It is more difficult to understand why the performance gap between the optimized baseline kernel and the warp synchronous kernel exists. From an abstract programmatic point of view, there is actually no difference between the optimized baseline and the (non-optimized) warp synchronous kernel, its just that the x and y axis of the computational grid have been flipped, the only real difference is the direction that the warp lane IDs increase (across samples in the baseline case, and across vector elements in the warp synchronous case). They both have a theoretical occupancy of 1, although the achieved occupancy for the warp synchronous kernel is lower. Additionally, the warp synchronous kernel does not have any stalls due to synchronization, this can be confirmed using the Nvidia visual profiler:



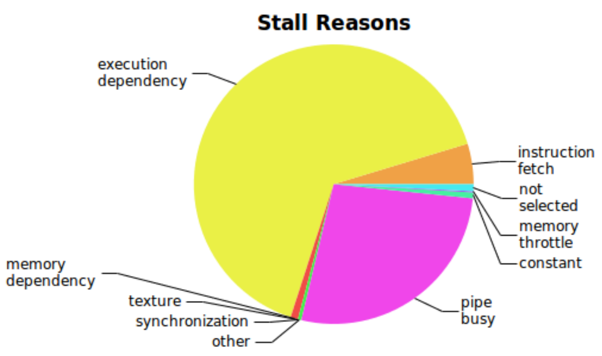


Figure B.5: Output of stall analysis from the NVIDIA visual profiler. Left: Optimized baseline kernel. Right: Warp Synchronous Kernel. Output is from Quadro K5200 with 2400 samples.

Based on this chart, one might conclude that the warp synchronous implementation has done its job. It eliminated the synchronization stalls, and now execution dependencies account for almost all the latency in the kernel. The only problem is that the kernel now runs over 6 times slower. The issue, is the access pattern for the constant memory array containing the neural network parameters. In the case of the baseline kernel, each lane in a half warp accesses the same element of constant memory at the same time. This is an ideal access pattern for constant memory. However, in the warp-synchronous case, threads in a warp access different elements at the same time, which is not ideal. This can be confirmed with the nvidia profiler, the Global Load Throughput (gld_throughput) for the optimized baseline is 11.595 GB/s, whereas the throughput for the warp synchronous kernel is nearly 6 times slower at 2.083 GB/s. The optimized warp synchronous kernel performs slightly better, due to its usage of shared and register memory. However, it is unable to perform as well as the optimized baseline. There are two primary reasons for this (1) When sharing registers across threads in a warp extra computations are required in order to determine which register to read/write to, and (2) The implementation uses too many registers, which means that occupancy must be reduced or local memory must be used (utilizing local memory by setting a maxrregcount was slightly faster).

APPENDIX C
DYNAMICS MODELS

C.1 Cart pole dynamics

The state of the cart-pole is described by the location of the cart, x, the velocity of the cart $\dot{x}$, the angle of the pole with the vertical, θ, and the angular velocity of the pole, $\dot{\theta}$. An additional state, f, is used to model the horizontal force supplied to the cart via a simple motor. The full state equations for the analytic model of the cart-pole dynamics are given below:

$$\ddot{x} = \frac{1}{m_c + m_p \sin^2\theta}\Big(f + m_p \sin\theta\Big(l\dot{\theta}^2 + g\cos\theta\Big)\Big)$$
$$\ddot{\theta} = \frac{1}{l(m_c + m_p \sin^2\theta)}\Big(-f\cos\theta - m_p l\dot{\theta}^2\cos\theta\sin\theta - \Big(m_c + m_p\Big)g\sin\theta\Big)$$
$$\dot{f} = 20\left(f_{des} - f\right)$$

Where f_{des} is the desired motor force, $g = 9.81\frac{\mathrm{m}}{\mathrm{sec}^2}$ is the gravitational acceleration, $m_c = 1.0$ kg is the mass of the cart, $m_p = 0.01$ kg is the mass of the pole, and $l = 0.25$ m is the length of the pole. The cart-pole swing-up task required the algorithm to bring the system from the initial state, stationary with the pole pointed straight down at the origin ($x = \dot{x} = \theta = \dot{\theta} = 0$), to the final state, stationary with the pole pointed straight up at the origin ($x = \dot{x} = \dot{\theta} = 0, \theta = \pi$).

C.2 Race car dynamics

We used an analytic model of vehicle dynamics derived in [66] as the ground truth model for the simulator. This model makes the simplifying assumption that the two front tires and two back tires are lumped into one tire at the front and the back (and is therefore known as a bicycle vehicle model), this assumption makes tractable the incorporation of lateral tire forces into the model which are an essential aspect of car dynamics when operating at high speeds. The full state equations are for the model are given below:

$$\begin{aligned}
&\dot{x} = v_x \cos(\psi) - v_y \sin(\psi), \quad \dot{y} = v_x \sin(\psi) + v_y \cos(\psi), \quad \dot{\psi} = r \\
&\dot{\beta} = \frac{F_{yF} + F_{yR}}{M v_x} - r, \quad \dot{v}_x = \frac{F_x - F_{yF}\sin(\delta)}{M} + r v_x \beta, \quad \dot{v}_y = \frac{F_{yF} + F_{yR}}{M} - r v_x \\
&\dot{r} = \frac{a F_{yF} - b F_{yR}}{I_z}, \quad \dot{\delta} = 10(\delta - \delta^{des}), \quad \dot{F}_x = 10(F_x - F_x^{des})
\end{aligned}$$

Where (x, y) is position, ψ is the heading, β is the side-slip angle, $(v_x, v_y$ are longitudinal and lateral velocity in the body frame of the vehicle, r is the heading (yaw) rate, δ is the steering angle, and F_x is the longitudinal force imparted by the rear wheels. The inputs to the model are desired commands δ^{des} and F_x^{des}. (a, b) are the distances from the center of mass to the front and rear axles. The terms F_{yF} and F_{yR} are the lateral tire forces imparted by the front and rear wheels respectively. This force is a function of the slip angle α which we compute based on a brush tire model.

$$\alpha_F = \tan^{-1}(\beta + a\frac{r}{v_x}) - \delta, \quad \alpha_R = \tan^{-1}(\beta - b\frac{r}{v_x})$$

and then the lateral forces are:

$$F_{yi} = \begin{cases} -C\tan(\alpha_i) + \frac{C^2}{3\xi\mu F_z}\frac{\tan(\alpha_i)^3}{|\tan(\alpha_i)|} - \frac{C^3}{27\mu^2\xi^2F_z^2}\tan(\alpha_i)^3 \\ -\mu\xi F_z\frac{|\alpha_i|}{\alpha_i} \quad \text{if} \quad \alpha_i \geq \gamma_i \end{cases}$$

Here C is the cornering stiffness of the tire, and μ is the co-efficient of friction between the tire and the ground. These values were set to 1200 and $.55$ respectively. The terms γ and ξ are computed as:

$$\xi = \left(\frac{\mu^2 F_z^2 - F_x^2}{\mu F_z}\right)^{1/2}, \quad \gamma = \left|\tan^{-1}\left(3\xi F_z\frac{|\alpha|}{\alpha}\right)\right| \tag{C.1}$$

C.3 Quadrotor dynamics

The dynamic model of the quad-rotor was introduced in [67]. The model includes 16 states: 3 for position ($\mathbf{r}$), 3 for translational velocity ($\dot{\mathbf{r}}$), 3 for Euler angles ($\boldsymbol{\Phi}$), 3 for the body angular rates ($\boldsymbol{\omega}$), and 4 for motor speeds ($\boldsymbol{\Omega}$). The full state equations are given below:

$$\dot{\boldsymbol{\Phi}} = \begin{pmatrix} c\theta & 0 & -c\phi s\theta \\ 0 & 1 & s\phi \\ s\theta & 0 & c\phi c\theta \end{pmatrix}^{-1} \begin{pmatrix} p \\ q \\ r \end{pmatrix}, \quad \ddot{\mathbf{r}} = \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix} + R \begin{pmatrix} 0 \\ 0 \\ \frac{1}{m}\sum F_i \end{pmatrix}$$

$$\dot{\boldsymbol{\omega}} = I^{-1} \left[\begin{pmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{pmatrix} - \begin{pmatrix} p \\ q \\ r \end{pmatrix} \times I \begin{pmatrix} p \\ q \\ r \end{pmatrix} \right], \quad \dot{\boldsymbol{\Omega}} = k_m \left[\begin{pmatrix} \omega_1^{des} \\ \omega_2^{des} \\ \omega_3^{des} \\ \omega_4^{des} \end{pmatrix} - \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{pmatrix} \right]$$

Where ϕ, θ, and ψ are the components of $\boldsymbol{\Phi}$, p, q, and r are the components of $\boldsymbol{\omega}$, and $\omega_1, \omega_2, \omega_3$, and ω_4 are the components of $\boldsymbol{\Omega}$. Additionally, $m = 0.25$ kg is the mass of the quad-rotor, $g = 9.81 \ \frac{\text{m}}{\text{sec}^2}$ is gravitational acceleration, $L = 0.1 \text{ m}$ is the length of the moment arm from the body of the quad-rotor to the motor,

$$I = \text{ diag}(2.5 \times 10^{-4}, 2.5 \times 10^{-4}, 1.2 \times 10^{-3}) \text{ kg m}^2,$$

is the mass moment of inertia matrix for the quad-rotor in the body frame, and $k_m = 20$ is the motor constant. The forces and moments F_i and M_i used above are given by:

$$F_i = k_F {\omega_i}^2$$

$$M_i = k_M {\omega_i}^2$$

Where k_F is the motor force constant equal to $6.11 \times 10^{-8}\ \frac{\text{N}}{\text{rpm}^2}$ and k_M is the motor moment constant equal to $1.5 \times 10^{-9}\ \frac{\text{N m}}{\text{rpm}^2}$. The rotation matrix, R, between the body and inertial frames is given by:

$$R = \begin{pmatrix} c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\psi & s\psi s\theta - c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{pmatrix}$$

C.4 Basis Function Model

We used an analytic model of vehicle dynamics derived in [66] as a source of physics based knowledge about vehicle dynamics. Based on the equations in [66] we picked out the key non-linearities found in the previous model and used them to form 25 basis function. The equations of motion are then:

$$\dot{\mathbf{x}} = \theta^{\mathrm{T}} \Phi(\mathbf{x}),$$

In the following we define:

$$\alpha_f = \arctan\left(\frac{v_y}{v_x} + .45\frac{r}{v_x} - u_\delta\right), \alpha_r = \arctan\left(\frac{v_y}{v_x} - .35\frac{r}{v_x}\right)$$

The basis functions that we choose for the AutoRally model are then:

$$\phi_1 = u_F,\ \phi_2 = v_x/10,\ \phi_3 = \sin(u_\delta)\tan(\alpha_f)/1200$$

$$\phi_4 = \sin(u_\delta)\tan(\alpha_f)\|\tan(\alpha_f)\|/1200^2$$

$$\phi_5 = \sin(u_\delta)\tan(\alpha_f)^3/1200^3$$

$$\phi_6 = rv_y/25,\ \phi_7 = r/10,\ \phi_8 = v_y/10,\ \phi_9 = \sin(u_\delta)$$

$$\phi_{10} = \begin{cases} \frac{v_y}{v_x}/40 & \text{if } v_x > 0.1 \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_{11} = \tan(\alpha_f)/1400,\ \phi_{12} = \tan(\alpha_f)\|\tan(\alpha_f)\|/1400^2$$

$$\phi_{13} = \tan(\alpha_f)^3/1400^3,\ \phi_{14} = \tan(\alpha_r)/40$$

$$\phi_{15} = \tan(\alpha_r)\|\tan(\alpha_r)\|/40^2,\ \phi_{16} = \tan(\alpha_r)^3/40^3$$

$$\phi_{17} = rv_x/50,\ \phi_{18} = \theta$$

$$\phi_{19} = \theta r,\ \phi_{20} = \theta v_x/3,\ \phi_{21} = \theta v_x r/5,\ \phi_{22} = v_x^2/100$$

$$\phi_{23} = v_x^3/1000,\ \phi_{24} = u_F^2,\ \phi_{25} = u_F^3.$$

REFERENCES

[1] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras, and J. M. Rehg, “Autorally: An open platform for aggressive autonomous driving,” *IEEE Control Systems Magazine*, vol. 39, no. 1, pp. 26–55, 2019.

[2] D. J. Fagnant and K. Kockelman, “Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations,” *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[3] S. Thrun, “Toward robotic cars,” *Communications of the ACM*, vol. 53, no. 4, pp. 99–106, 2010.

[4] P. A. Theodosis and J. C. Gerdes, “Generating a racing line for an autonomous racecar using professional driving techniques,” in *Dynamic Systems and Control Conference and Bath/ASME Symposium on Fluid Power and Motion Control*, American Society of Mechanical Engineers, 2011, pp. 853–860.

[5] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, “A survey of motion planning and control techniques for self-driving urban vehicles,” *Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[6] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, “Stanley: The robot that won the darpa grand challenge,” *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[7] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, “Autonomous driving in urban environments: Boss and the urban challenge,” *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[8] A. De Luca, G. Oriolo, and C. Samson, “Feedback control of a nonholonomic car-like robot,” *Robot motion planning and control*, pp. 171–253, 1998.

[9] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, *et al.*, “A perception-driven autonomous urban vehicle,” *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[10] *Baidu apollo*, `http://apollo.auto/`, Accessed: 2019-02-08.

[11] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[12] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo EM motion planner," *CoRR*, vol. abs/1807.08048, 2018. arXiv: `1807.08048`.

[13] W. Xu, J. Wei, J. M. Dolan, H. Zhao, and H. Zha, "A real-time motion planner with trajectory optimization for autonomous vehicles," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2012.

[14] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, "Motion planning for urban driving using rrt," in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2008, pp. 1681–1686.

[15] P. Polack, F. Altch, B. d'Andra Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" In *Intelligent Vehicles Symposium*, IEEE, 2017.

[16] N. R. Kapania, J. Subosits, and J. C. Gerdes, "A sequential two-step algorithm for fast generation of vehicle racing trajectories," *Journal of Dynamic Systems, Measurement, and Control*, vol. 138, no. 9, p. 091 005, 2016.

[17] J. V. Frasch, A. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, and M. Diehl, "An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles," in *Control Conference (ECC), 2013 European*, IEEE, 2013, pp. 4136–4141.

[18] J. Wurts, J. L. Stein, and T. Ersal, "Increasing computational speed of nonlinear model predictive control using analytic gradients of the explicit integration scheme with application to collision imminent steering," in *Conference on Control Technology and Applications (CCTA)*, IEEE, 2018.

[19] D. Q. Mayne, "Differential dynamic programming–a unified approach to the optimization of dynamic systems," in *Control and Dynamic Systems*, vol. 10, Elsevier, 1973, pp. 179–254.

[20] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *ICINCO*, 2004.

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

[22] J. Morimoto, G. Zeglin, and C. G. Atkeson, “Minimax differential dynamic programming: Application to a biped walking robot,” in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, vol. 2, 2003, pp. 1927–1932.

[23] P. Abbeel, V. Ganapathi, and A. Y. Ng, “Learning vehicular dynamics, with application to modeling helicopters,” in *Advances in Neural Information Processing Systems*, 2006, pp. 1–8.

[24] J. Van Den Berg, S. Patil, and R. Alterovitz, “Motion planning under uncertainty using iterative local optimization in belief space,” *International Journal of Robotics Research (IJRR)*, vol. 31, no. 11, pp. 1263–1278, 2012.

[25] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, “An integrated system for real-time model predictive control of humanoid robots,” in *International Conference on Humanoid Robots (Humanoids)*, IEEE, 2013, pp. 292–299.

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

[27] J. Peters and S. Schaal, “Policy gradient methods for robotics,” in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2006, pp. 2219–2225.

[28] H. Benbrahim and J. A. Franklin, “Biped dynamic walking using reinforcement learning,” *Robotics and Autonomous Systems*, vol. 22, no. 3, pp. 283–302, 1997.

[29] J. Kober and J. R. Peters, “Policy search for motor primitives in robotics,” in *Advances in neural information processing systems (NIPS)*, 2009, pp. 849–856.

[30] P. Kormushev, S. Calinon, and D. G. Caldwell, “Robot motor skill coordination with EM-based reinforcement learning,” in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2010, pp. 3232–3237.

[31] J. Peters and S. Schaal, “Reinforcement learning for parameterized motor primitives,” in *International Joint Conference on Neural Networks (IJCNN)*, 2006.

[32] ——, “Reinforcement learning of motor skills with policy gradients,” *Neural Networks*, vol. 21, no. 4, pp. 682–97, 2008.

[33] ——, “Learning to control in operational space,” *International Journal of Robotics Research (IJRR)*, vol. 27, pp. 197–212, 2008.

[34] E. Theodorou, J. Buchli, and S. Schaal, “A generalized path integral control approach to reinforcement learning,” *Journal of Machine Learning Research*, vol. 11, no. Nov, pp. 3137–3181, 2010.

[35] P. Abbeel, A. Coates, and A. Y. Ng, “Autonomous helicopter aerobatics through apprenticeship learning,” *The International Journal of Robotics Research*, 2010.

[36] C. G. Atkeson and J. C. Santamaria, “A comparison of direct and model-based reinforcement learning,” in *International Conference on Robotics and Automation (ICRA)*, IEEE, 1997.

[37] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, “Aggressive driving with model predictive path integral control,” in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 1433–1440.

[38] G. Williams, A. Aldrich, and E. A. Theodorou, “Model predictive path integral control: From theory to parallel computation,” *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

[39] 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)*, IEEE, 2017, pp. 1714–1721.

[40] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, “Information-theoretic model predictive control: Theory and applications to autonomous driving,” *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1603–1622, 2018.

[41] G. Williams, B. Goldfain, P. Drews, K. Saigol, J Rehg, and E. A. Theodorou, “Robust sampling based model predictive control with sparse objective information,” in *Robotics Science and Systems (RSS)*, 2018.

[42] G. Williams, B. Goldfain, P. Drews, J. M. Rehg, and E. A. Theodorou, “Best response model predictive control for agile interactions between autonomous ground vehicles,” in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2018.

[43] G. Williams, B. Goldfain, J Rehg, and E. A. Theodorou, “Locally weighted regression pseudo-rehearsal for online learning of vehicle dynamics,” in *Robotics Science and Systems (RSS)*, Currently Under Review, 2019.

[44] S. E. Shreve, *Stochastic calculus for finance 2, Continuous-time models*. New York, NY; Heidelberg: Springer, 2004, ISBN: 0387401016 9780387401010.

[45] T. Sauer, “Numerical solution of stochastic differential equations in finance,” in *Handbook of computational finance*, Springer, 2012, pp. 529–550.

[46] R. Stengel, *Optimal Control and Estimation*. Dover, 1994, ISBN: 0-486-68200-5.

[47] H. Royden, *Real Analysis*, 3rd ed. Upper Saddle River, NJ 07458: Prentice Hall, Inc, 1968, ISBN: 0-02-404151-3.

[48] I. V. Girsanov, “On transforming a certain class of stochastic processes by absolutely continuous substitution of measures,” *Theory of Probability & Its Applications*, vol. 5, no. 3, pp. 285–301, 1960.

[49] H. J. Kappen, “Path integrals and symmetry breaking for optimal control theory,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 11, P11011, 2005.

[50] ——, “Linear theory for control of nonlinear stochastic systems,” *Phys Rev Lett*, vol. 95, p. 200 201, 2005, Journal Article United States.

[51] M. Kac, “On distributions of certain wiener functionals,” *Transactions of the American Mathematical Society*, vol. 65, no. 1, pp. 1–13, 1949.

[52] E. A. Theodorou and E. Todorov, “Relative entropy and free energy dualities: Connections to path integral and kl control,” in *Conference on Decision and Control (CDC)*, IEEE, 2012, pp. 1466–1473.

[53] E. A. Theodorou, “Nonlinear stochastic control and information theoretic dualities: Connections, interdependencies and thermodynamic interpretations,” *Entropy*, vol. 17, no. 5, pp. 3352–3375, 2015.

[54] H. J. Kappen and H. C. Ruiz, “Adaptive importance sampling for control and inference,” *Journal of Statistical Physics*, vol. 162, no. 5, pp. 1244–1266, 2016.

[55] E. Todorov and M. I. Jordan, “A minimal intervention principle for coordinated movement,” in *Advances in neural information processing systems (NIPS)*, 2003, pp. 27–34.

[56] P. Dayan and G. E. Hinton, “Using expectation-maximization for reinforcement learning,” *Neural Computation*, vol. 9, no. 2, pp. 271–278, 1997.

[57] J. Peters and S. Schaal, “Reinforcement learning by reward-weighted regression for operational space control,” in *International Conference on Machine learning (ICML)*, 2007, pp. 745–750.

[58] Z. B. Zabinsky, *Stochastic adaptive search for global optimization*. Springer Science & Business Media, 2013.

[59] K. Rawlik, M. Toussaint, and S. Vijayakumar, “On stochastic optimal control and reinforcement learning by approximate inference,” *Robotics Science and Systems (RSS)*, p. 353, 2013.

[60] M. Kobilarov, “Cross-entropy motion planning,” *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.

[61] A. Weinstein and M. L. Littman, “Open-loop planning in large-scale stochastic domains.,” in *Conference on Artificial Intelligence*, AAAI, 2013.

[62] M. Kobilarov and S. Pellegrino, “Trajectory planning for cubesat short-time-scale proximity operations,” *Journal of Guidance, Control, and Dynamics*, vol. 37, no. 2, pp. 566–579, 2014.

[63] A. Doucet, N. De Freitas, and N. Gordon, “An introduction to sequential monte carlo methods,” in *Sequential Monte Carlo methods in practice*, Springer, 2001, pp. 3–14.

[64] F. Stulp and O. Sigaud, “Path integral policy improvement with covariance matrix adaptation,” in *International Conference on Machine Learning (ICML)*, IMLS, 2012, pp. 1547–1554.

[65] A. Savitzky and M. J. Golay, “Smoothing and differentiation of data by simplified least squares procedures.,” *Analytical chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[66] R. Hindiyeh, “Dynamics and control of drifting in automobiles,” PhD thesis, Stanford University, 2013.

[67] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, “The grasp multiple micro-uav testbed,”

[68] K. Lundahl, J. Åslund, and L. Nielsen, “Investigating vehicle model detail for close to limit maneuvers aiming at optimal control,” in *International Symposium of Dynamics on Vehicle on Roads and Tracks*, 2011.

[69] M. S. Burhaumudin, P. M. Samin, H. Jamaluddin, R. A. Rahman, S. Sulaiman, *et al.*, “Integration of magic formula tire model with vehicle handling model,” *International Journal of Research in Engineering and Technology*, vol. 1, no. 3, pp. 139–145, 2012.

[70] A. Venkatraman, M. Hebert, and J. A. Bagnell, “Improving multi-step prediction of learned time series models.,” in *Conference on Artificial Intelligence*, AAAI, 2015, pp. 3024–3030.

[71] G. Hinton, N. Srivastava, and K. Swersky, “Lecture 6a overview of mini–batch gradient descent.”

[72] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, “Agile off-road autonomous driving using end-to-end deep imitation learning,” *Robotics Science and Systems (RSS)*, 2017.

[73] D. Q. Mayne, M. M. Seron, and S. Raković, “Robust model predictive control of constrained linear systems with bounded disturbances,” *Automatica*, vol. 41, no. 2, pp. 219–224, 2005.

[74] D. Q. Mayne and E. C. Kerrigan, “Tube-based robust nonlinear model predictive control1,” *IFAC Proceedings Volumes*, vol. 40, no. 12, pp. 36–41, 2007.

[75] D. Q. Mayne, E. C. Kerrigan, and P. Falugi, “Robust model predictive control: Advantages and disadvantages of tube-based methods,” *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 191–196, 2011.

[76] S. V. Rakovic, B. Kouvaritakis, M. Cannon, C. Panos, and R. Findeisen, “Parameterized tube model predictive control,” *Transactions on Automatic Control*, vol. 57, no. 11, pp. 2746–2761, 2012.

[77] V. Desaraju, A. Spitzer, and N. Michael, “Experience-driven predictive control with robust constraint satisfaction under time-varying state uncertainty,” in *Robotics Science and Systems (RSS)*, 2017.

[78] M. McCloskey and N. J. Cohen, “Catastrophic interference in connectionist networks: The sequential learning problem,” in *Psychology of learning and motivation*, vol. 24, Elsevier, 1989, pp. 109–165.

[79] R. Ratcliff, “Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions.,” *Psychological review*, vol. 97, no. 2, p. 285, 1990.

[80] C. G. Atkeson, A. W. Moore, and S. Schaal, “Locally weighted learning for control,” in *Lazy learning*, Springer, 1997, pp. 75–113.

[81] S. Schaal and C. G. Atkeson, “Constructive incremental learning from only local information,” *Neural computation*, vol. 10, no. 8, pp. 2047–2084, 1998.

[82] S. Vijayakumar, A. D’souza, and S. Schaal, “Incremental online learning in high dimensions,” *Neural computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

[83] D. Mitrovic, S. Klanke, and S. Vijayakumar, “Adaptive optimal control for redundantly actuated arms,” in *International Conference on Simulation of Adaptive Behavior*, Springer, 2008, pp. 93–102.

[84] G. Williams, E. Rombokas, and T. Daniel, “Gpu based path integral control with learned dynamics,” in *Neural Informations Processing Systems: Autonomously Learning Robots Workshop*, 2014.

[85] A. Robins, “Catastrophic forgetting in neural networks: The role of rehearsal mechanisms,” in *First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, IEEE, 1993, pp. 65–68.

[86] ——, “Catastrophic forgetting, rehearsal and pseudorehearsal,” *Journal of Neural Computing, Artificial Intelligence and Cognitive Research*, vol. 7, pp. 123–146, 1995.

[87] ——, “Sequential learning in neural networks: A review and a discussion of pseudorehearsal based methods,” *Intelligent Data Analysis*, vol. 8, no. 3, pp. 301–322, 2004.

[88] R. M. French, “Using pseudo-recurrent connectionist networks to solve the problem of sequential learning,” in *Proceedings of the 19th Annual Cognitive Science Society Conference*, vol. 16, 1997.

[89] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, “Icarl: Incremental classifier and representation learning,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2017, pp. 5533–5542.

[90] H. Shin, J. K. Lee, J. Kim, and J. Kim, “Continual learning with deep generative replay,” in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 2990–2999.

[91] C. Atkinson, B. McCane, L. Szymanski, and A. V. Robins, “Pseudo-rehearsal: Achieving deep reinforcement learning without catastrophic forgetting,” *CoRR*, vol. abs/1812.02464, 2018. arXiv: `1812.02464`.

[92] D. Mellado, C. Saavedra, S. Chabert, and R. Salas, “Pseudorehearsal approach for incremental learning of deep convolutional neural networks,” in *Latin American Workshop on Computational Neuroscience*, Springer, 2017, pp. 118–126.

[93] R. Kemker and C. Kanan, “Fearnet: Brain-inspired model for incremental learning,” *CoRR*, vol. abs/1711.10563, 2017. arXiv: `1711.10563`.

[94] S. Wan and L. E. Banta, “Parameter incremental learning algorithm for neural networks,” *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1424–1438, 2006.

[95] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, “Overcoming catas-

trophic forgetting in neural networks," *Proceedings of the national academy of sciences*, p. 201 611 835, 2017.

[96] Z. Li and D. Hoiem, "Learning without forgetting," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 12, pp. 2935–2947, 2018.

[97] I. Clavera, A. Nagabandi, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt: Meta-learning for model-based control," *CoRR*, vol. abs/1803.11347, 2018. arXiv: `1803.11347`.

[98] S. Klanke, S. Vijayakumar, and S. Schaal, "A library for locally weighted projection regression," *Journal of Machine Learning Research*, vol. 9, pp. 623–626, 2008.

[99] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. arXiv: `1412.6980`.

[100] R. Miller and Q. Huang, "An adaptive peer-to-peer collision warning system," in *Vehicular Technology Conference*, 2002.

[101] S. Ammoun and F. Nashashibi, "Real time trajectory prediction for collision risk estimation between vehicles," in *International Conference on Intelligent Computer Communication and Processing (ICCP)*, IEEE, 2009, pp. 417–422.

[102] M. Liebner, M. Baumann, F. Klanner, and C. Stiller, "Driver intent inference at urban intersections using the intelligent driver model," in *Intelligent Vehicles Symposium*, 2012.

[103] C. Hermes, C. Wohler, K. Schenk, and F. Kummert, "Long-term vehicle motion prediction," in *Intelligent Vehicles Symposium*, IEEE, 2009, pp. 652–657.

[104] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *Robomech Journal*, vol. 1, no. 1, p. 1, 2014.

[105] D. Fudenberg and D. K. Levine, *The theory of learning in games*. MIT Press, 1998, vol. 2.

[106] D. Sadigh, S. Sastry, S. Seshia, and A. Dragan, "Planning for autonomous cars that leverage effects on human actions," in *Robotics Science and Systems (RSS)*, 2016.

[107] R. Spica, D. Falanga, E. Cristofalo, E. Montijano, D. Scaramuzza, and M. Schwager, "A real-time game theoretic planner for autonomous two-player drone racing," in *Robotics Science and Systems (RSS)*, 2018.

[108] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

VITA

Grady Williams grew up in Seattle, Washington. He graduated from Ballard High School in 2009, and received a B.S in mathematics from the University of Washington in 2014. He did his PhD work in Robotics at the Georgia Institute of Technology in the Autonomous Control and Decisions Systems Laboratory under the guidance of Prof. Evangelos Theodorou. He received the Qualcomm Innovation Fellowship in October of 2017, and was also a finalist for the best paper award at ICRA 2017.