# Online Constrained Model-based Reinforcement Learning

**Benjamin van Niekerk**
School of Computer Science
University of the Witwatersrand
South Africa

**Andreas Damianou***
Amazon.com
Cambridge, UK

**Benjamin Rosman**
Council for Scientific and Industrial Research,
and School of Computer Science
University of the Witwatersrand
South Africa

## Abstract

Applying reinforcement learning to robotic systems poses a number of challenging problems. A key requirement is the ability to handle continuous state and action spaces while remaining within a limited time and resource budget. Additionally, for safe operation, the system must make robust decisions under hard constraints. To address these challenges, we propose a model based approach that combines Gaussian Process regression and Receding Horizon Control. Using sparse spectrum Gaussian Processes, we extend previous work by updating the dynamics model incrementally from a stream of sensory data. This results in an agent that can learn and plan in real-time under non-linear constraints. We test our approach on a cart pole swing-up environment and demonstrate the benefits of online learning on an autonomous racing task. The environment's dynamics are learned from limited training data and can be reused in new task instances without retraining.

## 1 INTRODUCTION

Reinforcement learning has become an important approach to the planning and control of autonomous agents in complex environments. However, recent interest in reinforcement learning is yet to be reflected in robotics applications; possibly due to their specific challenges.

In robotic systems, reinforcement learning methods must deal with continuous, and potentially high-dimensional, state and control spaces. For example, the dynamics of autonomous vehicles are most naturally described in terms of continuous variables like position, velocity, orientation and steering angle. Data efficiency is also critical since collecting experience, or evaluating control policies, with a robot in the loop can be costly and time consuming. Poorly modeled dynamics, due to a slow learning rate, may result in unstable trajectories and dangerous collisions. The system also needs to handle physical constraints — from joint and actuator limitations to the boundaries defined by obstacles. Finally, in fast-paced applications rapid decisions, requiring real-time planning, must be made.

In this paper we propose a method which tackles all of the above challenges simultaneously. Our method achieves this through the marriage of two key components:

1. a learned dynamics model based on sparse spectrum Gaussian processes (GPs), and
2. a planner based on receding horizon control (RHC), and the structure exploiting interior point solver FORCES (Domahidi and Jerez, 2014).

Sparse spectrum GPs allow us to update the learned model online and the RHC framework provides the ability plan in real-time while naturally handling constraints. This means that we can learn and plan online in constrained environments where data collection is expensive. We believe that these are important features for real world applications of reinforcement learning, particularly in safety critical systems. Our proposed method will be referred to as Gaussian Process-Receding Horizon Control (GP-RHC hereafter).

The remainder of the paper is structured as follows. In section 2 we provide an overview of related approaches in model-based reinforcement learning. The receding horizon control framework is presented in section 3. This is followed, in section 4, by a discussion on the application of Gaussian process regression to model learning.

*Work done while this author was at the University of Sheffield.

In section 5 we apply GP-RHC to a cart pole swing-up environment and a challenging autonomous racing task. The results demonstrate that (a) models can be learned quickly from limited data, (b) complex non-linear constraints can be handled in real-time, and (c) online updates improve the rate of learning and result in more consistent performance.

## 2 RELATED WORK

GP-RHC falls into the class of model-based reinforcement learning methods. These are generally the methods of choice in robotics primarily due to their impressive data efficiency (Kober et al., 2013). Model-based approaches can be broadly classified according to (a) how the dynamics model is learned and, (b) the choice of planner.

PILCO (Deisenroth and Rasmussen, 2011), for example, combines policy search for planning with a Gaussian process model of the dynamics. The policy search relies on analytic gradients of closed form solutions to the long term expected cost. This requires the cost function and policy to take specific functional forms, making it difficult or impossible to incorporate general constraints. In a similar vein, Kim et al. (2004) use policy search with locally weighted linear regression to learn a controller for helicopter flight.

As an alternative, trajectory optimization based on differential dynamic programming is often used for planning. Methods such as PDDP (Pan and Theodorou, 2014) and AGP-iLQR (Boedecker et al., 2014) make use of this idea by combining dynamics models learned by locally weighted projection regression with either an iterative linear quadratic regulator or Gaussian as the planner. These planners can take simple box constraints into account but cannot handle general non-linear constraints.

Our work is most closely related to the RL-RCO method (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 agent 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, rendering GP-RHC overall more applicable to realistic scenarios.

## 3 RECEDING HORIZON CONTROL

In this paper, we consider an agent operating in a continuous environment described by a set of differential equations, $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$, where $\mathbf{x} \in \mathbb{R}^n$ represents the agent's state and $\mathbf{u} \in \mathbb{R}^m$ the control signal. The objective of the agent is to select a control $\mathbf{u}$ to minimize a cost function (1a), while conforming to the system dynamics, and additional constraints (1b)-(1d).

We address this overall problem using receding horizon control where the idea is to plan over a finite horizon by iteratively solving an optimization problem. Given a measurement of the current state $\hat{\mathbf{x}}_0$, a trajectory through the state-control space is calculated, and the first step of the control signal is applied to the system. The horizon is then shifted forward and the process repeats. Formally, at each time step the agent must minimize the cost function

$$J(\mathbf{x}_0) = h(\mathbf{x}(t_0 + T)) + \int_{t_0}^{t_0+T} \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t))dt, \quad \text{(1a)}$$

where $t_0$ is the current time, $T$ denotes the length of the planning horizon, $\mathcal{L}$ is an intermediate cost function and $h$ is a terminal cost function. In order to ensure that the trajectory is feasible, the optimization is subject to the constraints:

$$\begin{aligned}
&\mathbf{x}(t_0) = \hat{\mathbf{x}}_0, \\
&\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \\
&\underline{\mathbf{u}} \leq \mathbf{u}(t) \leq \overline{\mathbf{u}}, && \text{(1b)} \\
&\underline{\mathbf{x}} \leq \mathbf{x}(t) \leq \overline{\mathbf{x}}, && \text{(1c)} \\
&\mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)) \leq \mathbf{0}, && \text{(1d)} \\
&\text{for all } t \text{ in } [t_0, t_0 + T] \,.
\end{aligned}$$

Inequalities (1b) and (1c) specify box constraints on the control and state spaces (described by upper and lower bounds), and can represent operational limits on actuators or joints. On the other hand, (1d) can express non-linear, non-convex constraints through the function $\mathbf{g}$. This can be used, for example, to define road boundaries or obstacles in an autonomous driving task.

The receding horizon formulation differs from other reinforcement learning approaches in a few important ways. First, instead of explicitly maintaining a representation of a policy or value function, the control is recalculated (over the shifting horizon) at each time step. This has the advantage of not requiring a representation for the entire problem *a priori*, but incurs the additional computational burden of repeatedly replanning. Second, hard constraints can be explicitly specified and handled through (1d). This allows agents to safely learn by avoiding dangerous regions of the state and control spaces.

Without some simplifying assumptions we cannot solve problem (1) directly. However, efficient approximate solutions can be found by following the "first discretize, then optimize" approach described in the sections below.

### 3.1 DIRECT MULTIPLE SHOOTING

Using direct multiple shooting (Bock and Plitt, 1984), problem (1) can be transformed into a structured non-linear program (NLP). First, the time horizon $[t_0, t_0 + T]$ is partitioned into $N$ equal subintervals $[t_k, t_{k+1}]$ for $k = 0, \ldots, N-1$. Then, taking a piecewise constant approximation of the control signal over each interval, a sequence of initial value problems,

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}_k),\ \mathbf{x}(t_k) = \mathbf{x}_k,\ t \in [t_k, t_{k+1}], \quad (2)$$

can be set up for the state trajectory. Here, the variables $\mathbf{x}_k$ have been added as initial values. Each of these problems can then be integrated to obtain a discretized trajectory. However, in order to enforce continuity over the planning horizon, matching constrains,

$$\mathbf{x}_{k+1} = \mathbf{F}_k(\mathbf{x}_k, \mathbf{u}_k), \qquad k = 0, \ldots, N-1,$$

are placed on $\mathbf{x}_k$ at the boundary of each subinterval. The functions $\mathbf{F}_k$ represent the solutions to the initial value problems (2) at time $t_{k+1}$.

Finally, the cost function is discretized over each time interval, resulting in the following NLP:

$$\min_{\mathbf{x},\mathbf{u}} \quad h(\mathbf{x}_N) + \sum_{k=0}^{N-1} \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k), \quad (3a)$$

subject to

$$\mathbf{x}_0 = \hat{\mathbf{x}}_0, \quad (3b)$$
$$\mathbf{x}_{k+1} = \mathbf{F}_k(\mathbf{x}_k, \mathbf{u}_k), \qquad k = 0, \ldots, N-1 \quad (3c)$$
$$\underline{\mathbf{u}} \leq \mathbf{u}_k \leq \overline{\mathbf{u}}, \qquad k = 0, \ldots, N-1 \quad (3d)$$
$$\underline{\mathbf{x}} \leq \mathbf{x}_k \leq \overline{\mathbf{x}}, \qquad k = 1, \ldots, N \quad (3e)$$
$$\mathbf{g}(\mathbf{x}_k, \mathbf{u}_k) \leq \mathbf{0}. \qquad k = 1, \ldots, N-1 \quad (3f)$$

In principle, any non-linear programming method can be used to solve the above problem (Nocedal and Wright, 2006). However, the computational burden of solving an NLP at each time step is a severe limitation in real-time applications. To address this issue, there has been much interest in efficient optimization for receding horizon control (Domahidi et al., 2012; Vukov et al., 2013) . A particularly successful approach has been the combination of sequential quadratic programming (SQP) with structure exploiting stage-wise solvers (Kouzoupis et al., 2015).

### 3.2 SEQUENTIAL QUADRATIC PROGRAMMING

In the SQP framework, the NLP (3) is linearized about a given nominal trajectory $\mathbf{w} = [\mathbf{x}_0, \mathbf{u}_0, \ldots, \mathbf{x}_N]$. The trajectory can then be improved according to the update (Nocedal and Wright, 2006),

$$\mathbf{w}^+ = \mathbf{w} + \alpha \Delta \mathbf{w}^*, \quad (4)$$

where $\alpha$ is the step size and $\Delta \mathbf{w}^*$ denotes the solution to the following quadratic program:

$$\min_{\Delta\mathbf{x},\Delta\mathbf{u}} \frac{1}{2} \sum_{k=0}^{N-1} \begin{bmatrix} \Delta\mathbf{u}_k \\ \Delta\mathbf{x}_k \\ 1 \end{bmatrix}^\intercal \begin{bmatrix} \mathbf{R}_k & \mathbf{S}_k & \mathbf{r}_k \\ \mathbf{S}_k^\intercal & \mathbf{Q}_k & \mathbf{q}_k \\ \mathbf{r}_k^\intercal & \mathbf{q}_k^\intercal & \rho_k \end{bmatrix} \begin{bmatrix} \Delta\mathbf{u}_k \\ \Delta\mathbf{x}_k \\ 1 \end{bmatrix} + \frac{1}{2} \Delta\mathbf{x}_N^\intercal \mathbf{Q}_N \Delta\mathbf{x}_N + \Delta\mathbf{x}_N \mathbf{q}_N + \rho_N, \quad (5a)$$

subject to the constraints

$$\Delta\mathbf{x}_0 = \hat{\mathbf{x}}_0 - \mathbf{x}_0, \quad (5b)$$
$$\Delta\mathbf{x}_{k+1} = \mathbf{A}_k \Delta\mathbf{x}_k + \mathbf{B}_k \Delta\mathbf{u}_k + \mathbf{a}_k, \quad (5c)$$
$$\underline{\mathbf{u}} - \mathbf{u}_k \leq \Delta\mathbf{u}_k \leq \overline{\mathbf{u}} - \mathbf{u}_k, \quad (5d)$$
$$\underline{\mathbf{x}} - \mathbf{x}_k \leq \Delta\mathbf{x}_k \leq \overline{\mathbf{x}} - \mathbf{x}_k, \quad (5e)$$
$$\mathbf{G}_k \Delta\mathbf{x}_k + \mathbf{H}_k \Delta\mathbf{u}_k \leq \mathbf{g}_k. \quad (5f)$$

The objective function (5a) is a quadratic approximation of the discretized cost function (3a) and the constraints correspond to linearizations of (3b) to (3f).

Provided that $\mathbf{Q}_k$ is positive semidefinite and $\mathbf{R}_k$ is strictly positive definite, the trajectory can be shown to converge to a local optimum by iteratively linearizing and optimizing (Nocedal and Wright, 2006). This is guaranteed for least squares cost functions popular in tracking and stabilization tasks. When a more general cost function is desired a positive definite approximation to the Hessian can be obtained using BFGS updates.

In this paper we use FORCES (Domahidi and Jerez, 2014) as the stage-wise solver for the quadratic program (5). FORCES is an interior-point method tailored for problems arising in RHC. In particular, the block diagonal structure of the Hessian (5a) and the fact that the states are only directly coupled to the previous time step are exploited to give linear computational complexity in the planning horizon.

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 1. Given an initial trajectory, the car must maximize its progress along the race track.

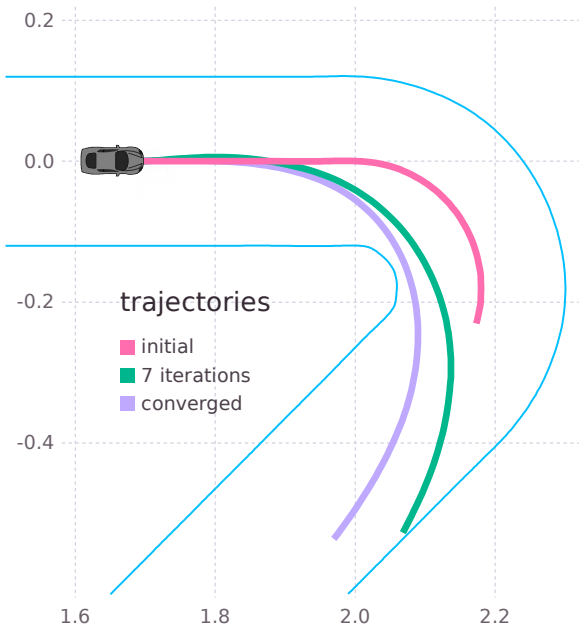


Figure 1: An example of planned trajectories in an autonomous racing task. The agent's objective is to maximize its progress along the racing track over the planning horizon. Note how early termination yields a solution trajectory that is still near-optimal over a short horizon.

The complete algorithm is presented in algorithm 1. In the next section we discuss how sparse online Gaussian process can be used to learn the non-linear dynamics model (3c).

# 4 DYNAMICS MODEL LEARNING

Typically, trajectory optimization based on receding horizon control relies on an analytic model of the system's dynamics. However, in the setting of reinforcement learning the dynamics are unknown and must be learned by interacting with the environment. In order to effectively plan over the horizon, an accurate model of the system's dynamics needs to be learned quickly.

Gaussian process (GP) regression is a non-parametric, Bayesian approach to model learning that has demonstrated impressive data-efficiency on both simulated and real robotic systems (Deisenroth and Rasmussen, 2011). Unfortunately, GPs do not scale well to large data sets, limiting their applicability in practice. There are, however, a number of approximation schemes that significantly reduce computational costs (Quiñonero-Candela and Rasmussen, 2005). In this paper, we use a sparse spectrum approximation of the kernel function (Lázaro-Gredilla et al., 2010). Sparse spectrum GPs were chosen

**Data:** number of features $D$, initial training data
**foreach** *episode* **do**
  train GPs on accumulated data (minimize (11));
  linearise dynamics and constraints about $\mathbf{x}_0$;
  solve (5) until convergence;
  **while** *not terminal* **do**
    shift previous trajectory;
    **for** $i = 1$ *to max iterations* **do**
      linearise about current trajectory;
      solve (5) to get step direction;
      update trajectory (4);
    **end**
    apply control to the system;
    update dynamics model (12);
  **end**
**end**

**Algorithm 1:** The complete GP-RHC algorithm

because (a) online data can be incorporated through incremental updates (Gijsberts and Metta, 2013), and (b) the learned model can be efficiently linearized to form the sequential quadratic programs.

## 4.1 GAUSSIAN PROCESS REGRESSION

Given an input set of $N$ state-control pairs $\tilde{\mathbf{x}} = (\mathbf{x}, \mathbf{u}) \in \mathbb{R}^{n+m}$ and the resulting (possibly noisy) state transitions $\Delta\mathbf{x} \in \mathbb{R}^n$, GP regression can be used to learn a model of the underlying dynamics. Following Deisenroth and Rasmussen (2011), we train conditionally independent GPs on each component of the state transition vector, so for the remainder of this section we will represent target data as $y \in \mathbb{R}$.

Formally, a Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution (Rasmussen and Williams, 2006). To completely specify a GP we need to choose a mean function $m(\tilde{\mathbf{x}})$ and a covariance function $k(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}')$, parametrized by a set of hyperparameters. A common, but flexible choice for the covariance function is the squared exponential kernel

$$k(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}') = \sigma^2 \exp\left(-\frac{1}{2}\sum_{i=1}^{n+m}\left(\frac{\tilde{x}_i - \tilde{x}'_i}{l_i}\right)^2\right),$$

where the signal variance $\sigma^2$ and characteristic length scales $l_i$ constitute the kernel's set of hyperparameters. When modelling noisy targets, an additive noise term with variance $\sigma_n^2$ is included in the covariance function.

Since we model the relative rather than absolute state transitions, we choose a zero mean prior. This means that in the absence of data, the state is expected to re-

main unchanged regardless of the control input.

After defining the input matrix $\mathbf{X} = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_N]$ and the corresponding target vector $\mathbf{y} = [y_1, \ldots, y_N]^\intercal$, the posterior predictive distribution for a set of test points, $\mathbf{X}_*$, is a multivariate Gaussian with mean

$$\mathbb{E}[\mathbf{y}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}] = \mathbf{K}(\mathbf{X}_*, \mathbf{X})\mathbf{Q}^{-1}\mathbf{y}, \tag{6}$$

and covariance

$$\mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) - \mathbf{K}(\mathbf{X}_*, \mathbf{X})\mathbf{Q}^{-1}\mathbf{K}(\mathbf{X}, \mathbf{X}_*), \tag{7}$$

where $\mathbf{Q} = \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2\mathbf{I}$, and $\mathbf{K}(\mathbf{X}, \mathbf{X})$ denotes the matrix of covariances evaluated at all pairs of input points. Computing the predictive mean and covariance are $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$ respectively.

A common approach for learning the hyperparameters and the noise variance $\sigma_n^2$ is to maximize the marginal likelihood (Rasmussen and Williams, 2006). The negative log marginal likelihood, given by

$$\mathcal{L} = \frac{1}{2}\log|\mathbf{Q}| + \frac{1}{2}\mathbf{y}^\intercal\mathbf{Q}^{-1}\mathbf{y} + \frac{n}{2}\log(2\pi), \tag{8}$$

can be minimized using gradient based optimizers. Since $\mathbf{Q}$ needs to be inverted each time the log marginal likelihood is evaluated, which is an $\mathcal{O}(N^3)$ operation in general, hyperparameter inference represents the main bottleneck for GP regression.

### 4.2 SPARSE SPECTRUM APPROXIMATION

In this section, we assume that the covariance function is stationary, i.e. that $k(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}')$ is a function of $\mathbf{r} = \tilde{\mathbf{x}} - \tilde{\mathbf{x}}'$. In this case, Bochner's theorem (Rudin, 2011) states that $k(\mathbf{r})$ can be represented as the Fourier transform,

$$k(\mathbf{r}) = \int_{\mathbb{R}^{n+m}} e^{i\boldsymbol{\omega}^\intercal\mathbf{r}} d\mu(\boldsymbol{\omega}), \tag{9}$$

of a positive finite measure $\mu$. If $\mu(\boldsymbol{\omega})$ has a density, then it is called the power spectrum $S(\boldsymbol{\omega})$ of the covariance function and, by the Wiener-Khintchine theorem (Carlson et al., 2009), $S(\boldsymbol{\omega})$ is the Fourier dual of $k(\mathbf{r})$. In particular, this means that $S$ is proportional to some probability measure $p$ over $\mathbb{R}^{n+m}$, and so equation (9) can be rewritten as an expectation

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

where $\alpha$ is the constant of proportionality.

To approximate the expectation, we draw $D$ sample frequencies $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D$ from $p$ and take averages. Since the power spectrum is symmetric about zero, we also include $\boldsymbol{\omega}_{-j} = -\boldsymbol{\omega}_j$ for each sample frequency, in order to guarantee that $k(\mathbf{r})$ is real valued for all $\mathbf{r}$. This results in the sparse spectrum approximation (Lázaro-Gredilla et al., 2010) of the covariance function:

$$k(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}') \approx \frac{\alpha^2}{2D}\sum_{j=-D}^{D} e^{i\boldsymbol{\omega}_j^\intercal(\tilde{\mathbf{x}}-\tilde{\mathbf{x}}')}. \tag{10}$$

In the particular case of the squared exponential kernel $\alpha^2 = \sigma^2$, and the frequencies are drawn from the normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{\Lambda}^{-1})$, where $\boldsymbol{\Lambda} = \mathrm{diag}([l_1^2, \ldots, l_{n+m}^2])$.

For convenience, we define the feature mapping $\boldsymbol{\phi} : \mathbb{R}^{n+m} \to \mathbb{R}^{2D}$ by,

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

In order to make use of the approximation (10), the matrix inversion lemma is applied to equations (6) and (7), giving

$$\mathbb{E}[\mathbf{y}_*] = \boldsymbol{\phi}(\mathbf{X}_*)^\intercal\mathbf{A}^{-1}\boldsymbol{\phi}(\mathbf{X})^\intercal\mathbf{y},$$
$$\mathrm{cov}[\mathbf{y}_*] = \sigma_n^2\boldsymbol{\phi}(\mathbf{X}_*)^\intercal\mathbf{A}^{-1}\boldsymbol{\phi}(\mathbf{X}_*),$$

where $\mathbf{A} = \boldsymbol{\phi}(\mathbf{X})^\intercal\boldsymbol{\phi}(\mathbf{X}) + \sigma_n^2\mathbf{I}$, and $\boldsymbol{\phi}(\mathbf{X})$ is the matrix obtained by applying $\boldsymbol{\phi}$ to each column of $\mathbf{X}$. Instead of inverting the $N \times N$ matrix $\mathbf{Q}$, we now only require the inverse of the $2D \times 2D$ matrix $\mathbf{A}$, which constitutes a significant saving in computational cost if $D \ll N$. Importantly, the size of $\mathbf{A}$ 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 negative log likelihood (8) gives the expression

$$\begin{aligned}\mathcal{L} = \frac{1}{2}\log|\mathbf{A}| - \frac{D}{2}\log\sigma_n^2 + \frac{n}{2}\log(2\pi\sigma_n^2) \\ + \frac{1}{2\sigma_n^2}\left(\mathbf{y}^\intercal\mathbf{y} - \mathbf{y}^\intercal\boldsymbol{\phi}(\mathbf{X}_*)\mathbf{A}^{-1}\boldsymbol{\phi}(\mathbf{X}_*)^\intercal\mathbf{y}\right).\end{aligned} \tag{11}$$

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

### 4.3 INCREMENTAL UPDATES

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

$$\mathbf{A} \leftarrow \mathbf{A} + \phi(\tilde{\mathbf{x}})\phi(\tilde{\mathbf{x}})^\intercal \quad \text{and} \quad \mathbf{b} \leftarrow \mathbf{b} + \phi(\tilde{\mathbf{x}})y. \tag{12}$$

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).

## 5 EXPERIMENTS

In the following sections we evaluate the performance of GP-RHC on a cart-pole swing up task and an autonomous racing scenario. In all the experiments GP-RHC was able to run in real-time with appropriate choices of the sparsity $D$ and planning horizon $N$. The choice of $D$ essentially involves a trade-off between model accuracy and the computational costs of both the learning and the prediction routines. Empirically, we found that a value between 20 and 100 allowed us to learn a sufficiently accurate dynamics model while remaining within the real-time constraints. Similarly, the choice of $N$ involves a balance between computational costs and the quality of the controller. Ideally, one would choose a large value of $N$ so that the controller can optimally react to future changes in dynamics or constraints.

### 5.1 CARTPOLE SWINGUP

Cartpole experiments are a common benchmark in both reinforcement learning and control theory (Kober et al., 2013). The basic set-up consists of a cart with an attached pendulum running along a track. In the swing-up task, the pendulum is initially pointing downwards and the objective is to apply horizontal forces to the cart in order to swing the pendulum up and balance it above the cart in the center of the track. This is a relatively difficult control problem as the dynamics are fairly nonlinear. 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.

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 $-10$N to 10N can be applied to the cart at a sampling rate of $0.025$s. The cost function is given by a least squares objective penalizing the distance from the set point $[0, 0, \pi, 0]$.

In the first experiment we compare GP-RHC against PILCO (Deisenroth and Rasmussen, 2011) and the ground-truth analytical model. To initialize each trial, 80 data points were collected from an episode with random control inputs. For the sparse spectrum approximation 50 sample frequencies were drawn and a planning horizon of 50 time steps was used. After each episode the GP models were retrained on all the preceding data using 10 random restarts to avoid poor local 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 2 shows that GP-RHC is competitive with PILCO both in terms of sample efficiency and overall performance.

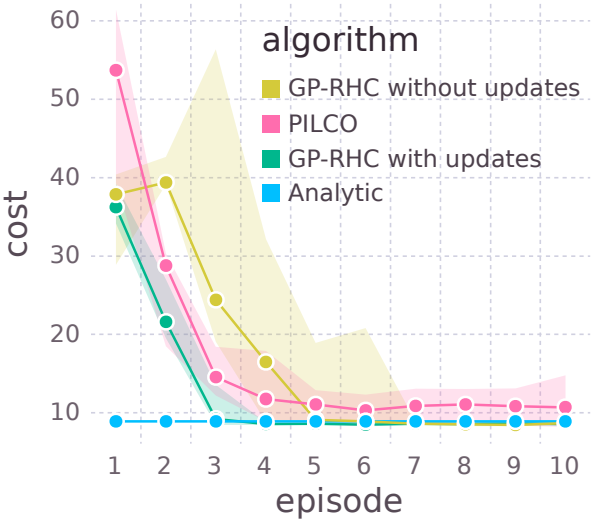


Figure 2: Median cost of the unconstrained cartpole swing-up task (using the saturated cost function). The lower and upper bounds of the confidence envelope represents the first and third quantiles respectively. The results were aggregated over 10 runs.

In the second experiment the length of the track is limited; constraining the cart's position to between $-2$m and 2m. Since PILCO does not handle constraints we just compare GP-RHC to the analytical model. The GP models were initialized with 20 points of training data collected from an episode with random inputs.

As seen in figures 2 and 3, GP-RHC learns to solve the cartpole swing-up task in just a few episodes. In particular, by updating the dynamics model online we can achieve performance comparable to the optimal analytic model at least a full episode earlier. Another noticeable feature is the reduced variability in cost. GP-RHC, with online updates, performs more consistently and is less susceptible to variations in the initial training data. Finally, table 1 shows that fewer constraint violations can be expected when using online updates. In fact, after the first episode only a single constraint violation was encountered, in contrast to the method without updates which incurred significantly more violations. These results indicate that GP-RHC enables fast learning and planning in safety critical conditions.

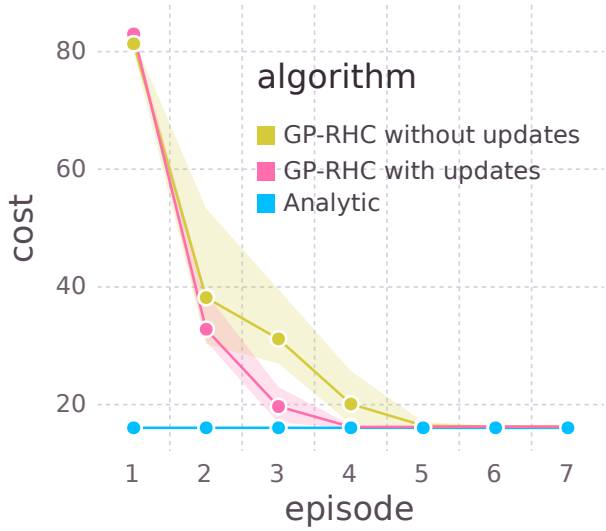


Figure 3: Median cost of the constrained cartpole swing-up task (using the saturated cost function). The lower and upper bounds of the confidence envelope represents the first and third quantiles respectively. The results were aggregated over 22 runs.

Table 1: Percentage of experiments terminated due to constraint violations

| EPISODE | UPDATES | NO UPDATES |
|---|---|---|
| 1 | 100.0 | 100.0 |
| 2 | 4.5 | 18.2 |
| 3 | 0.0 | 13.6 |
| 4 | 0.0 | 4.5 |
| 5 | 0.0 | 0.0 |
| 6 | 0.0 | 0.0 |
| 7 | 0.0 | 0.0 |

### 5.2 AUTONOMOUS RACING

In this section we apply GP-RHC to the autonomous racing of 1:43 scale remote control cars. Liniger et al. (2015) originally investigated this problem using an analytical model of the cars incorporated into a contouring control framework (see section 5.2.2). The objective is to maximize progress along the race course (depicted in figure 6) while remaining within the track boundaries.

#### 5.2.1 BICYCLE MODEL

Following Liniger et al. (2015), the cars are modeled using a bicycle model. The cars are treated as rigid bodies and symmetry is used to approximate the pairs of front and back tyres as single wheels. Pitch and roll dynamics are neglected so only in-plane motion is considered. In our experiments the additional complexity of tyre dynamics is ignored by assuming a no-slip model.

The state space is described by the vector $[x, y, v, \phi]$, where $x$ and $y$ denote the position of the car, $v$ the longitudinal velocity of the car, and $\phi$ the car's orientation. The control signal consists of the PWM duty cycle of the electric drive train motor and the steering angle of the front wheels. The duty cycle is constrained to the interval $[0, 1]$ and the steering angle cannot exceed 18 degrees.

#### 5.2.2 CONTOURING CONTROL

Contouring control was originally designed for industrial applications like machine tool control and laser profiling (Lam et al., 2010). 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. By specifying velocities and orientations the contouring controller is free to determine how the path is followed.

Here we assume that the reference path is given by an arc length parametrized curve

$$\Gamma = \{\mathbf{x} \in \mathbb{R}^q : \mathbf{x} = \gamma(s), s \in [0, l]\},$$

where $l$ is the total length of the path. In our experiments, the center line of the race track is used as the reference path. To find an arc length parametrization, the center line is interpolated by a cubic spline, using the method described by Wang et al. (2002).

Let $p_k = [x_k, y_k]$ denote the position of the car at time $t_k$. Then, the contouring error,

$$\varepsilon_k^c = \mathbf{n}(s_k^*) \cdot (p_k - \gamma(s_k^*)),$$

is defined as the normal deviation from the path $\gamma$, where $s_k^*$ is the value of the path parameter which minimizes the distance between the point $p_k$ and the path, and $\mathbf{n}(s_k^*)$ is the unit normal to $\gamma$ at $s_k^*$.

Calculating the contouring error requires us to determine the value of $s_k^*$ at each point along the planned trajectory. This is too computationally intensive to use as a cost function in the iterative SQP framework. To address this issue, approximations to $s_k^*$ at each point are introduced into the state. The dynamics are then augmented by the equation

$$s_{k+1} = s_k + \Delta t v_k, \quad v_k \in [0, v_{\max}],$$

where $s_k$ denotes the approximation to $s_k^*$ 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 $s_k$ to be a useful approximation we introduce a lag error term $\varepsilon^l$ defined as the distance between the points $\gamma(s_k^*)$ and $\gamma(s_k)$ along the reference path.

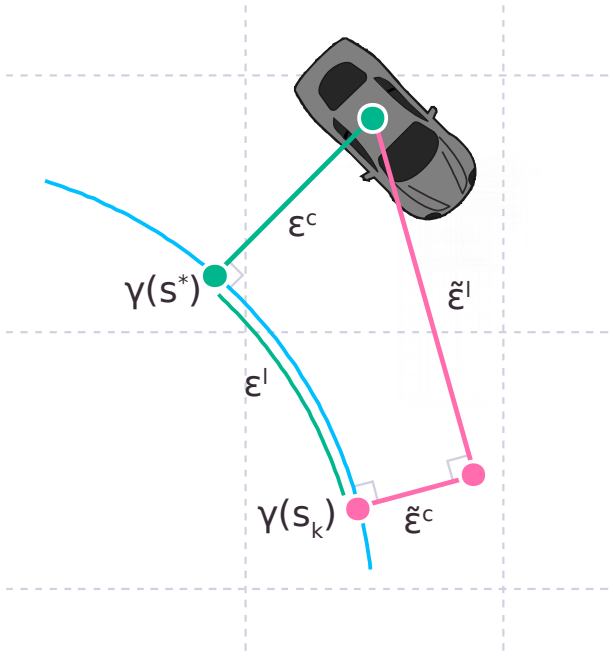


Figure 4: Contouring error $\varepsilon_c$, lag error $\varepsilon_l$ and their respective approximations $\tilde{\varepsilon}_c$ and $\tilde{\varepsilon}_l$.

Since neither the contouring nor lag error can be used directly in the cost function, approximations defined only in terms of $p_k$ and $s_k$ 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 $p_k$ and $\gamma(s_k)$,

$$\tilde{\varepsilon}_k^c = \mathbf{n}(s_k) \cdot (p_k - \gamma(s_k)),$$
$$\tilde{\varepsilon}_k^l = \mathbf{t}(s_k) \cdot (p_k - \gamma(s_k)),$$

where $\mathbf{t}(s_k)$ is the unit tangent to $\gamma$ at $s_k$. It is clear from figure 4 that $\tilde{\varepsilon}_k^c$ approaches $\varepsilon_k^c$, and $s_k$ approaches $s_k^*$ as the lag error is reduced. Therefore, in order to get a good approximation of $s_k^*$ the lag error $\tilde{\varepsilon}^l$ is heavily penalized in the cost function (13).

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

$$\mathcal{L} = ||\tilde{\varepsilon}^l(x, y, s)||_{q_l}^2 + ||\tilde{\varepsilon}^c(x, y, s)||_{q_c}^2 - \alpha \Delta t v. \quad (13)$$

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

### 5.2.3 TRACK CONSTRAINTS

To ensure that the car remains within the track, limits are placed on the $x$ and $y$ components of the car's state. Each point on the planned trajectory is constrained to lie within two half spaces defined by the left and right track boundaries (see figure 5). The relevant tangent lines are found by projecting the point $\gamma(s_k)$ (an approximation to the closest point on the center line) onto the track boundaries.

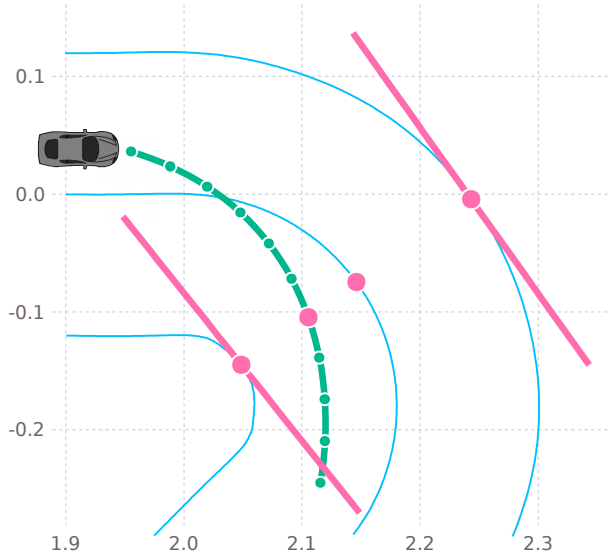


Figure 5: Track constraints (pink lines) corresponding to one point (pink dot) along the planned trajectory. The central dot is the closest point lying on the center line. Tangent lines are found by projecting this point onto the track boundaries.

The planned trajectory is often along the limit of these constraints so in order to avoid infeasibility problems in practice, the constraints are softened by adding slack variables. By penalizing the slack variables heavily in the cost function the original solution of the hard constrained problem is recovered where it would admit a solution.

### 5.2.4 RESULTS

Initially, 70 points of data were collected from a demonstrated trajectory around a simple oval track. 100 sample frequencies were chosen for the sparse spectrum approximation. A planning horizon of 20 time steps was used at a sampling rate of 0.03s. We found that we could plan in real time by limiting the number of SQP iterations to 30. The race track is shown in figure 6 along with the driven trajectory. The car initially starts at rest the point $[0, 0]$ and the race is completed after one full lap. The track is 7.23m long (measured at the center line).

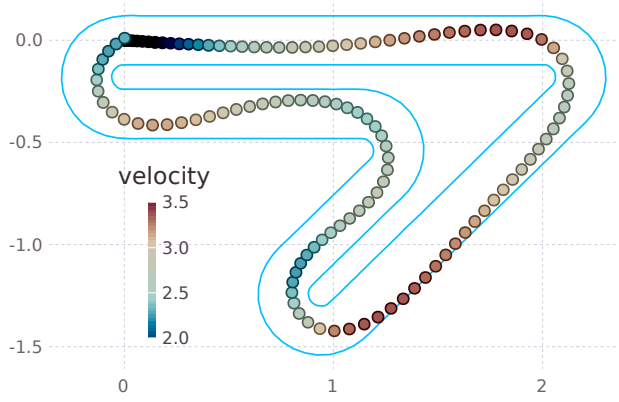


Figure 6: Driven trajectory of the racing car with velocity profile given by GP-RHC. The model was initially trained on 70 points of data on a simple oval track, and was updated online during this task.

Table 2: Lap Times

| ANALYTIC | UPDATES | NO UPDATES |
|---|---|---|
| 2.565 | 2.639 | 2.745 |

Using a learned dynamics model, GP-RHC quickly accelerates from rest and drives trajectories that satisfy the complex track constraints (see figure 6). The sparse spectrum Gaussian processes are very data efficient, learning a sufficiently accurate model of the car's dynamics from just 70 data points. In particular, the training data was collected by driving around a much simpler oval track yet the learned dynamics were able to generalize well enough to effectively navigate the sharp left turn in the new race course. Table 2 shows the lap times of GP-RHC with and without online updates compared to the baseline analytic model. By updating the model online we improve the lap time by more than 0.1s (a relative improvement of about 3.86 percent). This represents a significant saving considering the high speeds and small length scales involved in the problem. In fact, naively scaling up the domain results in a 311m track with a lap time improvement of about 4.56 seconds. This corresponds to a 16.4km/h increase in average speed around the track when using online updates.

In addition to the racing task we also consider an obstacle avoidance problem depicted in figure 7. Static obstacles, represented by the blue cars, are included by adjusting the track constraints. Liniger et al. (2015) determine these adjustments using a high-level planner based on dynamic programming. In our experiments, we manually specify the obstacle constraints but a high level planner could be employed in principle. Since the dynamics are independent of the constraints and cost function, GP-RHC can implicitly take advantage of all the information gained in the racing task by simply reusing the learned model with *no further learning*. This could be useful in safety critical tasks where costly trials can avoided by safely learning a model of the dynamics in a simpler, safer environment.

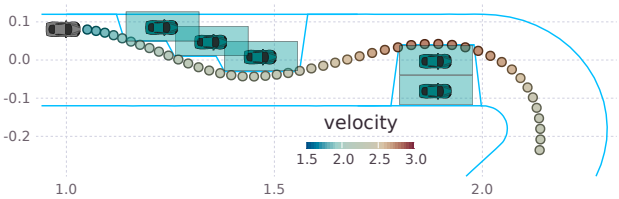


Figure 7: The driven trajectory in an obstacle avoidance task. The car is able to avoid dangerous collisions by planning around the obstacles.

## 6 CONCLUSION AND FUTURE WORK

In this paper we introduce GP-RHC for online learning and planning in continuous environments with non-linear constraints. This is achieved by combining receding horizon control for planning with data efficient sparse spectrum Gaussian processes for model learning. We show that incorporating online data results in faster convergence to optimal behaviour while significantly reducing the number of constraint violations during learning — an important feature for safety critical applications. We demonstrate our approach on a complex autonomous racing task, showing that GP-RHC enables learning from only few training points, and the ability to apply the learned model to new tasks with *no additional training*. This method provides a promising approach to deploying online reinforcement learning algorithms on complex systems such as robots.

In future work plan to apply GP-RHC to real robotic systems such as quadrotors or manipulators. To achieve this goal, a possible improvement to the current method would be the ability to handle input noise. GP regression methods typically assume that the training inputs are noise free. However, in real robotic systems, sensors and filtering algorithms can introduce noise into the state estimation. This issue could be addressed, by incorporating ideas from Mchutchon and Rasmussen (2011) for example, to make GP-RHC more robust.

### Acknowledgements

We thank the reviewer for their helpful insights and feedback.

## References


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.

Hans Georg Bock and Karl-Josef Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings of the IFAC world congress*, 1984.

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.

A.B. Carlson, P. Crilly, and P.B. Crilly. *Communication Systems*. McGraw-Hill Education, 2009.

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.

Alexander Domahidi and Juan Jerez. FORCES Professional. embotech GmbH (`http://embotech.com/FORCES-Pro`), July 2014.

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.

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

H. J. Kim, Michael I. Jordan, Shankar Sastry, and Andrew Y. Ng. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 16*, pages 799–806. MIT Press, 2004.

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

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.

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.

Andrew Mchutchon and Carl E. Rasmussen. Gaussian process training with input noise. In *Advances in Neural Information Processing Systems 24*, pages 1341–1349. Curran Associates, Inc., 2011.

J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

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

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.

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

Walter Rudin. *Fourier analysis on groups*. John Wiley & Sons, 2011.

Milan Vukov, Alexander Domahidi, Hans Joachim Ferreau, Manfred Morari, and Moritz Diehl. Auto-generated algorithms for nonlinear model predictive control on long and on short horizons. In *52nd IEEE Conference on Decision and Control*, pages 5113–5118. IEEE, 2013.

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, 2002.