Learning Safe Predictive Control with Gaussian Processes

Benjamin van Niekerk
School of Computer Science and Applied Mathematics
University of the Witwatersrand
benjamin.l.van.niekerk@gmail.com

August 12, 2019

Contents

Abstract

Learning-based methods have recently become popular in control engineering, achieving good performance on a number of challenging tasks. However, in complex environments where data efficiency and safety are critical, current methods remain unsatisfactory. As a step toward addressing these shortcomings, we propose a learning-based approach that combines Gaussian process regression with model predictive control. Using sparse spectrum Gaussian processes, we extend previous work by learning a model of the dynamics incrementally from a stream of sensory data. Utilizing learned dynamics and model uncertainty, we develop a controller that can learn and plan in real-time under non-linear constraints. We test our approach on pendulum and cartpole swing up problems and demonstrate the benefits of learning on a challenging autonomous racing task. Additionally, we show that learned dynamics models can be transferred to new tasks without any additional training.

Declaration

I, Benjamin van Niekerk, hereby declare the contents of this dissertation to be my own work unless otherwise explicitly referenced. This dissertation is submitted for the degree of Master of Science (Dissertation) at the University of the Witwatersrand, Johannesburg. This work has not been submitted to any other university, nor for any other degree.

Signature	Date
	08/12/2019

Chapter 1

Introduction

Since its introduction in the early eighties (Cutler and Ramaker, 1980), model predictive control (MPC) has become a standard tool in control engineering. Today MPC is widely used in industry with applications in power electronics (Vazquez et al., 2014), aeronautics (Di Cairano et al., 2012) and robotics (Erez et al., 2013). MPC is a class of feedback control methods that use a model to predict the future behavior of a process. By taking the predicted dynamics into account, MPC can plan a sequence of control inputs that drives the process to a desired state. For example, consider an autonomous car racing along a track. Using predicted responses to acceleration and steering, MPC determines how best to control the car while optimizing for speed and avoiding collisions.

The success of MPC is reliant on accurate predictions and depends on a precise description of the process dynamics. As a result, considerable engineering effort is spent on developing models of complex systems. However, modeling from first principles can be difficult. Non-linear dynamics are often ignored and time consuming experiments are required to identify and characterize effects like friction and tyre slip (Voser et al., 2010).

Data-driven methods try to address this issue by learning a controller or a process model directly from data. Learning-based approaches sidestep the need for known dynamics but come with their own set of challenges. Firstly, data-efficiency is essential. Running experiments on physical hardware can be onerous and costly, highlighting the need for methods that learn quickly from limited data. Secondly, to avoid damage and potentially dangerous behavior, safety constraints must be accounted for. Learning-based methods need to handle obstacles, boundaries, and actuator limits consistently and reliably. Finally, real-time feasibility is a key requirement. For processes with fast dynamics, learning-based control must operate on time scales of a few milliseconds.

To tackle these challenges we propose GP-RHC — Gaussian Processes for Receding Horizon Control. GP-RHC combines Gaussian process regression for data-efficient model learning with real-time model predictive control. This work includes three main contributions and was first presented in our paper Van Niekerk et al. (2017).

i) For many applications, we can approximately model complex dynamics by mak-

ing some simplifying assumptions. GP-RHC can take advantage of an approximate nominal model by learning to correct for unmodeled disturbances. By exploiting prior knowledge GP-RHC significantly improves controller performance in settings where learning from scratch is difficult or impossible.

ii) Using sparse spectrum Gaussian processes (Lázaro-Gredilla et al., 2010), GP-RHC extends previous work by incrementally updating the dynamics model from a stream of sensory data. Experiments demonstrate that incorporating online data results in more efficient and robust learning.

iii) GP-RHC integrates Gaussian process models with structure exploiting interior point methods for control. At each time step the control problem is formulated as a large convex program. By leveraging structure in the convex program and efficient interior point optimization, GP-RHC is able to plan in real-time while safely handling constraints.

The remainder of this thesis is structured as follows. In chapter 2 we explore Gaussian processes and discuss their application to model learning. We review sparse approximations for scaling up GPs and demonstrate how to incorporate new data through incremental updates. Chapter 3 introduces constrained convex optimization. Following Domahidi et al. (2012) we develop a structure exploiting interior point method for real-time model predictive control. In chapter 4 we introduce autonomous racing (Verschueren et al., 2014; Liniger et al., 2015; Rosolia et al., 2017) as a test-bed for GP-RHC. Autonomous racing is a challenging test-bed for learning-based control, highlighting challenges of data-efficiency, safety, and real-time feasibility. Finally, in chapter 5 we evaluate GP-RHC on three different tasks: pendulum swing up, cartpole swing up, and autonomous racing. Our experiments demonstrate that: a) models can be learned quickly from limited data, b) complex non-linear constraints can be handled safely in real-time, and c) online updates improve learning and result in more consistent performance.

1.1 Related Work

Due to their data-efficiency and ability to estimate model uncertainty, Gaussian processes have become increasingly popular in learning-based control. Early work by Kocijan et al. (2004) demonstrated that GPs can be successfully combined with model predictive control on a small trajectory tracking problem.

More recently, trajectory optimization based on differential dynamic programming has been applied to learned dynamics. Methods such as PDDP (Pan and Theodorou, 2014), AGP-iLQR (Boedecker et al., 2014), and MPC via probabilistic trajectory optimization (Pan et al., 2017) combine models learned using sparse Gaussian processes with an iterative linear quadratic regulator for planning and control. These methods can take simple box constraints into account but cannot handle general non-linear constraints.

Building on these ideas Kamthe and Deisenroth (2017) propose GP-MPC which uses Pontryagin's maximum principle and sequential quadratic programming for control. GP-

MPC demonstrates impressive data-efficiency and reliably handles constraints. However, experiments are limited to cartpole and pendulum swing up problems with simple box constraints.

Our work is most closely related to RL-RCO (Andersson et al., 2015) which leverages sparse Gaussian processes for learning the dynamics and trajectory optimization based sequential quadratic programming. We improve upon RL-RCO by proposing an approach which allows the dynamics model to be updated online as the controller interacts with the environment. Furthermore, in contrast to the work of Andersson et al. (2015), we present results that highlight the ability to handle non-linear constraints. We show how these enhancements improve data efficiency, learning rate and constraint handling.

Finally, as an alternative to model predictive control, PILCO (Deisenroth and Rasmussen, 2011) combines policy search with a Gaussian process model of the dynamics. The policy search relies on analytic gradients of the long term expected cost and requires the cost function and policy to take specific functional forms. This makes it difficult or impossible to incorporate general constraints.

Chapter 2

Gaussian Processes

In this chapter we consider the problem of learning a model of the system dynamics. In general, we have a training set of observations and want to learn the relationship between the inputs and outputs. Using this relationship, we can make predictions for a new input that was not part of the initial data. For example, given inputs of steering angle and throttle of a car we'd like to predict its position and velocity at the next time step. To tackle this problem we make use of **Gaussian processes** or **GPs**, introduced in section 2.1. In sections 2.2 and 2.3 we discuss Gaussian process regression and outline a common method for learning hyperparameters from data. Finally, section 2.4 describes the sparse spectrum approximation - a practical approach to reducing the computational costs of GPs.

2.1 Defining a Gaussian Process

Gaussian processes define a prior distribution over functions which potentially describe some dataset. To fit a GP to the data, this prior is updated using Bayesian inference. Formally, a Gaussian process is a collection of random variables, any finite subset of which have a joint Gaussian distribution (Rasmussen and Williams, 2006). In other words, a collection of random variables $\{f(\mathbf{x}) : \mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^m\}$ is said to be drawn from a Gaussian process with **mean function** $m(\cdot)$ and **kernel** $k(\cdot, \cdot)$ if for any finite set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$, the associated set of outputs or target values $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)$ are jointly normally distributed:

$$\begin{bmatrix} f(\mathbf{x}_1) \\ \vdots \\ f(\mathbf{x}_n) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(\mathbf{x}_1) \\ \vdots \\ m(\mathbf{x}_n) \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \right). \tag{2.1}$$

It is clear from this definition that $k(\cdot, \cdot)$ must be a positive semidefinite function in order to specify a valid distribution for any choice of $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

The kernel captures a measure of similarity between outputs in a Gaussian process.

Under assumptions of smoothness and stationarity, we expect nearby inputs from $\mathcal{X}$ to produce similar target values. This allows us to make predictions about the behavior of f at a point $\mathbf{x}$ based on neighboring input-target pairs in the training data.

In this work we use the popular **squared exponential** (SE) kernel defined by

$$k(\mathbf{x}, \mathbf{x}') := \sigma_s^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\intercal \Lambda^{-1}(\mathbf{x} - \mathbf{x}')\right), \tag{2.2}$$

where $\Lambda := \text{diag}(l_1^2, \ldots, l_m^2)$ is a positive diagonal matrix and σ_s is the signal amplitude. Functions drawn from a Gaussian process with a SE kernel will tend to be locally smooth since correlation drops off as a function of distance in input space. The parameters l_i determine how quickly the correlation drops off and define the characteristic length-scales of the GP. Figure 2.1 demonstrates the effect of each hyperparameter on a GP fit to six data points sampled from the sine function.

2.2 Regression with Gaussian Processes

Suppose we have a training set of n measurements $\{(\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$, where each $\mathbf{x}_i \in \mathbb{R}^m$ is an input vector, $y_i = f(\mathbf{x}_i) + \varepsilon$ denotes a noisy output or target, and ε is Gaussian noise with zero mean and variance σ_n^2. The aim of this section is to describe how to predict the value of f at a test point $\mathbf{x}_*$.

Assuming a zero mean GP prior[1] on f, the joint distribution of target data and test output is:

$$\begin{bmatrix} \mathbf{y} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X, X) + \sigma_n^2 I & K(X, \mathbf{x}_*) \\ K(\mathbf{x}_*, X) & K(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix}\right), \tag{2.3}$$

where $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ is a stacked matrix of inputs, $\mathbf{y} = [y_1, \ldots, y_n]^\intercal$ is the corresponding vector of targets, and $K(\cdot, \cdot)$ denotes the covariance matrix evaluated at each pair of inputs i.e. the entries of the covariance matrix are defined by $K(X, X)_{i,j} := k(\mathbf{x}_i, \mathbf{x}_j)$. Then, conditioning on the training data results in a normal distribution

$$f_* | \mathbf{y}, X \sim \mathcal{N}(\mu(\mathbf{x}_*), \Sigma(\mathbf{x}_*)), \tag{2.4}$$

with mean and covariance given by:

$$\mu(\mathbf{x}_*) := K(\mathbf{x}_*, X) Q^{-1} \mathbf{y}, \tag{2.5a}$$

$$\Sigma(\mathbf{x}_*) := K(\mathbf{x}_*, \mathbf{x}_*) - K(\mathbf{x}_*, X) Q^{-1} K(X, \mathbf{x}_*), \tag{2.5b}$$

where $Q := K(X, X) + \sigma_n^2 I$. By sampling from this distribution we can predict the value of f at the test point $\mathbf{x}_*$.

[1] Gaussian process priors with non-zero mean functions are not used in this work so for simplicity we only consider the zero mean case.

A practical implementation of GP regression is shown in algorithm 1. To invert the $n \times n$ matrix Q we first find its Cholesky decomposition and then use forward and backward substitution to find the mean and variance in (2.5). Note that evaluating the mean and variance are $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ operations respectively. This makes the computational costs prohibitively expensive for large data sets or applications with fast real-time constraints. Fortunately, there are a number of approximation schemes that significantly reduce the computational costs (see section 2.4 for details).

Algorithm 1: Gaussian process regression

Data: Inputs X, targets $\mathbf{y}$, covariance function K, noise variance σ_n, test input $\mathbf{x}_*$

1 Compute the matrix $Q := K(X, X) + \sigma_n^2 I$
2 Compute the Cholesky decomposition $Q = LL^\intercal$
3 Find $\boldsymbol{\alpha} := L^\intercal \backslash L \backslash \mathbf{y}$ using backward and forward substitution
4 Compute the predictive mean (2.5a): $\mu(\mathbf{x}_*) = K(\mathbf{x}_*, X)\boldsymbol{\alpha}$
5 Compute the predictive variance (2.5b): $\Sigma(\mathbf{x}_*) = K(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\intercal \mathbf{v}$, with $\mathbf{v} := L \backslash K(\mathbf{x}_*, X)$
6 **return** mean $\mu(\mathbf{x}_*)$, variance $\Sigma(\mathbf{x}_*)$

2.3 Learning the Hyperparameters

A kernel will typically have some free hyperparameters $\boldsymbol{\theta}$ that must be tuned to find a good fit of the data. For example, the parameters $\boldsymbol{\theta} = \{\sigma_s, l_1, \ldots, l_m\}$ in (2.2) can be varied to alter the signal amplitude and characteristic length-scales of the squared exponential kernel. In this section we describe a common method for determining the hyperparameters from the data.

To find the hyperparameters we maximize the **marginal likelihood** — the likelihood of the training data given the hyperparameters:

$$p(\mathbf{y}|X, \boldsymbol{\theta}) = \int p(\mathbf{y}|\mathbf{f}, X, \boldsymbol{\theta}) p(\mathbf{f}|X, \boldsymbol{\theta}) d\mathbf{f}. \tag{2.6}$$

The likelihood $\mathbf{y}|\mathbf{f}, X, \boldsymbol{\theta} \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I)$ is normally distributed, and under a Gaussian process prior we have $\mathbf{f}|X, \boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, K(X, X))$. In this case the marginal likelihood is an integral over the product of two Gaussians and can be computed analytically giving:

$$\log p(\mathbf{y}|X, \boldsymbol{\theta}) = -\frac{1}{2} \log |Q| - \frac{1}{2} \mathbf{y}^\intercal Q^{-1} \mathbf{y} - \frac{n}{2} \log(2\pi). \tag{2.7}$$

The log marginal likelihood in (2.7) can then be minimized using gradient based optimizers (Rasmussen and Williams, 2006). Since Q needs to be inverted each time the log marginal likelihood is evaluated — an $\mathcal{O}(N^3)$ operation in general — hyperparameter learning represents the main computational bottleneck for GP regression.

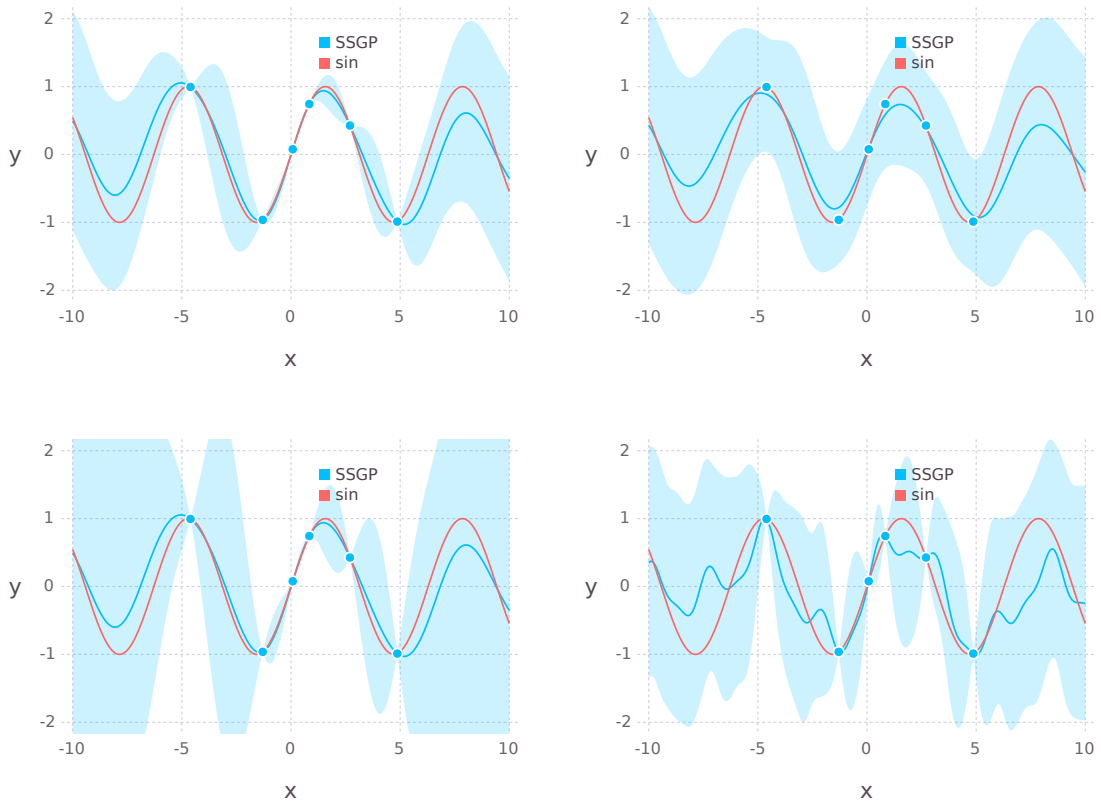

Figure 2.1: Gaussian process hyperparameters. The noise variance σ_n, signal amplitude σ_s, and length-scale l are independently varied while the remaining parameters are kept fixed. **Top-left**: Hyperparemeters are determined by minimizing the negative log likelihood ($\sigma_n^2 = 0.001$, $\sigma_s^2 = 0.8$, $l = 1.58$) for the sparse spectrum Gaussian process (SSGP) discussed in section 2.4. **Top-right**: Noise variance set to $\sigma_n^2 = 0.1$. The predictive mean no longer interpolates the data points exactly and has a larger envelope of uncertainty. **Bottom-left**: Signal amplitude set to $\sigma_s^2 = 2$. The predictive variance increases away from the data points. **Bottom-right**: Length-scale set to $l = 0.6$. A smaller length scale results in a rapidly varying signal.

2.4 Sparse Approximations

Due to their computational costs, GPs do not scale well to large datasets. There are, however, a number of approximation schemes designed to scale GPs (Quiñonero-Candela and Rasmussen, 2005). In this section, we discuss sparse spectrum approximations of the kernel (Lázaro-Gredilla et al., 2010). Sparse spectrum GPs were chosen for this work because: (a) online data can be incorporated through incremental updates (Gijsberts and Metta, 2013), and (b) the learned model can be efficiently linearized for model predictive control (see section 4.8).

2.4.1 Sparse Spectrum Approximation

The idea behind sparse spectrum GPs is to use random Fourier features to approximate the kernel (Rahimi and Recht, 2008). By mapping the input data into the low-dimensional space defined by the features we can significantly reduce computational costs.

In this section, we assume that the kernel is stationary, i.e. that $k(\mathbf{x}, \mathbf{x}')$ is a function of $\mathbf{r} = \mathbf{x} - \mathbf{x}'$ only. In this case, Bochner's theorem (Lázaro-Gredilla et al., 2010) states that $k(\mathbf{r})$ can be represented as the Fourier transform,

$$k(\mathbf{r}) = \int_{\mathbb{R}^m} e^{i\boldsymbol{\omega}^\intercal \mathbf{r}} d\mu(\boldsymbol{\omega}), \tag{2.8}$$

of a positive finite measure μ. With proper scaling, this means that equation (2.8) can be rewritten as an expectation

$$k(\mathbf{r}) = \alpha^2 \mathbb{E}_p[e^{i\boldsymbol{\omega}^\intercal \mathbf{r}}], \tag{2.9}$$

where p is a probability measure over $\mathbb{R}^m$ and α is a constant of proportionality.

To approximate the expectation, we use Monte Carlo integration drawing D sample frequencies $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D$ from p. In order to guarantee that $k(\mathbf{r})$ is real valued for all $\mathbf{r}$, we also include $\boldsymbol{\omega}_{-j} = -\boldsymbol{\omega}_j$ for each sample frequency. This results in the sparse spectrum approximation (Lázaro-Gredilla et al., 2010):

$$k(\mathbf{x}, \mathbf{x}') \approx \frac{\alpha^2}{2D} \sum_{j=-D}^{D} e^{i\boldsymbol{\omega}_j^\intercal (\mathbf{x} - \mathbf{x}')}. \tag{2.10}$$

In the particular case of the squared exponential kernel $\alpha^2 = \sigma_s^2$, and the frequencies are drawn from the normal distribution $\mathcal{N}(\mathbf{0}, \Lambda^{-1})$.

Since the exponential in (2.10) can be represented in terms of sinusoidal functions, we define the feature mapping $\boldsymbol{\phi} : \mathbb{R}^m \to \mathbb{R}^{2D}$ by,

$$\boldsymbol{\phi}(\mathbf{x}) := \frac{\alpha}{\sqrt{D}} [\cos(\boldsymbol{\omega}_1^\intercal \mathbf{x}), \sin(\boldsymbol{\omega}_1^\intercal \mathbf{x}), \ldots, \cos(\boldsymbol{\omega}_D^\intercal \mathbf{x}), \sin(\boldsymbol{\omega}_D^\intercal \mathbf{x})]^\intercal. \tag{2.11}$$

In order to make use of the sparse spectrum approximation, the matrix inversion lemma is applied to equations (2.5a) and (2.5b), giving

$$\mu(\mathbf{x}_*) = \boldsymbol{\phi}(\mathbf{x}_*)^\intercal A^{-1} \boldsymbol{\phi}(X)^\intercal \mathbf{y}, \tag{2.12a}$$

$$\Sigma(\mathbf{x}_*) = \sigma_n^2 + \sigma_n^2 \boldsymbol{\phi}(\mathbf{x}_*)^\intercal A^{-1} \boldsymbol{\phi}(\mathbf{x}_*), \tag{2.12b}$$

where $A = \boldsymbol{\phi}(X)^\intercal \boldsymbol{\phi}(X) + \sigma_n^2 I$, and $\boldsymbol{\phi}(X)$ is the matrix obtained by applying $\boldsymbol{\phi}$ to each column of X. Instead of inverting the $N \times N$ matrix Q, we now only require the inverse of the $2D \times 2D$ matrix A. This constitutes a significant saving in computation if $D \ll N$. Importantly, the size of A is independent of the number of training points, which makes it amenable to incremental updates (Gijsberts and Metta, 2013).

Applying the same idea to the log marginal likelihood (2.7) gives the expression

$$\begin{aligned}\log \mathbb{P}(\mathbf{y}|X,\boldsymbol{\theta}) = &-\frac{1}{2}\log|A| + \frac{D}{2}\log\sigma_n^2 - \frac{n}{2}\log(2\pi\sigma_n^2) \\ &-\frac{1}{2\sigma_n^2}\left(\mathbf{y}^\intercal\mathbf{y} - \mathbf{y}^\intercal\boldsymbol{\phi}(X)A^{-1}\boldsymbol{\phi}(X)^\intercal\mathbf{y}\right).\end{aligned} \tag{2.13}$$

Again, the smaller size of A results in reduced computational complexity for hyperparameter inference. In particular, each step of the gradient based optimization is $\mathcal{O}(nD^2)$.

2.4.2 Incremental Updates

To incrementally handle a stream of data, the matrix $\mathbf{A}$ and the vector $\mathbf{b} = \boldsymbol{\phi}(\mathbf{X})^\intercal\mathbf{y}$ need to be updated in real-time. Given a new sample, $(\mathbf{x}, y)$, the updates are computed according to the rules (Gijsberts and Metta, 2013):

$$\mathbf{A} \leftarrow \mathbf{A} + \boldsymbol{\phi}(\mathbf{x})\boldsymbol{\phi}(\mathbf{x})^\intercal \quad \text{and} \quad \mathbf{b} \leftarrow \mathbf{b} + \boldsymbol{\phi}(\mathbf{x})y. \tag{2.14}$$

Since $\mathbf{A}$ remains positive semidefinite after each update, we do not need to store it explicitly. Instead, we can keep track of its upper triangular Cholesky factor. This allows us to make use of fast, numerically stable rank-1 Cholesky updates (Gijsberts and Metta, 2013). A demonstration of incremental updating is shown in figure 2.2. Additional samples can be incorporated to improve predictions in areas far away from our initial training dataset.

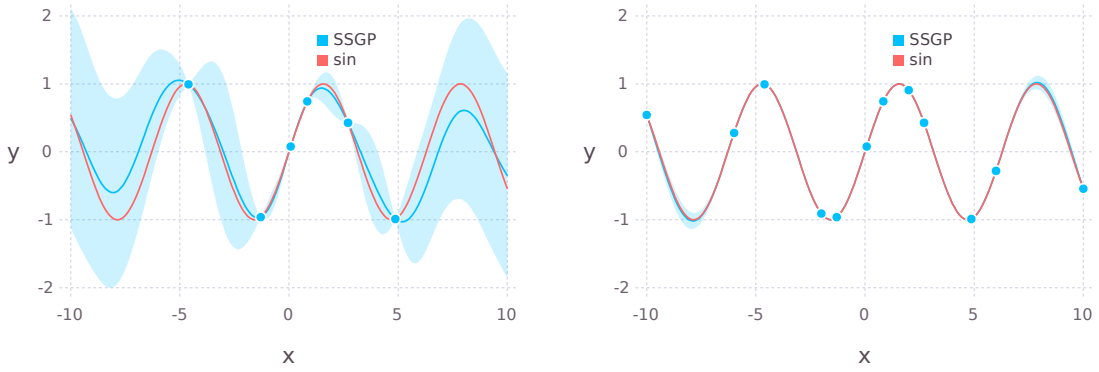

Figure 2.2: Incrementally updating a sparse spectrum GP. **Left**: SSGP fit to six data points sampled from the sine function. Hyperparameters set to ($\sigma_n^2 = 0.001$, $\sigma_s^2 = 0.8$, $l = 1.58$). **Right**: Incrementally updated SSGP using an additional six data points.

Chapter 3

Constrained Convex Optimization

This chapter reviews some fundamental ideas from convex optimization and introduces structure exploiting solvers for **model predictive control**. We begin by defining **convex programs** in section 3.1 and then discuss **primal-dual interior point** methods in section 3.2. Next, we introduce **convex multistage problems** in section 3.3 and show how to exploit their structure for real-time optimization. Finally, section 3.4 provides some examples and applications of the fast interior point methods outlined in this chapter.

3.1 Basic Definitions

This section serves as a brief summary of chapters 4 and 5 of Boyd and Vandenberghe (2004). We introduce convex programs and some basic definitions relevant to the rest of the chapter.

Consider the convex program:

$$\begin{aligned} \underset{\mathbf{z}}{\text{minimize}} \quad & f(\mathbf{z}) \\ \text{subject to} \quad & A\mathbf{z} = \mathbf{b}, \\ & \mathbf{g}(\mathbf{z}) \leq 0, \end{aligned} \tag{3.1}$$

where A is a $p \times n$ matrix with full row rank, $\mathbf{b}$ is a vector in $\mathbb{R}^p$, and $\mathbf{z}$ is a vector of **primal variables** in $\mathbb{R}^n$. The **objective function** $f : \mathbb{R}^n \to \mathbb{R}$ and **non-linear constraints** $g : \mathbb{R}^n \to \mathbb{R}^m$ are convex.

Optimality: We say that a point $\mathbf{z}^*$ in $\mathbb{R}^n$ is **locally optimal** if:

i) it is (primal) **feasible** — that is, $A\mathbf{z}^* = \mathbf{b}$ and $\mathbf{g}(\mathbf{z}^*) \leq 0$; and
ii) there is a scalar $\rho > 0$ such that $f(\mathbf{z}) \geq f(\mathbf{z}^*)$ for all feasible $\mathbf{z}$ with $||\mathbf{z} - \mathbf{z}^*|| < \rho$.

If condition ii) holds for any $\rho > 0$ then $\mathbf{z}^*$ is **globally optimal**. A fundamental property of convex problems is that any local solution is also a global solution (Boyd and Vandenberghe, 2004).

Lagrangian: An important function in mathematical optimization is the **Lagrangian**, defined by

$$\mathcal{L}(\mathbf{z}, \boldsymbol{\nu}, \boldsymbol{\lambda}) := f(\mathbf{z}) + \nu^\intercal (A\mathbf{z} - \mathbf{b}) + \boldsymbol{\lambda}^\intercal \mathbf{g}(\mathbf{z}). \tag{3.2}$$

The components of the vectors $\boldsymbol{\nu} \in \mathbb{R}^p$ and $\boldsymbol{\lambda} \in \mathbb{R}^m$ are known as **dual variables**. We also define a closely related function — the **Lagrangian dual** — to be the minimum of (3.2) over the primal variables $\mathbf{z} \in \mathbb{R}^n$ i.e.

$$d(\boldsymbol{\nu}, \boldsymbol{\lambda}) := \inf_{\mathbf{z}} \mathcal{L}(\mathbf{z}, \boldsymbol{\nu}, \boldsymbol{\lambda}). \tag{3.3}$$

Dual problem: For $\boldsymbol{\lambda} \geq 0$ and any feasible point, the Lagrangian dual is a lower bound on f. We can find the tightest lower bound by solving the **dual problem** associated with (3.1):

$$\begin{aligned} \underset{\boldsymbol{\nu}, \boldsymbol{\lambda}}{\text{maximize}} \quad & d(\boldsymbol{\nu}, \boldsymbol{\lambda}) \\ \text{subject to} \quad & \boldsymbol{\lambda} \geq 0. \end{aligned} \tag{3.4}$$

Quantifying the relationship between primal and dual problems is a major area of study in convex optimization. We refer the reader to Boyd and Vandenberghe (2004) and Wright (1997) for an overview of duality theory and its application to the design and analysis of interior point methods.

Duality gap: As a final note, duality allows us to bound the sub-optimality of a feasible point $\mathbf{z} \in \mathbb{R}^n$. If $(\boldsymbol{\nu}, \boldsymbol{\lambda})$ is dual feasible and p^* denotes the solution to the primal problem, then

$$f(\mathbf{z}) - p^* \leq f(\mathbf{z}) - d(\boldsymbol{\nu}, \boldsymbol{\lambda}). \tag{3.5}$$

The difference between the primal and dual objectives is called the **duality gap** and provides a useful stopping criterion for iterative optimization methods.

3.1.1 Karush-Kuhn-Tucker Conditions

Under the assumption that (3.1) is strictly feasible, the **Karush-Kuhn-Tucker** (KKT) conditions are necessary and sufficient conditions for a primal-dual pair to be optimal. Let $\mathbf{z}^* \in \mathbb{R}^n$, $\boldsymbol{\nu}^* \in \mathbb{R}^p$, $\boldsymbol{\lambda}^* \in \mathbb{R}^m$, and let $\mathbf{s}^* \in \mathbb{R}^m$ be a nonnegative vector of slacks. Then $\mathbf{z}^*$ and $(\boldsymbol{\nu}^*, \boldsymbol{\lambda}^*)$ are primal-dual optimal if and only if they satisfy the KKT conditions:

$$\begin{aligned} \nabla f(\mathbf{z}^*) + A^\intercal \boldsymbol{\nu}^* + G(\mathbf{z}^*)^\intercal \boldsymbol{\lambda}^* = 0, & \qquad \text{(stationarity)} \\ A\mathbf{z}^* - \mathbf{b} = 0, \quad \mathbf{g}(\mathbf{z}^*) + \mathbf{s}^* = 0, & \qquad \text{(primal feasibility)} \\ s_i^* \lambda_i^* = 0, \text{ for } i = 1, \ldots, m & \qquad \text{(complementary slackness)} \\ (\mathbf{s}^*, \boldsymbol{\lambda}^*) \geq 0. & \qquad \text{(dual feasibility)} \end{aligned}$$

3.2 Primal-dual Methods

Primal-dual methods find a solution to problem (3.1) by applying Newton's method to the KKT conditions. This defines an iterative procedure where we compute a **search direction** by linearizing the KKT conditions and solving the resulting system of equations:

$$\begin{bmatrix} H(\mathbf{z},\boldsymbol{\lambda}) & A^\intercal & G(\mathbf{z})^\intercal & 0 \\ A & 0 & 0 & 0 \\ G(\mathbf{z}) & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta\mathbf{z} \\ \Delta\boldsymbol{\nu} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = - \begin{bmatrix} \nabla\mathbf{f}(\mathbf{z}) + A^\intercal\boldsymbol{\nu} + G(\mathbf{z})^\intercal\boldsymbol{\lambda} \\ A\mathbf{z} - \mathbf{b} \\ \mathbf{g}(\mathbf{z}) + \mathbf{s} \\ \Lambda S\mathbf{1} \end{bmatrix}, \tag{3.7}$$

where $\Lambda := \texttt{diag}(\lambda_1, \ldots, \lambda_m)$ and $S := \texttt{diag}(s_1, \ldots, s_m)$ are diagonal matrices and

$$H(\mathbf{z},\boldsymbol{\lambda}) := \nabla^2 f(\mathbf{z}) + \sum_{i=1}^{m} \lambda_i \nabla^2 \mathbf{g}_i(\mathbf{z}). \tag{3.8}$$

Taking a full step in the Newton direction $(\Delta\mathbf{z}, \Delta\boldsymbol{\nu}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s})$ is usually not admissible, since it would violate the bounds $(\mathbf{s}, \boldsymbol{\lambda}) \geq 0$ (Wright, 1997). Consequently, we perform a line search to ensure that the next iterate:

$$(\mathbf{z}, \boldsymbol{\nu}, \boldsymbol{\lambda}, \mathbf{s}) \leftarrow (\mathbf{z}, \boldsymbol{\nu}, \boldsymbol{\lambda}, \mathbf{s}) + \alpha(\Delta\mathbf{z}, \Delta\boldsymbol{\nu}, \Delta\boldsymbol{\lambda}, \Delta\mathbf{s}), \tag{3.9}$$

is dual feasible for some $\alpha \in (0, 1]$. Unfortunately, this often restricts us to small steps in the Newton direction — hindering progress toward a solution in practice. Primal-dual methods address this issue by biasing the search direction toward the interior of the non-negative orthant where $(\mathbf{s}, \boldsymbol{\lambda}) \geq 0$. This allows us to take larger steps without violating the feasibility conditions.

3.2.1 The Central Path

To bias the Newton direction, primal-dual methods typically track the **central path** to the solution. The central path is a set of points $(\mathbf{z}_\tau, \boldsymbol{\nu}_\tau, \boldsymbol{\lambda}_\tau, \mathbf{s}_\tau)$, parameterized by $\tau > 0$, that solves the relaxed KKT conditions:

$$\nabla f(\mathbf{z}) + A^\intercal\boldsymbol{\nu} + G(\mathbf{z}))^\intercal\boldsymbol{\lambda} = 0, \tag{3.10a}$$

$$A\mathbf{z} - \mathbf{b} = 0, \quad \mathbf{g}(\mathbf{z}) + \mathbf{s} = 0, \tag{3.10b}$$

$$s_i\lambda_i = \tau, \text{ for } i = 1, \ldots, m \tag{3.10c}$$

$$(\mathbf{s}, \boldsymbol{\lambda}) > 0. \tag{3.10d}$$

The key idea is to solve (3.10) for a decreasing sequence of τ. As τ decreases to zero, the central path guides us toward a solution while keeping the products $s_i\lambda_i$ strictly positive.

More concretely, we introduce a **centering parameter** $\sigma \in [0, 1]$ and the **duality measure**, $\mu := \mathbf{s}^\intercal\boldsymbol{\lambda}/m$. The biased search direction is found by solving the following system

of equations:

$$\begin{bmatrix} H(\mathbf{z},\boldsymbol{\lambda}) & A^\intercal & G(\mathbf{z})^\intercal & 0 \\ A & 0 & 0 & 0 \\ G(\mathbf{z}) & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta\mathbf{z} \\ \Delta\boldsymbol{\nu} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = - \begin{bmatrix} \nabla\mathbf{f}(\mathbf{z}) + A^\intercal\boldsymbol{\nu} + G(\mathbf{z})^\intercal\boldsymbol{\lambda} \\ A\mathbf{z} - \mathbf{b} \\ \mathbf{g}(\mathbf{z}) + \mathbf{s} \\ \Lambda S\mathbf{1} - \sigma\mu\mathbf{1} \end{bmatrix}. \tag{3.11}$$

At one extreme, choosing $\sigma = 0$ results in the standard Newton direction (also known as the **affine-scaling** direction). Moving in this direction typically pushes iterates close to the boundary of the feasible set, limiting step size in practice.

At the other extreme, setting $\sigma = 1$ gives us a **centering direction** pointing towards the point $(\mathbf{z}_\mu, \boldsymbol{\nu}_\mu, \boldsymbol{\lambda}_\mu, \mathbf{s}_\mu)$ on the central path. Centering steps are usually biased strongly towards the interior of the nonnegative orthant but make little headway at reducing μ. However, moving closer to the central path allows us to take relatively large steps in subsequent iterations without violating feasibility.

A basic implementation of the primal-dual interior point method is shown in algorithm 2. Typically, we use the duality gap or the infinity norm of the residuals as stopping criteria.

Algorithm 2: Basic primal-dual interior point method

Data: Initial iterates $\mathbf{z}_0$, $\boldsymbol{\nu}_0$, $\boldsymbol{\lambda}_0 > 0$ and $\mathbf{s}_0 > 0$, centering parameter $\sigma \in (0, 1]$

for $k = 0, 1, 2, \ldots$ **do**
- Compute the duality measure $\mu_k = \mathbf{s}_k^\intercal \boldsymbol{\lambda}_k / m$
- Find the search direction $(\Delta\mathbf{z}_k, \Delta\boldsymbol{\nu}_k, \Delta\boldsymbol{\lambda}_k, \Delta\mathbf{s}_k)$ by solving (3.11)
- Update the solution:
  $(\mathbf{z}_{k+1}, \boldsymbol{\nu}_{k+1}, \boldsymbol{\lambda}_{k+1}, \mathbf{s}_{k+1}) = (\mathbf{z}_k, \boldsymbol{\nu}_k, \boldsymbol{\lambda}_k, \mathbf{s}_k) + \alpha_k(\Delta\mathbf{z}_k, \Delta\boldsymbol{\nu}_k, \Delta\boldsymbol{\lambda}_k, \Delta\mathbf{s}_k)$
  choosing a step size α_k such that $\mathbf{s}_{k+1} > 0$ and $\boldsymbol{\lambda}_{k+1} > 0$

end

3.2.2 Mehrotra's Predictor-Corrector Method

Since its introduction in 1992, Mehrotra's predictor-corrector method (Mehrotra, 1992) has become a standard tool in convex optimization (Mattingley and Boyd, 2012; Czyzyk et al., 1999; Vanderbei, 1999). Mehrotra's algorithm builds on the basic primal-dual method by adding:

i) an affine-scaling "predictor" step;
ii) an adaptive heuristic for choosing the centering parameter; and
iii) a "corrector" step that addresses linearization error in the complementary slackness conditions.

Predictor step: Given a point $(\mathbf{z}, \boldsymbol{\nu}, \boldsymbol{\lambda}, \mathbf{s})$ with $(\boldsymbol{\lambda}, \mathbf{s}) > 0$, we begin by by solving (3.11) with $\sigma = 0$ to compute the affine-scaling direction $(\Delta\mathbf{z}^{\text{aff}}, \Delta\boldsymbol{\nu}^{\text{aff}}, \Delta\boldsymbol{\lambda}^{\text{aff}}, \Delta\mathbf{s}^{\text{aff}})$. Then we perform a line search to find the maximum feasible step size in this direction:

$$\alpha^{\text{aff}} := \max\{\alpha \in [0,1] : \mathbf{s} + \alpha\Delta\mathbf{s}^{\text{aff}} \geq 0, \boldsymbol{\lambda} + \alpha\Delta\boldsymbol{\lambda}^{\text{aff}} \geq 0\}. \tag{3.12}$$

As a measure of progress, we can then predict the duality resulting from a step in the affine-scaling direction:

$$\mu^{\text{aff}} := (\mathbf{s} + \alpha\Delta\mathbf{s}^{\text{aff}})^\intercal(\boldsymbol{\lambda} + \alpha\Delta\boldsymbol{\lambda}^{\text{aff}})/m. \tag{3.13}$$

Centering step: If $\mu^{\text{aff}} \ll \mu$, we can make good progress toward the solution by following the affine-scalling direction. This motivates choosing a small value for σ. However if μ^{aff} is roughly equal to μ, we should take a centering step by setting σ close to one. Mehrotra (1992) proposes the following heuristic to determine the centering parameter:

$$\sigma := (\mu^{\text{aff}}/\mu)^3. \tag{3.14}$$

The centering direction can then be computed by solving the following system:

$$\begin{bmatrix} H(\mathbf{z},\boldsymbol{\lambda}) & A^\intercal & G(\mathbf{z})^\intercal & 0 \\ A & 0 & 0 & 0 \\ G(\mathbf{z}) & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta\mathbf{z}^{\text{cent}} \\ \Delta\boldsymbol{\nu}^{\text{cent}} \\ \Delta\boldsymbol{\lambda}^{\text{cent}} \\ \Delta\mathbf{s}^{\text{cent}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \sigma\mu\mathbf{1} \end{bmatrix}. \tag{3.15}$$

Corrector step: To motivate the corrector step, consider the complementary slackness conditions after a step in the affine-scaling direction is taken:

$$\begin{aligned} (\mathbf{s}_i + \Delta\mathbf{s}_i^{\text{aff}})(\boldsymbol{\lambda}_i + \Delta\boldsymbol{\lambda}_i^{\text{aff}}) &= \mathbf{s}_i\boldsymbol{\lambda}_i + \boldsymbol{\lambda}_i\Delta\mathbf{s}_i^{\text{aff}} + \mathbf{s}\Delta\boldsymbol{\lambda}_i^{\text{aff}} + \Delta\mathbf{s}_i^{\text{aff}}\Delta\boldsymbol{\lambda}_i^{\text{aff}} && \text{(3.16a)} \\ &= \Delta\mathbf{s}_i^{\text{aff}}\Delta\boldsymbol{\lambda}_i^{\text{aff}}. && \text{(3.16b)} \end{aligned}$$

Instead of going to zero, the pairwise products $\mathbf{s}_i\boldsymbol{\lambda}_i$ become $\Delta\mathbf{s}_i^{\text{aff}}\Delta\boldsymbol{\lambda}_i^{\text{aff}}$. To compensate for this error a corrector direction can be found by solving the system:

$$\begin{bmatrix} H(\mathbf{z},\boldsymbol{\lambda}) & A^\intercal & G(\mathbf{z})^\intercal & 0 \\ A & 0 & 0 & 0 \\ G(\mathbf{z}) & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta\mathbf{z}^{\text{cor}} \\ \Delta\boldsymbol{\nu}^{\text{cor}} \\ \Delta\boldsymbol{\lambda}^{\text{cor}} \\ \Delta\mathbf{s}^{\text{cor}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -\Delta S^{\text{aff}}\Delta\Lambda^{\text{aff}}\mathbf{1} \end{bmatrix}, \tag{3.17}$$

where $\Delta S^{\text{aff}} := \texttt{diag}(\Delta\mathbf{s}^{\text{aff}})$ and $\Delta\Lambda^{\text{aff}} := \texttt{diag}(\Delta\boldsymbol{\lambda}^{\text{aff}})$ are diagonal matrices.

The final search direction is then a sum of the affine-scaling, centering and corrector terms. Note that the coefficient matrices in (3.11), (3.15) and (3.17) are identical so we can save on computation by combining the right-hand sides to form the following system:

$$\begin{bmatrix} H(\mathbf{z},\boldsymbol{\lambda}) & A^\intercal & G(\mathbf{z})^\intercal & 0 \\ A & 0 & 0 & 0 \\ G(\mathbf{z}) & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta\mathbf{z} \\ \Delta\boldsymbol{\nu} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = -\begin{bmatrix} \nabla\mathbf{f}(\mathbf{z}) + A^\intercal\boldsymbol{\nu} + G(\mathbf{z})^\intercal\boldsymbol{\lambda} \\ A\mathbf{z} - \mathbf{b} \\ \mathbf{g}(\mathbf{z}) + \mathbf{s} \\ \Lambda S\mathbf{1} - \sigma\mu\mathbf{1} + \Delta S^{\text{aff}}\Delta\Lambda^{\text{aff}}\mathbf{1} \end{bmatrix}. \tag{3.18}$$

Figure 3.2.2 highlights the differences between the basic primal-dual method and Mehrotra's predictor-corrector method.

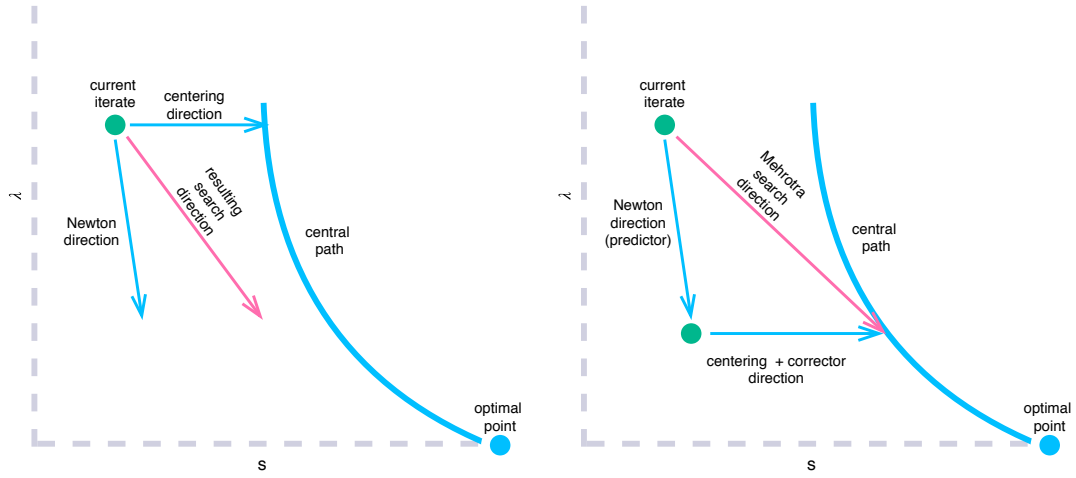

Figure 3.1: Contrasting the basic primal-dual interior point method with Mehrotra's predictor-corrector method. **Left:** Using centering the basic primal-dual method biases the affine-scaling search direction. Moving towards the central path allows for larger steps in subsequent iterations. **Right:** Mehrotra's predictor-corrector computes a search direction using a combination of predictor, centering and corrector terms. The resulting iterates track the central path more closely, giving better performance in practice.

Algorithm 3: Mehrotra's predictor-corrector method

Data: Initial iterates $\mathbf{z}_0$, $\boldsymbol{\nu}_0$, $\boldsymbol{\lambda}_0 > 0$ and $\mathbf{s}_0 > 0$

for $k = 0, 1, 2, \ldots$ **do**

- Compute the duality measure $\mu_k = \mathbf{s}^\intercal \boldsymbol{\lambda}/m$
- Find the affine direction $(\Delta\mathbf{z}_k^{\text{aff}}, \Delta\boldsymbol{\nu}_k^{\text{aff}}, \Delta\boldsymbol{\lambda}_k^{\text{aff}}, \Delta\mathbf{s}_k^{\text{aff}})$ by solving (3.11)
- Perform a line search to find the affine step size:
  $\alpha_k^{\text{aff}} = \max\{\alpha \in [0,1] : \mathbf{s}_k + \alpha\Delta\mathbf{s}_k^{\text{aff}} \geq 0, \boldsymbol{\lambda}_k + \alpha\Delta\boldsymbol{\lambda}_k^{\text{aff}} \geq 0\}$
- Compute the predicted affine duality measure:
  $\mu_k^{\text{aff}} = (\mathbf{s}_k + \alpha\Delta\mathbf{s}_k^{\text{aff}})^\intercal(\boldsymbol{\lambda}_k + \alpha\Delta\boldsymbol{\lambda}_k^{\text{aff}})/m$
- Compute the centering parameter $\sigma_k = (\mu_k^{\text{aff}}/\mu_k)^3$
- Find the combined direction $(\Delta\mathbf{z}_k, \Delta\boldsymbol{\nu}_k, \Delta\boldsymbol{\lambda}_k, \Delta\mathbf{s}_k)$ by solving (3.18)
- Perform a line search to find the final step size:
  $\alpha_k = \max\{\alpha \in [0,1] : \mathbf{s}_k + \alpha\Delta\mathbf{s}_k \geq 0, \boldsymbol{\lambda}_k + \alpha\Delta\boldsymbol{\lambda}_k \geq 0\}$
- Update the solution:
  $(\mathbf{z}_{k+1}, \boldsymbol{\nu}_{k+1}, \boldsymbol{\lambda}_{k+1}, \mathbf{s}_{k+1}) = (\mathbf{z}_k, \boldsymbol{\nu}_k, \boldsymbol{\lambda}_k, \mathbf{s}_k) + \alpha_k(\Delta\mathbf{z}_k, \Delta\boldsymbol{\nu}_k, \Delta\boldsymbol{\lambda}_k, \Delta\mathbf{s}_k)$

end

3.3 Exploiting Problem Structure

In principle, the interior point methods introduced in section 3 can be used to solve optimization problems arising in a broad range of applications. However, the computational

burden of solving large linear systems at each iteration can be limiting in practice. In light of this issue, there has been growing interest in efficient optimization for real-time applications (Wang and Boyd, 2010; Domahidi et al., 2012; Frison et al., 2014). A particularly successful approach has been the combination of interior point methods with structure exploiting multistage solvers (Kouzoupis et al., 2015). In this section we introduce **convex multistage problems** and show how to leverage their structure to efficiently compute the search directions in algorithm 3. The approach outlined in this section follows Domahidi et al. (2012).

3.3.1 Convex Multistage Problems

Problem 1. Consider the multistage problem:

$$\min_{\mathbf{z}_1, \mathbf{z}_2, \ldots} \quad \sum_{k=0}^{N} l_k(\mathbf{z}_k), \qquad \text{(stage costs)}$$

subject to

$$\begin{aligned}
&\mathbf{L}_0(\mathbf{z}_0) = 0 && && \text{(initial equality)}\\
&\mathbf{L}_k(\mathbf{z}_{k-1}, \mathbf{z}_k) = 0, && k = 1, \ldots, N && \text{(inter-stage equality constraints)}\\
&\mathbf{g}_k(\mathbf{z}_k) \leq 0 && k = 0, \ldots, N && \text{(inequality constraints)}
\end{aligned}$$

with $N+1$ stage variables $\mathbf{z}_k \in \mathbb{R}^{n_k}$, convex stage costs $l_k : \mathbb{R}^{n_k} \to \mathbb{R}$, convex inequality constraints $\mathbf{g}_k : \mathbb{R}^{n_k} \to \mathbb{R}^{m_k}$ with non-empty interior and affine inter-stage equality constraints $\mathbf{L}_0 : \mathbb{R}^{n_0} \to \mathbb{R}^{p_0}$ and $\mathbf{L}_k : \mathbb{R}^{n_{k-1}} \times \mathbb{R}^{n_k} \to \mathbb{R}^{p_k}$ defined by:

$$\begin{aligned}
\mathbf{L}_0(\mathbf{z}_0) &:= D_0 \mathbf{z}_0 + \mathbf{c}_0,\\
\mathbf{L}_k(\mathbf{z}_{k-1}, \mathbf{z}_k) &:= C_{k-1}\mathbf{z}_{k-1} + D_k \mathbf{z}_k + \mathbf{c}_k \qquad k = 1, \ldots, N
\end{aligned}$$

Here C_k and D_k are $p_k \times n_k$ matrices such that $[C_{k-1} D_k]$ has full row rank and $\mathbf{c}_k \in \mathbb{R}^{p_k}$. Altogether we have $n := n_0 + \ldots + n_N$ stage variables, $m := m_0 + \ldots + m_N$ inequality constraints and $p := p_0 + \ldots + p_N$ affine equality constraints.

Model predictive control: Note that any model predictive control problem with affine time-varying dynamics can be reformulated as a multistage problem. For example, consider a system described by the dynamics

$$\mathbf{x}_{k+1} = A_k \mathbf{x}_k + B_k \mathbf{u}_k + \mathbf{b}_k, \text{ and } \mathbf{x}_0 = \hat{\mathbf{x}}_0, \tag{3.20}$$

where $\mathbf{x}_k$ and $\mathbf{u}_k$ are the state and control vectors at time step k. Our goal is to compute a sequence of control inputs that minimize some cost function. We can rewrite (3.20) as a sequence of inter-stage constraints by setting

$$\mathbf{z}_k := [\mathbf{u}_k, \mathbf{x}_{k+1}], \qquad k = 0, \ldots, N-1 \tag{3.21a}$$

$$C_{k-1} := [0, A_k], \quad D_k := [B_k, -I], \quad \mathbf{c}_k := \mathbf{b}_k, \qquad k = 1, \ldots, N \tag{3.21b}$$

$$D_0 := [B_0, -I], \text{ and } \mathbf{c}_0 := A_0 \hat{\mathbf{x}}_0 + \mathbf{b}_0. \tag{3.21c}$$

The resulting multistage problem can then be solved to find the optimal control.

Search directions: For multistage problems search directions can be found by solving the following linear system:

$$\begin{bmatrix} H(\mathbf{z},\boldsymbol{\lambda}) & A^\intercal & G(\mathbf{z})^\intercal & 0 \\ A & 0 & 0 & 0 \\ G(\mathbf{z}) & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta\mathbf{z} \\ \Delta\boldsymbol{\nu} \\ \Delta\boldsymbol{\lambda} \\ \Delta\mathbf{s} \end{bmatrix} = - \begin{bmatrix} \mathbf{r}_c \\ \mathbf{r}_e \\ \mathbf{r}_i \\ \mathbf{r}_s \end{bmatrix}, \tag{3.22a}$$

where the residuals $\mathbf{r}_c \in \mathbb{R}^n$, $\mathbf{r}_e \in \mathbb{R}^p$, $\mathbf{r}_i \in \mathbb{R}^m$ and $\mathbf{r}_s \in \mathbb{R}^m$ are defined by the right hand side of (3.11). Stage variables, duals and slacks are concatenated into vectors

$$\mathbf{z} := \big[\mathbf{z}_0, \ldots, \mathbf{z}_N\big] \in \mathbb{R}^n, \quad \boldsymbol{\nu} := \big[\boldsymbol{\nu}_0, \ldots, \boldsymbol{\nu}_N\big] \in \mathbb{R}^p, \tag{3.22b}$$

$$\boldsymbol{\lambda} := \big[\boldsymbol{\lambda}_0, \ldots, \boldsymbol{\lambda}_N\big] \in \mathbb{R}^m, \quad \mathbf{s} := \big[\mathbf{s}_0, \ldots, \mathbf{s}_N\big] \in \mathbb{R}^m, \tag{3.22c}$$

and the remaining terms are given by:

$$\mathbf{b} := \big[-\mathbf{c}_0, \ldots, -\mathbf{c}_N\big] \in \mathbb{R}^p, \tag{3.22d}$$

$$\mathbf{g}(\mathbf{z}) := \big[\mathbf{g}_0(\mathbf{z}_0), \ldots, \mathbf{g}_0(\mathbf{z}_0)\big] \in \mathbb{R}^m, \tag{3.22e}$$

$$\nabla\mathbf{f}(\mathbf{z}) := \big[\nabla l_0(\mathbf{z}_0), \ldots, \nabla l_N(\mathbf{z}_N)\big] \in \mathbb{R}^n, \tag{3.22f}$$

$$A := \begin{bmatrix} D_0 & 0 & \cdots & 0 & 0 \\ C_0 & D_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & C_{N-1} & D_N \end{bmatrix} \in \mathbb{R}^{p\times n}, \tag{3.22g}$$

$$G(\mathbf{z}) := \texttt{blkdiag}\big(\nabla\mathbf{g}_0(\mathbf{z}_0), \ldots, \nabla\mathbf{g}_N(\mathbf{z}_N)\big) \in \mathbb{R}^{m\times n}, \tag{3.22h}$$

$$H(\mathbf{z},\boldsymbol{\lambda}) := \texttt{blkdiag}\big(H_0(\mathbf{z}_0,\boldsymbol{\lambda}_0), \ldots, H_N(\mathbf{z}_N,\boldsymbol{\lambda}_N)\big) \in \mathbb{R}^{n\times n}, \tag{3.22i}$$

$$\text{with} \quad H_k(\mathbf{z}_k,\boldsymbol{\lambda}_k) := \nabla^2 l_k(\mathbf{z}_k) + \sum_{j=1}^{q_k} \lambda_{kj}\nabla^2 g_{kj}(\mathbf{z}_k) \in \mathbb{R}^{n_k\times n_k}. \tag{3.22j}$$

Note that the multistage structure results in block diagonal matrices $H(\mathbf{z},\boldsymbol{\lambda})$ and $G(\mathbf{z})$, and a block banded equality constraint matrix A.

3.3.2 Reduction by Block Elimination

As a first step towards exploiting the structure of (3.22), we eliminate $\Delta\mathbf{s}$ to obtain the following **symmetric indefinite** system:

$$\begin{bmatrix} H(\mathbf{z},\boldsymbol{\lambda}) & A^\intercal & G(\mathbf{z})^\intercal \\ A & 0 & 0 \\ G(\mathbf{z}) & 0 & -\Lambda^{-1}S \end{bmatrix} \begin{bmatrix} \Delta\mathbf{z} \\ \Delta\boldsymbol{\nu} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \mathbf{r}_c \\ \mathbf{r}_e \\ \mathbf{r}_w \end{bmatrix}, \tag{3.23a}$$

$$\Delta\mathbf{s} = -\Lambda^{-1}(\mathbf{r}_s + S\Delta\boldsymbol{\lambda}), \tag{3.23b}$$

where we define $\mathbf{r}_w := \mathbf{r}_i - \Lambda^{-1}\mathbf{r}_s$. Note that (3.23b) is well-defined since the elements of Λ are strictly positive at any step of Mehrotra's method. To further reduce (3.23) we can eliminate $\Delta\boldsymbol{\lambda}$ to obtain the **augmented system**:

$$\begin{bmatrix} \Phi & A^\intercal \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta\mathbf{z} \\ \Delta\boldsymbol{\nu} \end{bmatrix} = - \begin{bmatrix} \mathbf{r}_d \\ \mathbf{r}_e, \end{bmatrix}, \tag{3.24a}$$

$$\Delta\boldsymbol{\lambda} = S^{-1}\Lambda(G(\mathbf{z})\Delta\mathbf{z} + \mathbf{r}_w), \tag{3.24b}$$

with the matrix $\Phi \in \mathbb{R}^{n\times n}$ and residual $\mathbf{r}_d \in \mathbb{R}^n$ defined by

$$\Phi := H(\mathbf{z}, \boldsymbol{\lambda}) + G(\mathbf{z})^\intercal S^{-1}\Lambda G(\mathbf{z}), \tag{3.25a}$$

$$\mathbf{r}_d := \mathbf{r}_c + G(\mathbf{z})^\intercal S^{-1}\Lambda\mathbf{r}_w. \tag{3.25b}$$

Finally, the most compact representation of the step equations can be obtained by finding the Schur complement of the coefficient matrix in (3.24a). This results in the **normal equations** form:

$$Y\Delta\boldsymbol{\nu} = \boldsymbol{\beta}, \tag{3.26a}$$

$$\Delta\mathbf{z} = -\Phi^{-1}(\mathbf{r}_d + A^\intercal\Delta\boldsymbol{\nu}), \tag{3.26b}$$

where the matrix $Y \in \mathbb{R}^{p\times p}$ and the vector $\boldsymbol{\beta} \in \mathbb{R}^p$ are defined by

$$Y := A\Phi^{-1}A^\intercal, \tag{3.27a}$$

$$\boldsymbol{\beta} := -\Phi^{-1}(\mathbf{r}_d - A^\intercal\Delta\boldsymbol{\nu}). \tag{3.27b}$$

Primal-dual methods based on the normal equations use sparse Cholesky decomposition to solve (3.26a). For multistage problems we can leverage tailored block-wise Cholesky factorization for additional speedups (see section 3.3.4).

3.3.3 Efficiently Computing the Matrix Y

Since the augmented Hessian Φ is block diagonal, its inverse is block diagonal too. Coupled with the block banded structure of A, the coefficient matrix $Y := A\Phi^{-1}A^\intercal$ is symmetric and block tri-diagonal i.e.

$$Y := \begin{bmatrix} Y_{0,0} & Y_{1,0}^\intercal & & & \\ Y_{1,0} & Y_{1,1} & Y_{2,1}^\intercal & & \\ & Y_{2,1} & \ddots & \ddots & \\ & & \ddots & \ddots & Y_{N,N-1}^\intercal \\ & & & Y_{N,N-1} & Y_{N,N} \end{bmatrix} \tag{3.28}$$

with blocks defined as follows:

$$Y_{0,0} := D_0\Phi_0^{-1}D_0^\intercal, \tag{3.29a}$$

$$Y_{k,k} := C_{k-1}\Phi_{k-1}^{-1}C_{k-1}^\intercal + D_k\Phi^{-1}D_k^\intercal, \qquad k = 1, \ldots, N \tag{3.29b}$$

$$Y_{k+1,k} := C_k\Phi^{-\intercal}D_k^\intercal. \qquad k = 0, \ldots, N-1 \tag{3.29c}$$

Domahidi et al. (2012) propose the following three steps to form the matrix Y from the inter-stage equality constraints:

Step 1: Compute the Cholesky decomposition of each block in Φ, i.e. find lower triangular factors L_k such that $\Phi_k = L_k L_k^\intercal$.

Step 2: Solve $V_{k-1} L_{k-1}^\intercal = C_{k-1}$ for V_k and $W_k L_k^\intercal = D_k$ for V_k for W_k using matrix forward substitution.

Step 3: Compute the terms in (3.29), i.e.

$$Y_{0,0} = W_0 W_0^\intercal, \tag{3.30a}$$
$$Y_{k,k} = V_{k-1} V_{k-1}^\intercal + W_k W_k^\intercal, \qquad k = 1, \dots, N \tag{3.30b}$$
$$Y_{k+1,k} = V_k W_k^\intercal. \qquad k = 0, \dots, N-1 \tag{3.30c}$$

The above procedure improves on method of Wang and Boyd (2010) both in terms of numerical stability and efficiency. Computational costs are shown in table 3.1.

Table 3.1: Cost of computing $Y_{k,k}$ and $Y_{k+1,k}$

Step	Operation	Cost (flops)
1	Factor $\Phi_k = L_k L_k^\intercal$	$1/3 n_k^3$
2	Solve $V_{k-1} L_{k-1}^\intercal = C_{k-1}$ for V_k	$n_{k-1}^2 p_k$
2	Solve $W_k L_k^\intercal = D_k$ for W_k	$n_k^2 p_k$
3	Compute $Y_{k,k} = V_{k-1} V_{k-1}^\intercal + W_k W_k^\intercal$	$(n_{k-1} + n_k) p_k^2$
3	Compute $Y_{k+1,k} = V_k W_k^\intercal$	$2 n_k p_k^2$

3.3.4 Block-wise Cholesky Factorization of Y

Finally, by finding the Cholesky decomposition of Y we can solve (3.26) for $\Delta\boldsymbol{\nu}$. Since the matrix Y is block tri-diagonal, its Cholesky factor L_Y has a block banded structure:

$$L_Y := \begin{bmatrix} L_{0,0} & 0 & 0 & \cdots & 0 & 0 \\ L_{1,0} & L_{1,1} & 0 & \cdots & 0 & 0 \\ 0 & L_{2,1} & L_{2,2} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & L_{N-1,N-1} & 0 \\ 0 & 0 & 0 & \cdots & L_{N,N-1} & L_{N,N} \end{bmatrix}, \tag{3.31}$$

where $L_{k,k} \in \mathbb{R}^{r_k \times r_k}$ are lower triangular matrices and $L_{k+1,k} \in \mathbb{R}^{r_{k+1} \times r_k}$ are dense. Leveraging this structure, we see that

$$Y_{0,0} = L_{0,0} L_{0,0}^\intercal, \tag{3.32a}$$
$$Y_{k+1,k} = L_{k+1,k} L_{k,k}^\intercal, \qquad k = 0, \dots, N-1 \tag{3.32b}$$
$$Y_{k,k} - L_{k,k-1} L_{k,k-1}^\intercal = L_{k,k} L_{k,k}^\intercal. \qquad k = 1, \dots, N \tag{3.32c}$$

We can solve (3.32b) for $L_{k+1,k}$ with a forward substitution, while each $L_{k,k}$ is found by applying Cholesky factorization to (3.32a) or (3.32c) (Wang and Boyd, 2010). The computational costs for the Cholesky factorization are given in table 3.2.

Table 3.2: Cost of block-wise Cholesky factorization of Y

Step	Operation	Cost (flops)
(3.32a)	Factor $Y_{0,0} = L_{0,0}L_{0,0}^\intercal$	$1/3p_k^3$
(3.32b)	Solve $Y_{k+1,k} = L_{k+1,k}L_{k,k}^\intercal$ for $L_{k+1,k}$	$p_k^2 p_{k+1}$
(3.32c)	Solve $Y_{k,k} - L_{k,k-1}L_{k,k-1}^\intercal = L_{k,k}L_{k,k}^\intercal$ for $L_{k,k}$	$p_k^2 r_k$

After factorizing Y we can find $\Delta\boldsymbol{\nu}$ by solving the normal equations (3.26a). Then we successively compute $\Delta\mathbf{z}$, $\Delta\boldsymbol{\lambda}$ and $\Delta\mathbf{s}$ using (3.26b), (3.24b) and (3.23b) respectively.

3.4 Examples and Applications

In this section we apply Mehrotra's method (with structure exploiting solver) to some simple model predictive control problems. In the first example we investigate the control of a simple double integrator system. The second example discusses slew rate (or input rate) constraints, and the final example demonstrates soft constraints.

For performance we generate problem specific C-99 code taking advantage of static memory allocation and fixed problem dimensions. Note that model predictive control problems typically have a small number of stage variables so custom linear algebra routines often out perform BLAS/LAPACK (Domahidi et al., 2012; Houska et al., 2011).

Example 1. In this example we consider the following linear MPC problem:

$$\begin{aligned} \underset{\mathbf{x}_k,\mathbf{u}_k}{\text{minimize}} \quad & \mathbf{x}_N^\intercal P \mathbf{x}_N + \sum_{k=0}^{N-1} \mathbf{x}_k^\intercal Q \mathbf{x}_k + \mathbf{u}_k^\intercal R \mathbf{u}_k \\ \text{subject to} \quad & \mathbf{x}_0 = \hat{\mathbf{x}}_0, \\ & \mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k, \\ & \underline{\mathbf{u}} \leq \mathbf{u}_k \leq \overline{\mathbf{u}}, \end{aligned} \tag{3.33a}$$

where

$$\begin{aligned} & Q := \begin{bmatrix} 10 & 0 \\ 0 & 15 \end{bmatrix}, P := 15Q, R := 1, A := \begin{bmatrix} 0.7115 & -0.4345 \\ 0.4345 & 0.8853 \end{bmatrix}, \\ & B := \begin{bmatrix} 0.2173 \\ 0.0573 \end{bmatrix}, \underline{\mathbf{u}} := -0.5, \overline{\mathbf{u}} := 2, \hat{\mathbf{x}}_0. := \begin{bmatrix} -2 \\ 6 \end{bmatrix}. \end{aligned} \tag{3.33b}$$

At each time set we solve (3.33) and apply the first control input to the system. We use a planning horizon of $N := 15$ and plot the controlled dynamics over 30 time steps (see figure 3.2). The goal is to regulate the initial state $\hat{\mathbf{x}}_0 = [-2, 6]^\intercal$ to the origin $[0, 0]^\intercal$.

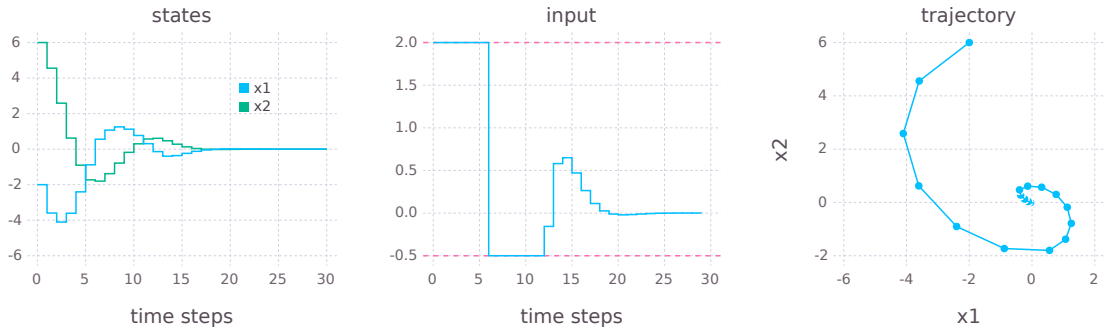

Figure 3.2: Controlled dynamics of the system in example 1. **Left:** State dynamics. Starting from the point $\hat{\mathbf{x}}_0 = [-2, 6]^\intercal$ the system is regulated to $[0, 0]^\intercal$. **Middle:** Control signal. The input limits are shown by the dashed pink lines. **Right:** Phase portrait of the dynamics.

Example 2. In this example we add an additional slew rate constraint to the controller:

$$-1 \leq \mathbf{u}_k - \mathbf{u}_{k-1} \leq 0.5. \tag{3.34}$$

Slew rate constraints stabilize the controller by preventing rapid changes in the control signal. To incorporate the slew rate constraint we augment the state with the previous control input and define the augmented dynamics:

$$\tilde{\mathbf{x}}_k := \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_{k-1} \end{bmatrix}, \quad \tilde{\mathbf{x}}_{k+1} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \tilde{\mathbf{x}}_k + \begin{bmatrix} B \\ I \end{bmatrix} \tilde{\mathbf{u}}_k, \text{ where } \tilde{\mathbf{u}}_k := \mathbf{u}_k - \mathbf{u}_{k-1}. \tag{3.35}$$

Then the slew rate and input constraints become simple constraints on $\tilde{\mathbf{u}}_k$ and $\tilde{\mathbf{x}}_k$ respectively. The state dynamics, control inputs, and slew rate are shown in figure 3.3.

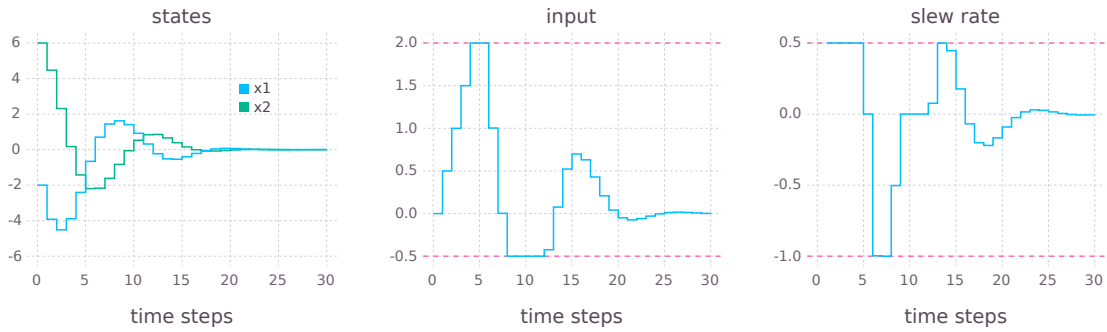

Figure 3.3: Controlled dynamics with slew rate constraint. **Time steps** $0 - 5$**:** The input signal is maximally increased (with a slew rate of 0.5) until it reaches an upper bound of 2. **Time steps** $6 - 8$**:** the input signal is maximally reduced (with a slew rate of -1) until it reaches a lower bound of -0.5.

Example 3. In the final example we demonstrate how to incorporate soft state constraints into the control problem. First, we introduce and penalty matrix S and additional variables $\boldsymbol{\delta}_k$ which represent the magnitude of the constraint violation. Then the soft constrained MPC problem is:

$$\begin{aligned} \underset{\mathbf{x}_k, \mathbf{u}_k}{\text{minimize}} \quad & \mathbf{x}_N^\intercal P \mathbf{x}_N + \sum_{k=0}^{N-1} \mathbf{x}_k^\intercal Q \mathbf{x}_k + \mathbf{u}_k^\intercal R \mathbf{u}_k + \boldsymbol{\delta}_{k+1}^\intercal S \boldsymbol{\delta}_{k+1} \\ \text{subject to} \quad & \mathbf{x}_0 = \hat{\mathbf{x}}_0, \\ & \mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k, \\ & \underline{\mathbf{u}} \leq \mathbf{u}_k \leq \overline{\mathbf{u}}, \\ & \underline{\mathbf{x}} - \boldsymbol{\delta}_k \leq \mathbf{x}_k \leq \overline{\mathbf{x}} + \boldsymbol{\delta}_k, \\ & \boldsymbol{\delta}_k \geq 0. \end{aligned} \tag{3.36}$$

Typically S is chosen to be much larger that Q and R so that the soft formulation does not have a significant effect when all constraints can be satisfied. We set $S := 10^4$ and constrain the inputs and the first component of the sate vector:

$$-1.2 \leq \mathbf{u}_k \leq 0.5, \quad \text{and} \quad -4 - \delta_k \leq \mathbf{x}_k[0] \leq 1.3 + \delta_k. \tag{3.37}$$

Finally, the initial state is set to $\hat{\mathbf{x}} := [-4, 1]^\intercal$. The resulting system dynamics and constraint violation are shown in figure 3.4.

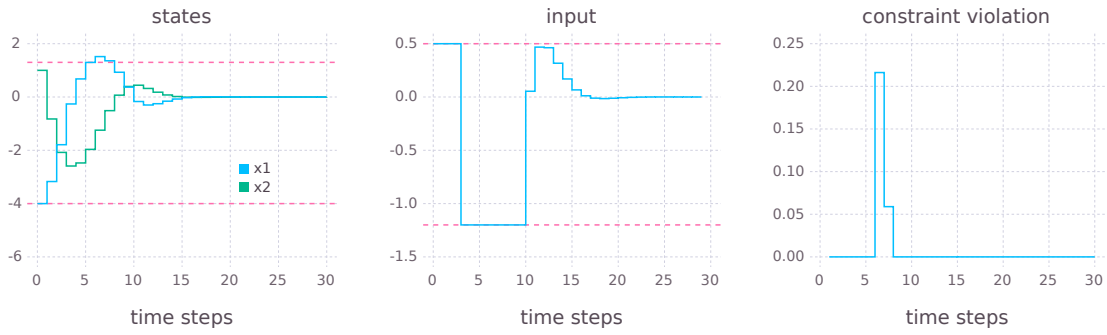

Figure 3.4: Controlled dynamics with soft constraints. **Left:** State dynamics. The state constraints are shown by the pink dashed lines. Between time steps $6 - 8$ the upper bound is violated. **Right:** Magnitude of the constraint violation.

Chapter 4

Problem Formulation

In this chapter we formulate autonomous racing as a **model predictive control** problem. The idea is to plan a trajectory around the race track by solving a sequence of optimization problems with the objective of maximizing progress while avoiding collisions. To ensure that the plan is feasible, the optimization is also constrained by the vehicle dynamics outlined in section 4.1 and the control input constraints discussed in section 4.3. In section 4.2 we describe the race track model and introduce ribbon coordinates. Finally, section 4.4 reviews the **contouring control** framework.

4.1 Car Dynamics

Formally, we consider the control of a 1:43 scale racing car described by the dynamics:

$$\dot{\mathbf{x}} = \mathbf{f}_c(\mathbf{x}, \mathbf{u}) + W\left(\boldsymbol{\delta}_c(\mathbf{x}, \mathbf{u}) + \boldsymbol{\varepsilon}\right), \tag{4.1}$$

composed of a known nominal model $\mathbf{f}_c$ and unmodeled disturbances $\boldsymbol{\delta}_c$. The disturbances are assumed to lie in a subspace spanned by the columns of the matrix W and are subject to additive process noise $\boldsymbol{\varepsilon}$. The state vector $\mathbf{x} = [x, y, \phi, v_x, v_y, \omega]^\intercal$ consists of the position (x, y) of the car, its orientation ϕ, longitudinal and lateral velocities (v_x, v_y), and yaw rate ω. The control inputs $\mathbf{u} = [d, \delta]^\intercal$ are the duty cycle d of the drive train motor and the steering angle δ.

4.1.1 Bicycle Model

Following Liniger et al. (2015), the cars are modeled using a bicycle model with nonlinear tyre forces (see figure 4.1). Pitch and roll dynamics are neglected so only in-plane motion is considered. The symmetry of the car is used to approximate the pairs of front and back

tyres as single wheels resulting in the equations of motion:

$$\mathbf{f}_c(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} v_x \cos\phi - v_y \sin\phi \\ v_x \sin\phi + v_y \cos\phi \\ \omega \\ \frac{1}{m}\big(F_{r,x}(\mathbf{x}, \mathbf{u}) - F_{f,y}(\mathbf{x}, \mathbf{u}) \sin\delta + m v_y \omega\big) \\ \frac{1}{m}\big(F_{r,y}(\mathbf{x}, \mathbf{u}) + F_{f,y}(\mathbf{x}, \mathbf{u}) \cos\delta - m v_x \omega\big) \\ \frac{1}{I_z}\big(F_{f,y}(\mathbf{x}, \mathbf{u}) l_f \cos\delta - F_{r,y}(\mathbf{x}, \mathbf{u}) l_r\big) \end{bmatrix}, \tag{4.2}$$

where m is the mass of a car, I_z is the moment of inertia about the z-axis and l_f and l_r are the distances of the center of gravity from the front and rear wheels respectively. Finally, $F_{r,x}$, $F_{r,y}$ and $F_{f,y}$ are tyre forces which describe the interaction between the car and the road (see section 4.1.2 for details).

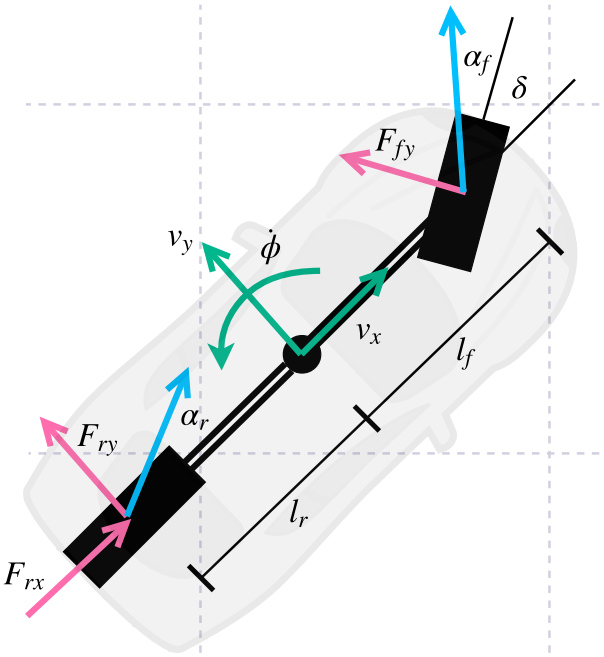

Figure 4.1: Schematic diagram of the bicycle model. Tyre forces (pink), slip angles (blue), velocities (green).

4.1.2 Tyre Model

Tyre forces play an important role in vehicle dynamics and have been investigated extensively since the introduction of Pacejka's Magic Tyre Formula (Pacejka, 2005). In this

work, $F_{r,x}$, $F_{f,y}$ and $F_{r,y}$ are modeled using a simplified Pacejka model (Liniger et al., 2015):

$$F_{f,y} = D_f \sin(C_f \arctan(B_f \alpha_f)) \text{ where } \alpha_f = -\arctan\left(\frac{\omega l_f + v_y}{v_x}\right) + \delta,$$
$$F_{r,y} = D_r \sin(C_r \arctan(B_r \alpha_r)) \text{ where } \alpha_r = \arctan\left(\frac{\omega l_r - v_y}{v_x}\right).$$

Subscripts f and r are used to distinguish the parameters of the front and rear tyres and the constants B, C and D define the shape of the force curve resulting from a slip angle α. Figure 4.2 shows the tyre force curves for the parameters in table 4.1.

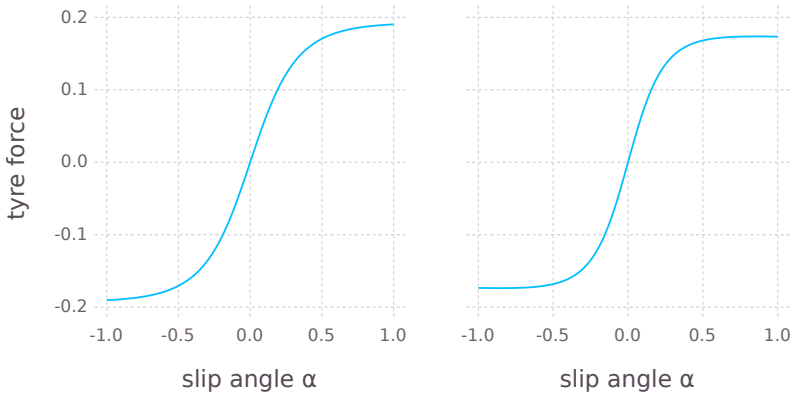

Figure 4.2: Tyre force curves for the front (left) and rear (right) tyres.

Finally, the drivetrain force $F_{r,x}$ is a combination of a DC motor term and a friction term:

$$F_{r,x} = (C_{m1} - C_{m2} v_x) d - C_r - C_d v_x^2.$$

The model identification procedure has been discussed in previous work (Voser et al., 2010). The parameters used in this thesis are reported in table 4.1:

Table 4.1: The model parameters for the bicycle model introduced in section 4.1.

Physical Parameters		Tyre Parameters				Drivetrain Parameters	
I_z	$2.78e^{-5}$	B_f	2.579	B_r	3.3852	C_{m1}	0.2870
m	0.041	C_f	1.200	C_r	1.2691	C_{m2}	0.0545
l_f	0.029	D_f	0.192	D_r	0.1737	C_{r0}	0.0518
l_r	0.033					C_{r2}	$3.5e^{-4}$

4.1.3 Model Discretization

For use in discrete-time model predictive control (see chapter 3) we integrate equation (4.1) using the 4th order Runge-Kutta method. This results in the dynamics:

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) + W(\boldsymbol{\delta}(\mathbf{x}_t, \mathbf{u}_t) + \boldsymbol{\varepsilon}_t), \tag{4.3}$$

where $\mathbf{f}$ is the known nominal component, $\boldsymbol{\delta}$ is the unknown disturbance model and $\boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \Sigma_n)$ is uncorrelated Gaussian noise.

4.1.4 Learning the Disturbance Model

To learn the disturbance model in equation (4.3) we apply Gaussian process regression (see chapter 2). Using data collected from interactions with the environment we train a GP to predict deviations from the nominal system dynamics:

$$\mathbf{g}(\mathbf{x}_k, \mathbf{u}_k) = \boldsymbol{\delta}(\mathbf{x}_k, \mathbf{u}_k) + \varepsilon_k = W^{\dagger}\left(\mathbf{x}_{k+1} - \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)\right), \tag{4.4}$$

where $W^{\dagger}$ is the pseudo-inverse of W. Since $\mathbf{g}$ is modeled as a GP, the states $\mathbf{x}_k$ should be treated as random variables.

4.2 The Race Track

The race track, pictured in figure 4.4, is modeled as a ribbon-like surface described by a center line and a fixed track width. The ribbon defines a curvilinear coordinate system in which points are specified by a distance l along the center line and an offset ρ (see figure 4.3).

To represent the center line we use a parametric cubic spline. For our application we fit an arc-length parametrized curve, $\boldsymbol{\gamma}(l) = [x_c(l), y_c(l)]$, where l is in the interval $[0, L]$ and L is the total length of the path. To find an arc-length parametrization, the center line is interpolated using the method described in Wang et al. (2002a).

4.2.1 Arc-length Parametrization

Starting from a spline curve $\boldsymbol{\lambda}(t)$ with parameter t in $[t_0, t_n]$ and break points $\{t_0, t_1, \ldots, t_n\}$ we compute an approximate arc-length parametrization in three steps. First, the arc-lengths of each segment in $\boldsymbol{\lambda}(t)$ are calculated and summed to find the total path length L. Next, we find $m + 1$ points spaced equally along $\boldsymbol{\lambda}(t)$. Finally, we fit a new cubic spline using the equally spaced points as knots. The full procedure can be described as follows:

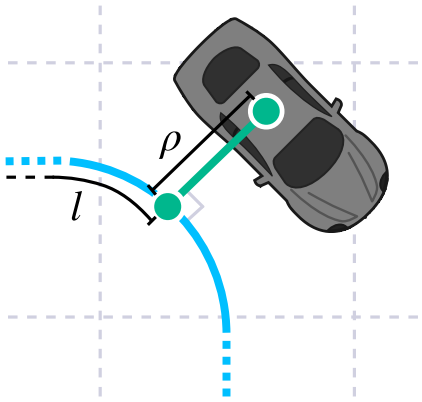

Figure 4.3: The ribbon coordinate system: l is the distance along the center line of the track and ρ is the offset from the center line.

Step 1: The arc-length of the i^{th} segment in $\boldsymbol{\lambda}(t)$ is given by the integral

$$L_i = \int_{t_i}^{t_{i+1}} ||\boldsymbol{\lambda}'(t)||dt, \tag{4.5}$$

which can be evaluated using standard numerical methods e.g. adaptive Gauss-Kronrod quadrature (Calvetti et al., 2000). After computing the arc-length of each segment, the total length the curve is given by the sum $L = \sum_{i=0}^{n} L_i$.

Step 2: Next, we find $m+1$ points on $\boldsymbol{\lambda}(t)$ positioned at increments of $\tilde{L} = L/m$ along the curve. These points are defined in terms of parameters values $\tilde{t}_i$ satisfying

$$\int_0^{\tilde{t}_i} ||\boldsymbol{\lambda}'(t)||dt = i\tilde{L}, \quad \text{where } i = 0, \ldots, m. \tag{4.6}$$

To determine $\tilde{t}_i$ we first find a segment, indexed by j, which satisfies:

$$\sum_{k=0}^{j-1} L_k \leq i\tilde{L} < \sum_{k=0}^{j} L_k,$$

ensuring that $\tilde{t}_i$ lies in the interval $[t_j, t_{j+1})$. After the segment has been identified, the integral in (4.6) can be rewritten as:

$$\int_0^{\tilde{t}_i} ||\boldsymbol{\lambda}'(t)||dt = \int_0^{t_j} ||\boldsymbol{\lambda}'(t)||dt + \int_{t_j}^{\tilde{t}_i} ||\boldsymbol{\lambda}'(t)||dt = \sum_{k=0}^{j-1} L_k + \int_{t_j}^{\tilde{t}_i} ||\boldsymbol{\lambda}'(t)||dt,$$

and the problem can be reduced to finding the value of $\tilde{t}_i$ such that

$$\int_{t_j}^{\tilde{t}_i} ||\boldsymbol{\lambda}'(t)||dt = i\tilde{L} - \sum_{k=0}^{j-1} L_k. \tag{4.7}$$

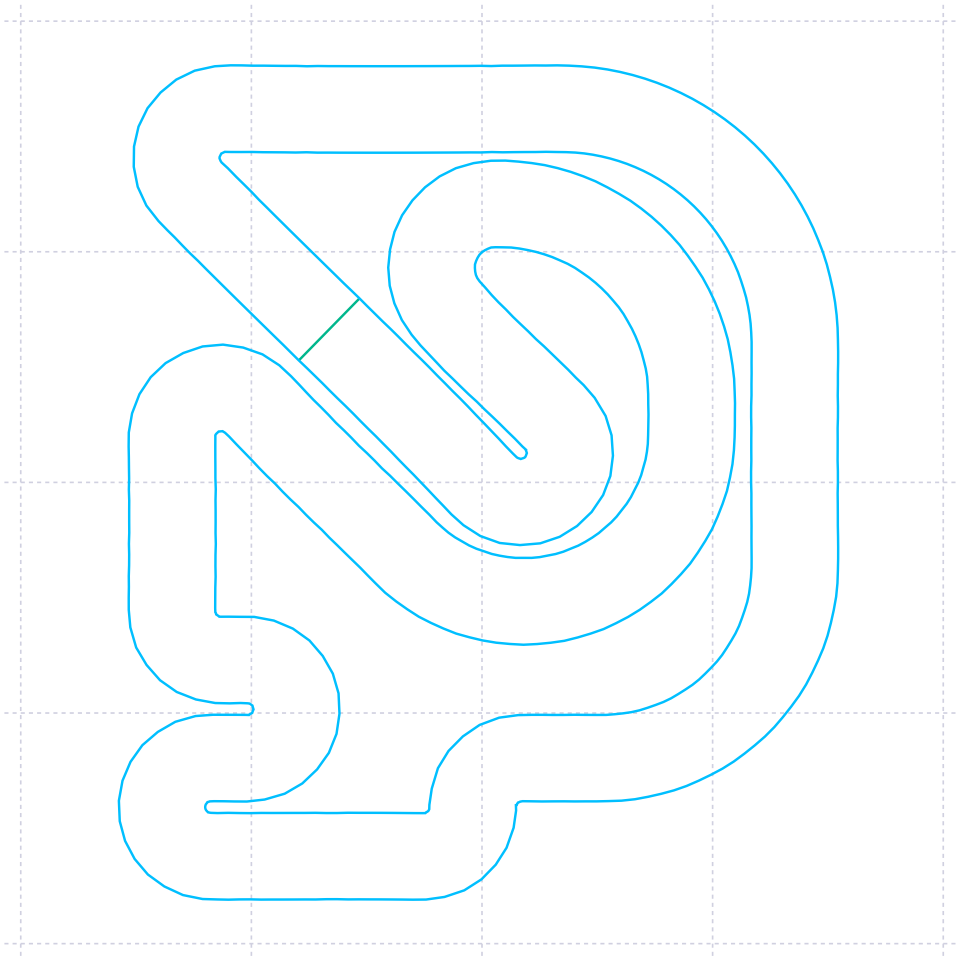

Figure 4.4: The race track is 18.86 meters long and has a width of 0.375 meters. The track contains a number of chicanes, straights and turns to encourage interesting racing lines. The start line is marked in green.

Since the solution to (4.7) lies in the interval $[t_j, t_{j+1}]$ we can use the bisection method to compute $\tilde{t}_i$ to a desired level of accuracy.

Step 3: Applying the above procedure we can compute a set of break points $\{\tilde{t}_0, \ldots, \tilde{t}_m\}$ that divide $\boldsymbol{\lambda}(t)$ into segments of equal length. Then, evaluating $\boldsymbol{\lambda}(t)$ at each $\tilde{t}_i$ gives us a set of equally spaced points which are used as knots for fitting a new spline $\boldsymbol{\gamma}(l)$.

4.2.2 Mapping Between Cartesian and Ribbon Coordinates

Since the car's dynamics are simulated in Cartesian coordinates but the track constraints are expressed in ribbon coordinates we need an efficient mapping between the two coordinate

systems. In this work we use the method proposed by Wang et al. (2002b).

Ribbon to Cartesian Coordinates: Suppose that we are given the ribbon coordinates (l, ρ) of a point, where l is the distance along the center line and ρ is an offset (see figure 4.3). Then mapping from Cartesian coordinates can be computed in three steps:

Step 1: Evaluate the spline to find the point $\boldsymbol{\gamma}(l)$ a distance l along the center line.

Step 2: Compute the unit normal $\mathbf{n}(l)$ to the center line at the point $\boldsymbol{\gamma}(l)$.

Step 3: Find the point offset a distance ρ in the direction of $\mathbf{n}(l)$, i.e. the Cartesian coordinates of the point are given by $\boldsymbol{\gamma}(l) + \rho\mathbf{n}(l)$.

Cartesian to Ribbon Coordinates: The inverse mapping, from Cartesian to ribbon coordinates, presents a bigger computational challenge. Given the car's position $\mathbf{p} = (x, y)$, the corresponding ribbon coordinates can be computed using the following three steps:

Step 1: Project $\mathbf{p}$ onto the center line of the track i.e. find the parameter l^* which minimizes the function

$$\mathtt{d}(l) = \left\|\mathbf{p} - \boldsymbol{\gamma}(l)\right\|^2. \tag{4.8}$$

Note that the width of the track is less than the minimum radius of curvature so (4.8) has a unique minimum.

Step 2: Compute the unit normal $\mathbf{n}(l^*)$ to the center line at the point $\boldsymbol{\gamma}(l^*)$.

Step 3: Find the scalar projection of $\boldsymbol{\gamma}(l^*) - \mathbf{p}$ onto $\mathbf{n}(l^*)$ i.e. $\rho = \mathbf{n}(l^*) \cdot (\boldsymbol{\gamma}(l^*) - \mathbf{p})$.

The key step in this procedure is computing the closest point to $\mathbf{p}$ lying on the center line of the track (step 1 above). Although standard optimization techniques can minimize equation (4.8), Wang et al. (2002b) note that these approaches often fail in practice. To address this issue they propose a combination of quadratic minimization and Newton's method.

Quadratic minimization: The idea behind quadratic minimization is to successively optimize a local quadratic approximation to the objective function. Given l_0, l_1 and l_2 as initial estimates of l^*, we can fit a quadratic polynomial,

$$\mathtt{p}(l) = \frac{(l-l_1)(l-l_2)}{(l_0-l_1)(l_0-l_2)}\mathtt{d}(l_0) + \frac{(l-l_0)(l-l_2)}{(l_1-l_0)(l_1-l_2)}\mathtt{d}(l_1) + \frac{(l-l_0)(l-l_1)}{(l_2-l_0)(l_2-l_1)}\mathtt{d}(l_2), \tag{4.9}$$

that interpolates (4.8) at l_0, l_1 and l_2. The minimizer of $\mathtt{d}(l)$ is then approximated by the minimizer of $\mathtt{p}(l)$, given by

$$l_k^* = \frac{1}{2}\frac{(l_1^2-l_2^2)\mathtt{d}(l_0) + (l_2^2-l_0^2)\mathtt{d}(l_1) + (l_0^2-l_1^2)\mathtt{d}(l_2)}{(l_1-l_2)\mathtt{d}(l_0) + (l_2-l_0)\mathtt{d}(l_1) + (l_0-l_1)\mathtt{d}(l_2)}. \tag{4.10}$$

The parameter resulting in the largest value of $\mathtt{p}(l)$ is replaced by l_k^* and we fit a new quadratic using (4.9). This procedure repeats for a fixed number of iterations or until some error tolerance is reached. With a sufficiently good set of initial guesses, this process will converge to l^* at a superlinear rate (Luenberger et al., 1984).

A key step in quadratic minimization is determining a good set of initial guesses. Fortunately, we can usually guess which segment of the track contains l^* based on the previous position and velocity of the car.

Newton's method: The parameter l^* that minimizes (4.8) satisfies the condition $\mathrm{d}'(l^*) = 0$. So, given an initial guess l_0^*, we can use Newton's method to find the root of this equation:

$$l_{k+1}^* = l_k^* - \frac{\mathrm{d}'(l_k^*)}{\mathrm{d}''(l_k^*)}, \text{ for } k = 0, 1, 2, \ldots \tag{4.11}$$

Since Newton's method utilizes gradients it typically converges faster than quadratic minimization. However, Wang et al. (2002b) observe that a poor starting point can severely impede progress. On the other hand, quadratic minimization is good at refining course initial guesses, motivating a combination of the two methods. The composite algorithm uses a few iterations of quadratic minimization to initialize Newton's method.

4.3 Constraints

To plan a safe trajectory around the track we impose a set of constraints on the car's position and the control inputs. This ensures that the controller will avoid collisions while remaining safely within operational limits.

4.3.1 Track Constraints

First, the car must remain within the boundaries of the track. Let $\mathbf{p} = (x, y)$ denote the position of the car and let (l^*, ρ) be the corresponding ribbon coordinates. Then the distance between the car and the point $\boldsymbol{\gamma}(l^*)$ on the center line must not exceed half the track width (see figure 4.6) i.e. the car must lie within the set

$$\mathcal{X}(l^*) = \left\{ \mathbf{x} : \left\| \mathbf{p} - \boldsymbol{\gamma}(l^*) \right\| \leq r/2 \right\}, \tag{4.12}$$

where r is the track width.

4.3.2 Input Constraints

Second, we limit the maximum steering angle of the car and constrain the duty cycle to lie between zero and one resulting in the input constraint set:

$$\mathcal{U} = \left\{ \mathbf{u} : \begin{bmatrix} 0 \\ -\delta_{\text{max}} \end{bmatrix} \leq \begin{bmatrix} d \\ \delta \end{bmatrix} \leq \begin{bmatrix} 1 \\ \delta_{\text{max}} \end{bmatrix} \right\}. \tag{4.13}$$

For our experiments we set δ_{max} to $\pi/9$ radians.

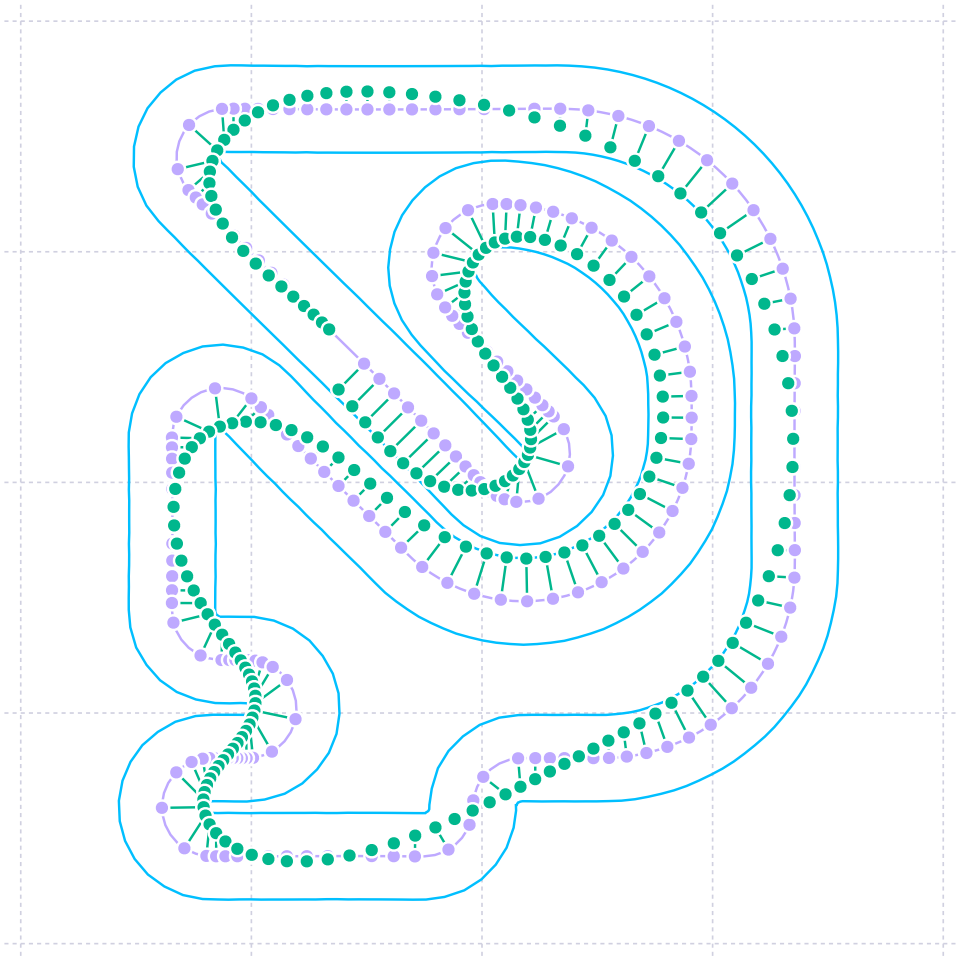

Figure 4.5: To demonstrate the combined approach each point of a typical racing trajectory (green) is projected onto the center line of the track (purple).

4.4 Contouring Control

Contouring control was originally designed for industrial applications such as machine tool control and laser profiling (Lam et al., 2010). Recently, contouring control has also been shown to be an effective framework for autonomous racing (Liniger et al., 2015). The objective of the controller is to track a given reference path while maximizing some measure of progress. Often these are competing interests and we need to find a balance between speed and tracking accuracy.

In contrast to standard tracking approaches, the reference path is described only in terms of spatial coordinates. This allows the contouring controller to trade-off speed and accuracy by setting the velocities and orientations of the planned trajectory.

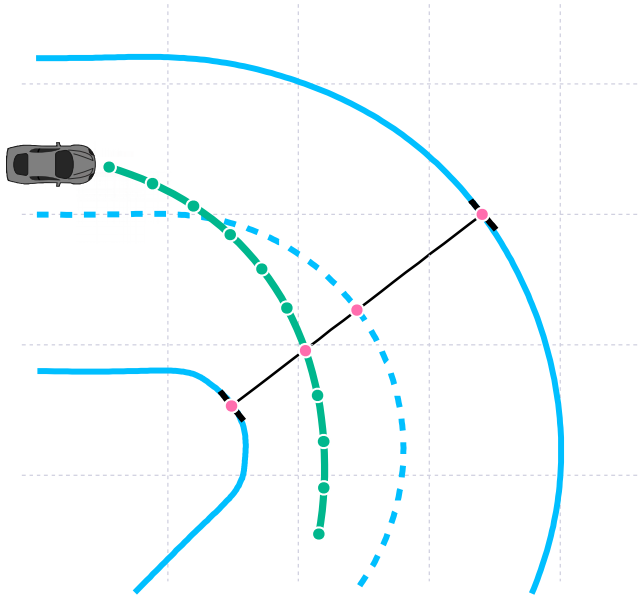

Figure 4.6: Any point along the planned trajectory must lie within the track boundaries descibed by the constraint set $\mathcal{X}$ in equation (4.12).

Let $\mathbf{p}_k = (x_k, y_k)$ denote the position of the car at time t_k. Then, the contouring error,

$$\varepsilon_k^c = \mathbf{n}(l_k^*) \cdot (\mathbf{p}_k - \boldsymbol{\gamma}(l_k^*)), \tag{4.14}$$

is defined as the normal deviation from the path $\boldsymbol{\gamma}$. Here l_k^* is the projection of the point $\mathbf{p}_k$ onto the path, and $\mathbf{n}(l_k^*)$ is the unit normal to $\boldsymbol{\gamma}$ at l_k^*. Calculating the contouring error requires us to determine the value of l_k^* at each point along the planned trajectory. This is too computationally expensive to use as a cost function for real-time model predictive control. To address this issue, we introduce an approximation to l_k^* as a state variable. The car's dynamics are augmented by the equation:

$$l_{k+1} = l_k + \Delta t v_k, \quad v_k \in [0, v_{\text{max}}], \tag{4.15}$$

where l_k denotes the approximation to l_k^* at time t_k, and v_k is a virtual control input. Since the path is parameterized by arc length, v_k can be thought of as the velocity of the car along the center line.

For the auxiliary state l_k to be a useful approximation we introduce a lag error term ε^l defined as the distance between the points $\boldsymbol{\gamma}(l_k^*)$ and $\boldsymbol{\gamma}(l_k)$ along the track. Since neither the contouring nor lag error can be used directly in the cost function, approximations defined only in terms of $\mathbf{p}_k$ and l_k are made. The approximate contouring error $\tilde{\varepsilon}_k^c$ and approximate lag error $\tilde{\varepsilon}_k^l$ are defined as the orthogonal and tangential component of the error between the points $\mathbf{p}_k$ and $\boldsymbol{\gamma}(l_k)$, i.e.

$$\tilde{\varepsilon}_k^c = \mathbf{n}(l_k) \cdot (\mathbf{p}_k - \boldsymbol{\gamma}(s_k)) \text{ and } \tilde{\varepsilon}_k^l = \mathbf{t}(l_k) \cdot (\mathbf{p}_k - \boldsymbol{\gamma}(s_k)), \tag{4.16}$$

where $\mathbf{t}(l_k)$ is the unit tangent to γ at l_k. It is clear from figure 4.7 that $\tilde{\varepsilon}_k^c$ approaches ε_k^c, and l_k approaches l_k^* as the lag error is reduced. Therefore, in order to get a good approximation of l_k^* the lag error $\tilde{\varepsilon}^l$ is heavily penalized in the cost function (4.17).

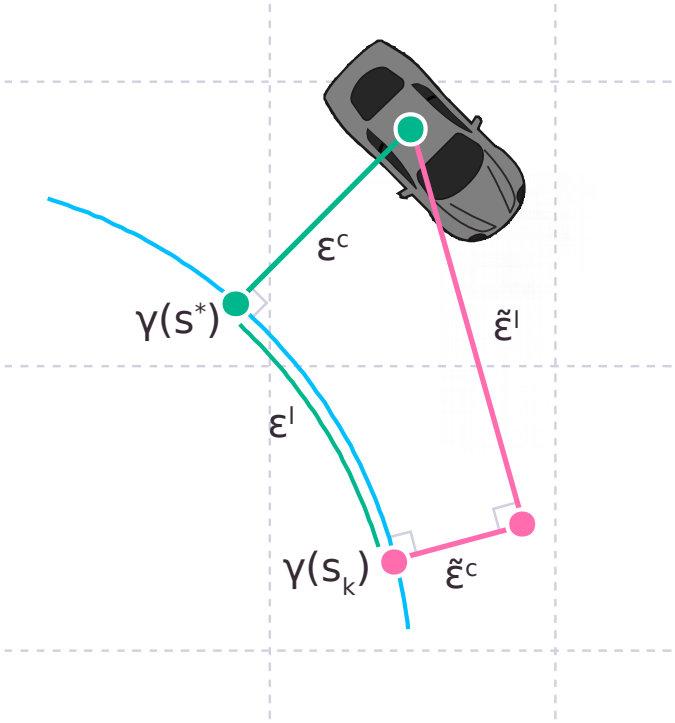

Figure 4.7: Contouring error ε_c, lag error ε_l and their respective approximations $\tilde{\varepsilon}_c$ and $\tilde{\varepsilon}_l$.

Using the approximate contouring and lag errors the stage cost function can be defined

$$\mathcal{L} = ||\tilde{\varepsilon}^l(x_k, y_k, l_k)||_{q_l}^2 + ||\tilde{\varepsilon}^c(x_k, y_k, l_k)||_{q_c}^2 - \alpha \Delta t v_k + \mathcal{L}_{reg}(\Delta \mathbf{u}_k, \Delta v_k). \tag{4.17}$$

The term $-\alpha \Delta t v$ can be thought of as a reward for progressing along the track and the weights q_c and α represent the relative importance of fast progress and accurate path tracking. Finally, the term

$$\mathcal{L}_{reg}(\Delta \mathbf{u}_k, \Delta v_k) = ||\mathbf{u}_k - \mathbf{u}_{k-1}||_{q_u}^2 + ||v_k - v_{k-1}||_{q_v}^2, \tag{4.18}$$

is a slew-rate cost penalizing large changes in control inputs.

4.5 Control Problem

Based on the contouring control framework, we define the autonomous racing problem which incorporates a learned disturbance model, track constraints and input limits:

Problem 2. The Autonomous Racing Problem.

Given: $\mathcal{L}$ – the contouring stage cost function (section 4.4);
$\mathbf{f}$ – a known nominal model of the system's dynamics (section 4.1);
$\mathbf{g}$ – a learned disturbance model (section 4.1.4);
$\mathcal{X}$ – the track constraint set (section 4.3.1);
$\mathcal{U}$ – the control constraint set (section 4.3.2); and
$\hat{\mathbf{x}}_0$ – the initial state of the car.

The autonomous racing problem is defined as the following stochastic non-linear program over a finite horizon of length N:

$$\begin{aligned}
\underset{\mathbf{x}_k, \mathbf{u}_k}{\text{minimize}} \quad & \mathbb{E}\left[\sum_{k=0}^{N-1} \mathcal{L}(\mathbf{x}_k, l_k, \mathbf{u}_k, v_k)\right], \\
\text{subject to} \quad & \mathbf{x}_0 = \hat{\mathbf{x}}_0,\ l_0 = 0, \\
& \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + W\mathbf{g}(\mathbf{x}_k, \mathbf{u}_k), && k = 0, \dots, N-1 \\
& l_{k+1} = l_k + \Delta t v_k, && k = 0, \dots, N-1 \\
& p(\mathbf{x}_{k+1} \in \mathcal{X}(l_{k+1})) > 1 - \varepsilon, && k = 1, \dots, N-1 \\
& \mathbf{u}_k \in \mathcal{U}, && k = 0, \dots, N-1 \\
& 0 \leq v_k \leq v_{\text{max}}, && k = 0, \dots, N-1
\end{aligned} \tag{4.19}$$

where the disturbances are assumed to lie in a subspace spanned by the columns of the matrix W and the track constraints have been reformulated as chance constraints with probability of violation less that ε.

4.6 Approximate Uncertainty Propagation

Since the disturbance term in problem 2 is modeled as a Gaussian process, we need to account for input uncertainty when making predictions. Given a control input and a distribution over our current state, we can predict the next state by computing the marginal:

$$p(\mathbf{x}_{k+1}) = \int p(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k) p(\mathbf{x}_k) d\mathbf{x}_k. \tag{4.20}$$

Unfortunately, this integral is intractable in general and the resulting distribution is non-Gaussian.

To address this issue, Candela et al. (2003) and Girard et al. (2003) propose i) approximating all the distributions over states as Gaussian i.e.

$$\mathbf{x}_k \sim \mathcal{N}(\bar{\mathbf{x}}_k, \Sigma_k), \text{ for all } k \geq 0, \tag{4.21}$$

and ii) applying a first order Taylor expansion to the dynamics. Using the law of iterated expectation and the law of total variance, assumption i) allows us to completely describe the marginal (4.20) by the following mean and variance dynamics:

$$\begin{aligned}
\bar{\mathbf{x}}_{k+1} &:= \mathbb{E}\left[\mathbf{x}_{k+1}\right] = \mathbb{E}\left[\mathbb{E}\left[\mathbf{x}_{k+1} \mid \mathbf{x}_k\right]\right] && \text{(4.22a)}\\
&= \mathbb{E}\left[\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + W\boldsymbol{\mu}(\mathbf{x}_k, \mathbf{u}_k)\right], && \text{(4.22b)}\\
\Sigma_{k+1} &:= \mathrm{Var}(\mathbf{x}_{k+1}) = \mathbb{E}\left[\mathrm{Var}(\mathbf{x}_{k+1} \mid \mathbf{x}_k)\right] + \mathrm{Var}(\mathbb{E}\left[\mathbf{x}_{k+1} \mid \mathbf{x}_k\right]) && \text{(4.22c)}\\
&= \mathbb{E}\left[W\Sigma(\mathbf{x}_k, \mathbf{u}_k)W^\intercal\right] + \mathrm{Var}\left(\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + W\boldsymbol{\mu}(\mathbf{x}_k, \mathbf{u}_k)\right). && \text{(4.22d)}
\end{aligned}$$

Taking a Taylor expansion of $\mathbf{f}$, $\boldsymbol{\mu}$ and Σ about the point $\bar{\mathbf{x}}_k$, equations (4.22b) and (4.22d) can be approximated by:

$$\begin{aligned}
\bar{\mathbf{x}}_{k+1} &\approx \mathbf{f}(\bar{\mathbf{x}}_k, \mathbf{u}_k) + W\boldsymbol{\mu}(\bar{\mathbf{x}}_k, \mathbf{u}_k), && \text{(4.23a)}\\
\Sigma_{k+1} &\approx W\Sigma(\bar{\mathbf{x}}_k, \mathbf{u}_k)W^\intercal + (A_k + WC_k)\Sigma_k(A_k + WC_k)^\intercal\\
&= \begin{bmatrix} A_k & W \end{bmatrix} \begin{bmatrix} \Sigma_k & \Sigma_k^\intercal C_k^\intercal \\ C_k\Sigma_k & \Sigma(\bar{\mathbf{x}}_k, \mathbf{u}_k) \end{bmatrix} \begin{bmatrix} A_k & W \end{bmatrix}^\intercal, && \text{(4.23b)}
\end{aligned}$$

where

$$A_k := \left.\frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\right|_{\bar{\mathbf{x}}_k, \mathbf{u}_k}, \text{ and } C_k := \left.\frac{\partial \boldsymbol{\mu}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\right|_{\bar{\mathbf{x}}_k, \mathbf{u}_k}.$$

Together, the approximations (4.23a) and (4.23b) offer a simple method of propagating uncertainty through the system dynamics. As an alternative, exact moment matching is often used for uncertainty propagation (Deisenroth et al., 2009; Kuss, 2006). Although moment matching is more accurate than the linearization approach, the added computational cost may be prohibitive for real-time applications. For more detail on propagating uncertainty with sparse spectrum Gaussian processes see Pan et al. (2017).

4.6.1 Chance Constraint Reformulation

Making use of the above approximations also allows us to simplify the chance constraint (4.19). In particular, we use Lemma 1 (Hewing et al., 2018) to reformulate the constraint as

$$\tilde{\mathcal{X}}(l_{k+1}) := \left\{\mathbf{x}_{k+1} : \left\|\mathbf{p}_{k+1} - \boldsymbol{\gamma}(l_{k+1})\right\| \leq r/2 - \sqrt{\chi_2^2(1-\varepsilon)\lambda_{\max}(\Sigma_{k+1}^{\mathrm{p}})}\right\}, \qquad \text{(4.24)}$$

where χ_2^2 is the quantile function of the χ-squared distribution with 2 degrees of freedom, $\Sigma_{k+1}^{\mathrm{p}}$ is the marginal variance of the joint distribution of $\mathbf{p}_{k+1}$, and $\lambda_{\max}(\Sigma_{k+1}^{\mathrm{p}})$ is the maximum eigenvalue of $\Sigma_{k+1}^{\mathrm{p}}$.

4.7 Simplified Autonomous Racing Problem

Using approximate uncertainty propagation and the chance constraint reformulation discussed in section 4.6, the autonomous racing problem can be simplified as follows:

Problem 3. The simplified Autonomous Racing Problem.

$$
\begin{aligned}
\underset{\mathbf{x}_k, \mathbf{u}_k}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \mathcal{L}(\bar{\mathbf{x}}_k, l_k, \mathbf{u}_k, v_k), \\
\text{subject to} \quad & \bar{\mathbf{x}}_0 = \hat{\mathbf{x}}_0,\ l_0 = 0,\ \Sigma_0 = 0, \\
& \bar{\mathbf{x}}_{k+1} \text{ according to (4.23a)}, && k = 0, \dots, N-1 \\
& l_{k+1} = l_k + \Delta t v_k, && k = 0, \dots, N-1 \\
& \Sigma_{k+1} \text{ according to (4.23b)}, && k = 0, \dots, N-1 \\
& \bar{\mathbf{x}}_{k+1} \in \tilde{\mathcal{X}}(l_{k+1}) \text{ according to (4.24)}, && k = 0, \dots, N-1 \\
& \mathbf{u}_k \in \mathcal{U}, && k = 0, \dots, N-1 \\
& 0 \leq v_k \leq v_{\text{max}}, && k = 0, \dots, N-1
\end{aligned}
\tag{4.25}
$$

In principle, any non-linear programming method can be used to solve the above problem. However, to meet real-time requirements we combine **sequential convex programming** (SCP) with the structure exploiting solver introduced in section 3.3.

4.8 Sequential Convex Programming

In the SCP framework, problem 3 is repeatedly approximated by a convex program formed by linearizing (4.28a) about a nominal trajectory

$$
\mathbf{z} := [\bar{\mathbf{x}}_0^{\text{guess}}, \mathbf{u}_0^{\text{guess}}, \bar{\mathbf{x}}_1^{\text{guess}}, \mathbf{u}_1^{\text{guess}}, \dots, \bar{\mathbf{x}}_N^{\text{guess}}]\,. \tag{4.26}
$$

The trajectory can then be improved according to the update,

$$
\mathbf{z} \leftarrow (1 - \alpha)\mathbf{z} + \alpha \mathbf{z}^*, \tag{4.27}
$$

where α is the step size and $\mathbf{z}^*$ is the solution to the convex program defined by problem 4. We then re-linearize about the updated trajectory and the process repeats until convergence. The SCP algorithm for model predictive control is shown in algorithm 4.

Algorithm 4: Sequential convex programming for model predictive control

Data: Initial state $\hat{\mathbf{x}}_0$, solution at previous time step $\mathbf{z}$, and step size α

Shift $\mathbf{z}$ backwards according to (4.29)

for $i = 0, 1, 2, \dots$ **do**

  Form the convex program 4

  Solve the convex program to find $\mathbf{z}^*$ (see algorithm 3)

  Update the trajectory: $\mathbf{z} \leftarrow (1 - \alpha)\mathbf{z} + \alpha \mathbf{z}^*$

end

Problem 4. Sequential convex program.

$$
\begin{aligned}
\underset{\mathbf{x}_k, \mathbf{u}_k}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \begin{bmatrix} \mathbf{x}_k \\ l_k \end{bmatrix}^\mathsf{T} H_k \begin{bmatrix} \mathbf{x}_k \\ l_k \end{bmatrix} + \mathbf{f}_k s^\mathsf{T} \begin{bmatrix} \mathbf{x}_k \\ l_k \end{bmatrix} - \alpha \Delta t v_k + \begin{bmatrix} \Delta \mathbf{u}_k \\ \Delta v_k \end{bmatrix}^\mathsf{T} R \begin{bmatrix} \Delta \mathbf{u}_k \\ \Delta v_k \end{bmatrix}, \\
\text{subject to} \quad & \bar{\mathbf{x}}_0 = \hat{\mathbf{x}}_0,\ l_0 = 0, \\
& \bar{\mathbf{x}}_{k+1} = A_k \bar{\mathbf{x}}_k + B_k \mathbf{u}_k + \mathbf{b}_k, \\
& l_{k+1} = l_k + \Delta t v_k, \\
& \bar{\mathbf{x}}_{k+1}^\mathsf{T} Q \bar{\mathbf{x}}_{k+1} + \mathbf{p}^\mathsf{T} \bar{\mathbf{x}}_{k+1} \leq r_{k+1}, \\
& \underline{\mathbf{u}} \leq \mathbf{u}_k \leq \overline{\mathbf{u}}, \\
& 0 \leq v_k \leq v_{\max}.
\end{aligned} \tag{4.28a}
$$

The cost function is a quadratic approximation of the contouring cost (4.18) and the dynamics are a linearization of (4.23a). Specifically, for the cost function, we have:

$$
\begin{aligned}
H_k := {} & q_l \nabla \tilde{\varepsilon}^l(\bar{\mathbf{x}}_k^{\text{guess}}, l_k^{\text{guess}}) \nabla \tilde{\varepsilon}^l(\bar{\mathbf{x}}_k^{\text{guess}}, l_k^{\text{guess}})^\mathsf{T} \\
& + q_c \nabla \tilde{\varepsilon}^c(\bar{\mathbf{x}}_k^{\text{guess}}, l_k^{\text{guess}}) \nabla \tilde{\varepsilon}^c(\bar{\mathbf{x}}_k^{\text{guess}}, l_k^{\text{guess}})^\mathsf{T},
\end{aligned} \tag{4.28b}
$$

$$
\mathbf{f}_k := 2 \begin{bmatrix} \nabla \tilde{\varepsilon}^l(\bar{\mathbf{x}}_k^{\text{guess}}, l_k^{\text{guess}})^\mathsf{T} \\ \nabla \tilde{\varepsilon}^c(\bar{\mathbf{x}}_k^{\text{guess}}, l_k^{\text{guess}})^\mathsf{T} \end{bmatrix} \begin{bmatrix} q_l & 0 \\ 0 & q_c \end{bmatrix} \begin{bmatrix} \tilde{\varepsilon}^l(\bar{\mathbf{x}}_k^{\text{guess}}, l_k^{\text{guess}}) \\ \tilde{\varepsilon}^c(\bar{\mathbf{x}}_k^{\text{guess}}, l_k^{\text{guess}}) \end{bmatrix} - 2 H_k \begin{bmatrix} \bar{\mathbf{x}}_k^{\text{guess}} \\ l_k^{\text{guess}} \end{bmatrix}, \tag{4.28c}
$$

and for the linearized dynamics:

$$
A_k := \left. \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \right|_{\bar{\mathbf{x}}_k^{\text{guess}}, \mathbf{u}_k^{\text{guess}}} + W \left. \frac{\partial \boldsymbol{\mu}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}} \right|_{\bar{\mathbf{x}}_k^{\text{guess}}, \mathbf{u}_k^{\text{guess}}}, \tag{4.28d}
$$

$$
B_k := \left. \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \right|_{\bar{\mathbf{x}}_k^{\text{guess}}, \mathbf{u}_k^{\text{guess}}} + W \left. \frac{\partial \boldsymbol{\mu}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}} \right|_{\bar{\mathbf{x}}_k^{\text{guess}}, \mathbf{u}_k^{\text{guess}}}, \tag{4.28e}
$$

$$
\mathbf{b}_k := \mathbf{f}(\bar{\mathbf{x}}_k^{\text{guess}}, \mathbf{u}_k^{\text{guess}}) + W \boldsymbol{\mu}(\bar{\mathbf{x}}_k^{\text{guess}}, \mathbf{u}_k^{\text{guess}}) - A_k \bar{\mathbf{x}}_k^{\text{guess}} - B_k \mathbf{u}_k^{\text{guess}}. \tag{4.28f}
$$

To reduce computational costs the variances can be evaluated using the nominal trajectory and kept constant over the optimization. Variances are then reevaluated using the updated trajectory for the next optimization. Using this idea we can exclude the variance dynamics from problem 4 while still making use of model uncertainty in the formulation of the constraints. For model predictive control problems we usually have a good guess for the nominal trajectory based on the solution at the previous time step. As a result, fixing the variance dynamics over an optimization step should not significantly affect the accuracy of the solution. The previous trajectory is shifted backwards to warm start the optimization:

$$
\bar{\mathbf{x}}_k^{\text{guess}} \leftarrow \bar{\mathbf{x}}_{k+1}^*, \quad k = 0, \ldots, N-1 \tag{4.29a}
$$

$$
\mathbf{u}_k^{\text{guess}} \leftarrow \mathbf{u}_{k+1}^*, \quad k = 0, \ldots, N-2 \tag{4.29b}
$$

$$
\mathbf{u}_{N-1}^{\text{guess}} \leftarrow \mathbf{u}_{N-2}^{\text{guess}}, \quad \bar{\mathbf{x}}_N \leftarrow \mathbf{f}(\bar{\mathbf{x}}_{N-1}^{\text{guess}}, \mathbf{u}_{N-1}^{\text{guess}}) + W \boldsymbol{\mu}(\bar{\mathbf{x}}_{N-1}^{\text{guess}}, \mathbf{u}_{N-1}^{\text{guess}}). \tag{4.29c}
$$

Given a good initial guess, it is often unnecessary to iterate to convergence before a reasonable improvement is found. In real-time settings this is important because we need to maintain a balance between efficiency and accuracy. An example of this trade-off can be seen in figure 4.8.

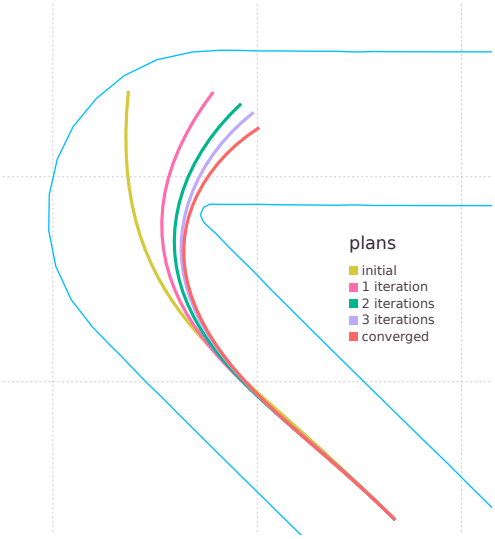

Figure 4.8: An example of planned trajectories in the autonomous racing task. Note how early termination yields a solution trajectory that is still near-optimal over a short horizon.

4.9 Gaussian Process Receeding Horizon Control

Finally, putting all the pieces together gives us the complete GP-RHC algorithm shown in algorithm 5:

Algorithm 5: The complete GP-RHC algorithm

Data: A stage-cost function $\mathcal{L}$, a known nominal model of the dynamics $\mathbf{f}$, state and control constraints $\mathcal{X}$ and $\mathcal{U}$, number of sparse spectrum frequencies D, planning horizon N, step size α, an initial nominal trajectory $\mathbf{z}$, and an initial set of training data.

Fit a sparse spectrum GP to the initial set of training data (see algorithm 1);
foreach *episode* **do**
  foreach *step* **do**
    Measure the current state of the system $\hat{\mathbf{x}}_0$;
    Apply sequential convex programming (see algorithm 4);
    Apply the control $\mathbf{u}_0$ to the system;
    Optionally update dynamics model (see (2.14));
    Using the nominal model predict the state of the system given the input $\mathbf{u}_0$;
    Store the error between the current state and predicted state;
  end
  Retrain the dynamics model on all available data;
end

Chapter 5

Experiments

Although we focused on autonomous racing in chapter 4, GP-RHC is applicable to a broad range of control problems. In this chapter we evaluate GP-RHC on three different tasks: pendulum swing up (section 5.1), cartpole swing up (section 5.2), and autonomous racing (section 5.3). The experiments are designed to assess the feasibility and performance of GP-RHC. In particular, we aim to answer the following questions:

i) Can GP-RHC learn an accurate dynamics model for predictive control?
ii) Do incremental updates improve sample efficiency and performance?
iii) Is GP-RHC competitive with state-of-the-art learning-based methods like (Deisenroth and Rasmussen, 2011)?
iv) Can GP-RHC make effective use of known nominal dynamics by learning to correct for unmodeled disturbances?
v) How well does GP-RHC handle non-linear constraints?

In all the experiments GP-RHC was able to run in real-time with appropriate choices for the planning horizon and the number of frequencies in the sparse spectrum approximation.

5.1 Pendulum Swing Up

For our first experiment we apply GP-RHC to a simple pendulum of mass $m = 1kg$ and length $l = 1m$. By applying a torque u, the objective is to swing up and balance the pendulum in the inverted position (see figure 5.1). The system state $[\theta, \omega]$ is described by the pendulum's angle θ and its angular velocity ω. The torque u is limited to the range$[-2, 2]Nm$ and is applied to the pendulum at intervals of $0.1s$. A full description of the system dynamics can be found in Deisenroth (2010).

The pendulum is simulated using a 4th order Runge-Kutta method, adding Gaussian measurement noise to the result. Each episode is 4 seconds long and the cost function is given by a least squares objective penalizing the distance between the tip of the pendulum

and the set point $[0, l]$:

$$\texttt{cost}(\mathbf{x}) := l^2 \sin^2 \theta + (l + l\cos\theta)^2. \tag{5.1}$$

We mark an episode as successful if the tip of the pendulum remains within $6cm$ of the set point for the last 0.5 seconds of the episode.

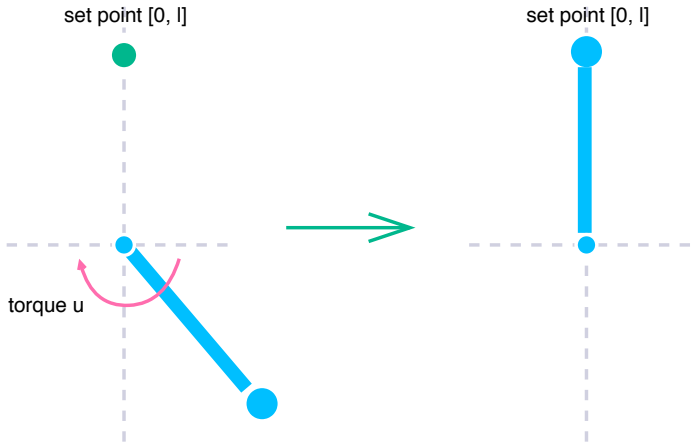

Figure 5.1: By applying a torque u the goal is to swing up and balance the pendulum.

For this experiment we compare GP-RHC against PILCO and a baseline controller using the ground truth dynamics. We assume that no nominal model is given and simply set $\mathbf{f}$ in (4.3) to the identity function. For the sparse spectrum approximation, 50 sample frequencies are drawn and a planning horizon of 20 time steps is used. Choosing the number of sample frequencies essentially involves a trade-off between model accuracy and computational costs. The pendulum swing up task has relatively simple dynamics and only requires a few sample frequencies to learn a sufficiently accurate model. We found no difference in performance for a range of sample frequencies between 20 and 200.

Each trial is initialized with 40 data points collected using random control inputs. After each episode the GP dynamics model is retrained on all the preceding data using a random restart to avoid occasional poor minima. PILCO can potentially have problems with least squares costs (Deisenroth, 2010) so we used (the preferred) saturating cost function and post-processed the results.

Figure 5.2 shows that GP-RHC matches the performance of the ground truth controller after 2 to 3 episodes and is competitive with PILCO in terms of sample efficiency and success rate. However, PILCO is able to balance the pendulum earlier than GP-RHC and performs slightly better in this task. Note that the saturating cost function (which incorporates model uncertainty) used by PILCO may account for the difference in performance. An interesting direction for future work would be to incorporate a saturating cost function into GP-RHC.

Finally, comparing the two variants of GP-RHC shows that incremental updates improve efficiency and reduce variance. Overall GP-RHC with updates performs more con-

sistently and is less sensitive to the initial training data. As expected, the asymptotic performance of the two variants are similar, however GP-RHC with updates achieves a success rate of about 75% with 50% less data. Figure 5.3 shows an example rollout using GP-RHC after completing training.

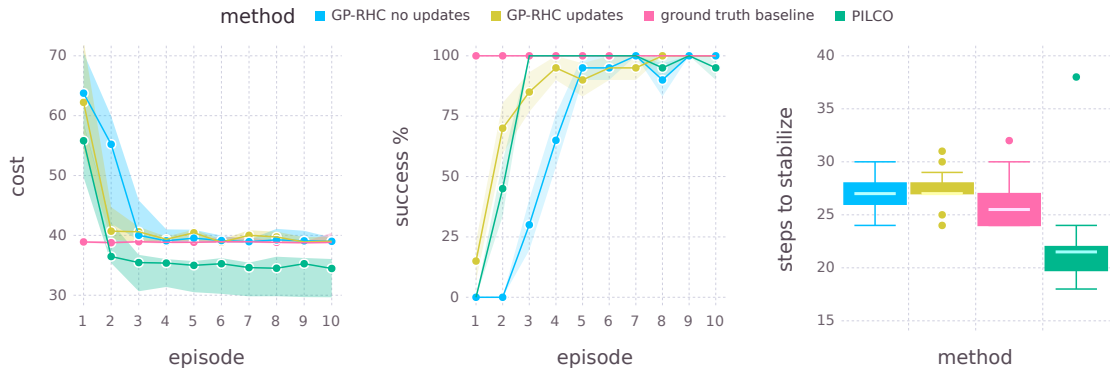

Figure 5.2: Comparison of GP-RHC, PILCO, and a ground truth controller. Results are aggregated over 20 trials. **Left:** Median cost of the pendulum swing up task, bounds on the confidence envelope represent the interquartile range. GP-RHC is able to quickly learn a model of the dynamics and matches the performance of the ground truth controller after 2 to 3 episodes. **Middle:** Success rate, the confidence envelope gives the standard error. Online updates improve data efficiency and performance. **Right:** Number of time steps until the pendulum is stabilized during the final episode. Due to differences in the cost function PILCO is able to stabilize the pendulum earlier, leading to lower costs.

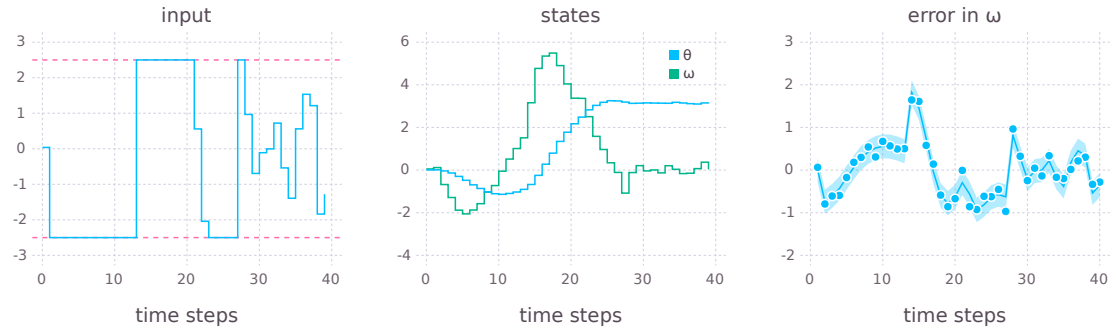

Figure 5.3: Example trajectory from GP-RHC. **Left and Middle:** Control signal and state dynamics. The input limits are shown by the dashed pink lines. Initially GP-RHC rotates the pendulum clockwise. At 1.2 seconds the torque switches direction, swinging the pendulum up and balancing it at $\theta = \pi$. Towards the end of the episode, fluctuations in the control signal towards correct for measurement noise. **Right:** Learned error in the angular velocity ω. The GP's predictive mean is indicated by the blue line while the shaded region represents the 95% confidence envelope.

5.2 Cartpole Swing Up

Cartpole experiments are a common benchmark in both reinforcement learning and optimal control (Furuta et al., 1991; Kober et al., 2013). The basic set-up consists of a cart, with an attached pendulum, running along a track. By applying horizontal forces to the cart, the goal is to swing the pendulum up and balance it at the center of the track (see figure 5.4). This is a difficult control problem with fairly non-linear dynamics. Additionally, a long planning horizon is required because the cart must be pushed back and forth in order to develop enough momentum to swing the pendulum up.

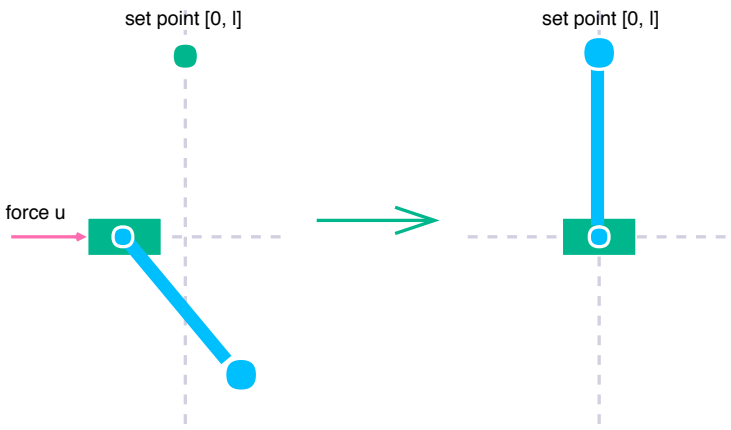

Figure 5.4: By applying a force u to the cart, the goal is to swing the pendulum up and balance it at the center of the track.

The state of the system, $\mathbf{x} = [x, v, \theta, \omega]$, is described by the position of the cart, the velocity of the cart, the angle of the pendulum and its angular velocity. A horizontal force u in the range of -10N to 10N can be applied to the cart at time steps of $0.1s$. For a full description of the system dynamics see Deisenroth (2010).

Again, we compare GP-RHC against PILCO and the ground truth controller. Additionally, given a nominal model of the dynamics, we demonstrate that GP-RHC can learn to correct for unmodelled disturbances. The nominal part of the dynamics is given by randomly perturbing the system parameters by up to $\pm 10\%$ of their original value. GP-RHC must learn to account for process noise and mismatch in the dynamics. Disturbance models are only learned for the cart's velocity v and the pendulum's angular velocity ω.

Trials are initialized with 40 data points collected using a uniform random controller. We sample 200 frequencies for the sparse spectrum approximation and use a planning horizon of 20 time steps. Each episode is 4 seconds long and the cost function is given by a least squares objective penalizing the distance between the tip of the pendulum and the set point $[0, l]$:

$$\texttt{cost}(\mathbf{x}) := (x + l \sin\theta)^2 + (l + l\cos\theta)^2. \tag{5.2}$$

Again, an episode is considered successful if the tip of the pendulum remains within $6cm$

of the set point for the last 0.5 seconds of the episode.

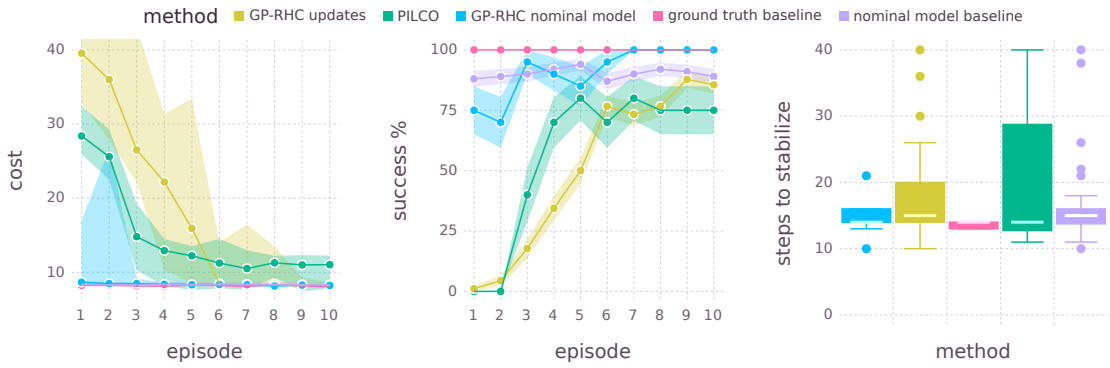

Figure 5.5: Comparison of GP-RHC, PILCO, and a ground truth controller. Results are aggregated over 20 trials. **Left:** Median cost of the cartpole swing up task, bounds of the confidence envelope represent the interquartile range. GP-RHC approximately matches the performance of the ground truth controller after about 6 episodes. **Middle:** Success rate, the confidence envelope gives the standard error. GP-RHC is able to learn a disturbance model to compensate for model mismatch. **Right:** Number of time steps until the pendulum is stabilized during the final episode.

Figure 5.5 shows that GP-RHC is slightly less efficient than PILCO for cartpole swing up but achieves a better final success rate. GP-RHC can also take advantage of known nominal dynamics — learning to compensate for model mismatch. With a learned disturbance model, GP-RHC is able to match the performance of the ground truth controller and consistently stabilize the pendulum. Figure 5.6 shows an example rollout using GP-RHC after completing training.

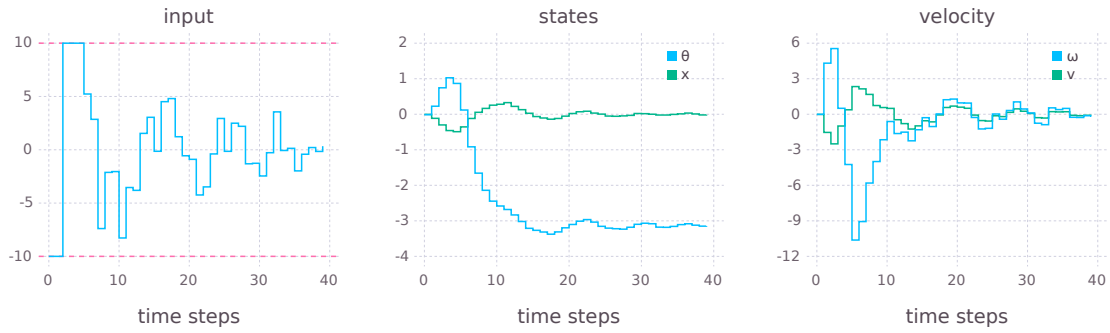

Figure 5.6: Example trajectory from GP-RHC. **Left:** Control signal, the input limits are shown by the dashed pink lines. **Middle:** State dynamics, the pendulum is balanced at an angle of $\theta = -\pi$. **Right:** Velocity dynamics. GP-RHC pushes the cart to the left then rapidly switches direction to swing the pendulum up and balance it. Towards the end of the episode, fluctuations in the control signal correct for measurement noise.

Finally, we evaluate the effect that the number of sample frequencies has on performance. First, we collect a small dataset of dynamics using the ground truth cartpole controller. We train SSGPs to predict changes in the angle of the pendulum given the control input and current state of the system. The dataset is split into 350 training points and 50 test points. The left panel of figure 5.7 reports the normalized mean square error as a function of the number of sample frequencies used in the sparse spectrum approximation. Additionally, we apply GP-RHC to the cartpole task over a range of sample frequencies. The center and right panels of figure 5.7 show that using a larger number of sample frequencies improves the success rate and reduces the steps to stabilize. However, there is a trade-off between performance and computational costs.

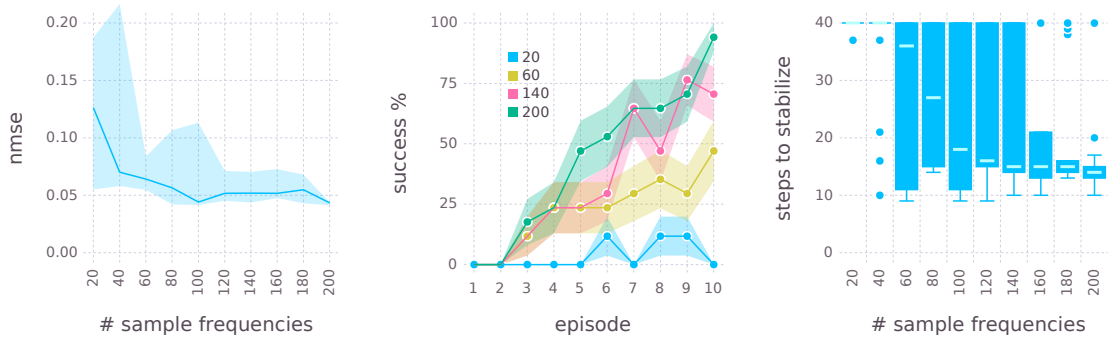

Figure 5.7: Effect of number of sample frequencies on performance. **Left:** Median normalized mean square error as a function of number of sample frequencies. Bounds of the confidence envelope represent the interquartile range. **Middle:** Success rate for a selection of sample frequencies. The confidence envelope gives the standard error. Increasing the number of sample frequencies improves performance. **Right:** Number of time steps until the pendulum is stabilized during the final episode as a function of the number of sample frequencies.

5.3 Autonomous Racing of 1:43 Scale Cars

In our final experiment we apply GP-RHC to the autonomous racing problem discussed in chapter 4. For the nominal dynamics we randomly perturb the parameters in table 4.1 by up to $\pm 15\%$ of their original value. The car is then driven around the track for a single lap and the errors between the nominal and true dynamics are recorded. This dataset is split into 350 training points and 100 test points. We learn disturbance models for the longitudinal velocity v_x, lateral velocity v_y, and yaw rate ω. For the yaw rate model, figure 5.8 reports normalized mean square error as a function of the number of sample frequencies. For our final models we use 150 sample frequencies for a balance of accuracy and computational cost. Predicted deviations from the nominal dynamics are shown in figure 5.9.

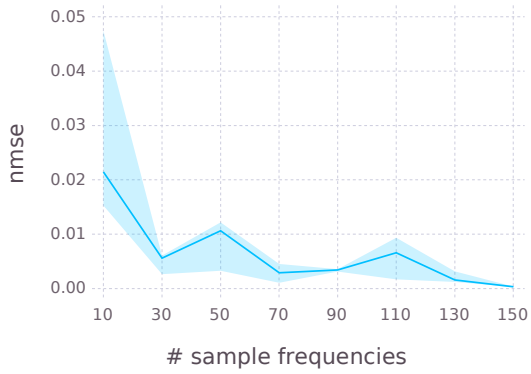

Figure 5.8: Normalized mean square error as a function of the number of basis functions used in the sparse spectrum approximation.

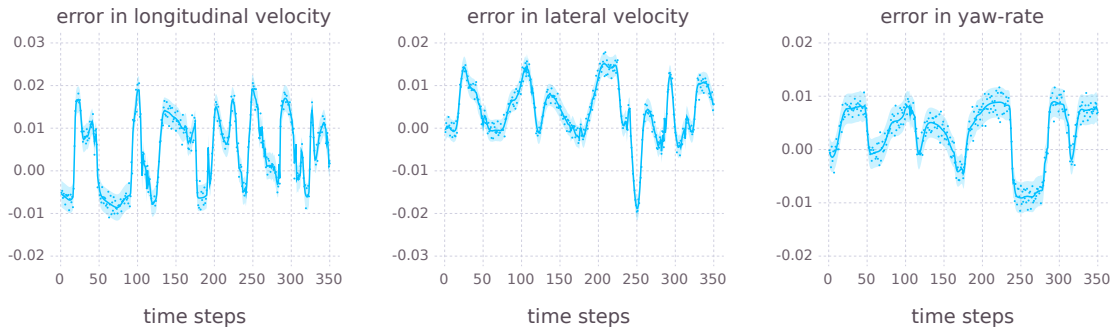

Figure 5.9: Learned disturbance models for the longitudinal velocity v_x, lateral velocity v_y, and yaw rate ω.

We use a planning horizon of 40 steps and simulate the system using a sampling time of $\Delta t = 20ms$. Since the optimal racing line operates at the limits of the constraints we use a soft constraint formulation (see example 3 in chapter 3) to avoid infeasibility problems.

In figure 5.10 we compare the lap times of GP-RHC, the nominal baseline, and the ground truth controller. Using a learned disturbance model, GP-RHC significantly improves on the nominal baseline achieving an average lap time of $9.3s$ (a relative improvement of about 8.65%). This represents a dramatic saving considering the high speeds and small length scales involved in the problem. In fact, a back-of-the-envelope calculation shows that naively scaling up to full size gives an average speed increase of $27.16km/h$ over a $811m$ track.

Figure 5.11 shows the racing lines for GP-RHC and the nominal baseline over 10 different trials. Due to model mismatch the nominal baseline cannot guarantee safety and occasionally collides with the track boundary. This is evident around sharp corners and the

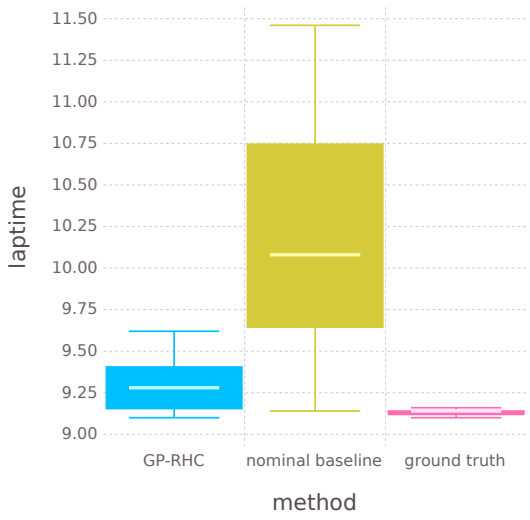

Figure 5.10: Lap times of GP-RHC, the nominal baseline and the ground truth controller. Results are aggregated over 10 runs.

chicane at the bottom left of the track. GP-RHC learns to correct for errors in the nominal model, resulting in fast, safe, and consistent racing lines. Figure 5.12 shows the control inputs and lateral velocity of the car. The lateral velocity is non-zero over large portions of the lap, showing that GP-RHC can exploit drift dynamics and operate at the limits of friction.

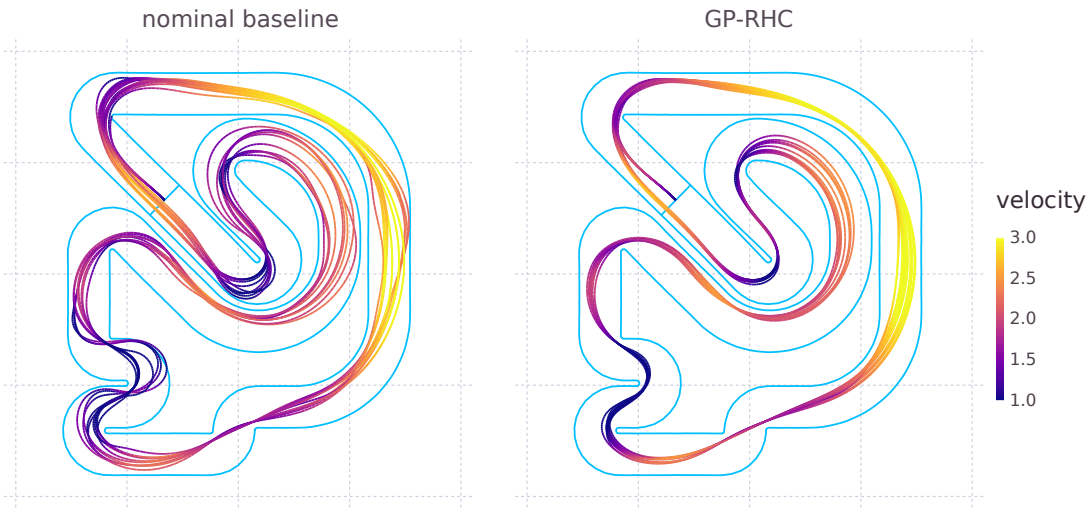

Figure 5.11: Control inputs and lateral velocity of the car over a single lap around the racing track.

Finally, we demonstrate that GP-RHC can transfer to new tasks without any further learning. To do this we adapt the autonomous racing problem, adding a set of static obstacles to the track. Obstacles are included in the MPC constraints by manually adjusting the track boundaries following our work in Van Niekerk et al. (2017). GP-RHC can implicitly take advantage of data collected in the autonomous racing task by simply reusing the learned disturbance model. With no additional learning GP-RHC is able to avoid collisions by planning around the obstacles (see figure 5.13).

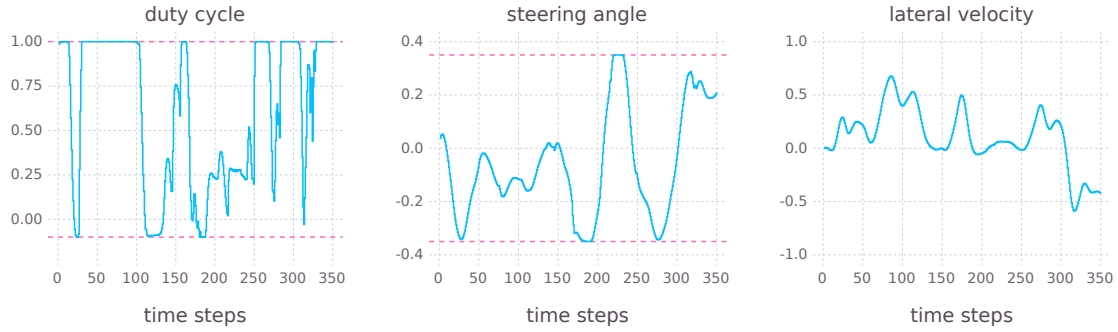

Figure 5.12: Racing lines of GP-RHC and the nominal baseline over 10 trials. The color scale displays the longitudinal velocity of the car.

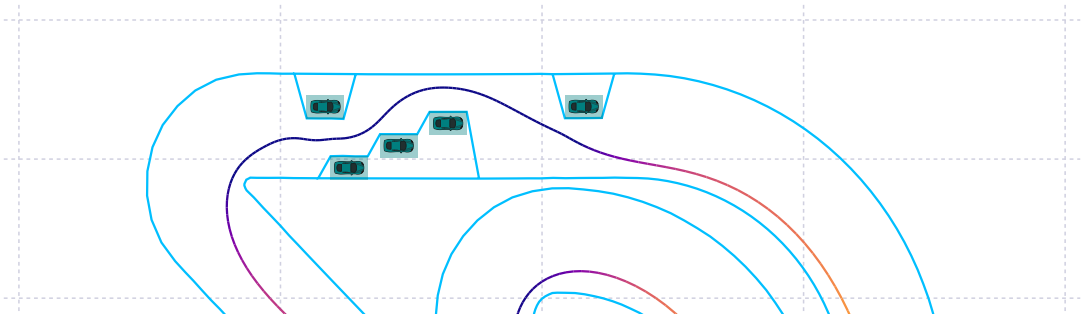

Figure 5.13: Obstacle avoidance problem. GP-RHC is able to avoid collisions by planning around the obstacles.

Chapter 6

Conclusion and Future Work

In this thesis we introduce GP-RHC for learning-based control. By combining data-efficient sparse spectrum Gaussian processes with model predictive control, GP-RHC is able to learn and plan in real-time. We show that incorporating online updates results in faster learning and more reliable performance. We test our method on a complex autonomous racing task, showing that unmodelled disturbance can be learned from limited data. By learning directly from data, GP-RHC is able to improve both lap times and constraint handling — an important feature for safety critical applications. Finally, learned models can be transferred to new tasks with no additional training. GP-RHC provides a promising approach to deploying learning-based control on real-world systems.

In future work it would be interesting to extend GP-RHC and address some of its shortcomings. Firstly, Deisenroth (2010) have noted that least squares costs do not work well with probabilistic dynamics models. To address this issue they propose the use of saturating cost functions. With relatively minor modifications to the optimization methods discussed in chapter 3 saturating costs could be incorporated into GP-RHC. Secondly, in order to meet real-time requirements, we have made a number of trade-offs between accuracy and computational costs. Specifically, we excluded the variance dynamics from the SCP optimization (see section 4.8) and we chose to use a sparse spectrum approximation instead of full Gaussian processes to model the dynamics (see section 2.4). It would be interesting to quantify the effect of these decisions through a set of ablation studies.

Bibliography

Olov Andersson, Fredrik Heintz, and Patrick Doherty. Model-based reinforcement learning in continuous environments using real-time constrained optimization. In *Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI15)*, 2015.

Joschka Boedecker, Jost Tobias Springenberg, Jan Wülfing, and Martin Riedmiller. Approximate real-time optimal control based on sparse gaussian process models. In *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 1–8. IEEE, 2014.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Daniela Calvetti, G Golub, W Gragg, and Lothar Reichel. Computation of gauss-kronrod quadrature rules. *Mathematics of computation*, 69(231):1035–1052, 2000.

Joaquin Quinonero Candela, Agathe Girard, Jan Larsen, and Carl Edward Rasmussen. Propagation of uncertainty in bayesian kernel models-application to multiple-step ahead forecasting. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 2, pages II–701. IEEE, 2003.

Charles R Cutler and Brian L Ramaker. Dynamic matrix control: A computer control algorithm. In *joint automatic control conference*, number 17, page 72, 1980.

Joseph Czyzyk, Sanjay Mehrotra, Michael Wagner, and Stephen J Wright. Pcx: An interior-point code for linear programming. *Optimization Methods and Software*, 11(1-4):397–430, 1999.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Marc Peter Deisenroth. *Efficient reinforcement learning using Gaussian processes*. KIT Scientific Publishing, 2010.

Marc Peter Deisenroth, Marco F Huber, and Uwe D Hanebeck. Analytic moment-based gaussian process filtering. In *Proceedings of the 26th annual international conference on machine learning*, pages 225–232. ACM, 2009.

S Di Cairano, H Park, and I Kolmanovsky. Model predictive control approach for guidance of spacecraft rendezvous and proximity maneuvering. *International Journal of Robust and Nonlinear Control*, 22(12):1398–1427, 2012.

Alexander Domahidi, Aldo U Zgraggen, Melanie N Zeilinger, Manfred Morari, and Colin N Jones. Efficient interior point methods for multistage problems arising in receding horizon control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 668–674. IEEE, 2012.

Tom Erez, Kendall Lowrey, Yuval Tassa, Vikash Kumar, Svetoslav Kolev, and Emanuel Todorov. An integrated system for real-time model predictive control of humanoid robots. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 292–299. IEEE, 2013.

Gianluca Frison, Henrik Brandenborg Sørensen, Bernd Dammann, and John Bagterp Jørgensen. High-performance small-scale solvers for linear model predictive control. In *Control Conference (ECC), 2014 European*, pages 128–133. IEEE, 2014.

Katsuhisa Furuta, Masaki Yamakita, and Seiichi Kobayashi. Swing up control of inverted pendulum. In *Proceedings IECON'91: 1991 International Conference on Industrial Electronics, Control and Instrumentation*, pages 2193–2198. IEEE, 1991.

Arjan Gijsberts and Giorgio Metta. Real-time model learning using incremental sparse spectrum gaussian process regression. *Neural Networks*, 41:59–69, 2013.

Agathe Girard, Carl Edward Rasmussen, Joaquin Quinonero Candela, and Roderick Murray-Smith. Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting. In *Advances in neural information processing systems*, pages 545–552, 2003.

Lukas Hewing, Alexander Liniger, and Melanie Nicole Zeilinger. Cautious nmpc with gaussian process dynamics for autonomous miniature race cars. In *European Control Conference (ECC 2018)*, 2018.

Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. Acado toolkitan open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

Sanket Kamthe and Marc Peter Deisenroth. Data-efficient reinforcement learning with probabilistic model predictive control. *arXiv preprint arXiv:1706.06491*, 2017.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 2013.

Juš Kocijan, Roderick Murray-Smith, Carl Edward Rasmussen, and Agathe Girard. Gaussian process model based predictive control. In *Proceedings of the 2004 American Control Conference*, volume 3, pages 2214–2219. IEEE, 2004.

D Kouzoupis, A Zanelli, Helfried Peyrl, and Hans Joachim Ferreau. Towards proper assessment of qp algorithms for embedded model predictive control. In *Control Conference (ECC), 2015 European*, pages 2609–2616. IEEE, 2015.

Malte Kuss. *Gaussian process models for robust regression, classification, and reinforcement learning*. PhD thesis, Technische Universität, 2006.

Denise Lam, Chris Manzie, and Malcolm Good. Model predictive contouring control. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 6137–6142. IEEE, 2010.

Miguel Lázaro-Gredilla, Joaquin Quiñonero-Candela, Carl Edward Rasmussen, and Aníbal R Figueiras-Vidal. Sparse spectrum gaussian process regression. *The Journal of Machine Learning Research*, 11:1865–1881, 2010.

Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods*, 36 (5):628–647, 2015.

David G Luenberger, Yinyu Ye, et al. *Linear and nonlinear programming*, volume 2. Springer, 1984.

Jacob Mattingley and Stephen Boyd. Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, 2(4):575–601, 1992.

Hans Pacejka. *Tire and vehicle dynamics*. Elsevier, 2005.

Yunpeng Pan and Evangelos Theodorou. Probabilistic differential dynamic programming. In *Advances in Neural Information Processing Systems*, pages 1907–1915, 2014.

Yunpeng Pan, Xinyan Yan, Evangelos A Theodorou, and Byron Boots. Prediction under uncertainty in sparse spectrum gaussian processes with applications to filtering and control. In *International Conference on Machine Learning*, pages 2760–2768, 2017.

Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec): 1939–1959, 2005.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

Carl Edward Rasmussen and Christopher K I Williams. Gaussian processes for machine learning. 2006.

Ugo Rosolia, Ashwin Carvalho, and Francesco Borrelli. Autonomous racing using learning model predictive control. In *2017 American Control Conference (ACC)*, pages 5115–5120. IEEE, 2017.

Benjamin Van Niekerk, Andreas Damianou, and Benjamin Rosman. Online constrained model-based reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence*, 2017.

Robert J Vanderbei. Loqo: An interior point code for quadratic programming. *Optimization methods and software*, 11(1-4):451–484, 1999.

Sergio Vazquez, Jose Leon, Leopoldo Franquelo, Jose Rodriguez, Hector A Young, Abraham Marquez, and Pericle Zanchetta. Model predictive control: A review of its applications in power electronics. *IEEE Industrial Electronics Magazine*, 8(1):16–31, 2014.

Robin Verschueren, Stijn De Bruyne, Mario Zanon, Janick V Frasch, and Moritz Diehl. Towards time-optimal race car driving using nonlinear mpc in real-time. In *53rd IEEE conference on decision and control*, pages 2505–2510. IEEE, 2014.

Christoph Voser, Rami Y Hindiyeh, and J Christian Gerdes. Analysis and control of high sideslip manoeuvres. *Vehicle System Dynamics*, 48(S1):317–336, 2010.

Hongling Wang, Joseph Kearney, and Kendall Atkinson. Arc-length parameterized spline curves for real-time simulation. In *In in Proc. 5th International Conference on Curves and Surfaces*, pages 387–396, 2002a.

Hongling Wang, Joseph Kearney, and Kendall Atkinson. Robust and efficient computation of the closest point on a spline curve. In *Proceedings of the 5th International Conference on Curves and Surfaces*, pages 397–406, 2002b.

Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IEEE Transactions on control systems technology*, 18(2):267, 2010.

Stephen J Wright. *Primal-dual interior-point methods*, volume 54. Siam, 1997.