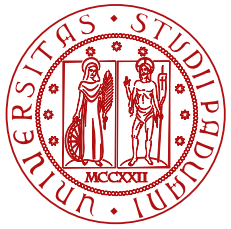

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering

Learning-Based Non-Linear Model Predictive Control with Gaussian Processes: A Computational Efficiency Analysis of open-source toolboxes

Master Candidate

Stefano Trenti

Student ID 2079450

Supervisor

Prof. Mattia Bruschetta

University of Padova

Academic Year
2024/2025

To my parents,
family and friends

Abstract

This work investigates the computational efficiency in Learning-Based Non-Linear Model Predictive Control (NMPC) with Gaussian Processes (GPs), focusing on the comparison of two open-source toolboxes: L4ACADOS and LbMATMPC. NMPC methods augmented with GPs enable accurate modeling of unknown dynamics enabling precise modeling of the real system dynamics but are computationally demanding, especially as the volume of training data increases. A key challenge is the growth in computational time associated with GP inference as the dataset size expands causing. This work evaluates how GPU acceleration and dimensionality reduction techniques can mitigate these limitations.

The study compares different approaches employed by these two toolboxes. A shared toy problem serves as the benchmark, allowing consistent evaluation of computation times, particularly that of sequential quadratic programming iterations.

Results highlight that GPU acceleration significantly reduces the computational overhead of GP computations, making real-time NMPC feasible for large datasets, once the dataset size reaches a threshold where the GPU's initial overhead is outweighed by its processing efficiency.

Additionally, dimensionality reduction methods, including sparse GP approximations via Stochastic Variational Gaussian Processes, are tested to explore whether comparable performance can be achieved with fewer training points. These techniques may obviate the need for extensive datasets, further reducing the computational burden and questioning the necessity of GPU acceleration in certain cases.

The findings provide insights into the trade-offs between GP model accuracy, computational efficiency, and hardware requirements in learning-based NMPC. By leveraging both GPU acceleration and advanced GP approximation techniques, this thesis aims to advance the practical deployment of efficient NMPC strategies in real-world applications.

Sommario

Questa tesi indaga l'efficienza computazionale nel Controllo Predittivo Non Lineare basato sull'Apprendimento (Learning-Based Nonlinear Model Predictive Control, NMPC) con Processi Gaussiani (Gaussian Processes, GP), con particolare attenzione al confronto tra due toolbox open-source: L4ACADOS e LbMATMPC. L'integrazione dei GP nei metodi NMPC consente una modellazione più accurata delle dinamiche sconosciute, migliorando la rappresentazione del sistema reale. Tuttavia, tali metodologie risultano computazionalmente onerose, specialmente all'aumentare della quantità di dati e complessità di modello. Una sfida cruciale è rappresentata dalla crescita del tempo computazionale associato allinferenza Gaussiana, che aumenta significativamente con l'espansione del dataset di training. Questo studio valuta in che misura laccelerazione tramite hardware grafico (GPU) e le tecniche di riduzione della dimensionalità possano mitigare tali limitazioni. Vengono confrontati diversi approcci adottati dai due toolbox considerati. Un Toy-problem di riferimento condiviso funge da benchmark, consentendo una valutazione coerente dei tempi di calcolo, con particolare attenzione alle iterazioni di Sequential Quadratic Programming. I risultati evidenziano che laccelerazione tramite GPU riduce significativamente il carico computazionale delle operazioni GP, rendendo il NMPC in tempo reale praticabile anche per dataset di grandi dimensioni. Tuttavia, lefficienza della GPU emerge solo quando il dataset supera una soglia critica, oltre la quale il costo iniziale della GPU viene compensato dai benefici in termini di velocità di computazione. Inoltre, vengono analizzate tecniche di riduzione della dimensionalità, tra cui la approssimazione sparsa dei GP basata su Stochastic Variational Gaussian Processes, per verificare se sia possibile ottenere prestazioni comparabili con un numero ridotto di punti di training. Tali strategie potrebbero ridurre la necessità di dataset estesi, abbattendo ulteriormente il costo computazionale e mettendo in discussione leffettiva necessità dellaccelerazione GPU in determinati scenari. Le conclusioni di questo lavoro offrono una prospettiva sui compromessi tra accuratezza del modello GP, efficienza computazionale e requisiti hardware nel contesto del learning-based NMPC. Sfruttando sia laccelerazione GPU sia tecniche avanzate di approssimazione GP, questa ricerca mira a favorire l'adozione pratica di strategie NMPC efficienti in applicazioni reali.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AC Actor-Critic

ACADOS Automatic Control And Dynamic Optimization Software

CPU Central Processing Unit

CSV Comma Separated Values

EKF Extended Kalman Filter

GP Gaussian Process

GPR Gaussian Process Regression

GPU Graphics Processing Unit

ILQR Iterative Linear Quadratic Regulator

PID ProportionalIntegralCerivative controller

MIMO Multi-input Multi-output

IoU Intersection over Union

IPOPT Interior Point OPTimizer

KF Kalman Filter

L-BFGS Limited-memory Broyden-Fletcher-Goldfarb-Shanno

LQR Linear Quadratic Regulator

LQG Linear Quadratic Gaussian

MPC Model Predictive Control

MSE Mean Squared Error

MAE Mean Absolute Error

R-squared Mean Squared Error

NMPC Nonlinear Model Predictive Control

NLP Nonlinear Programming

NN Neural Network

ODE Ordinary Differential Equation

OCP Optimal Control Problem

RBF Radial Basis Function

RL Reinforcement Learning

RMSE Root Mean Square Error

RK Runge-Kutta

RTI Real-Time Iteration

VFE Variational Free Energy

ZORO Zero-Order Robust Optimization

SVGP Stochastic Variational Gaussian Process

IPT Inducing Point

DMS Direct Multiple Shooting

ERK-4 Fourth-order Explicit Runge-Kutta

1

Introduction

Model Predictive Control (MPC) has established itself as one of the most effective and widely applied advanced control strategies in modern engineering. By leveraging an explicit dynamic model of the system, MPC predicts future behavior and determines optimal control inputs within a given horizon while ensuring constraint satisfaction. Its capability to systematically handle multi-input, multi-output systems and constraints has made it a dominant methodology in fields such as chemical process control, automotive systems, aerospace applications, robotics, and energy management. The extension of MPC to Nonlinear Model Predictive Control (NMPC) has further broadened its applicability, allowing for the control of highly complex and nonlinear systems where linear approximations fail to capture the intricacies of system dynamics.

The versatility and robustness of NMPC have led to its successful application in various real-world scenarios. In the automotive industry, NMPC has been utilized for trajectory planning [1], obstacle avoidance [2, 3], and stability control in both autonomous [4] and assisted driving systems [5]. It has been applied to high-performance racing vehicles, where real-time optimization is critical for minimizing lap times while ensuring safety and adherence to constraints [6, 7]. In aerospace applications, NMPC has been used for spacecraft attitude control [8, 9], UAV path planning [10], and missile guidance [11], demonstrating its effectiveness in scenarios requiring high precision and adaptability. The field of robotics has also benefited from NMPC, with applications in humanoid locomotion [12, 13], robotic manipulation [14, 15], and exoskeleton control [16], where complex nonlinear dynamics and constraints play a fundamental role.

The core principle of NMPC involves solving a constrained nonlinear optimization problem at each sampling instant. Given the current state of the system, NMPC computes an optimal sequence of control inputs over a prediction horizon, ensuring that constraints on inputs and states are respected. However, only the first computed input is applied to the system, and the process is repeated at the next time step using updated state measurements. This receding horizon approach allows NMPC to react adaptively to disturbances and modeling inaccuracies. The key advantage of NMPC lies in its ability to explicitly handle nonlinear dynamics and constraints while optimizing performance. Unlike traditional linear controllers, NMPC does not require system linearization and can directly incorporate state and control limits, making it particularly suitable for complex control tasks.

Despite its advantages, NMPC faces significant challenges. The computational complexity of solving a nonlinear optimization problem in real-time imposes stringent requirements on hardware and algorithm efficiency. The choice of prediction and control horizons, the formulation of cost functions, and the handling of constraints all contribute to the complexity of NMPC design. Additionally, NMPC relies heavily on an accurate mathematical model representation of the system to be controlled. Model inaccuracies can lead to suboptimal or even unstable behavior, necessitating extensive system identification and tuning efforts. Parameter selection, including cost function weights and constraint formulations, often requires labor-intensive tuning [17], which may not generalize well across different operating conditions.

To mitigate the challenges associated with model inaccuracies and tuning difficulties, learning-based NMPC has emerged as a promising approach [18]. By leveraging data-driven techniques, learning-based NMPC aims to enhance control performance through the integration of system identification, online adaptation, and parameter learning. This paradigm shift allows controllers to compensate for model mismatches, learn system dynamics from data, and refine control policies over time. Within this framework, various methodologies have been explored, including reinforcement learning [19], neural networks [20], and Gaussian Process Regression for system modeling and prediction [21].

Gaussian Process Regression has proven to be a particularly effective tool for learning-based NMPC due to its probabilistic nature and ability to quantify uncertainty. This allows NMPC to incorporate both the mean prediction and uncertainty estimates when necessary, enabling robust control strategies that

account for model inaccuracies [22]. The primary advantage of GPR lies in its capability to generate smooth and continuous function approximations, even when the available data is limited.

However, GPR also has limitations. The computational complexity of standard GPR scales poorly with the number of training data points, making it challenging to deploy in real-time applications. To mitigate this issue, several approaches have been proposed in the literature, including sparse Gaussian Process (GP) approximations [23], local GP models [24], kernel approximations such as Random Fourier Features [25]. Furthermore, the choice of kernel functions, hyperparameter tuning, and dataset selection play crucial roles in the performance of GPR-based controllers.

The integration of GPR within NMPC frameworks has led to the development of Gaussian Process-based NMPC (GP-NMPC), which combines the advantages of data-driven learning with the well-established principles of predictive control. GP-NMPC enables adaptive control strategies that refine system models based on real-world data while preserving the constraint-handling capabilities of NMPC. This hybrid approach has shown promise in various applications, including robotics [26], autonomous vehicles [27], and spacecraft attitude control [28], where accurate modeling and adaptability are paramount. As research progresses, advancements in computational techniques, parallelization, and hardware acceleration will further enhance the feasibility of GP-NMPC, paving the way for its broader adoption in real-time control applications.

In this thesis, we investigate the computational efficiency of open-source Nonlinear Model Predictive Control toolboxes in the context of learning-based NMPC with Gaussian Processes. The objective is to analyze the different computational behaviors between different implementations and explore GPU acceleration for GP predictions, when incorporating GP models into NMPC formulations. To asses these differences, two open-source GP-NMPC toolboxes will be utilized comparing the two implementation from a computational efficiency standpoint. By leveraging data-driven approaches, learning-based NMPC aims to improve control performance in uncertain and nonlinear environments while addressing the computational challenges inherent in traditional NMPC implementations. The findings of this study will contribute to the ongoing development of efficient, scalable, and robust control strategies for real-world applications.

2

Gaussian Process-based Nonlinear Model Predictive Control

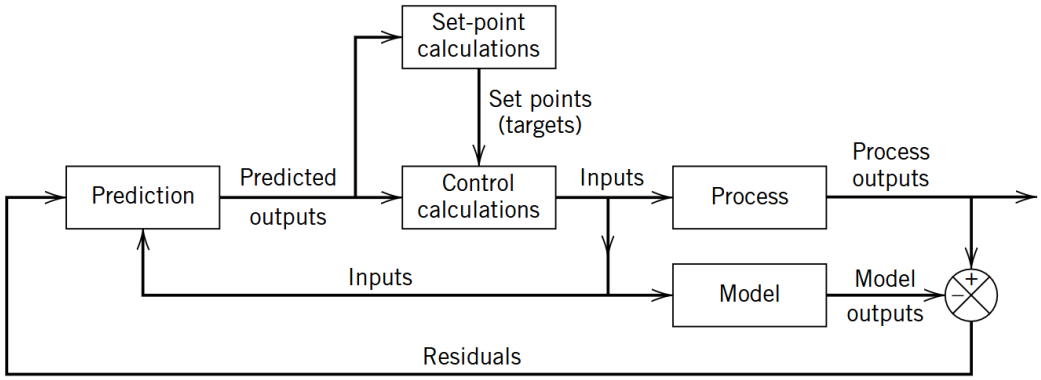

Figure 2.1: Block diagram for model predictive control [29, p. 369, Fig. 20.1].

Model Predictive Control, as explained in [30] and [31], originated in the 1970s for industrial process control, evolving from early heuristic methods like Model Predictive Heuristic Control (MPHC) and Dynamic Matrix Control (DMC). These approaches leveraged digital computing to optimize control actions over a finite horizon. The introduction of Quadratic Dynamic Matrix Control (QDMC) in the 1980s incorporated quadratic programming for handling constraints. Third-generation algorithms, such as IDCOM-M and SMOC, improved constraint management and introduced state-space representations. By the late 1990s, robust MPC (RMPCT) and DMC-plus integrated advanced optimization techniques and graphical interfaces. Stability, a key challenge due to

the finite horizon, was addressed through terminal constraints and dual-mode designs.

MPC has emerged as a powerful alternative for addressing control challenges, particularly in nonlinear and constrained systems. MPC explicitly incorporates system dynamics and constraints, into the control decision-making process. At each time step, MPC solves an optimization problem over a finite prediction horizon, yielding an optimal control sequence. Only the first control action is implemented, and the process repeats at the next time step, incorporating updated system information. This receding-horizon approach enables MPC to handle Multi-input Multi-output (MIMO) systems, incorporate constraints naturally, and proactively respond to anticipated disturbances when combined with appropriate estimation methods, making it highly effective in complex control scenarios. Despite its advantages, MPC has notable challenges that

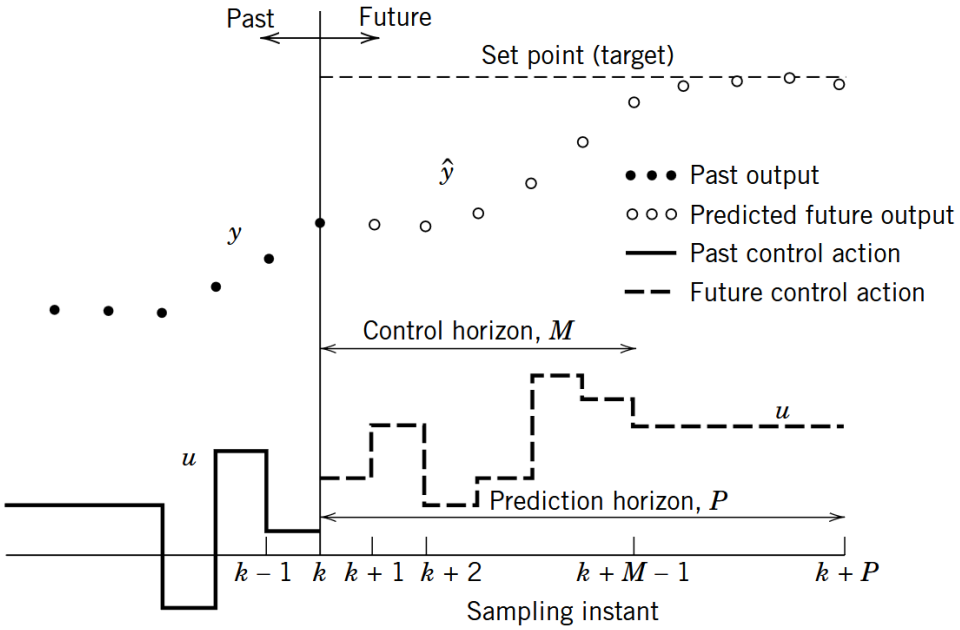

Figure 2.2: Basic concept for model predictive control [29, p. 370, Fig. 20.2].

must be addressed. The computational complexity of solving an optimization problem at each time step can be prohibitive, particularly for high-dimensional and fast-sampling systems. Traditional MPC implementations often rely on linearized models to reduce computational burden, especially when applied to nonlinear systems. While this approach can be effective in many cases, it may introduce inaccuracies when dealing with highly nonlinear dynamics.

2.1 The Nonlinear Model Predictive Control problem

MPC can be naturally extended to nonlinear systems through Nonlinear Model Predictive Control, which explicitly incorporates nonlinear system dynamics in the optimization framework. For many physical systems, the governing dynamics are most naturally expressed in continuous time as an Ordinary Differential Equation (ODE):

$$\dot{x}(t) = f(x(t), u(t)) \tag{2.1}$$

where x represents the system state and u denotes the control input. Given that these dynamics inherently evolve in continuous time, it is often preferable to formulate the optimization problem in continuous time as well. This continuous-time formulation serves as the foundation for the control problem that must be solved at each sampling instant. The resulting problem is an Optimal Control Problem (OCP), which can be expressed as follows:

$$\begin{aligned} \min_{x(\cdot),u(\cdot)} \quad & \int_0^T l(x(t), u(t))\, dt + l_f(x(T)) \\ \text{s.t.} \quad & x(0) = \hat{x}_0 \\ & \dot{x}(t) = f(x(t), u(t)), && t \in [0, T] \\ & g(x(t), u(t)) \leq 0, && t \in [0, T] \\ & g_f(x(t), u(t)) \leq 0 \end{aligned} \tag{2.2}$$

The objective function, given by the integral, represent the cost to be optimized over the prediction horizon in which $l(x(t), u(t))$ denotes the running cost and $l_f(x(T))$ the terminal cost. The system dynamics are described by the continuous-time ODE, capturing the evolution of the state as a function of the control input. The initial condition $x(0) = \hat{x}_0$ ensures that the optimization process starts from the estimated current state. Additionally, the inequality constraint $g(x(t), u(t)) \leq 0$ encodes state and control constraints that must be satisfied at all times within the prediction horizon, while $g_f(x(t), u(t)) \leq 0$ enforces terminal constraints to ensure feasibility and stability. In order to solve this problem numerically, the Direct Multiple Shooting (DMS) method can be

applied, obtaining the following Nonlinear Programming (NLP) problem:

$$\begin{aligned}
\min_{x_0,\dots,x_N,u_0,\dots,u_{N-1}} \quad & \sum_{k=0}^{N-1} l(x_k, u_k) \cdot (t_{k+1} - t_k) + l_f(x_N) \\
\text{s.t.} \quad & x_0 = \hat{x}_0 \\
& x_{k+1} = \phi(x_k, u_k), && \forall k \in \{0, \dots, N-1\} \\
& g(x_k, u_k) \leq 0, && \forall k \in \{0, \dots, N-1\} \\
& g_f(x_N) \leq 0
\end{aligned} \tag{2.3}$$

In this way the original continuous-time OCP is discretized into a finite-dimensional optimization problem. The objective function approximates the integral cost from the continuous formulation using a discrete summation over the control horizon, where $l(x_k, u_k)$ represents the stage cost, scaled by the discretization time step, and $l_f(x_N)$ accounts for the terminal cost. The system dynamics are enforced through the discrete-time mapping $x_{k+1} = \phi(x_k, u_k)$, where $\phi(x_k, u_k)$ represents the numerical integration of the continuous-time system, commonly computed using an Explicit fourth-order Runge-Kutta (ERK-4) scheme to approximate the state transition over each time step. These methods evaluate the system dynamics at multiple intermediate points within a single time step and compute a weighted sum of these evaluations to estimate the next state, ERK-4 provides a good balance between accuracy and computational efficiency by taking four intermediate steps per time step. The initial condition constraint $x_0 = \hat{x}_0$ ensures that the optimization process starts from the estimated initial state. Additionally, the inequality constraints $g(x_k, u_k) \leq 0$ enforce state and control feasibility at each discretization step, while $g_f(x_N) \leq 0$ ensures satisfaction of terminal constraints.

This discretized problem can be efficiently solved using nonlinear optimization solvers, facilitating real-time implementation of Nonlinear Model Predictive Control. In practice, NMPC problems are typically solved using either Sequential Quadratic Programming (SQP) [32] or Interior Point methods. In this work, the focus will be on SQP, an iterative algorithm that computes an approximate solution to the original nonlinear programming problem by sequentially solving a series of Quadratic Programming (QP) subproblems. The formulation of each QP problem involves the computation of the so called sensitivities, i.e., the first-order derivatives of the system dynamics, constraints, and objective function

with respect to the state and control variables. These derivatives are typically obtained using automatic differentiation techniques to enhance computational efficiency. While SQP is an effective method for solving NLPs, its computational complexity can be challenging for real-time applications. To address this issue, the Real-Time Iteration (RTI) scheme [33] can be employed. In its most basic formulation, as considered in this work, RTI consists of executing only a single SQP iteration per control update.

However, since full convergence is not achieved at each step, the RTI scheme significantly reduces the computational burden at the cost of yielding a suboptimal solution. As a result, this approach may introduce performance degradation in the closed-loop control, although it remains a practical compromise for real-time implementation. Furthermore, another challenge is that of representing with high fidelity the dynamics of the system that is to be controlled, as parameter uncertainty or unmodeled dynamics may lead to suboptimal or unstable performance. Since in practical scenarios obtaining an accurate analytical model is often challenging, over the years, researchers have thought of several ways of mitigating the issue of model-mismatch. Among these, this work focuses on the use of learning models, such as Gaussian Processes, to learn the dynamics (or just part if it) from data. Such approach will be detailed in the following section.

2.2 Gaussian Processes: Foundations, Theory, and Applications

Gaussian Processes represent a fundamental class of non-parametric probabilistic models that have been widely adopted in machine learning and control theory due to their ability to model complex functions while providing uncertainty quantification [34]. The theoretical foundation of Gaussian Processes can be traced back to early statistical work in the field of stochastic processes, with initial developments in the early 20th century by mathematicians such as Kolmogorov and Wiener [35]. Their adoption in machine learning was significantly advanced in the late 1990s and early 2000s, particularly following the seminal work of Williams and Rasmussen, who formalized their use in regression and classification tasks [36]. Since then, GPs have been extensively studied and applied to various domains, including robotics, climate modelling, reinforcement learning, and Model Predictive Control [37, 38, 18].

Mathematically, a Gaussian Process defines a distribution over functions, allowing for a principled way to infer function values at unobserved locations given a set of training data. Formally, a Gaussian Process is a infinite set of random variables representing the value of a function $f(x)$ which is said to be distributed as a Gaussian Process if, for any finite set of input points $x \in \mathbf{X}$, the corresponding function value follow a joint Gaussian distribution:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{2.4}$$

where the functions $m(x)$ or mean and $k(x, x')$ or covariance completely characterize the GP function $f(x)$. These functions are defined as:

$$m(x) = \mathbb{E}[f(x)] \tag{2.5}$$

$$k(x, x') = \mathbb{E}\left[(f(x) - m(x))(f(x') - m(x'))\right] \tag{2.6}$$

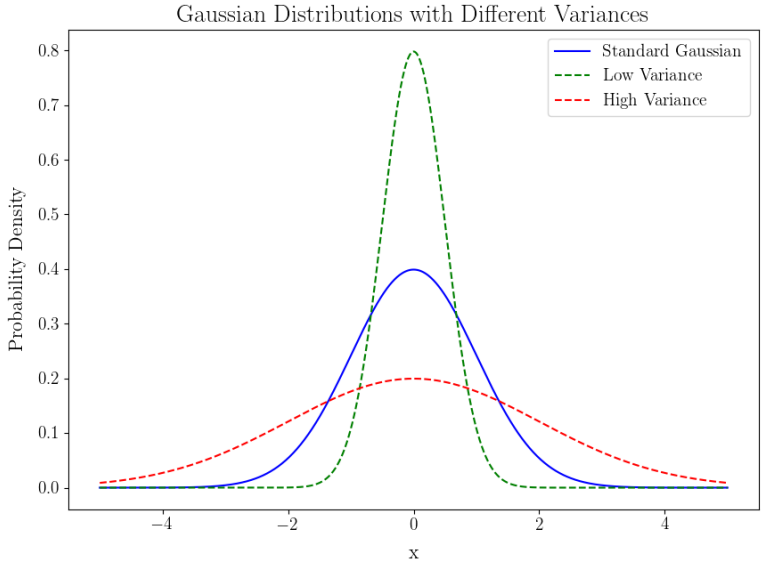

Figure 2.3: Zero-mean Gaussian Distribution with different variance values.

Within the context of machine learning, Gaussian Processes have been utilized for regression by modelling the relationship between an input dataset $\mathbf{X}$ and its corresponding target set $\mathbf{Y}$. Expanding this into the realm of system dynamics learning, the input set is formed by multidimensional vectors $x \in \mathbb{R}^n$ containing system states and eventual control inputs, whereas the targets consist of noisy observations $y \in \mathbb{R}^m$ with noise $e \in \mathbb{R}^m$ that follows a zero-mean

Gaussian distribution. The underlying function $f \sim \mathcal{N}(0, K_{xx})$ is modelled as a zero-mean Gaussian Process. $K_{xx} \in \mathbb{R}^{n \times n}$ is the covariance matrix, whose elements are specified by a kernel function $k(x_i, x_j)$. This covariance function (or kernel) is the key component of a Gaussian Process and encodes assumptions about the smoothness and structure of the underlying function being modelled [36]. Common choices for $k(x_i, x_j)$ include the squared exponential (SE) kernel, also known as Radial Basis Function (RBF) kernel:

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right) \tag{2.7}$$

where:

- σ_f^2 is the output variance (signal variance), controlling the overall scale of function variations.
- ℓ is the lengthscale parameter, determining how far inputs need to be apart before their function values become uncorrelated.
- $\|x - x'\|^2$ is the squared Euclidean distance between input points x and x'.

A Gaussian Process can be used to infer the function values at new test points $x_\star$ [39]. The predictive mean at a test point $x_\star$ is given by:

$$\mu(x_\star) = \mu_x + K_{x_\star x}(K_{xx} + \sigma_f^2 I)^{-1}(Y - \mu_x) \tag{2.8}$$

and the predictive variance is given by:

$$\Sigma(x_\star) = k(x_\star, x_\star) - K_{x_\star x}(K_{xx} + \sigma_f^2 I)^{-1} K_{x_\star x} \tag{2.9}$$

where $K_{x_\star x}$ is the vector of covariances between the test point $x_\star$ and the training points. The mean function $\mu(x_\star)$ provides a prediction of the function value at $x_\star$, while the variance function $\Sigma(x_\star)$ quantifies the uncertainty in this prediction.

The parameters of the kernel function, such as the lengthscale ℓ and signal variance σ_f^2, are typically learned by maximizing the Marginal Likelihood [39] of the observed data:

$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^T(K + \sigma^2 I)^{-1}\mathbf{y} - \frac{1}{2}\log||K + \sigma^2 I|| - \frac{n}{2}\log 2\pi. \tag{2.10}$$

This process, known as hyperparameter optimization, is usually performed using gradient-based methods such as the L-BFGS algorithm [34].

The ability of Gaussian Processes to provide uncertainty estimates makes them particularly useful in a wide range of applications. In robotics, GPs have been used to model the dynamics of robotic systems, allowing for adaptive control strategies that can compensate for model uncertainty [38]. In reinforcement learning, Gaussian Process regression has been applied to model the reward function and transition dynamics, enabling more data-efficient learning compared to deep learning methods [40].

2.3 Integrating Gaussian Processes into Model Predictive Control

In the context of Nonlinear Model Predictive Control, control performance is heavily dependent on the fidelity of the predictive model, as inaccuracies can lead to suboptimal control actions, constraint violations, or even instability. However, in many real-world scenarios, deriving an accurate first-principles model is challenging, this can be due to unknown system dynamics, disturbances, incorrect modeling or dynamics that may change over time. To address these challenges, Gaussian Process-based NMPC (GP-NMPC) has been proposed as a data-driven alternative to traditional NMPC. Gaussian Processes provide a non-parametric approach to modeling system dynamics by learning from data rather than relying on explicit equations. This allows GP-NMPC to adapt to uncertainties and improve model accuracy. In GP-MPC, the predictive model is updated at regular intervals based on observed data, allowing for improved tracking and disturbance rejection compared to traditional MPC [18]. GPs are particularly well-suited for learning unknown dynamics because they provide a probabilistic estimate of system behavior, allowing for uncertainty quantification into the control process. When explicitly leveraged, this probabilistic nature enables GP-NMPC to make more informed control decisions by considering the confidence in learned models, leading to improved robustness and generalization in dynamic environments. Gaussian Process-based Nonlinear Model Predictive Control has become more popular and mature over the years, some examples in literature where this technique is applied are: mobile robot path following [41], wind turbine load mitigation [42] and agile quadrotor flight [43].

A typical GP-NMPC control algorithm can be summarized as:

Algorithm 1 GP-NMPC Algorithm

Input: Initial state x_0, nonlinear system dynamics $x_{k+1} = f(x_k, u_k)$, Gaussian process model $p(y|x, u)$,stage cost function $l(x_k, u_k)$, final stage cost function $l_f(k + N)$, constraints $g(x_k, u_k) \leq 0$, prediction horizon N.
Output: Optimal control input sequence $\mathbf{u}^* = (u_0^*, u_1^*, \ldots, u_{N-1}^*)$.
1. Initialize: Set $x_k = x_0$, Gaussian Process model, time step $k = 0$.
while termination condition not met (e.g., $k < k_{\max}$) **do**
2. Prediction: Compute the future state predictions using the GP model:

$$\hat{y}_{i+1} = p(y|x_i, u_i)$$

where $\hat{y}_{i+1}$ represents the GP-predicted residual dynamics.
3. Model Correction: Update the integrated system using the GP prediction:

$$x_{i+1} = \phi(x_i, u_i) + \hat{\psi}_{i+1}$$

Where ϕ and ψ are the numerical integrator of the real-time system.
4. Optimization: Solve the following NLP:

$$\min_{\mathbf{u},\mathbf{x}} \sum_{i=k}^{k+N-1} l(x_i, u_i) \cdot (t_{i+1} - t_i) + l_f(k + N)$$

subject to:

$$\begin{aligned} x_{i+1} &= \phi(x_i, u_i) + \hat{\psi}_{i+1}, && \forall i = k, \ldots k + N - 1 \\ g(x_i, u_i) &\leq 0, && \forall i = k, \ldots, k + N - 1 \\ g_f(x_N) &\leq 0 \end{aligned}$$

Let $\mathbf{u}^* = (u_k^*, u_{k+1}^*, \ldots, u_{k+N-1}^*)$ be the solution.
5. Apply the control input: Apply the first control input u_k^*.
6. State measurement: Measure the new state x_{k+1}.
7. Increment time step: k++.
end while

Modelling approaches in GP-NMPC can be categorized into grey-box and black-box methods. Grey-box modelling incorporates prior knowledge of system dynamics, using GPs to model only the uncertain or unknown components. This hybrid approach benefits from known physics-based models while leveraging data-driven learning for unmodeled dynamics. In contrast, black-box modelling relies entirely on data to infer system behavior, making it useful for

systems with highly complex or poorly understood dynamics [44]. Both approaches offer significant advantages in control applications. Grey-box models enhance interpretability and reduce data requirements, whereas black-box models provide flexibility in capturing intricate, data-driven dynamics. The choice between these methods depends on the availability of prior knowledge, computational constraints, and the desired balance between accuracy and efficiency.

2.3.1 Computational Challenges of GP-NMPC

One of the main challenges of Learning-Based modeling is data collection and scarcity. Accurately modeling a complex dynamical system requires sufficient data covering a large portion of the state space to capture its underlying distributions. The integration of Gaussian Processes into Nonlinear Model Predictive Control also introduces significant computational challenges, primarily due to the high dimensionality of the optimization problem and the need for real-time execution.

GPs have gained widespread adoption in machine learning and control applications due to their ability to provide not only accurate predictions but also meaningful uncertainty estimates [34]. This characteristic makes them particularly well-suited for safety-critical applications where model confidence is essential. However, despite their advantages, one of the most significant limitations of GPs is their computational cost, particularly when applied to large datasets. The computational cost of GP training and inference grows significantly with the number of training points [45]. This is particularly problematic for NMPC, where solving the OCP at each time step is already computationally demanding (see sec. 2.1). Unlike traditional NMPC models, which evaluate system dynamics through closed-form equations, GP-NMPC requires computing posterior mean and variance predictions from a trained GP model at each SQP iteration. This significantly increases the computational burden and challenges real-time feasibility, particularly for fast-sampling, high-dimensional applications. The computational cost of making predictions using a Gaussian Process model primarily arises from the necessity of inverting the covariance matrix K_{xx}. Since K_{xx} is an $n \times n$ matrix, its inversion during training requires $\mathcal{O}(n^3)$ operations [34]. Once this inversion is computed, the mean prediction $\mu(x_\star)$ can be obtained by performing a matrix-vector multiplication, which requires $\mathcal{O}(n)$ operations. However, when computing the predictive variance $\Sigma(x_\star)$, an

additional quadratic operation is involved, as it requires the computation of the quadratic form $K_{x_\star x}(K_{xx} + \sigma_f^2 I)^{-1} K_{x_\star x}$, leading to an overall complexity of: $O(n^2)$ for the variance estimation. This means as the number of training points increases, the cost of making a single prediction grows at least linearly in n when only the mean is computed and quadratically in n when the variance is also required [45].

One tempting ansatz to avoid this scaling problem could be to rely on Neural Networks. In a typical feedforward neural network with L layers, the computational complexity of a single forward pass is given by $O(\sum_{l=1}^{L} d_{l-1} d_l)$, where d_l represents the number of neurons in layer l. Importantly, this complexity is independent of the number of training points [46]. Once a neural network has been trained, its inference time remains constant regardless of the size of the dataset used for training. This contrasts sharply with the behaviour of Gaussian Processes, where each prediction requires computation involving the entire training dataset. However, we discard Neural Networks due to their lack of inherent uncertainty quantification and data inefficiency. Since NMPC relies on an accurate model for real-time optimization, the black-box nature of NNs makes them less reliable, especially when dealing with limited or noisy data. Moreover, NN training requires large datasets, whereas Gaussian Processes inherently provide uncertainty estimates and can generalize well from fewer data points, making them preferable for safety-critical applications.

In the case of a complex system where the necessity for large amount of data to obtain precise GP predictions, the resulting increase in prediction time can significantly slow down the MPC computations, making it difficult to achieve the required control update rates. An interesting approach to mitigate this issue is that of utilizing GPU acceleration to offload the computationally intensive GP prediction to hardware that specialize in this type of computation. One of the most promising solutions to alleviate the computational burden of GP-NMPC is leveraging Graphics Processing Units (GPUs) through CUDA (Compute Unified Device Architecture) to compute GP inference. We discuss the benefits and limits of GPU acceleration in sec. 4.6.

3

Open Source Toolboxes

In the context of Gaussian Process-based Nonlinear Model Predictive Control, efficient numerical optimization and real-time feasibility are critical considerations. To systematically evaluate the computational performance and implementation requirements of GP-NMPC, two advanced software toolboxes, LbMATMPC and L4acados, will be employed as benchmark platforms. By analyzing these toolboxes, the study aims to assess their computational demands, solver efficiency, and suitability for executing a successful closed-loop control sequence, particularly in the presence of learned Gaussian Process models. This comparative analysis will provide valuable insights into the trade-offs between flexibility, computational overhead, and real-time applicability in GP-NMPC implementations.

3.1 LbMATMPC

LbMATMPC [47] is a Matlab-based, open-source toolbox for integrating Gaussian Processes as a way of modeling the unknown or uncertain dynamics of a dynamical system through Gaussian Process Regression into the Non-Linear Model Predictive Control framework. This integration allows for the combination of a nominal model of system dynamics with a learning-based model which can be used for either computing GP-based corrections in a grey-box fashion or a complete black-box modelling of the studied system. The exploitation of the data gathered from the system evolution could then greatly increase

the MPC reliability and performance, which are heavily dependent on the accuracy of the model description. This toolbox is designed as an extension of MATMPC [48], a Matlab-based software framework designed for Non-Linear Model Predictive Control. MATMPC was developed to be a state-of-the-art real-time NMPC solver, providing an easy to use and flexible algorithm and model testing framework for NMPC applications due to being directly implemented in Matlab code. MATMPC is designed for real-time NMPC implementation, achieving high computational efficiency by leveraging Matlabs C API for time-critical components. Its modular architecture offers key functionalities such as automatic differentiation, direct multiple shooting, and Real-Time Iterations (RTI): an efficient approach that reduces computational load by performing only a single optimization iteration per control step. MATMPC efficiently handles NMPC formulations through direct multiple shooting and integrates with high-performance QP solvers like HPIPM [49] for fast computations. It supports real-time execution via warm-starting and structure-exploiting techniques, further enhancing efficiency. Additionally, its seamless integration with Matlabs ecosystem allows users to easily define and test control problems.

3.1.1 Control Problem Definition

In LbMATMPC the nominal dynamics are numerically integrated with Explicit Runge-Kutta ϕ, the next step state evolution for a grey-box model of the system dynamics can be expressed as:

$$x_{k+1} = \phi(x_k, u_k) + \psi(x_k, u_k) \tag{3.1}$$

Where x is the state vector, u is the control inputs, and ψ are the GP predictions. Additionally, we can encode constraints, later referenced as r. At every discrete time instant k a NLP is formulated by applying direct multiple shooting to an OCP, the prediction horizon is then divided into N shooting intervals. The cost function Equation 3.2 minimizes a weighted sum of squared constraint violations, where $h_j(x_{j|k}, u_{j|k})$ and $h_N(x_{N|k})$ represent stage and terminal constraints, respectively. The weight matrices W and W_N ensure a penalty on constraint deviations. The initial condition Equation 3.3 enforces that the initial state $x_{0|k}$ matches the given reference $\bar{x}_{0|k}$, ensuring consistency in the optimization.

$$\min_{x,u} \sum_{j=0}^{N-1} \frac{1}{2}\|h_j(x_{j|k}, u_{j|k})\|^2_W + \frac{1}{2}\|h_N(x_{N|k})\|^2_{W_N} \tag{3.2}$$

$$s.t. \quad 0 = x_{0|k} - \bar{x}_{0|k} \tag{3.3}$$

$$0 = x_{j+1|k} - \left[\phi(x_{j|k}, u_{j|k}) + \psi(x_{j|k}, u_{j|k})\right], \quad j = 0, 1, \ldots, N-1 \tag{3.4}$$

$$\underline{r}_{j|k} \leq r(x_{j|k}, u_{j|k}) \leq \bar{r}_{j|k}, \quad j = 0, \ldots, N-1. \tag{3.5}$$

$$\underline{r}_{N|k} \leq r_N(x_{N|k}) \leq \bar{r}_{Nk} \tag{3.6}$$

The state dynamics Equation 3.4 are described by a discrete-time transition equation, where $\phi(x_{j|k}, u_{j|k})$ represents the nominal system evolution, while $\psi(x_{j|k}, u_{j|k})$ introduces an additional correction term, possibly modeling uncertainties or learned dynamics. The state and input constraints Equation 3.5 enforce bounded feasibility regions for the function $r(x_{j|k}, u_{j|k})$ at each stage and for $r_N(x_{N|k})$ at the terminal stage, ensuring the system operates within predefined limits. Despite its computational efficiency, the control performance of MATMPC heavily depends on the fidelity of the system model. Inaccurate models or unmodeled dynamics can degrade controller performance, leading to suboptimal control decisions. To address this, LbMATMPC extends MATMPC by incorporating Gaussian Process Regression (GPR), enabling data-driven learning of system dynamics. In the context of LbMATMPC (see [47]) the output prediction of a Gaussian Process is:

$$y_k^i = \dot{q}_k^i - \hat{\dot{q}}_k^i = \bar{\psi}_{\dot{q}}^i(\tilde{x_k}) + e_k^i \tag{3.7}$$

The given GP prediction models the discrepancy between the true and estimated system dynamics. The term y_k^i denotes the residual error between the actual time derivative of the state $\dot{q}_k^i$ and its estimated counterpart $\hat{\dot{q}}_k^i$. The function $\bar{\psi}_{\dot{q}}^i(\tilde{x_k})$ represents the GP mean prediction of this discrepancy based on the input feature vector $\tilde{x}_k$, which typically includes system states and controls. Lastly, e_k^i accounts for the residual uncertainty, capturing stochastic variations and model errors not represented by the GP mean function. LbMATMPC leverages MATMPC for solving NMPC problems while utilizing PyTorch, a Python-based Machine Learning library, for learning the system dynamics. This combination enables learning model uncertainties by training GPs on residual dynamics data in a grey-box fashion or the full system dynamics in a black-box fashion.

3.1.2 Control Loop Implementation - LbMATMPC

The LbMATMPC control loop follows these steps: at each time step, the system state is measured, the NMPC solver computes the OCP solution, the control input is applied, and the system evolves to a new state. Both the control loop and the NMPC solver are implemented in Matlab, with the OCP being solved using in this particular setup Sequential Quadratic Programming [32] with Real-time iterations [33]. In LbMATMPC, an SQP-RTI step performs the following phases:

1. **QP Preparation (RTI phase 1):**
   - Integrate the system dynamics using Explicit Runge-Kutta solver
   - Compute the GP predictions
   - Correct the system dynamics
   - Compute the QP formulation

2. **Feedback (RTI phase 2):**
   - Obtain the QP problem solution using the chosen solver (qpoases, hpipm, ...)

3. **Convergence Check:**
   - Compute the Karush-Kuhn-Tucker (KKT) conditions and residuals to check for optimality and feasibility

To ensure computational efficiency all of these different sub-phases of an SQP-RTI step are computed by utilizing a MEX compilation of CasADi functions [50] for the model, and only MEX files for other components. MEX-compiled CasADi's functions for MATLAB provide an efficient interface for solving nonlinear optimization and optimal control problems by generating highly optimized C++ code that is compiled into MEX executables. This approach enables seamless integration with MATLAB while leveraging just-in-time compilation for significantly faster execution compared to pure MATLAB implementations. By exploiting algorithmic differentiation and sparse linear algebra, CasADi efficiently computes gradients and Hessians, reducing computational overhead in large-scale optimization problems. This makes it particularly well-suited for real-time applications where speed and numerical accuracy are critical

such as the one studied in this work. The integration of a PyTorch Gaussian Process model into this control loop is not trivial, as explained in subsection 2.3.1, the computation of a GP model mean prediction requires kernel computations. The training of this Gaussian Process model is conducted offline, prior to the execution of the control loop. Once the model is trained, the relevant data characterizing the learned distribution is extracted and stored locally. Assuming a zero-mean prior ($\mu_x = 0$), the information required to compute the GP predictive mean, as given in Equation 2.8, includes the covariance vector between the test point and the training set, the covariance matrix of the training data, the output variance, and the training targets. While the covariance vector between the test point and the training set must be computed online for each new prediction, the remaining components of Equation 2.8 are independent of the test point and can therefore be efficiently precomputed offline. These precomputed values are combined into a single component, referred to as the alpha value. In the specific case of LbMATMPC, these precomputed values are loaded and used to generate a MEX-compiled function that efficiently performs the Gaussian Process predictive mean computation, given by:

$$\mu(x_\star) = K_{x_\star x}\alpha \tag{3.8}$$

LbMATMPC represents a significant advancement in the field of Learning-Based NMPC, providing a practical and efficient approach for integrating Gaussian Processes into real-time control frameworks. By extending MATMPC with data-driven modelling capabilities, LbMATMPC enables adaptive and accurate control in uncertain and nonlinear environments. Experimental evaluations using LbMATMPC have demonstrated its effectiveness in various control tasks, including the Furuta pendulum and the inverted cart-pole system, up to a real go-kart [51]. Results indicate a significant reduction in prediction errors and improved control performance compared to traditional NMPC approaches [47].

3.2 L4acados: Learning-Based Models for ACADOS

L4acados is an open-source framework designed to integrate learning-based residual models into Acados for Nonlinear Model Predictive Control. Unlike CasADi-based approaches, which rely on symbolic computation, non-CasADi learning-based residual models use external machine learning frameworks (e.g.,

PyTorch or TensorFlow) to capture unknown system dynamics. This modular setup allows for flexible customization of Gaussian Process-based NMPC algorithms while leveraging Acados' efficient optimization capabilities.

3.2.1 ACADOS and Zero-Order GP-MPC Implementation

L4acados originates from the prior research conducted by the Dynamic Systems and Control group at ETH Zurich on Zero-Order Robust Optimization for Gaussian Process-based Model Predictive Control [52]. Built upon the Acados framework [53], L4acados extends these methodologies to enable efficient, real-time nonlinear MPC with data-driven uncertainty quantification. Acados is a very popular open-source software package designed for fast and reliable solutions to optimal control problems [53]. It provides a collection of solvers and interfaces tailored for efficient NMPC and moving horizon estimation. A notable feature of Acados is its modularity, allowing users to customize and extend its functionalities to accommodate various control applications. Acados leverages CasADi to conveniently formulate control problems in one of the high-level interfaces, namely Python, Matlab and Octave. Just like MATMPC and its learning-based derivation, Acados utilizes state-of-the-art solvers like HPIPM [49] and QPOASES [54] while also relying on the high-performance linear algebra package BLASFEO [55].

In the context of learning-based control, the integration of GPs into MPC frameworks offers a data-driven approach to model system dynamics and uncertainties. The confidence of a prediction is a key aspect that is at the core of Gaussian Processes, this aspect can be leveraged in a control system context to enable an uncertainty-aware data-driven control method. However, incorporating GPs into an NMPC framework poses significant computational challenges, primarily due to the $O(n^2)$ complexity raising from the covariance propagation (see subsection 2.3.1). To address these challenges, the zero-order robust optimization (zoRO) was conceived. The zoRO algorithm mitigates the computational burden by efficiently propagating state covariances through a zero-order approximation between iterations of the Newton-type optimization method used to solve the OCP [52]. This approach avoids the need to include covariance propagation within the optimization problem itself, thereby reducing the number of optimization variables and enhancing computational efficiency. The implementation of zero-order GP-MPC in Acados has demonstrated successful applica-

tions in scenarios such as obstacle avoidance and friction-adaptive autonomous driving [52], where real-time performance is critical. The code implementation of the tests from [52] was re-implemented, and despite slight value differences that are a result of running these experiments on different hardware, the results were confirmed. Most notably the GPU-accelerated evaluations from these results are what sparked the start of this thesis work. Acados offers a robust platform for NMPC, but integrating non-CasADi learning-based models, such as GPyTorch Gaussian Processes or PyTorch neural networks, requires extra interfacing as Acados is primarily designed to work with CasADi-based models. To address this, L4acados has been developed as a versatile framework that simplifies the integration of learning-based residual models into Acados [56]. Building up on the groundwork of the zoRO algorithm , L4acados, enables users to define custom regression models in Python and seamlessly integrate them into the Acados optimization pipeline by exploiting the already proven PyTorch and GPyTorch libraries. By providing the required sensitivities through a user-defined Python module, L4acados supports the implementation of MPC controllers with learning-based residual models. This framework also allows for custom Jacobian approximations and parallelization of sensitivity computations during the preparation of quadratic subproblems, thereby enhancing computational efficiency. The versatility of L4acados has been demonstrated through various applications, including autonomous miniature racing and motion control of simulated autonomous vehicles performing complex maneuvers [56]. In comparison, L4CasADi is a different framework designed to bridge the gap between deep learning models, particularly those constructed in PyTorch, and the numerical optimization environment provided by CasADi [57]. By facilitating the use of learned process models within CasADi, L4CasADi broadens the scope of applications that can benefit from data-driven modelling in optimization problems. The applicability of L4CasADi has been illustrated through examples [57] such as optimizing a fish's trajectory in a turbulent river, where the flow dynamics are represented by a PyTorch model, and employing implicit Neural Radiance Field (NeRF) environment representations for optimal control tasks. While both frameworks aim to integrate learning-based models into optimization problems, L4CasADi leverages CasADi environment for seamless integration and hardware acceleration, whereas L4acados builds upon Acados, offering enhanced computational efficiency through custom sensitivity computations and parallelization strategies.

3.2.2 Control Loop Implementation - L4acados

L4acados lets the user implement their own main control loop, requiring them to provide a trained GP residual model and a AcadosOCP object containing the model non-linear dynamics definition and the OCP specifics. These two inputs will be used to create a L4acados class that will automatically modify the OCP and transform the nonlinear model dynamics into Linear Time-Varying (LTV) dynamics. A general LTV system can be described with the following state-space equations:

$$x_{k+1} = A_k x_k + B_k u_k + c_k \tag{3.9}$$

Where, differently from the constant matrices A and B for an LTI system, A_k is the time-varying state transition matrix, B_k is the time-varying input matrix and c_k is an optional time-dependent affine term. This is the same approach that is taken in the QP problem formulation step of LbMATMPC [47]. The affine dynamics OCP is then defined as:

$$\min_{u_0,\dots,u_{N-1},\mu_0^x,\dots,\mu_N^x} \sum_{k=0}^{N-1} l(\mu_k^x, u_k) + l_f(\mu_N^x) \tag{3.10}$$

$$\text{s.t.} \quad \mu_0^x = \bar{x}_0 \tag{3.11}$$

$$\mu_{k+1}^x = \hat{A}_k \mu_k^x + \hat{B}_k u_k + \hat{c}_k, \quad k = 0, \dots, N-1 \tag{3.12}$$

$$0 \geq h(\mu_k^x, u_k) + \tilde{\beta}_k, \quad k = 0, \dots, N-1 \tag{3.13}$$

$$0 \geq h_N(\mu_N^x) + \tilde{\beta}_N \tag{3.14}$$

The given OCP formulation consists of several key components. The objective function (3.10) aims to minimize the cumulative stage cost $l(\mu_k^x, u_k)$ and the terminal cost $l_f(\mu_N^x)$, balancing control effort and state deviation. The initial condition (3.11) enforces that the initial state mean is fixed at $\bar{x}_0$, ensuring a well-defined starting point. The system dynamics (3.12) follow an affine model characterized by the matrices $\hat{A}_k$, $\hat{B}_k$ and an affine term $\hat{c}_k$, which facilitates efficient optimization. State and input constraints (3.13) impose feasibility conditions through the function $h(\mu_k^x, u_k)$, while the back-off term $\tilde{\beta}_k$ accounts for uncertainties. Finally, the terminal constraint (3.14) ensures that the final state satisfies prescribed conditions, incorporating robustness via $\tilde{\beta}_N$. Backoff refers to a margin added to constraints to account for model uncertainties, disturbances, and numerical approximations, ensuring constraint satisfaction in the presence of prediction

errors. In the current implementation at every iteration L4acados follows the following phases:

1. **QP Preparation (RTI Phase 1):**
    - Computes the linearization of the system
    - Computes the GP sensitivities for the residual model
    - Integrates the nominal model utilizing the AcadosSimSolver from the nominal OCP
    - Computes the GP corrected linearized dynamics
    - Update the transformed OCP solver parameters

2. **Feedback (RTI Phase 2):**
    - Solve the QP using the AcadosOcpSolver from the transformed OCP

3. **Convergence Check:**
    - Check termination conditions and residuals to check for optimality and feasibility

During the preparation phase, sensitivity computations are carried out before the next initial state, $\bar{x}_0$, becomes available. Once this state is received, the process transitions to the feedback phase, where the precomputed quadratic program is solved, and the first control input is applied to the actual system. Following this same division, the approach implemented in L4acados can be interpreted as executing (parts of) the sensitivity computations during the RTI preparation phase, but externally to the Acados framework. Regarding the implementation of the control loop, in the Python interface, most components and SQP-RTI steps are computed using Python code. However, the AcadosOcpSolver, which computes the solution of the transformed OCP, and the AcadosSimSolver, which integrates the nominal system dynamics, can be compiled into efficient C code for improved performance. In Acados, the use of Cython [58] to compile the AcadosSimSolver and AcadosSimSolver significantly enhances computational efficiency by generating optimized C code for numerical optimization routines. This approach reduces the overhead associated with Python function calls, enabling faster execution of nonlinear optimization and simulation tasks. By leveraging Cythons ability to interact seamlessly with low-level C structures, Acados achieves real-time performance suitable for embedded and high-performance applications.

The L4acados implementation appears to be less computationally efficient compared to LbMATMPC, as most of LbMATMPC's SQP-RTI steps are precompiled into MEX functions before simulation. In general, Python is less computationally efficient than MEX-compiled CasADi C functions in MATLAB, since the latter benefit from highly optimized, low-level C code for numerical computations [50]. MEX functions enable direct execution of compiled code within MATLAB, reducing interpretation overhead and improving execution speed, particularly for computationally intensive tasks like nonlinear optimization. In comparison to standard MATLAB code, Python's efficiency depends on implementation details. While MATLABs built-in vectorized operations are highly optimized, naive Python implementations relying on loops can be significantly slower. However, when leveraging NumPy, Numba, or JIT compilation, Python can achieve performance comparable to or even exceeding standard MATLAB scripts.

Regarding the main observation aspect of this study, the GP predictions in L4acados are computed utilizing functions derived from the standard GPyTorch implementation whereas in LbMATMPC, as mentioned in subsection 3.1.2 the model prediction are performed through precomputed MEX functions.

The computational efficiency differences between these two approaches are one of the focus points of this work.

4 Results

The main goal of this work is to analyse the computational behaviours of the Gaussian Process-based Nonlinear Model Predictive Control methods which have been discussed in chapter 3, namely those of the two different software toolboxes LbMATMPC and L4acados. To study these differences and analyze the results a Toy problem is implemented, namely that of a Cart-Pole system.

4.1 The Cart-Pole system

The cart-pole system has consistently been recognized as a fundamental benchmark in control theory, attributed to its intricate amalgamation of characteristics that present significant challenges whilst remaining adaptable to advanced control approaches. A principal rationale for its significance is the fundamentally nonlinear nature of its dynamics. These nonlinearities introduce a complex control problem, particularly for methodologies such as nonlinear model predictive control, which necessitate precise modeling and forecasting to attain optimal performance. Additionally, the system is underactuated, possessing more degrees of freedomspecifically, the cart's position and the pendulum's anglethan actuators, as only the cart is under direct control. This underactuation demands advanced control algorithms capable of managing the coupled dynamics between the cart and the pendulum.

The control tasks associated with the cart-pole system further highlight its utility as a benchmark. It encompasses two fundamental and widely studied objectives in control theory: swing-up control and stabilization. Swing-up con-

trol involves transitioning the pendulum from its natural hanging position to an upright configuration, which requires precise trajectory planning to overcome the systems nonlinearities and underactuation. Stabilization, on the other hand, focuses on maintaining the pendulum in the upright position, counteracting external disturbances and uncertainties in the system model. These tasks encapsulate key challenges in control, such as trajectory generation and disturbance rejection, which make the cart-pole system an ideal platform for testing control methodologies.

Another significant aspect of the cart-pole system is its sensitivity to model inaccuracies. Variations in parameters such as the length or mass of the pendulum can significantly alter its dynamics, introducing a layer of uncertainty that must be accounted for in the control design. This sensitivity makes the system an excellent candidate for evaluating learning-based approaches, such as Gaussian Process Regression-augmented NMPC, which explicitly model and compensate for discrepancies between the nominal model and the true system behavior.

Finally, the cart-pole system is well-suited for studies of computational scalability, particularly in the context of integrating Gaussian Processes to manage model uncertainties. As the size of the dataset used for GPR increases, the computational burden grows significantly, posing a challenge to achieving real-time control. This makes the cart-pole system an ideal testbed for examining the performance improvements offered by GPU acceleration, which can offset the computational overhead associated with large datasets. In conclusion, the cart-pole system, with its amalgamation of nonlinear dynamics, underactuation, intricate control objectives, sensitivity to model uncertainties, and scalability demands, serves as an exemplary benchmark for the assessment of advanced control methodologies like the one tested in this work.

4.1.1 Cart-pole system: Equations of motion

The cart-pole system is an underactuated, nonlinear system, where the cart is controlled by a horizontal force F, but the pendulum's motion is influenced indirectly through the dynamics of the cart. A Lagrangian approach is commonly used to derive the equations of motion of the cart-pole system in which the task is to balance a simple free pendulum around its unstable equilibrium point. The cart-pole system consists of: a cart of mass M that moves freely along

a horizontal track and a pendulum of mass m and length l attached to the cart via a frictionless pivot point. A force F is applied to the cart horizontally parallel to the track allowing the movement of the cart via the generated acceleration. To begin the parametrization of this system the following state variables are defined:

- x: Position of the cart along the track.
- $\dot{x}$: Velocity of the cart.
- θ: Angle of the pendulum from the vertical.
- $\dot{\theta}$: Angular velocity of the pendulum.

The control task of the swing-up of this system consists in moving the system from the downward stable point position ($\theta = \pi$) to the upward unstable point position ($\theta = 0$) using only horizontal forces on the cart as a control input.

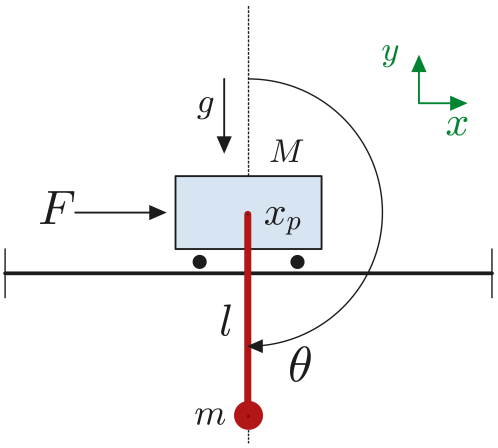

Figure 4.1:
Cart-Pole downward position

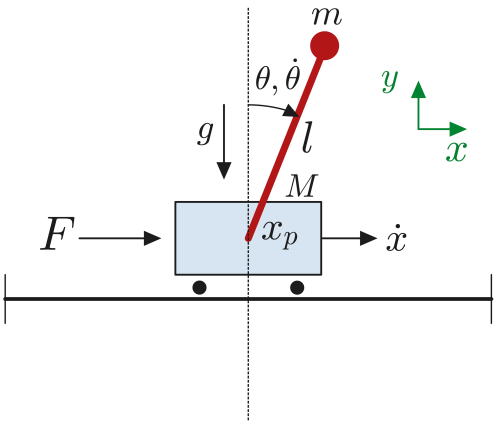

Figure 4.2:
Cart-Pole upward position

Kinetic and Potential Energy Contributions

To derive the equations of motion, we start by calculating the kinetic energy (T) and the potential energy (U) of the different components of the system:

Kinetic Energy of the Cart: The cart has a mass M and moves horizontally with a velocity $\dot{x}$. Its kinetic energy is straightforwardly given by:

$$T_{\text{cart}} = \frac{1}{2} M \dot{x}^2$$

Kinetic Energy of the Pendulum: The pendulum, treated as a point mass m located at a distance ℓ from the pivot, has both translational and rotational kinetic energy components. Its velocity is derived from the motion of the cart and the pendulum's own angular velocity $\dot{\theta}$. The total kinetic energy of the pendulum is expressed as:

$$T_{\text{pendulum}} = \frac{1}{2}m\left[(\dot{x} - \ell\dot{\theta}\cos\theta)^2 + (\ell\dot{\theta}\sin\theta)^2\right]$$

This expression accounts for both horizontal and vertical velocity components, arising from the coupled motion of the cart and pendulum. After simplifying, the **total kinetic energy** becomes:

$$T = \frac{1}{2}(M+m)\dot{x}^2 + \frac{1}{2}m\ell^2\dot{\theta}^2 - m\ell\dot{x}\dot{\theta}\cos\theta$$

Potential Energy of the Pendulum: The potential energy of the pendulum is due to gravity. Assuming that the vertical height of the cart is the reference point ($U = 0$), the pendulum height is proportional to $\ell\cos\theta$. Thus, its potential energy is:

$$U = mg\ell\cos\theta$$

Formulating the Lagrangian

The Lagrangian is defined as:

$$L = T - U$$

Substituting the expressions for T and U obtained above, the Lagrangian finally becomes:

$$L = \frac{1}{2}(M+m)\dot{x}^2 + \frac{1}{2}m\ell^2\dot{\theta}^2 - m\ell\dot{x}\dot{\theta}\cos\theta - mg\ell\cos\theta$$

Applying the Euler-Lagrange Equation

The dynamics of the system are derived from the Euler-Lagrange equation, which is expressed for each generalized coordinate q_i (in this case x and θ) as:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = Q_i$$

where Q_i represents the generalized forces acting on the system.

For x (horizontal cart motion): The generalized force is the externally applied horizontal force F:

$$Q_x = F$$

Substituting into the Euler-Lagrange equation for x:

$$F = \frac{d}{dt}\left[(M+m)\dot{x} - m\ell\dot{\theta}\cos\theta\right] - \left[-m\ell\dot{\theta}^2\sin\theta\right]$$

Expanding and simplifying gives:

$$F = (M+m)\ddot{x} - m\ell\ddot{\theta}\cos\theta + m\ell\dot{\theta}^2\sin\theta$$

For θ (angular pendulum motion): The pendulum experiences no direct torque from the applied force F, so $Q_\theta = 0$. Substituting into the Euler-Lagrange equation for θ:

$$\frac{d}{dt}\left[m\ell^2\dot{\theta} - m\ell\dot{x}\cos\theta\right] - \left[m\ell\dot{x}\dot{\theta}\sin\theta + mg\ell\sin\theta\right] = 0$$

Expanding and simplifying gives:

$$m\ell^2\ddot{\theta} - m\ell\ddot{x}\cos\theta - mg\ell\sin\theta = 0$$

Coupled Dynamics

The two equations are coupled, as $\ddot{x}$ and $\ddot{\theta}$ depend on each other. To isolate the two terms we must first solve the equation for x to express $\ddot{x}$ in terms of $\ddot{\theta}$ and finally substitute this into the equation for θ. After algebraic manipulation, the final equations of motion obtained are the following.

1. Cart Acceleration:

$$\ddot{x} = \frac{-m\ell\sin\theta\dot{\theta}^2 + mg\cos\theta\sin\theta + F}{M + m - m\cos^2\theta} \tag{4.1}$$

2. Pendulum angular acceleration:

$$\ddot{\theta} = \frac{-m\ell\cos\theta\sin\theta\dot{\theta}^2 + F\cos\theta + (M+m)g\sin\theta}{\ell(M + m - m\cos^2\theta)} \tag{4.2}$$

These equations highlight the non-linear, coupled nature of the cart-pole system: The cart's motion ($\ddot{x}$) is influenced by the pendulum's angle (θ), angular velocity ($\dot{\theta}$), and angular acceleration ($\ddot{\theta}$). The pendulum's motion ($\ddot{\theta}$)

is influenced by the cart's motion and the external force F. The term $m\cos^2\theta$ in the denominator reflects the coupling between translational and rotational dynamics. The final step involves expressing these equations in a state-space representation, which is particularly useful for control applications, simulation, and numerical integration. This process requires rewriting the system of second-order differential equations in terms of first-order equations, thereby facilitating their use in computational methods. To achieve this transformation, it is first necessary to define a state vector that encapsulates all relevant system variables. A common choice for the cart-pole system is the following four-dimensional state vector:

$$\mathbf{x} = \begin{bmatrix} x & \theta & \dot{x} & \dot{\theta} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \tag{4.3}$$

where x represents the position of the cart, $\dot{x}$ its velocity, θ the angular displacement of the pendulum, and $\dot{\theta}$ the angular velocity of the pendulum. Given this state representation, the system dynamics must be reformulated in terms of first-order differential equations. The first two state equations follow directly from the definitions of velocity and angular velocity:

$$\dot{x}_1 = x_3, \quad \dot{x}_2 = x_4 \tag{4.4}$$

The remaining equations are obtained by substituting the given expressions for the cart acceleration and pendulum angular acceleration into the definitions of $\dot{x}_3$ and $\dot{x}_4$:

$$\dot{x}_3 = \frac{-m\ell \sin x_2 x_4^2 + mg \cos x_2 \sin x_2 + F}{M + m - m\cos^2 x_2} \tag{4.5}$$

$$\dot{x}_4 = \frac{-m\ell \cos x_2 \sin x_2 x_4^2 + F\cos x_2 + (M+m)g \sin x_2}{\ell(M + m - m\cos^2 x_2)} \tag{4.6}$$

Having rewritten the system in first-order form, it is now possible to express it in the general nonlinear state-space form:

$$\dot{\mathbf{x}} = f(\mathbf{x}, u) \tag{4.7}$$

where $u = F$ denotes the external force applied to the cart. This representation is particularly advantageous for numerical integration methods, including Runge-Kutta schemes, and facilitates implementation in modern control frameworks such as Nonlinear Model Predictive Control. This structured representation

provides a foundation for both theoretical analysis and practical implementation in real-time control systems.

4.2 Testing platform and methodology

This section outlines the setup of the computational experiments, detailing the framework used to evaluate the numerical efficiency, scalability, and real-time feasibility of each approach. The testing ground is carefully designed to ensure consistency in environmental conditions, solver configurations, and evaluation metrics, thereby enabling a fair comparison. Key aspects include the formulation of benchmark scenarios, parameter selection, and performance metrics that capture both solution accuracy and computational cost. By standardizing these elements, the study aims to provide an objective assessment of the trade-offs inherent in different GP-MPC implementations. To start, all of the tests were performed on the same hardware platform which in this case is a computer laptop ASUS ROG G513QM. The hardware components of this system are:

- CPU: AMD Ryzen 9 5900HX
- GPU: NVIDIA GeForce RTX 3060 Laptop (6 GB VRAM)
- RAM: 16 GB DDR5

To ensure consistent hardware performance all tests where computational performance was measured were run with: high power profile, balanced fan mode, laptop charge plugged in, no external monitors and no other applications open other than Visual Studio Code for L4acados and Matlab for LbMATMPC on Windows 11. To ensure the correct installation of the two software, the tests performed in their own literature were carried out making sure that the results were comparable to the ones present in literature [52, 47].

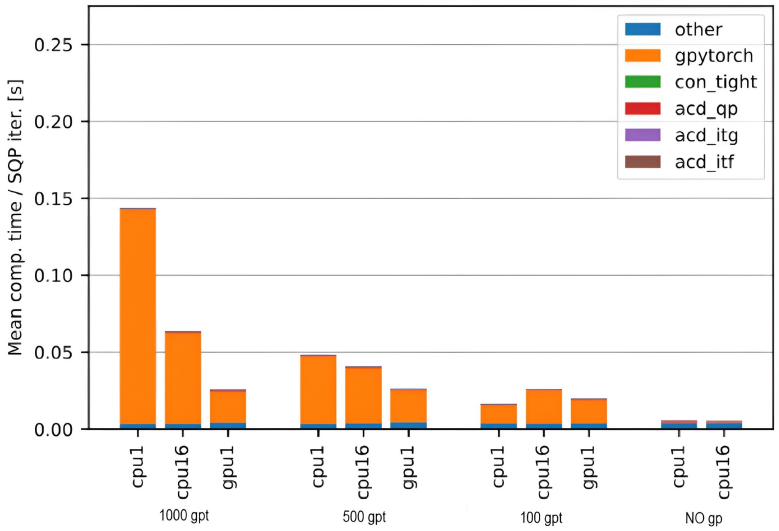

Figure 4.3: Timing results from running the zoRO algorithm in a simple pendulum swing-up simulation with different values of GP training points [0, 100, 500, 1000].

For example in Figure 4.3 are present the results obtained from running the simple pendulum code from [52] examples for different amounts of data points [1000, 500, 100, 0]. The simple pendulum is a dynamical system consisting of a rigid, massless rod of fixed length, with a point mass attached at one end, oscillating under the influence of gravity about a fixed pivot, where the goal is that of stabilizing the rod from the downright to the upright position with a swing-up motion. The results here confirm the trends noticed in [52] where GPU acceleration shows better performance for large data sets. Creating the correct installation for these two toolboxes requires the use of different compilers and subsystems as both of these software require Linux-based dependencies such as BLASFEO and HPIPM. While LbMATMPC runs natively on Windows through Matlab, L4acados and Acados in general require installation through Windows WSL. To assess any possible performance difference a set of tests were run on both Linux and Windows 11 and no notable performance differences were found. Once both software were installed and tested the set of tests could be out. Three main configurations were identified for testing:

- LbMATMPC - CPU
- L4acados - CPU
- L4acados - CPU + GPU GP prediction acceleration

To analyze the different behaviors and tradeoffs of these configurations it was decided to keep the same test problem and vary the amount of GP data for the different tests. This was chosen so to have a focused comparison on the computational differences in the GP prediction. Finally, GP datasets for testing were chosen with a varying number of training samples: [50, 100, 200, 500, 1000].

4.3 Optimal Control Problem setup

The optimal control problem for the cart-pole system is formulated to achieve stable and efficient control while accounting for model discrepancies between a nominal and a real system. The goal of the control strategy is to stabilize the pole in its upright position while minimizing control effort and tracking deviations. The nominal model assumes a pendulum length of $l = 0.8$ m, whereas the real system has a shorter pendulum length of $l = 0.5$ m.

This discrepancy induces variations in the system dynamics, which the Gaussian Process model is subsequently trained to capture and compensate for. Specifically, in this test case, the GP learning framework is configured with training inputs consisting of the system state $\mathbf{x}$ and control input u, while the target outputs correspond to the velocity deviations $\Delta\dot{x}$ and $\Delta\dot{\theta}$. These deviations represent the differences in the cart's velocity and angular velocity between the nominal and actual system dynamics. The GP-predicted corrections are then incorporated into the control formulation to enhance robustness against model uncertainties. The cost function used in the OCP consists of a quadratic tracking term weighted and regularized, ensuring a trade-off between accurate tracking and energy efficiency and it is given by:

$$J = \sum_{k=0}^{N-1} \left(\mathbf{x}_k^\top Q \mathbf{x}_k + \mathbf{u}_k^\top R \mathbf{u}_k \right) + \mathbf{x}_N^\top Q_f \mathbf{x}_N \tag{4.8}$$

In this expression, the cost function J represents the objective to be minimized over the prediction horizon N. The term $\mathbf{x}_k \in \mathbb{R}^n$ denotes the system state at time step k, while $\mathbf{u}_k \in \mathbb{R}^m$ corresponds to the control input at the same time step. The matrices $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{n \times m}$ are symmetric, positive semi-definite weight matrices that penalize deviations in the state trajectory and excessive control effort, respectively. Additionally, the final term $\mathbf{x}_N^\top Q_f \mathbf{x}_N$ introduces a terminal cost, where Q_f is typically selected as a function of Q to ensure sta-

bility and optimal convergence properties. For the specific NMPC setup under consideration, the state cost matrix is defined as: $Q = \text{diag}(10, 10, 0.001, 0.001)$ The weight matrix Q assigns higher penalties to deviations in cart position x and pole angle θ, emphasizing the importance of maintaining the cart's position and stabilizing the pendulum angle. In contrast, smaller weights are assigned to the velocity terms $\dot{x}$ and $\dot{\theta}$, implying that transient variations in these states are tolerated to some extent. The control cost matrix is given by: $R = 0.01$ where the control input consists of a single force applied to the cart. The relatively small weight on R suggests that moderate control efforts are permitted to regulate the system effectively without overly restricting force application. Additionally, the terminal cost matrix was set to be equal the standard weight matrix: $Q_f = Q$ This formulation ensures that the NMPC controller prioritizes the stabilization of the pendulum in the upright position and accurate tracking of the cart's position while balancing the trade-off between control effort and transient dynamics. This allows the system to achieve the swing-up maneuver with more rapid movement, placing a heavier cost on the velocity component would result in a less jerky and slower movement. The lower weight of the control input let the system utilize more energy to reach the goal state, setting this weight too could hinder the ability of the system to perform the maneuver at all as the total close-loop cost could be lower by not making any movement. The prediction horizon ($N = 80$) specifies the number of future time steps considered during optimization, directly influencing the controller's foresight and computational complexity. The discretization time step ($dt = 0.01$) seconds defines the temporal resolution of the control updates, affecting numerical accuracy and solver performance. The state constraints impose position limits on the system, with the state variable x bounded between [2,2] meters, ensuring the system operates within predefined spatial constraints. Similarly, the control input u bounds restrict the applied force (F) to the range [50,50] Newtons, preventing excessive actuation and ensuring feasibility within actuator limits. The key parameters defining the OCP formulation for the test simulations are displayed in the following Table 4.1.

Parameter	Value	Description
N	80	Prediction horizon, defines the number of future time steps considered in optimization.
dt	0.01 s	Discretization time step, determines the time interval between control updates.
T_{sim}	5 s	Simulation time
$\mathbf{x}$	$\begin{bmatrix} x & \theta & \dot{x} & \omega \end{bmatrix}^T$	System state, including cart position, velocity, pole angle, and angular velocity.
lb_x	-2 m	Lower bound on state position
ub_x	2 m	Upper bound on state position
u	F	Control input, representing the force applied to the cart.
lb_u	-50 N	Lower bound on control input
ub_u	50 N	Upper bound on control input
Q	diag(10, 10, 0.001, 0.001)	State cost matrix, penalizing deviations from the desired state.
R	diag(0.01)	Control cost matrix, penalizing excessive control effort.
x_{ref}	$\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$	Reference trajectory for tracking, the upright equilibrium state.
l (nominal)	0.8 m	Pendulum length assumed in the nominal model.
l (real)	0.5 m	Actual pendulum length in the real system.
GP Model	–	Learns the velocity deviations $(\Delta\dot{x}, \Delta\dot{\theta})$ between the nominal and real model.
QP Solver	HPIPM	QP solver using partial condensing to balance problem size reduction and computational efficiency in NMPC.
NLP Solver	SQP RTI	Solves NMPC by linearizing once per step, solving a QP, and applying immediately.

Table 4.1: Key Parameters of the OCP Setup, common for both LbMATMPC and L4acados.

4.4 GP MODEL TRAINING

Another way of ensuring a fair computational performance comparison is to ensure the two toolboxes behave similarly when it comes to GP predictions. To accomplish this goal a single dataset was created and utilized to train both a GPyTorch model in the case of L4acados, and two exact GP PyTorch model that are implemented in LbMATMPC. Utilizing a single dataset to train two different GP models ensures that both models are exposed to identical training distributions, thereby isolating the effects of the framework in which they are implemented. This approach guarantees that any observed differences in performance, accuracy, or computational efficiency arise solely from the framework-specific implementation details rather than variations in the learned underlying distribution.

The created dataset consists of a total of 10000 training samples, this samples where gathered by running twenty control sequence simulations in L4acados. Each simulation generated a total of 500 training samples with state and control input of each step representing the training inputs (train inputs = $[\mathbf{x_k}, u_k]$) and the difference between the next state predictions of the correct model and nominal model becoming the training targets (train targets = $[\mathbf{x_{k+1}^{real}} - \mathbf{x_{k+1}^{nom}}]$). To increase dataset variability and favor state-space exploration, randomized Gaussian noise was added to the control input u during simulation, promoting broader state space exploration.

From this large dataset, subsets of 50, 100, 200, 500, and 1000 samples were extracted using K-means clustering. K-means is an iterative unsupervised learning algorithm that partitions data into k clusters by minimizing within-cluster variance. In each iteration, the algorithm assigns data points to the nearest cluster centroid and then updates the centroids based on the mean of the assigned points. This process continues until the centroids stabilize, resulting in representative clusters that capture the essential variability of the data. In this approach, we first cluster the data into several clusters equal to the desired training subset size. However, since the cluster centroids lack associated labels, we then select, for each cluster, the sample closest to the centroid. This step ensures that each chosen sample is representative of its cluster while retaining the corresponding training label. In this way, we generate five distinct training subsets that effectively capture the variability present in the original dataset.

The GP models utilized in the L4acados variant is a single Multitask Exact Gaussian Process model capable of handling multiple output dimensions. It incorporates a zero-mean function and a covariance structure defined by a RBF kernel, wrapped within a scale kernel to introduce output-dependent variability. The model supports automatic relevance determination (ARD), allowing the length-scales to be learned independently for each input dimension when enabled. In LbMATMPC a single GP model with a constant mean function was created for every one of the two output dimensions, it's covariance is also defined by a RBF kernel and a single lenght-scale is learnt per input dimension.

In the context of Gaussian Process learning, various performance metrics are employed to evaluate the accuracy and reliability of the models predictions. One of the most commonly used metrics is the Mean Squared Error (MSE), which quantifies the average squared difference between the predicted and true values. This metric provides a measure of overall prediction accuracy, penalizing larger deviations more significantly due to the squaring operation, making it particularly sensitive to outliers. A lower MSE indicates a model that more precisely captures the underlying function governing the data:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.9}$$

where y_i represent the true output values, $\hat{y}_i$ represent the predicted values, $\bar{y}$ is the mean of the true observed values, n is the number of samples. Another relevant metric is the Mean Absolute Error (MAE), which represents the average absolute difference between the predicted and actual values. Unlike MSE, this metric does not square the errors, treating all deviations with equal weight. As a result, MAE provides a more interpretable measure of prediction error that is less sensitive to extreme outliers, making it useful in scenarios where large deviations should not disproportionately influence model assessment:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.10}$$

A higher-level indicator of model performance is the coefficient of determination (R^2), which evaluates how well the GP model explains the variance in the observed data. This metric compares the models predictive performance against a simple baseline that assumes the mean of the target variable as the best estimate:

$$\mathrm{R}^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4.11}$$

An R^2 value close to one suggests that the model explains most of the variance in the data, whereas a value near zero or negative implies that the model fails to provide meaningful predictive insight. In the context of GP learning, a high R^2 score indicates that the learned function successfully captures the underlying structure of the data, making it a crucial metric for assessing the overall effectiveness of the GP model. By jointly considering these metrics, a comprehensive evaluation of both predictive accuracy and model robustness can be achieved. The different GP model configuration were finally tested on a newly created test set consisting of 500 samples obtained from a single simulation. The results achieved for each output and GP configuration are displayed in Table 4.2 below. For a lower number of training points (50 GPT), both toolbox's models exhibit similar performance, with minimal differences in MAE and R^2 values. The MSE remains at an extremely low magnitude across all cases, indicating that the models maintain a high degree of accuracy even with a limited training dataset. However, as the number of training points increases to 100 GPT, a noticeable improvement in performance is observed. The MAE values significantly decrease for both $\dot{x}$ and $\dot{\theta}$, and the R^2 values approach unity, reflecting an enhanced predictive capability of the GP models. When the training dataset is further increased to 200 GPT, both toolboxes achieve near-perfect predictions, with MSE values effectively reaching zero and R^2 values exceeding 0.99995 across all outputs. This trend demonstrates the expected behavior that additional training data allows for better generalization and a reduction in prediction errors. Interestingly, at 500 GPT, a slight increase in MAE for both $\dot{x}$ and $\dot{\theta}$ is observed. This minor degradation may indicate overfitting effects or numerical artifacts in the training process. Finally, at 1000 GPT, the models maintain high performance, but the R^2 values exhibit a slight decrease for $\dot{\theta}$ compared to the 200 GPT configuration. This suggests that increasing the number of training points beyond a certain threshold may not necessarily yield further gains in predictive accuracy. Instead, diminishing returns appear to emerge, possibly due to model complexity and computational overhead. In comparing the GP implemented in the two toolboxes, these perform nearly identically across all tested configurations, with only marginal differences in MAE and R^2 values.

50 gpt	L4acados		LbMATMPC	
Output	$\dot{x}$	$\dot{\theta}$	$\dot{x}$	$\dot{\theta}$
MSE	0.000000	0.000001	0.000000	0.000001
MAE	0.000018	0.000745	0.000018	0.000724
R^2	0.999779	0.999747	0.999767	0.999800
100 gpt	L4acados		LbMATMPC	
Output	$\dot{x}$	$\dot{\theta}$	$\dot{x}$	$\dot{\theta}$
MSE	0.000000	0.000000	0.000000	0.000000
MAE	0.000013	0.000306	0.000014	0.000297
R^2	0.999866	0.999950	0.999868	0.999957
200 gpt	L4acados		LbMATMPC	
Output	$\dot{x}$	$\dot{\theta}$	$\dot{x}$	$\dot{\theta}$
MSE	0.000000	0.000000	0.000000	0.000000
MAE	0.000007	0.000127	0.000007	0.000123
R^2	0.999956	0.999979	0.999961	0.999985
500 gpt	L4acados		LbMATMPC	
Output	$\dot{x}$	$\dot{\theta}$	$\dot{x}$	$\dot{\theta}$
MSE	0.000000	0.000000	0.000000	0.000000
MAE	0.000022	0.000431	0.000021	0.000437
R^2	0.999877	0.999920	0.999889	0.999919
1000 gpt	L4acados		LbMATMPC	
Output	$\dot{x}$	$\dot{\theta}$	$\dot{x}$	$\dot{\theta}$
MSE	0.000000	0.000000	0.000000	0.000000
MAE	0.000015	0.000400	0.000015	0.000383
R^2	0.999905	0.999871	0.999911	0.999902

Table 4.2: Comparison of LbMATMPC and L4acados solvers for different numbers of GP points.

This indicates that both implementations are robust and capable of leveraging GP-based learning effectively for the cart-pole system. The results collectively highlight the importance of selecting an appropriate number of training points to balance model accuracy and computational efficiency in GP-based predictive control applications.

4.5 Simulation results

In this section the simulation results will be discussed and analyzed. Initially, the results must be assessed relative to a baseline. To this end, two configurations were implemented: one utilizing the nominal cart-pole dynamic model ($l = 0.8$) and another incorporating the real cart-pole dynamic model ($l = 0.5$), both without Gaussian Process-based corrective terms. In Figure 4.4 the state and control input evolution for the nominal and correct version of both solvers are displayed, on the x-axis of the graphs it is the time that for the simulations here tested totals 5 seconds.

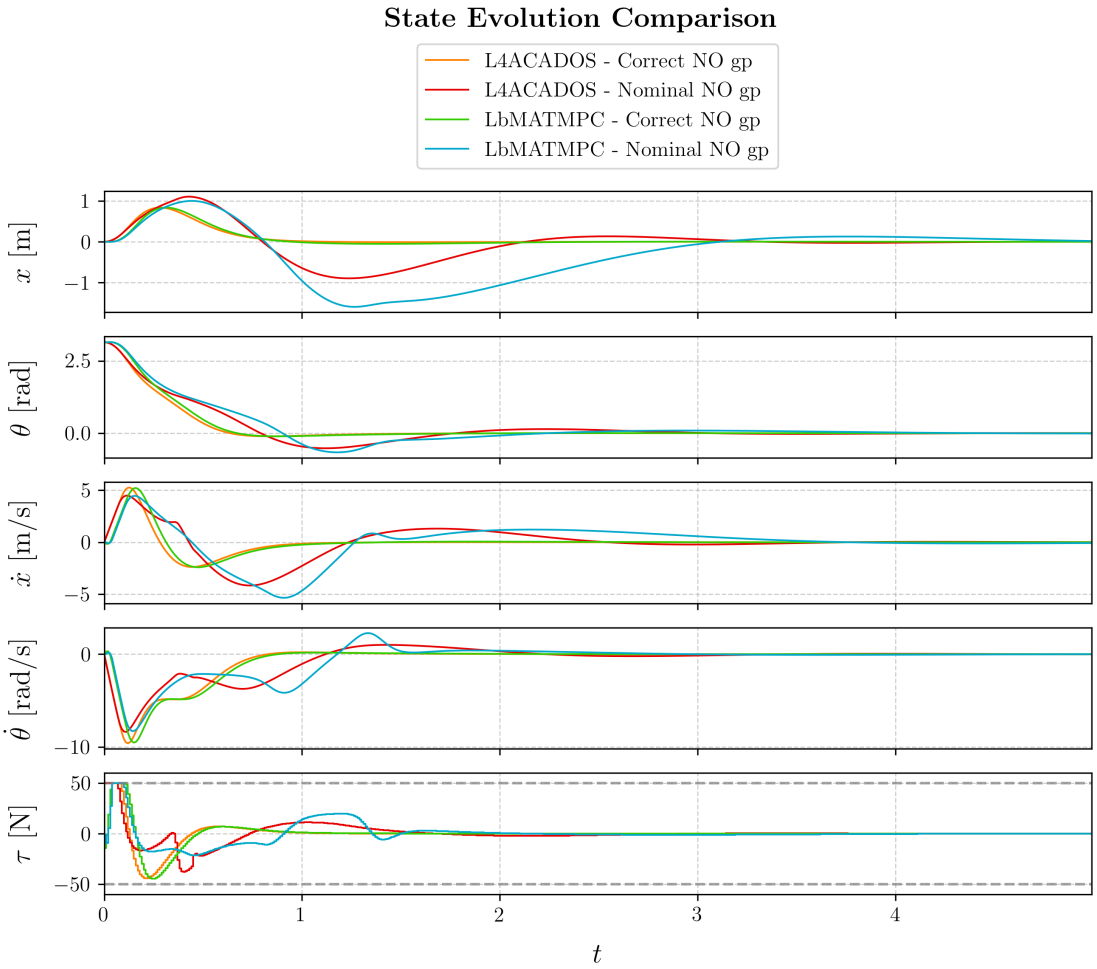

Figure 4.4: State **x** and control input u evolution for the nominal and correct model of LbMATMPC and L4acados, the correct model configurations have almost identical behavior while the nominal ones differ.

It is noticeable how the nominal model creates a big difference in the control

evolution of the cart-pole system. The system still manages to reach the final goal state but requires significantly more time to do so. Observing the control input evolution the correct model present much smoother transitions while the nominal controller requires much more jerky movements. A peak in the control input is also observable most probably due to the necessity of correcting an overshoot due to the different expected evolution. Looking at the nominal and correct simulations in LbMATMPC in Figure 4.4 a very similar situation can be observed. By doing so, the different behavior LbMATMPC and L4acados have when it comes to handling model mismatch scenario is apparent. LbMATMPC struggles much more to complete the swing-up maneuver successfully, requiring more time to fully complete the stabilization. Another remarkable difference is the initial control input sequence difference between the correct model configuration of L4acados and the one of LbMATMPC. By meticulously analyzing the initial control input sequence in Figure 4.5 it is apparent that L4acados begins immediately it's control sequence with input $u = +50$ while LbMATMPC takes a more cautions approach and instead slowly increase it's control output causing the whole control sequence to be slightly shifted forward in time with respect to L4acados, this could be possibly due to a smarter problem initialization by L4acados.

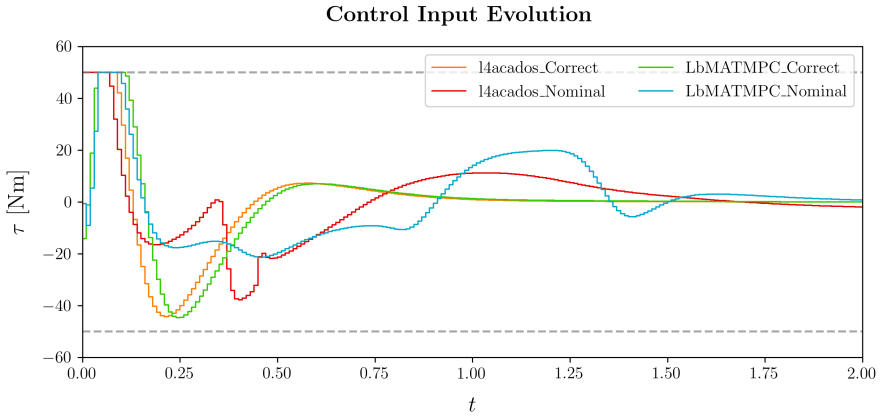

Figure 4.5: Control input difference in the first steps between the correct model of L4acados and LbMATMPC.

Observing the simulation results in Figure 4.6, obtained from the various GP configuration on the L4acados solver, a great overall control performance is noticeable. Notably, employing a K-means clustering approach to determine the GP points yields a significant improvement, with the model trained on just

50 GP points already closely approximating the corrected model. Beyond this threshold, further increases in the number of GP points do not lead to substantial performance enhancements, indicating a saturation effect.

Additional experiments were conducted to evaluate the sensitivity of the results to different GP point selection strategies. When the 50 GP points were selected randomly or spaced equidistantly from the initial dataset, the control performance deteriorated considerably. However, this decline in performance was largely mitigated when the number of GP points was increased to 100, suggesting that a sufficiently large dataset compensates for suboptimal selection strategies. This finding reinforces the notion that, for this specific system, 100 GP points are sufficient to adequately capture the relevant model discrepancies and with correct selection 50 GP points are also adequate.

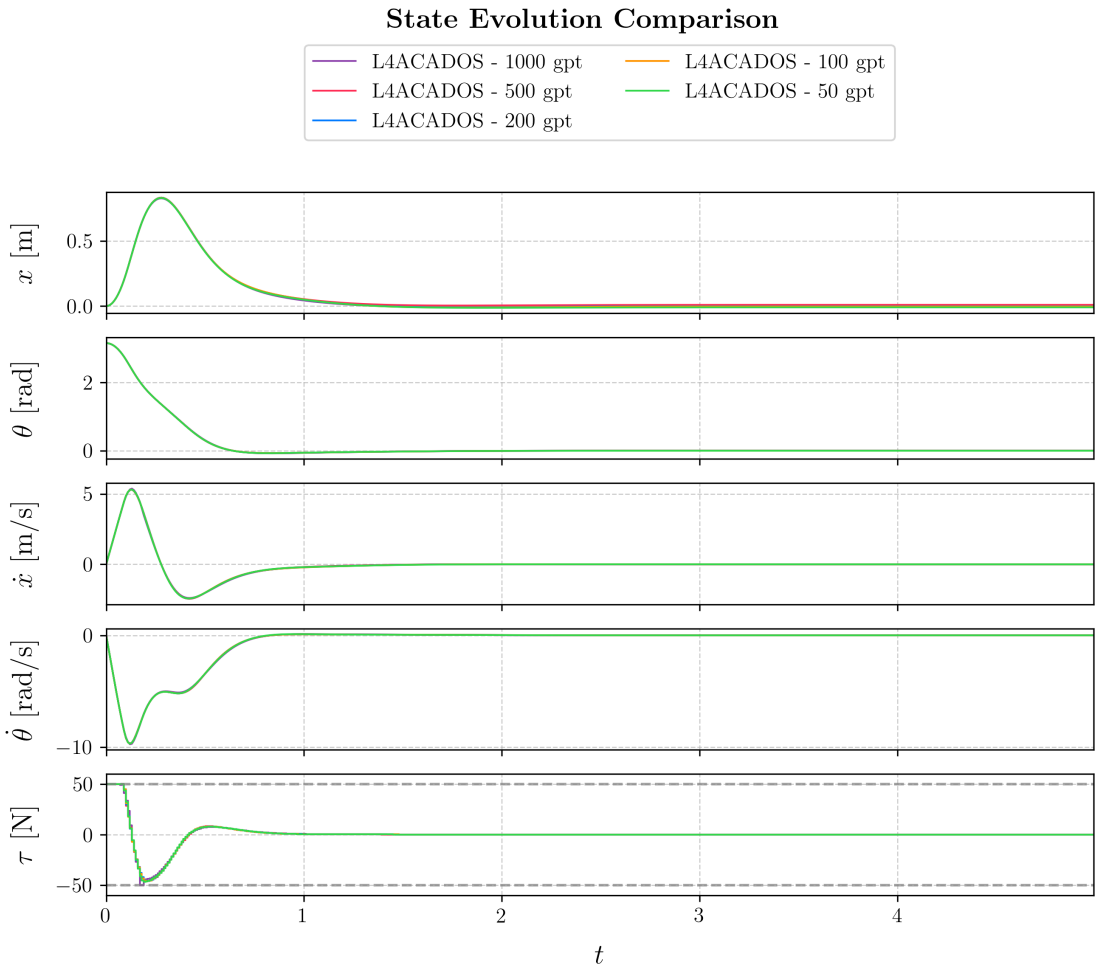

Figure 4.6: State and control input evolution of L4acados solvers with different GP values, all L4acados configurations behave almost identical.

A similar trend is observed in the results obtained from the LbMATMPC solver, as shown in Figure 4.7. Across all tested configurations, the GP-based corrections closely align with the corrected model, exhibiting only minor variations. This consistency further supports the validity of the chosen GP configurations and confirms that their impact on control performance is minimal in this context.

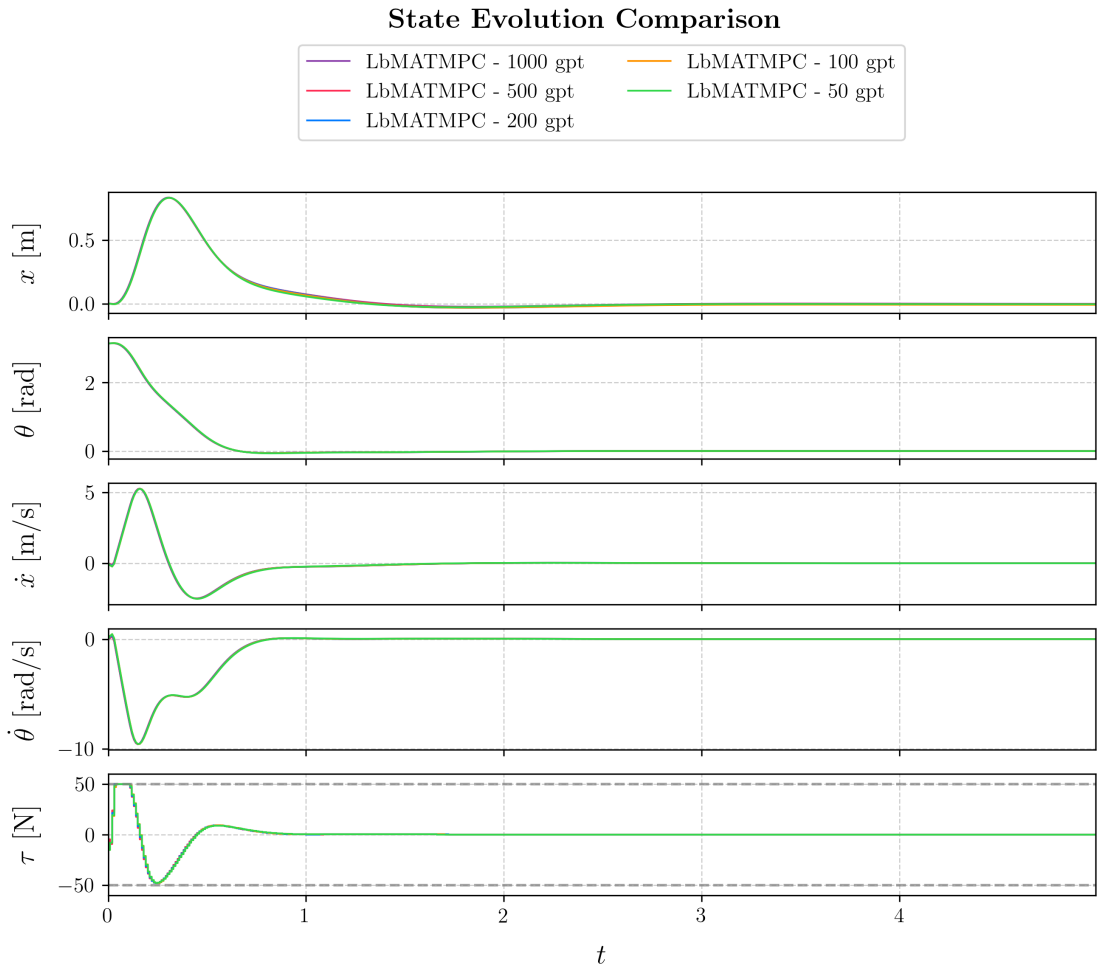

Figure 4.7: State and control input evolution of LbMATMPC solvers with different GP values, all L4acados configurations behave almost identical.

This outcome is particularly significant because it ensures that all tested configurations exhibit nearly identical control behavior during simulation. Consequently, the comparative evaluation of computational performance across toolboxes and GP configurations remains rigorous, unbiased, and methodologically sound. The relatively low GP model complexity required for this problem makes it well-suited for a fair benchmarking of the solvers. However, in more complex

scenarios, where greater generalization capabilities are required, the number of necessary GP points may increase substantially.

In cases where larger model corrections are needed, the system dynamics are inherently more complex, or a higher-dimensional state space must be predicted, the number of GP points required for an accurate model approximation can increase dramatically. Moreover, if the objective shifts towards learning the full system dynamics in a black-box fashion rather than correcting specific model discrepancies, the data requirements may grow even further. While the present study demonstrates that 50 GP points suffice to approximate an accurate model correction, making it suitable for the intended solver comparison, it does not provide insights into how many GP points would be required to ensure a satisfactory control performance in a more demanding scenario. This limitation precludes a comprehensive analysis of the trade-offs between computational complexity and control accuracy, which could be a crucial consideration in more challenging applications.

4.6 Accelerating GP Predictions with GPU

Here we present the numerical studies for the GPU acceleration of GP inference. GPUs are specialized hardware designed to perform massive parallel computations, making them well-suited for accelerating matrix operations and large-scale numerical computations required for GP inference.

Figure 4.8: Nvidia RTX 6000 GPU.

CUDA, developed by NVIDIA [59], provides a programming model that enables efficient execution of parallel algorithms on GPUs. Unlike traditional

Central Processing Units (CPUs), which execute tasks sequentially, GPUs feature thousands of smaller cores that can execute computations concurrently, for example in the test system used in this work the GPU possesses 3,840 CUDA cores [60] whilst the CPU only has 8 physical cores [61]. This parallelism is particularly advantageous for GP-based computations, where matrix-vector operations, kernel evaluations, and Cholesky decompositions can be distributed across multiple processing units.

The integration of CUDA-enabled GPUs into GP-NMPC could offer several advantages. Accelerating GP prediction computation can provide the boost in performance needed for physical implementation of real-time GP-NMPC controller. The scalability of the CUDA-accelerated computation is another key advantage, the parallel nature of CUDA enables scaling GP-NMPC implementations to handle larger datasets and more complex system dynamics.

However, CUDA-based implementations also introduce many challenges. The first problem is the hardware dependency of CUDA to NVIDIA-produced GPUs, meaning that any real-world implementation will mandate the utilization of one of NVIDIA's physical GPUs. For this reason other software and hardware options like FPGAs or specialized AI chips may be considered for embedded GP-NMPC controllers. Another disadvantage is that data transfers between CPU and GPU memory can introduce latencies, impacting real-time performance if not managed efficiently. This overhead becomes relevant when the train data set size of the GP is small.

One of the disadvantages of offloading the GP predictions is that when training a Gaussian Process Regression model on a GPU, memory limitations impose significant constraints on scalability and computational efficiency. Unlike deep learning models, which typically operate with batch-based training and localized parameter updates, Gaussian Process models require the storage and manipulation of full covariance matrices, whose size grows quadratically with the number of data points. As the dataset size increases, the memory footprint of these matrices can rapidly exceed the available GPU memory, leading to severe performance degradation or outright failure due to out-of-memory (OOM) errors. Computing Exact GP inference also requires full covariance matrices for computing the prediction. This limitation is exacerbated by CUDAs memory allocation constraints [62], which prevent seamless overflow into system RAM, unlike CPU-based implementations that can leverage virtual memory at the cost of reduced efficiency. Consequently, large-scale GPR requires approxima-

tion techniques, such as inducing point methods (e.g., Variational Free Energy or Nyström approximation), distributed computing strategies, or mixed-precision computations to mitigate memory bottlenecks. Effective memory management, including careful batching and asynchronous memory transfers, is also crucial in optimizing CUDA-based implementations of Gaussian Process models for large datasets.

Although the advantages in computation performance promised with the integration of CUDA computing and offloading many operations to the GPU may seem very enticing, this is not always the case. Analyzing the computational performance between a workload that strictly uses a CPU and one that offloads some computational burden to the GPU comes with the caveat that GPUs and CPUS vary greatly in performance between each other as they do for power requirements. The platform where the testing of this work is performed is based on the Ryzen 9 5900HX CPU with 8 cores and 16 threads and a NVIDIA GeForce RTX 3060 Laptop GPU. The comparison that will follow, between the timing result with CPU only computing and GPU offloaded computing can vary a lot depending on the hardware utilized. To prove this point some cart-pole swing-up simulation were ran with different sizes of GP model datasets and different CPU threading configurations. Moreover the threading configurations utilized were:

Number of threads	**Computation device**
1	CPU
8	CPU
16	CPU
16	CPU + GP predictions on GPU

Table 4.3: CPU threads and device test configurations .

A first test is run with a GP training data set of 50 points.The results show that the computation time is comparable between the single-thread and eight-thread configurations, while the sixteen-threads and sixteen-threads + GPU have a considerable performance deficit. When the load of the Gaussian Process predictions increases, the configuration with GPU acceleration surpasses all other configurations in performance.
The single-thread and eight-thread results show how this type of computation is still skewed toward being less multi-threading optimized but the lower performance of the single-thread configuration with respect to the eight-thread

configuration shows that as the computational burden increases, the reliance on single-threaded performance lowers. Finally, when running the simulation

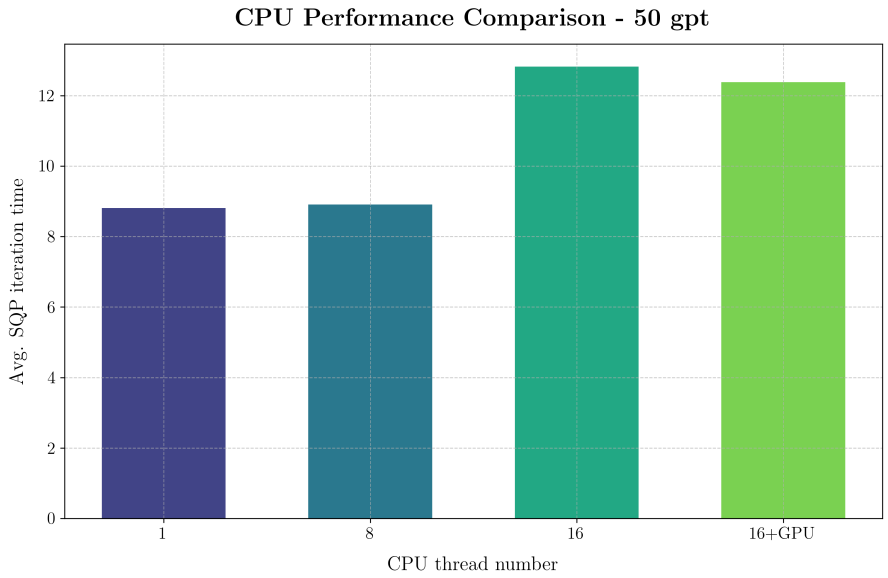

Figure 4.9: Comparison of CPU threading configurations and GPU timing results with a GP training set of 50 points. Single thread configuration is best.

with a GP training set composed of 5000 data points, the results really show off the advantages of offloading the GP prediction workload to the GPU. The computation times for the single-thread configuration increases by 450% with respect to the sixteen-thread + GPU configuration. The eight-thread configuration is still giving better performance than the sixteen-thread configuration. Whilst both of these configuration yield better results than the single-thread one, these results show a weakness of multi-threading for this type of workloads. The performance of a Python script utilizing multi-threading does not always improve with an increased number of threads. Due to Python's Global Interpreter Lock (GIL), CPU-bound tasks cannot execute multiple threads in parallel [63], leading to increased context switching overhead as thread count rises. Additionally, excessive threading can cause cache contention and memory bandwidth saturation, reducing overall efficiency. Thread synchronization mechanisms, such as locks and queues, may further introduce delays as contention increases. Moreover, if the number of threads exceeds the available physical cores, hyper-threading limitations may result in reduced per-thread performance.

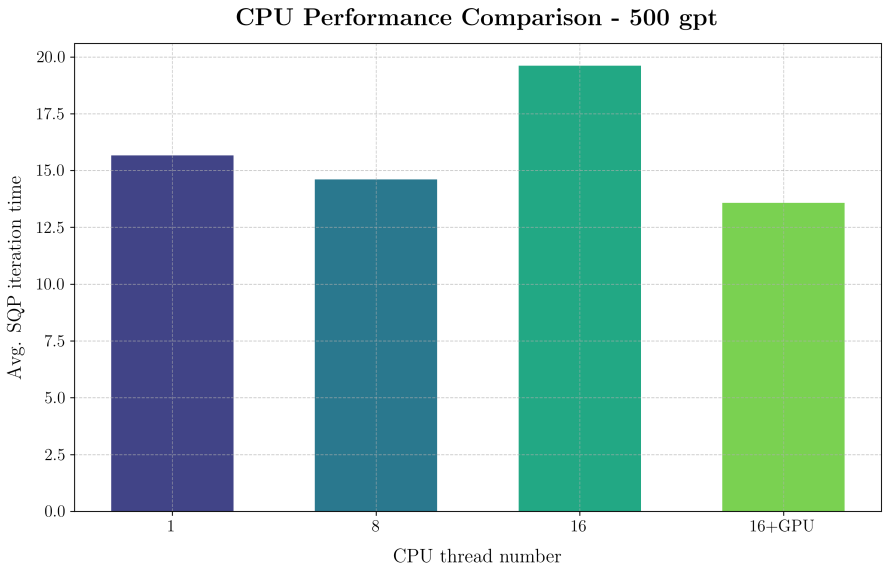

Figure 4.10: Comparison of CPU threading configurations and GPU timing results with a GP training set of 500 points. The 16 thread configuration achieve the worst result. The best result is obtained with GPU acceleration.

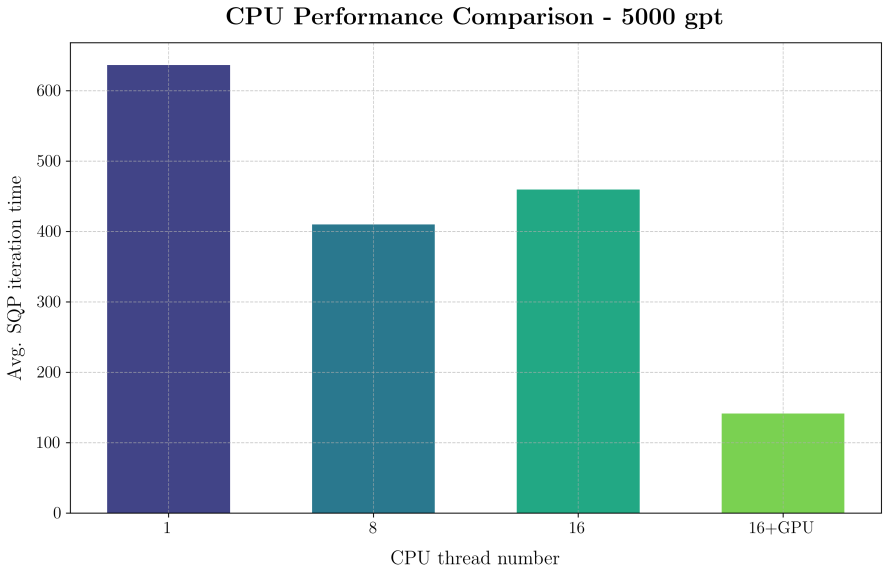

Figure 4.11: Comparison of CPU threading configurations and GPU timing results with a GP training set of 5000 points. GPU acceleration advantages become even more evident. 16 thread configuration still show disadvantage compared to 8 thread configuration.

The optimal number of threads depends on workload characteristics, hardware constraints, and memory access patterns. For these reasons, the results that will follow in this work will have to take into account the fact that, for different hardware combinations, the point where it is more advantageous to shift some of the computation tasks into a GPU will vary. Despite these challenges, CUDA-enabled GPU acceleration remains a key strategy for enhancing the computational efficiency of GP-NMPC, enabling its deployment in high-speed control applications.

4.7 Timing performance results

The computational performance timing data was collected by strategically placing timing functions within the solvers. In MATLAB, the built-in function `tic/toc` is used to start and stop a stopwatch timer to measure the elapsed time between code segments. In Python, a similar high-resolution timing mechanism is provided by the `perf_counter()` function from the time module, which offers precise measurements of execution time. These timers were inserted at key points in the control step computation to capture the time taken for each phase of the GP-NMPC solver. By segmenting the timing information, it was possible to analyze the computational cost of different solver components and compare their efficiency. As both toolboxes implement a very similar SQP-RTI scheme, we can compare the timing results for the following general sections:

- SQP-RTI update
  - RTI phase 1
    - Linearization
    - GP sensitivities prediction
    - Integration step
  - RTI phase 2

The final results utilized for evaluation will be averages of each of these operation over a full simulation sequence, this should ensure a reliable average expectation of the achievable performance.

The computational time results are analyzed by examining different configuration's performance across different GP training dataset sizes. The study focuses on evaluating the average sequential quadratic programming timing per

real-time iteration cycle, the computational time required for RTI phase 1 and phase 2, and the average GP prediction time. By systematically comparing these metrics, we aim to highlight the impact of GP inference complexity on toolbox performance and assess how each implementation handles the increasing computational burden as the number of training points grows. This timing data does not take into account the actual system dynamics simulation but only the operations followed to compute the optimal control input for a simulation step.

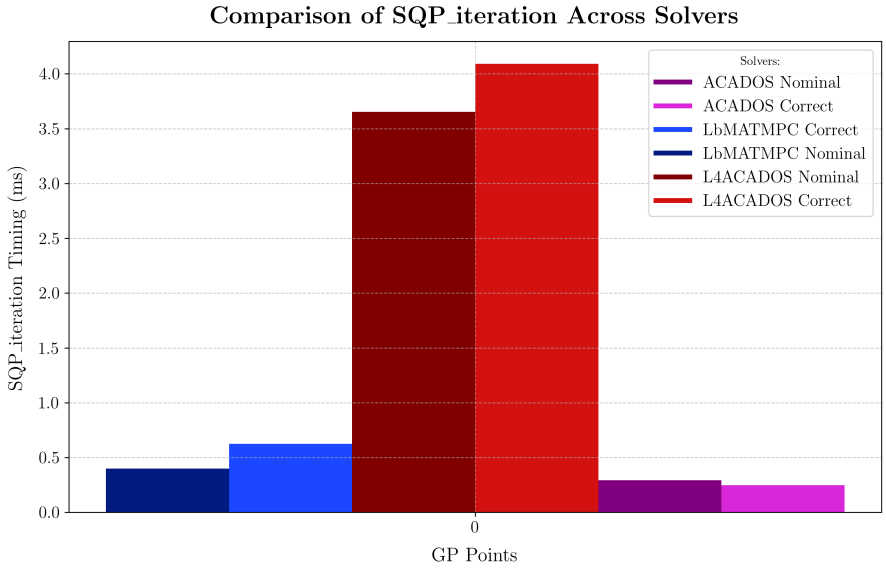

Figure 4.12: Average SQP timing result for solvers with no GP.

To finalize the analysis of the nominal and corrected solvers, a evaluation of the computational efficiency of the two toolboxes, excluding any GP considerations, is conducted. As outlined in subsection 3.2.2, the SQP-RTI cycle in L4acados is implemented in Python, with only specific components compiled into C code using Cython such as the final solver utilized in the feedback RTI phase and the integrator. In contrast, LbMATMPC implements the SQP-RTI computation in MATLAB, with the majority of its procedures compiled as MEX C functions. This difference in implementation results in a significant performance disparity, as evident in Figure 4.12, where the solvers are also benchmarked against the standard ACADOS implementation. Unlike the other two toolboxes, ACADOS fully compiles its OCP solver into C code, leading to the highest computational efficiency. Among the toolboxes compared, ACADOS exhibits the fastest performance, with LbMATMPC achieving similar results, whereas L4acados is observed to be more than ten times slower in comparison.

Now looking at all the tested configurations adding now also the CUDA accelerated predictions, the average results of SQP-RTI iteration time across is obtained. Analyzing the results displayed in Figure 4.13 much information can be deduced.

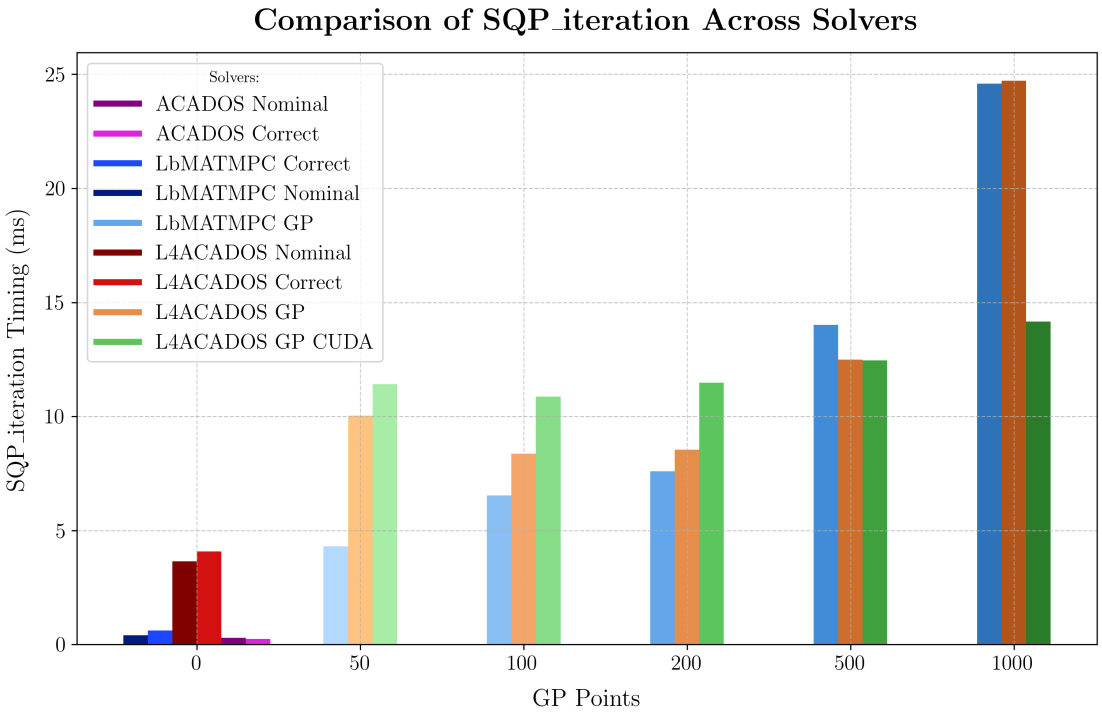

Figure 4.13: Average SQP-RTI iteration time across all configurations.

A clear overhead can be seen introduced by adding GP model predictions into L4acados as, even though the dataset size enlarges, the computational time decreases until the 200 GP points mark after which a much rapid increase is discernible. Adding GP predictions to these algorithms clearly introduces a heavy computational burden. Introducing now the additional GPU accelerated GP predictions three distinct trends can be seen for the three main toolbox configurations. These were once again: LbMATMPC - CPU, L4acados - CPU, L4acados - CPU + GPU. Starting with LbMATMPC a very steep increase in computational time is apparent when GP dataset size increases, similarly L4acados have a very rapid growth in computational time after the aforementioned overhead is surpassed.

A greatly different growth trend is instead present for the GPU accelerated configuration, for smaller GP set dimensions this is the worst performing configuration. It is only after the 500 GP points mark where much flatter rise curve of the GPU accelerated configuration is clearly observable. These results confirm

the initial assumptions which inferred that accelerating GP predictions with a GPU is beneficial for larger dataset with the downside of being worse for small dataset due to the computational cost of moving data in and out of the GPU memory for computations. A is the fact that LbMATMPC is much more efficient for smaller dataset, as assumed previously, the way LbMATMPC relies on MEX functions make this toolbox very efficient for small datasets but not so performing for large datasets.

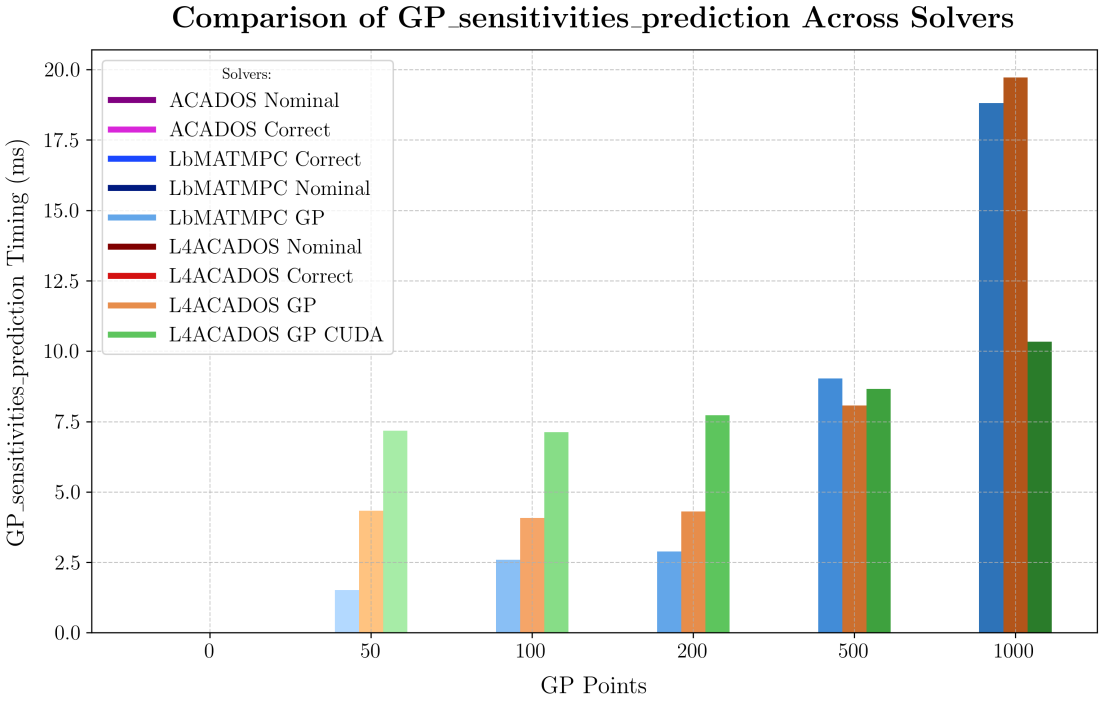

Figure 4.14: Average GP model prediction time per iteration across solver. No-GP model configurations do not compute GP predictions.

The exact same trends that are present in Figure 4.13 are also present in Figure 4.14, Figure 4.15 and Figure 4.16 with only two noteworthy remarks. Firstly, as it was expected, solvers with no GP prediction have zero time spent for GP related tasks. Secondly, in Figure 4.16 we can see that LbMATMPC necessitates more time to perform phase 2 or RTI.

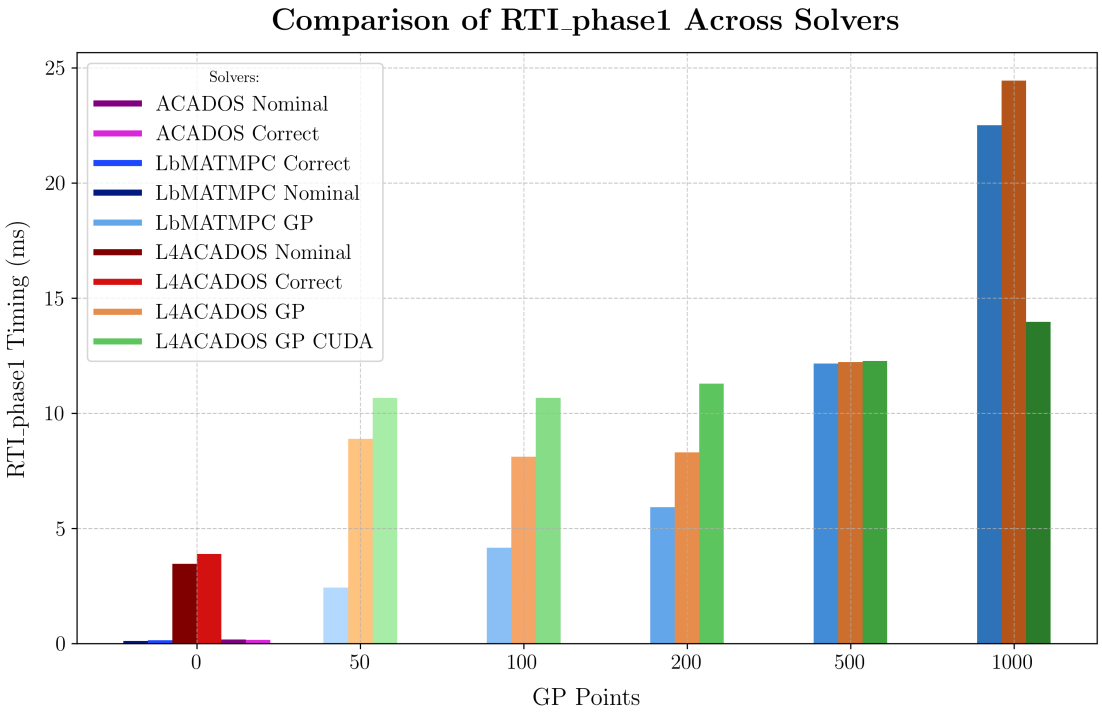

Figure 4.15: Average time results for RTI phase 1 iteration. LbMATMPC scales more linearly with the amount of GP points compared to L4acados.

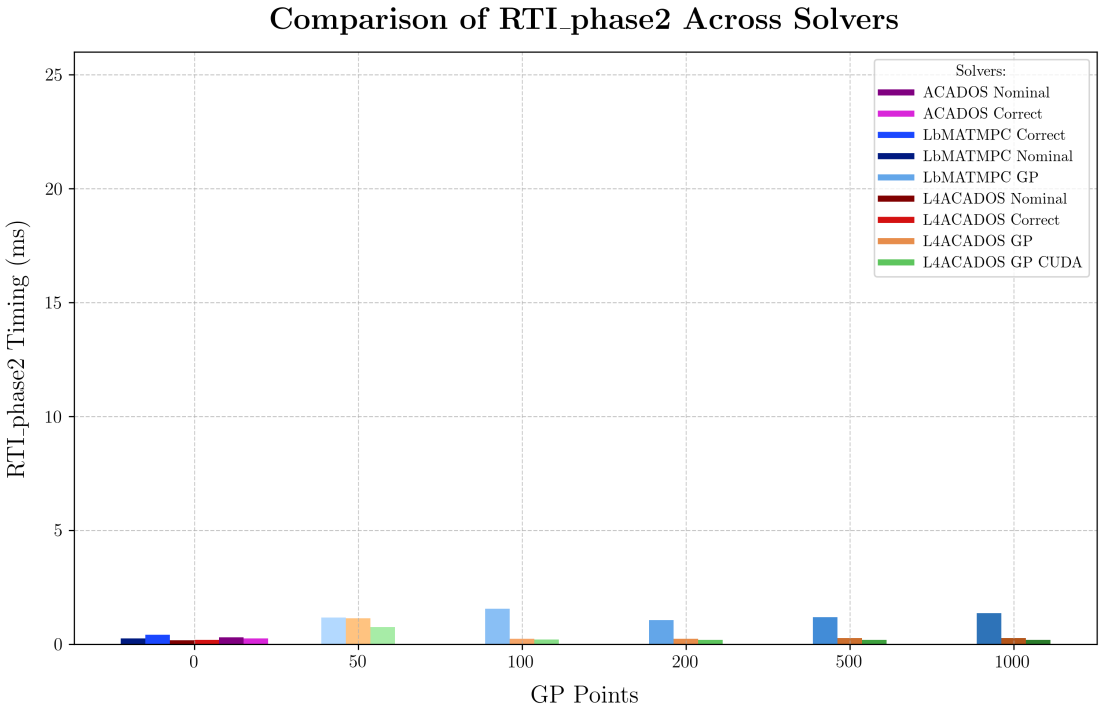

Figure 4.16: Average time results for RTI phase 2 iteration. LbMATMPC takes slighlty longer to compute this step.

5

Dimensionality Reduction for Gaussian Processes

Gaussian Processes have over time become widely used and powerful tools for modelling functions of all sorts, ranging from complex dynamical systems to financial evolution prediction. They are however less suitable for high-dimensionality problems where real-time performance and fast inference is necessary such as the Model Predictive Control framework. As mentioned earlier in this text, this is due to the computational complexity of a GP prediction, that for a training set of n data samples is $O(n^3)$ [39]. This comes from the complexity of the computation of the Cholesky decomposition of K_{xx} needed for the inverse term present in the mean computation formula which for an input point $x_\star$ is:

$$\mu(x_\star) = \mu_x + K_{x_\star x}(K_{xx} + \sigma_f^2 I)^{-1}(Y - \mu_x) \tag{5.1}$$

The computation of the mean and variance also requires to store the different values of K_{xx} which grows squarely with the dimension of the data samples $O(n^2)$. This high computational burden and necessity of storing vast amount of data makes Gaussian Process modeling impractical for large datasets and problems where a highly complex model is required jointly to fast prediction computations. Researchers have been experimenting with different methods of mitigating this issue. Some methods keep the Exact Gaussian Process structure and instead focus on lowering the dimensionality of training inputs of the Gaussian Process, some examples are Principal Component Analysis [34], Random

Features [64], Independent Component Analysis [65] and more recently the use of Autoencoders [66].

5.1 Sparse Gaussian Processes

Sparse Gaussian Processes are approximation techniques born from the necessity of mitigating the computational burden of Exact Gaussian Processes, these techniques aim at approximating the Exact GP by selecting a subset of data points from the training data, called inducing points $\mathbf{Z}$ which summarize the information of the full dataset, making the inference less data onerous. Optimizing the choice of inducing points becomes fundamental for the quality of the approximations [67]. The posterior is approximated through these training points in the form:

$$p(f \mid \mathbf{X}) \approx \int p(f \mid x, \mathbf{X}) p(x \mid \mathbf{Z})\, dx \tag{5.2}$$

The function $p(f \mid \mathbf{X})$ represents the posterior distribution of the function values given the observed inputs $\mathbf{X}$. The term $p(f \mid x, \mathbf{X})$ denotes the conditional distribution of function values given an input x and the training data $\mathbf{X}$, while $p(x \mid \mathbf{Z})$ represents the distribution of the input points conditioned on the inducing points $\mathbf{Z}$. The integral over x accounts for the approximation introduced by the sparse formulation, where the full posterior is inferred based on the subset $\mathbf{Z}$ rather than the entire dataset, reducing computational complexity.

Over the years several approaches for Sparse GP approximation have been proposed [68], these methods primarily focus on approximating the covariance structure or leveraging sparsity within the dataset. Some common Sparse GP approximation methods are Fully Independent Training Conditional (FITC) approximation [67], Partially Independent Training Conditional (PITC) [69], Nyström method [70], Expectation Propagation (EP) [71, 72], Deterministic Training Conditional (DTC) [73] and Variational Free Energy (VFE) [23].

In this study, we employ the sparse Gaussian Process approximation provided by the GPyTorch library, specifically the implementation available in `gpytorch.variational`. This approach is based on the variational strategy introduced in [74]. Throughout this work, we refer to this approximation method as the Stochastic Variational Gaussian Process (SVGP).

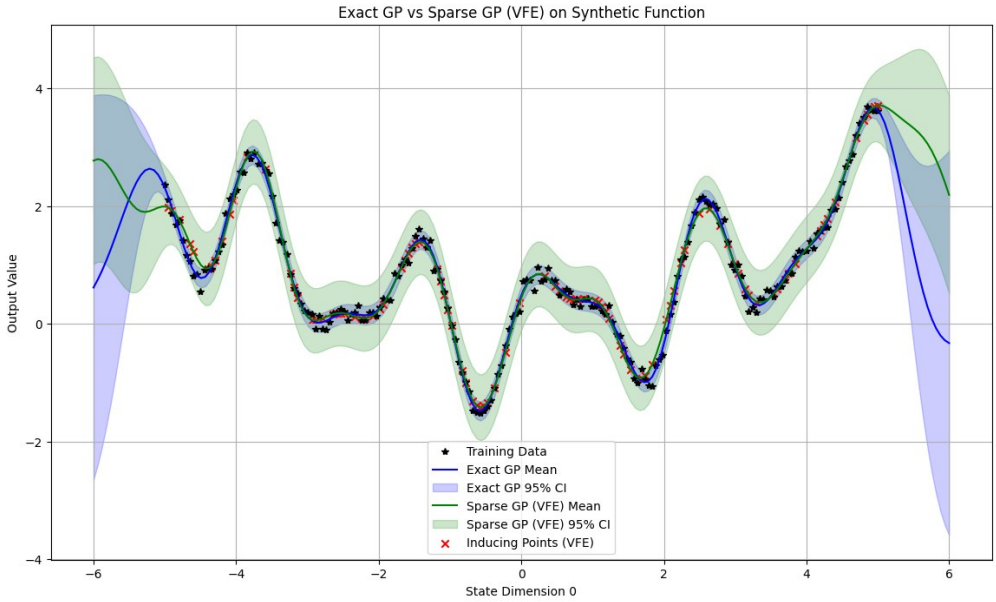

Figure 5.1: SVGP vs EXACT GPs for a synthetic function example.

Unlike traditional GPs, which perform exact Bayesian inference, SVGP leverages a set of m inducing points (ipt), $\mathbf{Z} = \{z_1, z_2, ..., z_m\}$, with $m \ll n$, to construct a lower-dimensional representation of the function. These inducing points serve as a compact summary of the dataset, reducing the computational burden while preserving the essential characteristics of the modeled function.

SVGP defines a variational distribution over the inducing variables $\mathbf{u} = f(\mathbf{Z})$:

$$q(\mathbf{u}) = \mathcal{N}(\mathbf{m}, \mathbf{S}) \tag{5.3}$$

where $\mathbf{m}$ and $\mathbf{S}$ are variational parameters optimized during training. The approximate posterior over the function values is given by:

$$q(f) = \int p(f \mid \mathbf{u}) q(\mathbf{u}) d\mathbf{u} \tag{5.4}$$

Here, $q(\mathbf{u})$ represents the variational distribution over the inducing variables, while $p(f \mid \mathbf{u})$ defines the conditional prior of the function values given the inducing variables, capturing the dependency structure imposed by the GP. In practice, the variational inference is often performed in a whitened space, where inducing variables are re-parameterized to improve numerical stability and optimization efficiency, see [75] for further explanation. SVGP employs the Evidence Lower Bound (ELBO) to optimize the variational parameters, lower-

bounding the marginal log-likelihood as:

$$\log p(\mathbf{y} \mid \mathbf{X}) \geq \mathbb{E}_{q(f)}[\log p(\mathbf{y} \mid f)] - \mathrm{KL}(q(\mathbf{u}) \parallel p(\mathbf{u} \mid \mathbf{Z})) \tag{5.5}$$

Maximizing this bound enables efficient approximate inference while ensuring computational feasibility. The SVGP framework reduces the complexity of training to $O(m^2 n)$ and inference to $O(m^2)$, making it significantly more scalable than standard GP regression.

A key advantage of SVGP is its ability to handle large-scale datasets efficiently through mini-batch optimization. Unlike exact GPs, which require the full dataset for inference, SVGP employs stochastic optimization techniques such as stochastic gradient descent, allowing training with small data subsets at each step. This enables SVGP to scale to large datasets that would be impractical for traditional GP models due to memory and computational constraints.

The variational inference framework in SVGP introduces a trade-off between computational efficiency and predictive accuracy. While exact GPs compute the true posterior, SVGP provides an approximation that ensures scalability while maintaining well-calibrated uncertainty estimates. The method is particularly advantageous in real-time applications, such as Gaussian Process-based Model Predictive Control, where rapid evaluations of the predictive model are required.

In summary, Stochastic Variational Gaussian Processes extend the flexibility of standard GPs to large-scale and real-time applications by leveraging inducing points, variational inference, and stochastic optimization. This enables scalable and efficient training and inference while preserving the core advantages of GP models in uncertainty quantification.

5.2 SVGP Results

To prove the capabilities of a SVGP implementation, three different SVGP solvers were implemented by leveraging the ApproximateGP library provided by GPyTorch for the L4acados toolbox on the cart-pole toy-problem. Three configurations of 10, 25, 50 inducing points were created and trained over the full available training set composed of ten thousand data points. By inspecting the state evolution in Figure 5.2, the configurations with 25 and 50 inducing points (ipt) proves to be almost equal to the correct model except for some erratic behavior at the initial control inputs for the 25 ipt points configuration. These model

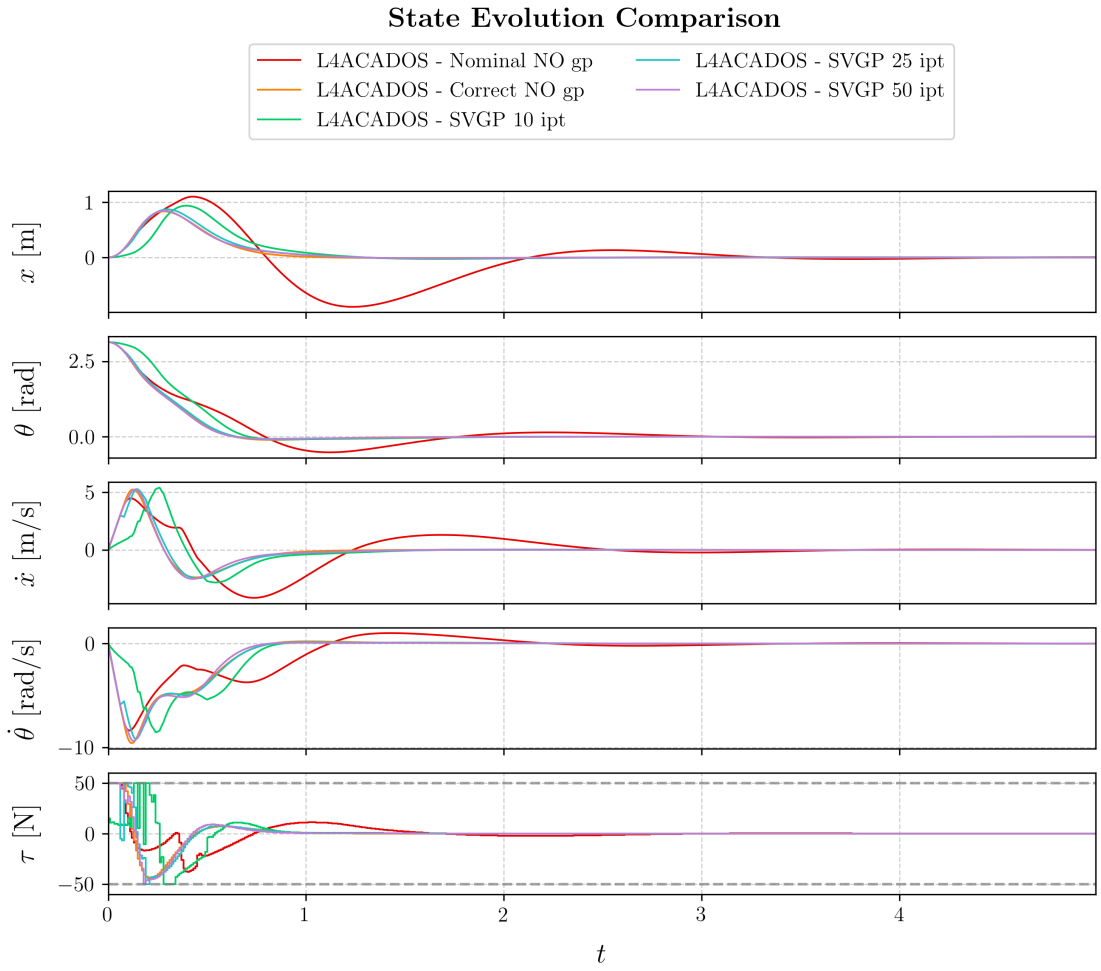

Figure 5.2: State and control input evolution of SVGP-L4acados configurations with different number of inducing points. All but the 10 ipt configurations perform almost like the correct model configuration.

show great improvements over the nominal model for the simulation results with even less than 50 GP points achieving correct model-like performance with only 25 ipt points. The 10 ipt configuration manage to complete the swing-up maneuver quicker than the nominal no GP configuration, although worse than all other GP models.

5.2.1 Control performance evaluation

Control performance in model predictive control and optimal control frameworks is commonly assessed using metrics that quantify both tracking accuracy and actuation efficiency. In evaluating the performance of a GP-NMPC con-

troller for the cart-pole system, three key performance metrics are considered: total control effort, closed-loop cost function, and cumulative tracking error. These metrics provide insights into control efficiency, optimality, and tracking accuracy, respectively. The total control effort quantifies the cumulative magnitude of the control inputs applied over the simulation horizon. It is defined as:

$$J_u = \sum_{k=0}^{N-1} \mathbf{u}_k^\top \mathbf{u}_k \tag{5.6}$$

where $\mathbf{u}_k$ is the control input at time step k. This metric is useful for assessing the energy expenditure required by the controller and serves as an indicator of actuator wear and efficiency. A lower control effort typically corresponds to a more energy-efficient controller but must be balanced with tracking accuracy.

The closed-loop cost function evaluates the cumulative objective function value over the entire simulation. Using the stage and terminal costs from the NMPC formulation, the closed-loop cost is computed as:

$$J_{\text{cl}} = \sum_{k=0}^{T-1} \left(\mathbf{x}_k^\top Q \mathbf{x}_k + \mathbf{u}_k^\top R \mathbf{u}_k \right) + \mathbf{x}_T^\top Q_f \mathbf{x}_T \tag{5.7}$$

where: T is the total number of discrete time steps in the closed-loop simulation, $\mathbf{x}_k$ is the state vector at time step k, Q and Q_f are positive semi-definite weighting matrices penalizing deviations from the desired state, R is a positive definite control weight matrix penalizing excessive control effort. This metric directly reflects the optimality of the closed-loop behavior. A lower cost function value indicates better performance in terms of tracking precision and control efficiency.

The cumulative tracking error quantifies the deviation of the system states from the desired reference trajectory $\mathbf{x}_{\text{ref},k}$. It is computed as:

$$J_e = \sum_{k=0}^{T} ||\mathbf{x}_k - \mathbf{x}_{\text{ref},k}||^2 \tag{5.8}$$

where $|| \cdot ||^2$ denotes the squared Euclidean norm. This metric is crucial for evaluating the controllers ability to maintain the cart-pole system on its intended trajectory. A lower cumulative tracking error suggests improved trajectory adherence and better overall control performance. These performance metrics provide a comprehensive evaluation of the GP-NMPC controller. Total control effort offers insight into the energy expenditure and actuator stress, while the

closed-loop cost function serves as an overall measure of control optimality. The cumulative tracking error directly assesses trajectory adherence, which is crucial in real-world applications where precision is required. By analyzing these metrics together, the study aims to highlight differences in control efficiency, accuracy, and computational trade-offs. SVGP configuration will be evaluated against the exact GP model configurations.

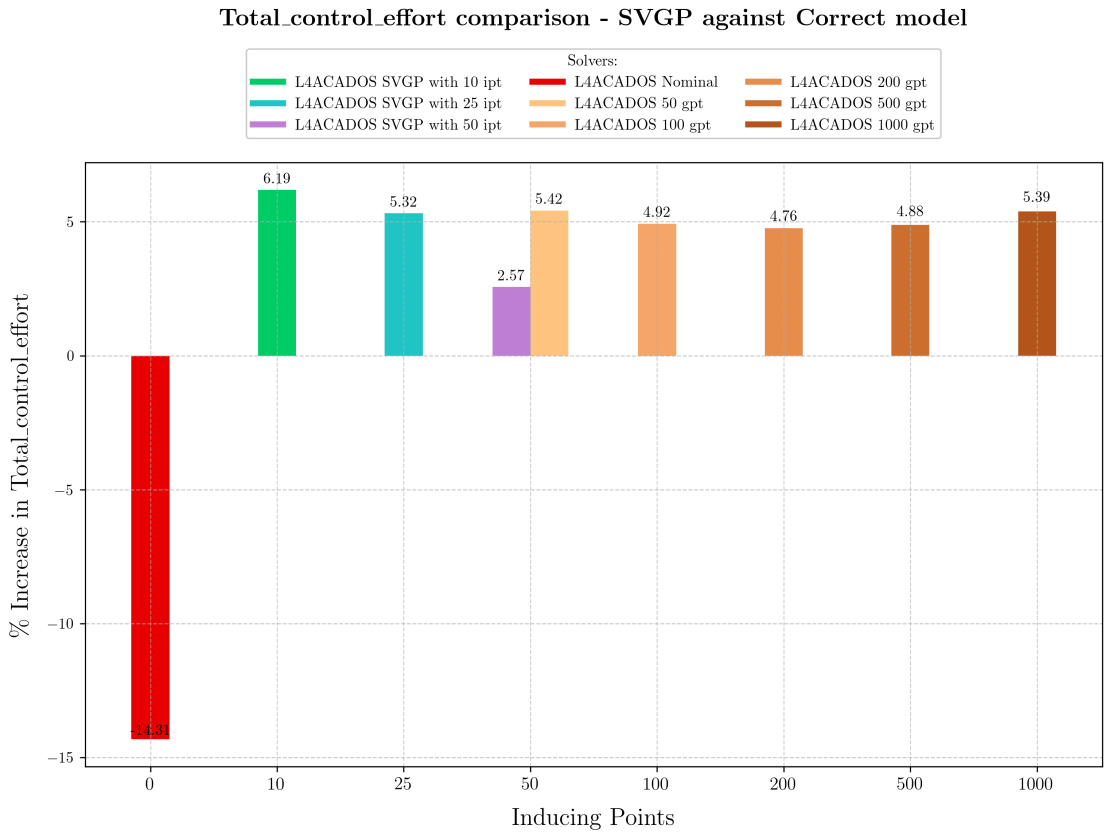

Figure 5.3: Total control effort percentage of improvement against configurations with correct model. All GP configurations use more control effort than non-GP configurations.

The total control effort shows very similar results to the Exact GP, notably the 50 inducing point model is the one with least amount of increase in the total control effort as seen in Figure 5.3

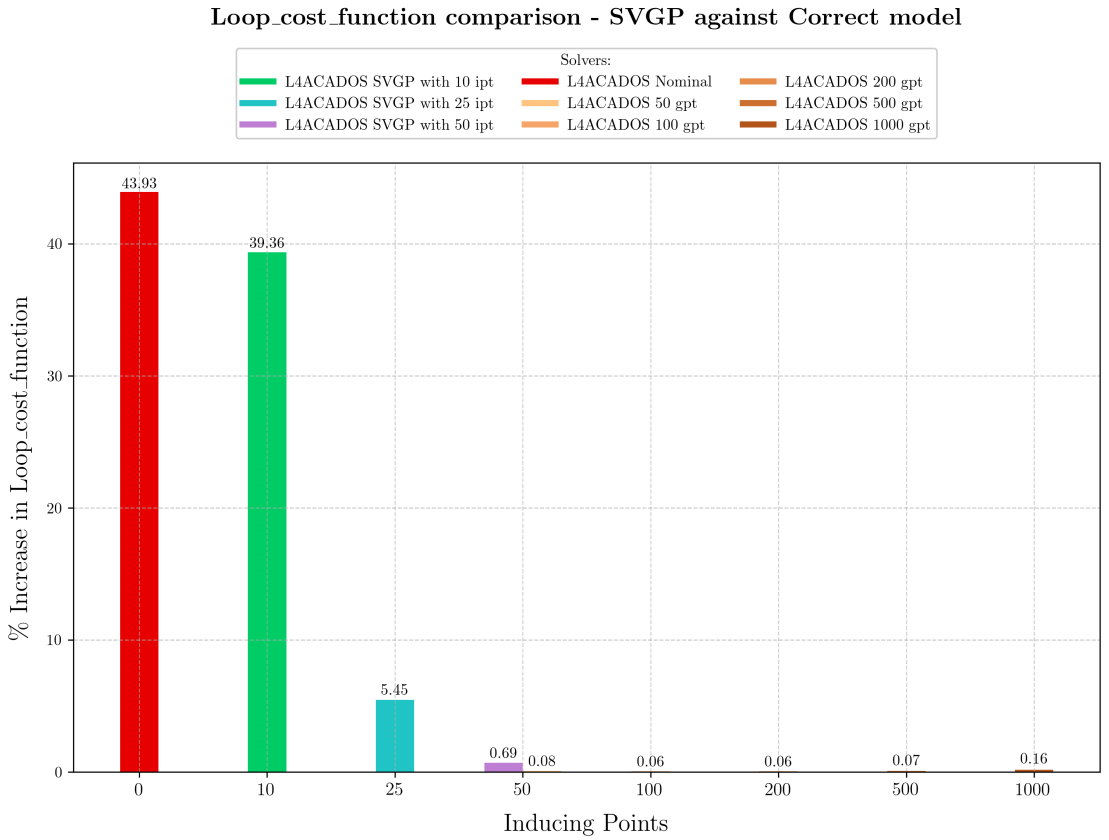

Figure 5.4: Closed-loop cost function percentage of improvement against configuration with correct model, the less inducing points the worse Closed-loop cost result.

Analyzing the results for Figure 5.4 a very slight deterioration in performance with the 50 inducing point model and a larger increase in total closed-loop cost function for the 25 inducing point model. An improvement in cumulative tracking error is instead present for the 25 ipt model decreasing it's value by around 2%, the 50 ipt model has a very slightly lower increase than the 1000 GP model configuration in tracking error when compared to the correct model. The closed-loop cost function for the 10 ipt configuration is very high, almost 40% worse than the correct model, this is expected from the state trajectory evolution seen in Figure 5.2.

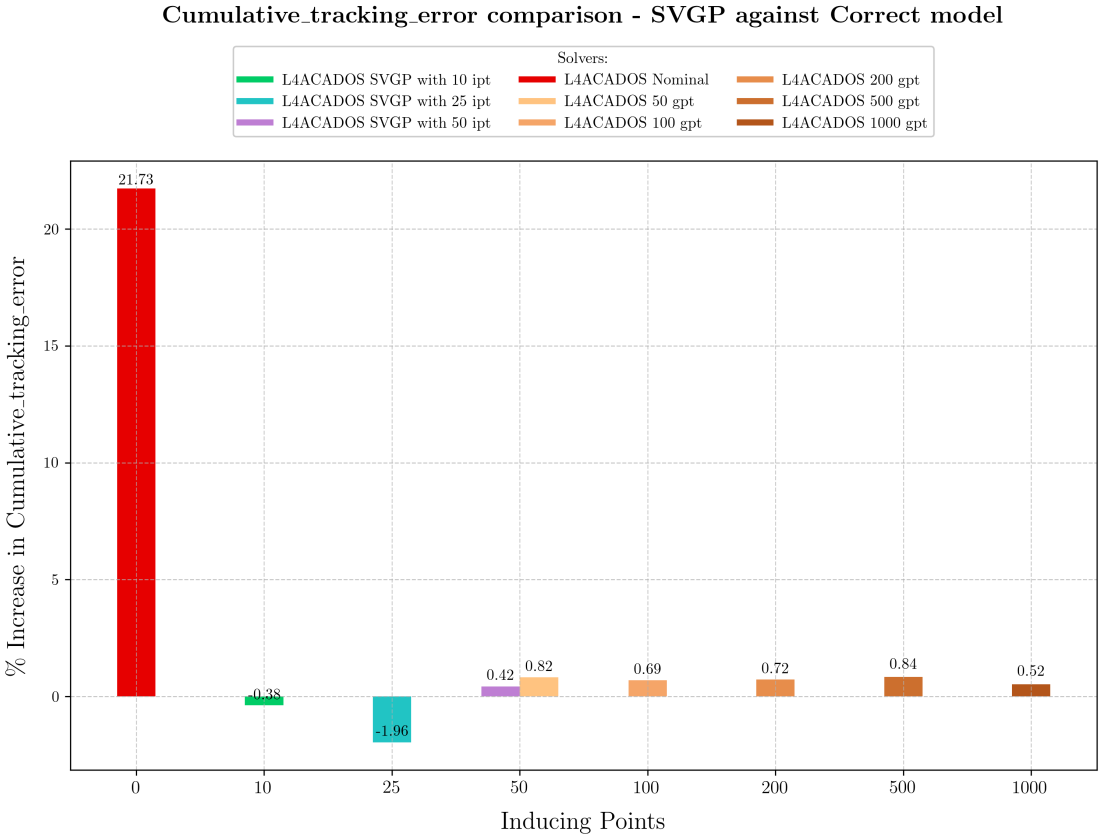

Figure 5.5: Cumulative tracking error percentage of improvement against configuration with correct model, all configurations show a very low tracking error.

5.2.2 SVGP COMPARISON OF TIMING PERFORMANCE

Finally, the computational timing results of the SVGP implementations are analyzed in Figure 5.6, highlighting the potential improvements in performance. The average SQP iteration times reveal two key observations: for the same number of GP training points and inducing points, the SVGP model exhibits a slight increase in computational time compared to the Exact GP counterpart. However SVGP's ability of leveraging the whole dataset to train makes the configuration with very little inducing points still a viable option where with the exact GP of equivalent training data amount the simulations fail to complete the swing-up maneuver. The 10 and 25 ipt configurations achieve marginally lower SQP-RTI cycle time than the 50 GP point exact GP model. However, the computational overhead introduced by L4acados adversely affects the performance of all models when using a small number of Gaussian Process or inducing points. The performance no longer scales efficiently with a reduced number of GP points.

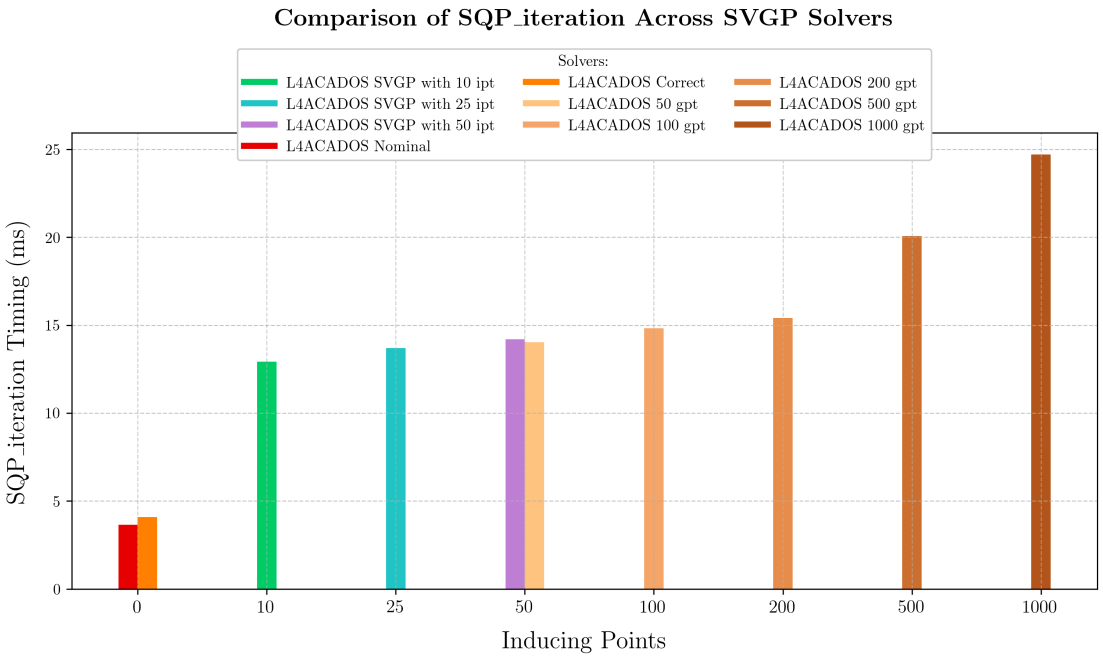

Figure 5.6: Average SQP-RTI iteration time, SVGP against no-GP solvers and exact GP models. SVGP models show little difference to 50 GP point model.

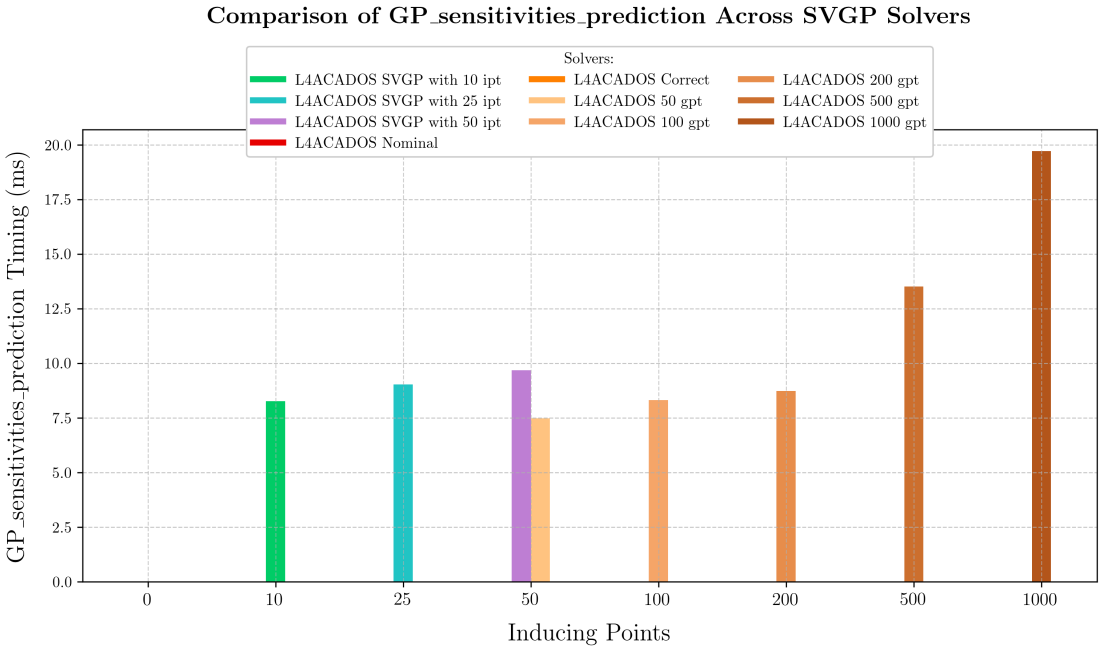

Figure 5.7: Average GP prediction time per iteration, SVGP against no-GP solvers and exact GP models. SVGP models have worse performance when computing gp predictions when compared to exact GP model prediction with less than 500 GP points.

6

Conclusions and Future Works

This thesis has explored the computational efficiency of learning-based nonlinear model predictive control using Gaussian Processes within two open-source toolboxes, L4acados and LbMATMPC. By leveraging both theoretical insights and practical implementation, a comprehensive analysis was conducted to assess the computational behaviors across different solver configurations. The study focused on three primary configurations: L4acados with CPU computations only, L4acados utilizing GPU acceleration for GP predictions, and LbMATMPC operating on CPU only. To systematically evaluate these approaches, a toy problem involving the swing-up control of a cart-pole system was employed, ensuring a structured and repeatable testing framework with a well-designed training dataset to enhance state-space exploration.

The results of the study have highlighted significant differences in computational efficiency between the two toolboxes. LbMATMPC demonstrated superior performance in low-data regimes, benefiting from its OCP solving loop being predominantly implemented in MEX compilations of CasADi functions. In contrast, L4acados exhibited a notable computational overhead, largely attributable to its SQP-RTI cycle, which is primarily executed in Python with only selective components compiled into efficient C code via Cython. However, as the GP dataset size increased, LbMATMPC showed a steeper rise in computational time, while L4acados maintained relatively stable performance up to approximately 500 training points, beyond which a sharp increase in computational cost was observed. The GPU-accelerated variant of L4acados, although less efficient for smaller datasets, exhibited a more favourable scaling behaviour, ul-

timately outperforming other configurations once the training dataset exceeded 500 samples.

Additionally, the study briefly examined performance differences between L4acados and LbMATMPC when operating without GP models, further contextualizing the observed computational trends. Moreover, an implementation of Stochastic Variational Gaussian Process through the GPyTorch library was tested for the L4acados toolbox. SVGP models exhibited slightly higher computational costs compared to an exact Gaussian Process model trained on a dataset of the same size as the SVGPs inducing points. However, it demonstrated the ability to effectively learn the underlying distribution even with a reduced number of inducing points, such as 25 or 10 cases in which the exact GP model was unable to achieve comparable performance. Nevertheless, due to the computational overhead introduced by L4acados, a significant improvement in computational efficiency was not observed. Further evaluations involving more complex control scenarios are necessary to better assess potential computational advantages.

Overall, this research has provided valuable insights into the computational trade-offs associated with different GP-NMPC solver configurations and GP approximations. The findings highlight the importance of solver architecture and GP training set size in determining real-time feasibility, with GPU acceleration and sparse GP techniques emerging as key strategies for improving computational performance in large-data regimes. These results offer a strong foundation for further research into optimizing GP-NMPC implementations, particularly in applications where real-time efficiency is paramount.

References

[1] Mattia Boggio, Carlo Novara, and Michele Taragna. "Trajectory planning and control for autonomous vehicles: a fast data-aided NMPC approach". In: *European Journal of Control* 74 (2023). 2023 European Control Conference Special Issue, p. 100857. ISSN: 0947-3580. DOI: https://doi.org/10.1016/j.ejcon.2023.100857. URL: https://www.sciencedirect.com/science/article/pii/S0947358023000869.

[2] Janick V. Frasch et al. "An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles". In: *2013 European Control Conference (ECC)* (2013), pp. 4136–4141. URL: https://api.semanticscholar.org/CorpusID:14390541.

[3] Mohammed Salim Qasim, Abdurahman Basil Ayoub, and Abdulla Ibrahim Abdulla. "NMPC Based-Trajectory Tracking and Obstacle Avoidance for Mobile Robots". In: *International Journal of Robotics and Control Systems* (2024). URL: https://api.semanticscholar.org/CorpusID:275487540.

[4] Ashwin Carvalho et al. "Predictive control of an autonomous ground vehicle using an iterative linearization approach". In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)* (2013), pp. 2335–2340. URL: https://api.semanticscholar.org/CorpusID:16075294.

[5] Nishant Chowdhri et al. "Integrated nonlinear model predictive control for automated driving". In: *Control Engineering Practice* 106 (2021), p. 104654. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2020.104654. URL: https://www.sciencedirect.com/science/article/pii/S0967066120302240.

[6] Alexander Liniger, Alexander Domahidi, and Manfred Morari. "Optimizationbased autonomous racing of 1:43 scale RC cars". In: *Optimal Con-*

trol Applications and Methods 36 (2015), pp. 628–647. URL: https://api.semanticscholar.org/CorpusID:11242645.

[7] Robin Verschueren et al. "Time-optimal race car driving using an online exact hessian based nonlinear MPC algorithm". In: *2016 European Control Conference (ECC)* (2016), pp. 141–147. URL: https://api.semanticscholar.org/CorpusID:22665887.

[8] J. T. Wen, Sanjeev Seereeram, and David S. Bayard. "Nonlinear predictive control applied to spacecraft attitude control". In: *Proceedings of the 1997 American Control Conference (Cat. No.97CH36041)* 3 (1997), 1899–1903 vol.3. URL: https://api.semanticscholar.org/CorpusID:121744138.

[9] Hyeongjun Park et al. "Nonlinear model predictive control for spacecraft rendezvous and docking with a rotating target". In: URL: https://api.semanticscholar.org/CorpusID:260442275.

[10] Gonzalo Andres Garcia and Shahriar Keshmiri. "Nonlinear Model Predictive Controller for Navigation, Guidance and Control of a Fixed-Wing UAV". In: 2011. URL: https://api.semanticscholar.org/CorpusID:111728553.

[11] Nathan J. Slegers, Jason Kyle, and Mark Costello. "Nonlinear Model Predictive Control Technique for Unmanned Air Vehicles". In: *Journal of Guidance Control and Dynamics* 29 (2006), pp. 1179–1188. URL: https://api.semanticscholar.org/CorpusID:27380894.

[12] Jiatao Ding et al. "Nonlinear model predictive control for robust bipedal locomotion: exploring angular momentum and CoM height changes". In: *Advanced Robotics* 35 (2019), pp. 1079–1097. URL: https://api.semanticscholar.org/CorpusID:237393452.

[13] Giulio Romualdi et al. *Online Non-linear Centroidal MPC for Humanoid Robot Locomotion with Step Adjustment*. 2022. arXiv: 2203.04489 [cs.RO]. URL: https://arxiv.org/abs/2203.04489.

[14] Anlong Zhang et al. "Nonlinear Model Predictive Control of Single-Link Flexible-Joint Robot Using Recurrent Neural Network and Differential Evolution Optimization". In: *Electronics* 10.19 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10192426. URL: https://www.mdpi.com/2079-9292/10/19/2426.

[15] Yuantao Yu et al. "Robust Nonlinear Model Predictive Control for Robot Manipulators with Disturbances". In: *2018 37th Chinese Control Conference (CCC)* (2018), pp. 3629–3633. URL: https://api.semanticscholar.org/CorpusID:52929190.

[16] Subham Samal and Oumar Barry. "Model Predictive Control for Tremor Suppressing Exoskeleton". In: *IFAC-PapersOnLine* 56.3 (2023). 3rd Modeling, Estimation and Control Conference MECC 2023, pp. 337–342. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2023.12.047. URL: https://www.sciencedirect.com/science/article/pii/S2405896323023807.

[17] Lars Grne and Jrgen Pannek. "Nonlinear Model Predictive Control: Theory and Algorithms". In: 2011. URL: https://api.semanticscholar.org/CorpusID:59632180.

[18] Lukas Hewing et al. "Learning-Based Model Predictive Control: Toward Safe Learning in Control". In: *Annu. Rev. Control. Robotics Auton. Syst.* 3 (2020), pp. 269–296. URL: https://api.semanticscholar.org/CorpusID:214052774.

[19] Lei Du et al. "A Learning-Based Nonlinear Model Predictive Control Approach for Autonomous Driving". In: *IFAC-PapersOnLine* 56.2 (2023). 22nd IFAC World Congress, pp. 2792–2797. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2023.10.1388. URL: https://www.sciencedirect.com/science/article/pii/S2405896323017950.

[20] Hendrik Schäfke et al. "Learning-Based Nonlinear Model Predictive Control of Articulated Soft Robots Using Recurrent Neural Networks". In: *IEEE Robotics and Automation Letters* 9.12 (Dec. 2024), pp. 11609–11616. ISSN: 2377-3774. DOI: 10.1109/lra.2024.3495579. URL: http://dx.doi.org/10.1109/LRA.2024.3495579.

[21] Ju Kocijan et al. "Predictive control with Gaussian process models". In: *The IEEE Region 8 EUROCON 2003. Computer as a Tool.* 1 (2003), 352–356 vol.1. URL: https://api.semanticscholar.org/CorpusID:7346905.

[22] Lukas Hewing, Alexander Liniger, and Melanie Nicole Zeilinger. "Cautious NMPC with Gaussian Process Dynamics for Autonomous Miniature Race Cars". In: *2018 European Control Conference (ECC)* (2017), pp. 1341–1348. URL: https://api.semanticscholar.org/CorpusID:44923829.

[23] Michalis K. Titsias. "Variational Learning of Inducing Variables in Sparse Gaussian Processes". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)* (2009), pp. 567–574.

[24] Duy Nguyen-Tuong and Jan Peters. "Local Gaussian process regression for real-time model-based robot control". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2008), pp. 380–385. URL: https://api.semanticscholar.org/CorpusID:14257778.

[25] Ali Rahimi and Ben Recht. "Random Features for Large-Scale Kernel Machines". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2007), pp. 1177–1184.

[26] Andrea Carron et al. "Data-Driven Model Predictive Control for Trajectory Tracking With a Robotic Arm". In: *IEEE Robotics and Automation Letters* 4 (2019), pp. 3758–3765. URL: https://api.semanticscholar.org/CorpusID:199489532.

[27] Juraj Kabzan et al. "Learning-Based Model Predictive Control for Autonomous Racing". In: *IEEE Robotics and Automation Letters* 4 (2019), pp. 3363–3370. URL: https://api.semanticscholar.org/CorpusID:198145409.

[28] Yuhan Liu and Pengyu Wang. "Model Predictive Control with Gaussian-Process-Enhanced Dynamics for Spacecraft Attitude Takeover Maneuvers". In: *Journal of Physics: Conference Series* 2762 (2024). URL: https://api.semanticscholar.org/CorpusID:270241487.

[29] Thomas F Edgar and Dale E Seborg. *Process dynamics and control*. Wiley, 2016.

[30] Ruchika and Neha Raghu. "Model Predictive Control: History and Development". In: *International Journal of Engineering Trends and Technology (IJETT)* 4.6 (2013), pp. 2600–2602. ISSN: 2231-5381. URL: http://www.ijettjournal.org.

[31] Max Schwenzer et al. "Review on model predictive control: an engineering perspective". In: *The International Journal of Advanced Manufacturing Technology* 117.5 (Nov. 2021), pp. 1327–1349. ISSN: 1433-3015. DOI: 10.1007/s00170-021-07682-3. URL: https://doi.org/10.1007/s00170-021-07682-3.

[32] Philip E. Gill and Elizabeth Wong. "Sequential Quadratic Programming Methods". In: *Mixed Integer Nonlinear Programming*. Ed. by Jon Lee and Sven Leyffer. New York, NY: Springer New York, 2012, pp. 147–224. ISBN: 978-1-4614-1927-3.

[33] Moritz Diehl, Hans Georg Bock, and Johannes P. Schlöder. "A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control". In: *SIAM J. Control. Optim.* 43 (2005), pp. 1714–1736. URL: https://api.semanticscholar.org/CorpusID:207079410.

[34] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. ISBN: 9780262182539.

[35] Norbert Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series: With Engineering Applications*. The MIT Press, Aug. 1949. ISBN: 9780262257190. DOI: 10.7551/mitpress/2946.001.0001. eprint: https://direct.mit.edu/book-pdf/2313079/book_9780262257190.pdf. URL: https://doi.org/10.7551/mitpress/2946.001.0001.

[36] Christopher K. I. Williams and Carl E. Rasmussen. "Gaussian Processes for Regression". In: *Neural Information Processing Systems*. 1995. URL: https://api.semanticscholar.org/CorpusID:2877073.

[37] Marc Tunnell, Nathaniel Bowman, and Erin Carrier. "Fast Gaussian Process Emulation of Mars Global Climate Model". In: *Earth and Space Science* 10 (2023). URL: https://api.semanticscholar.org/CorpusID:259319036.

[38] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. "Gaussian Processes for Data-Efficient Learning in Robotics and Control". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37 (2015), pp. 408–423. URL: https://api.semanticscholar.org/CorpusID:8452555.

[39] Carl Edward Rasmussen. "Gaussian Processes in Machine Learning". In: *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*. Ed. by Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 63–71. ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_4. URL: https://doi.org/10.1007/978-3-540-28650-9_4.

[40] 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)* (2011).

[41] Jie Wang, Michael T. H. Fader, and Joshua A. Marshall. "Learningbased model predictive control for improved mobile robot path following using Gaussian processes and feedback linearization". In: *Journal of Field Robotics* 40 (2023), pp. 1014–1033. URL: https://api.semanticscholar.org/CorpusID:257037418.

[42] Yanhua Liu, Ron John Patton, and Shuo Shi. "Wind Turbine Load Mitigation Using MPC with Gaussian Wind Speed Prediction". In: *2018 UKACC 12th International Conference on Control (CONTROL)* (2018), pp. 32–37. URL: https://api.semanticscholar.org/CorpusID:53641483.

[43] Wonoo Choo and Erkan Kayacan. "Computationally Efficient Data-Driven MPC for Agile Quadrotor Flight". In: *2023 American Control Conference (ACC)* (2023), pp. 2627–2632. URL: https://api.semanticscholar.org/CorpusID:258960259.

[44] Enrico Picotti et al. "A Learning-Based Nonlinear Model Predictive Controller for a Real Go-Kart Based on Black-Box Dynamics Modeling Through Gaussian Processes". In: *IEEE Transactions on Control Systems Technology* 31 (2023), pp. 2055–2065. URL: https://api.semanticscholar.org/CorpusID:258959333.

[45] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. "A Unifying View of Sparse Approximate Gaussian Process Regression". In: *Journal of Machine Learning Research* 6 (2005), pp. 1939–1959.

[46] Yoshua Bengio Ian Goodfellow and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[47] Enrico Picotti et al. "LbMATMPC: an open-source toolbox for Gaussian Process modeling within Learning-based Nonlinear Model Predictive Control". In: *2022 European Control Conference (ECC)*. IEEE. 2022, pp. 736–742.

[48] Yutao Chen et al. "MATMPC - A MATLAB Based Toolbox for Real-time Nonlinear Model Predictive Control". In: *2019 18th European Control Conference (ECC)*. 2019, pp. 3365–3370. DOI: 10.23919/ECC.2019.8795788.

[49] Gianluca Frison and Moritz Diehl. "HPIPM: a high-performance quadratic programming framework for model predictive control". In: *ArXiv* abs/2003.02547 (2020). URL: https://api.semanticscholar.org/CorpusID:212415182.

[50] Joel Andersson. "A general-purpose software framework for dynamic optimization". In: (2013).

[51] Enrico Picotti et al. "A Learning-Based Nonlinear Model Predictive Controller for a Real Go-Kart Based on Black-Box Dynamics Modeling Through Gaussian Processes". In: *IEEE Transactions on Control Systems Technology* 31.5 (2023), pp. 2055–2065. DOI: 10.1109/TCST.2023.3291532.

[52] Amon Lahr et al. "Zero-Order optimization for Gaussian process-based model predictive control". In: *European Journal of Control* (2023), p. 100862. ISSN: 0947-3580. DOI: 10.1016/j.ejcon.2023.100862.

[53] Robin Verschueren et al. "Towards a modular software package for embedded optimization". In: *IFAC-PapersOnLine* 51.20 (2018), pp. 374–380.

[54] Hans Joachim Ferreau et al. "qpOASES: a parametric active-set algorithm for quadratic programming". In: *Mathematical Programming Computation* 6 (2014), pp. 327–363. URL: https://api.semanticscholar.org/CorpusID:42031739.

[55] Gianluca Frison et al. "BLASFEO". In: *ACM Transactions on Mathematical Software (TOMS)* 44 (2017), pp. 1–30. URL: https://api.semanticscholar.org/CorpusID:10844444.

[56] Amon Lahr et al. *L4acados: Learning-based models for acados, applied to Gaussian process-based predictive control*. 2024. arXiv: 2411.19258 [eess.SY]. URL: https://arxiv.org/abs/2411.19258.

[57] Tim Salzmann et al. "Learning for CasADi: Data-driven Models in Numerical Optimization". In: *Learning for Dynamics and Control Conference (L4DC)*. 2024.

[58] Stefan Behnel et al. "Cython: The Best of Both Worlds". In: *Computing in Science & Engineering* 13 (2011), pp. 31–39. URL: https://api.semanticscholar.org/CorpusID:14292107.

[59] Wikipedia contributors. *CUDA*. https://it.wikipedia.org/wiki/CUDA. Accessed: 2024-10-16. 2024.

[60] TechPowerUp. *NVIDIA GeForce RTX 3060 Mobile Specs*. Accessed: 2024-03-12. 2024. URL: https://www.techpowerup.com/gpu-specs/geforce-rtx-3060-mobile.c3757.

[61] TechPowerUp. *AMD Ryzen 9 5900HX Specs*. Accessed: 2024-03-12. 2024. URL: https://www.techpowerup.com/cpu-specs/ryzen-9-5900hx.c2371.

[62] NVIDIA Corporation. *CUDA C Programming Guide*. 2022. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/.

[63] David Beazley. *Understanding the Python GIL*. Presentation at PyCon 2010. 2010. URL: https://www.dabeaz.com/python/GIL.pdf.

[64] Ali Rahimi and Benjamin Recht. "Random Features for Large-Scale Kernel Machines". In: *Neural Information Processing Systems*. 2007. URL: https://api.semanticscholar.org/CorpusID:877929.

[65] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. "Independent Component Analysis". In: 2005. URL: https://api.semanticscholar.org/CorpusID:1299275.

[66] Andreas C. Damianou and Neil D. Lawrence. "Deep Gaussian Processes". In: *ArXiv* abs/1211.0358 (2012). URL: https://api.semanticscholar.org/CorpusID:5945613.

[67] Edward Snelson and Zoubin Ghahramani. "Sparse Gaussian Processes using Pseudo-inputs". In: *Neural Information Processing Systems*. 2005. URL: https://api.semanticscholar.org/CorpusID:394337.

[68] Joaquin Quiñonero Candela and Carl Edward Rasmussen. "A Unifying View of Sparse Approximate Gaussian Process Regression". In: *J. Mach. Learn. Res.* 6 (2005), pp. 1939–1959. URL: https://api.semanticscholar.org/CorpusID:16005390.

[69] Edward Snelson and Zoubin Ghahramani. "Local and global sparse Gaussian process approximations". In: *International Conference on Artificial Intelligence and Statistics*. 2007. URL: https://api.semanticscholar.org/CorpusID:585696.

[70] Christopher Williams and Matthias Seeger. "Using the Nyström Method to Speed Up Kernel Machines". In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper_files/paper/2000/file/19de10adbaa1b2ee13f77f679fa1483a-Paper.pdf.

[71] Manfred Opper. "Sparse Online Gaussian Processes". In: 2008. URL: https://api.semanticscholar.org/CorpusID:8581477.

[72] Yuan Qi, Ahmed H. Abdel-Gawad, and Thomas P. Minka. "Sparse-posterior Gaussian Processes for general likelihoods". In: *ArXiv* abs/1203.3507 (2010). URL: https://api.semanticscholar.org/CorpusID:9560740.

[73] Matthias W. Seeger, Christopher K. I. Williams, and Neil D. Lawrence. "Fast Forward Selection to Speed Up Sparse Gaussian Process Regression". In: *International Conference on Artificial Intelligence and Statistics*. 2003. URL: https://api.semanticscholar.org/CorpusID:17404261.

[74] James Hensman, Alexander G. de G. Matthews, and Zoubin Ghahramani. "Scalable Variational Gaussian Process Classification". In: *International Conference on Artificial Intelligence and Statistics*. 2014. URL: https://api.semanticscholar.org/CorpusID:12892620.

[75] Alexander Graeme De Garis Matthews. "Scalable Gaussian process inference using variational methods". PhD thesis. Apollo - University of Cambridge Repository, 2017. DOI: 10.17863/CAM.25348. URL: https://www.repository.cam.ac.uk/handle/1810/278022.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor, professor Mattia Bruschetta, for his invaluable guidance, encouragement, and unwavering support throughout this research journey. His expertise and insights have been instrumental in shaping my work, and I am truly grateful for the opportunity to learn under his mentorship.

I also extend my sincere thanks to Antonio, Lorenzo, and Giacomo for their generous time, insightful discussions, and continuous support. Their contributions have significantly enriched my understanding and strengthened this research.

To my parents, I cannot thank you enough for your unconditional love, patience, and encouragement. Your support has been my greatest source of strength, and I dedicate this achievement to you.

To my brother, thank you for your generosity and high spirits. When I was in need, you gave me the computer on which this whole thesis work was written and created. This work would not have been possible without you.

To Asya, who has been by my side throughout this journey. Through every moment of stress and doubt, she shared the burden with me, offering her unwavering support and comfort when I needed it most.

Finally, to my friends, thank you for always being therewhether to celebrate small victories, provide motivation during challenging times, or simply offer a much-needed distraction. Your companionship has made this journey all the more memorable.

This work would not have been possible without each of you, and for that, I am deeply grateful.