University of West Bohemia

Faculty of Applied Sciences

Department of Cybernetics

DIPLOMA THESIS

Data and Hybrid Models of Dynamical Systems

Pilsen, 2023

Šmíd Matěj

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Matěj ŠMÍD**
Osobní číslo:	**A21N0122P**
Studijní program:	**N3918 Aplikované vědy a informatika**
Studijní obor:	**Kybernetika a řídicí technika**
Téma práce:	**Data and hybrid models of dynamical systems**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Review and summarise hybrid (physics-driven and data-driven) modelling approaches.
2. Analyse suitability of data-driven models (e.g., Gaussian process, neural network) for state estimation and control applications (e.g., in terms of expected accuracy, number of data, computational complexity).
3. Implement selected methods of hybrid modelling in Matlab/Python.
4. Assess contribution of physics-driven and data-driven parts.
5. Demonstrate and verify benefits of hybrid models over pure physics-driven or data-driven ones.
6. Summarise theoretical and simulation results.

Rozsah diplomové práce:	**40-50 stránek A4**
Rozsah grafických prací:	
Forma zpracování diplomové práce:	**tištěná**
Jazyk zpracování:	**Angličtina**

Seznam doporučené literatury:

Rasmussen, Carl Edward. „Gaussian processes in machine learning.“ *Summer school on machine learning*. Springer, Berlin, Heidelberg, 2003.
Torrente, Guillem, et al. „Data-driven MPC for quadrotors.“ *IEEE Robotics and Automation Letters* 6.2 (2021): 3769-3776.
Imbiriba, Tales, et al. „Hybrid Neural Network Augmented Physics-based Models for Nonlinear Filtering.“ *arXiv preprint arXiv:2204.06471* (2022).
Salzmann, Tim, et al. „Neural-MPC: Deep Learning Model Predictive Control for Quadrotors and Agile Robotic Platforms.“ *arXiv preprint arXiv:2203.07747* (2022).

Vedoucí diplomové práce:	**Doc. Ing. Jindřich Duník, Ph.D.** Katedra kybernetiky
Datum zadání diplomové práce:	**1. října 2022**
Termín odevzdání diplomové práce:	**22. května 2023**

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

L.S.

[illegible]

Doc. Ing. Miloš Železný, Ph.D.
děkan

[illegible]

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 1. října 2022

Abstract

This thesis presents a hybrid approach for modeling and of dynamics by combining first principles modeling and data-driven modeling. An unique property of the RGP is exploited, namely that it is able to fit the dynamics online, without the need for a training run, to fit time-varying aerodynamics. We propose a method RGP−MPC, which uses the hybrid model in a MPC controller, while changing the data-driven component of the hybrid model to account for model discrepancies. We demonstrate our method on a model of a quadrotor in simulation, using the Gazebo simulator. The RGP−MPC is able to track the desired trajectory and adapt to the changing drag forces present. This simulation experiment provides a proof of concept that the RGP−MPC method is able to improve the performance of the MPC controller in the presence of unknown discrepancies in the model.

Keywords – *Model Predictive Control, Gaussian Process regression, recursive Gaussian Process regression, quadrotor, hybrid model, data-driven Model, first principles model, Gazebo, simulation, adaptive control, online learning, time-varying Systems*

Tato práce prezentuje hybridní přístup k modelování dynamiky pomocí kombinace modelování prvních principů a modelování založeného na datech. Využita je unikátní vlastnost RGP, a to že je schopen přizpůsobit svojí dynamiku online, bez nutnosti předsběru dat během trénování, na časově proměnné aerodynamické síly. Navrhujeme metodu RGP−MPC, která používá hybridní model v MPC regulátoru, přičemž mění datově založenou složku hybridního modelu tak, aby zohledňovala rozdíly mezi modelem a reálným systémem. Metoda je demonstrována na modelu quadrotoru v simulaci, pomocí simulátoru Gazebo. RGP−MPC je schopen sledovat požadovanou trajektorii a přizpůsobit se měnícím se aerodynamickým silám. Tento simulační experiment poskytuje důkaz, že metoda RGP−MPC je schopna zlepšit výkon MPC regulátoru v přítomnosti neznámých rozdílů mezi modelem a reálným systémem.

Klíčová slova – *Model Predictive Control, Gaussovské procesy, rekurzivní Gausovské procesy, dron, hybridní model, modelování vedené daty, modelování z prvních principů, Gazebo, simulace, adaptivní řízení, online učení, časově proměnné systémy*

Dedication

I would like to dedicate this work to the many people I have had the good fortune of meeting during my studies at the University of West Bohemia. I would like to thank my supervisor, Ing. Jindřich Duník, Ph.D., for his guidance during the writing this thesis. I would also like to thank my family and friends for their support and encouragement.

Contents

Chapter 1

Introduction

ACCURATE models are a necessity for precise control of dynamical systems. In general, modeling approaches can be divided into two. First approach consists of using known physics of the studied system, perhaps using measurements of the system to identify the model parameters. The other approach is to use data-driven models, such as machine learning (ML) models, to describe the system. A general model with little inductive bias (the apriori information about the system encoded in the model structure) is fit to the data measured on the system. Both these approaches provide unique benefits to the modeling process, but also have their own drawbacks. This thesis deals with the problem of combining the two approaches to create a hybrid model that balances the properties of both approaches.

We present an approach to control a quadrotor using a first principles model augmented with an Recursive Gaussian Process (RGP) [18, 37] model that serves to describe the quadrotor's air drag characteristics. The augmented model is employed in a Model Predictive Control (MPC) framework allowing for precise trajectory tracking accounting for the quadrotor's air drag without assuming a priori knowledge of the air drag form or coefficients. The ML augmented models typically require a training dataset to be collected in a separate run which is then used to fit the ML model [33, 26]. In contrast, we fit the RGP model online at each time step in a recursive fashion. This allows us to adapt to changing environmental conditions *without* the need to collect a new training dataset.

We investigate the notion of *discrepancy modeling* to create a hybrid model, that minimizes the observed discrepancy between the model and the system. To this end, we seek to find discrepancies between the model and the actual system. This is a common problem in control theory, where the dynamics model is often a simplification of the actual system. The ability to adapt to changing conditions is a key feature of a successful autonomous system. ML models are a popular choice for data-augmenting a physics-based dynamics model [7, 21, 17, 33].

This thesis is organized as follows. The split between the two modeling approaches is discussed in chapter 2. Gaussian process regression is discussed in chapter 4 as our data-driven model of choice. Chapter 3 discusses the background theory on optimal control, in particular on MPC and its usege in trajectory tracking. In chapter 5 we discuss the simulation environment Gazebo used to test the proposed methods. Finally, in chapter 6 we present our method RGP−MPC [32] for combining the two modeling approaches and evaluate it in simulation.

Chapter 2

Modeling Approaches

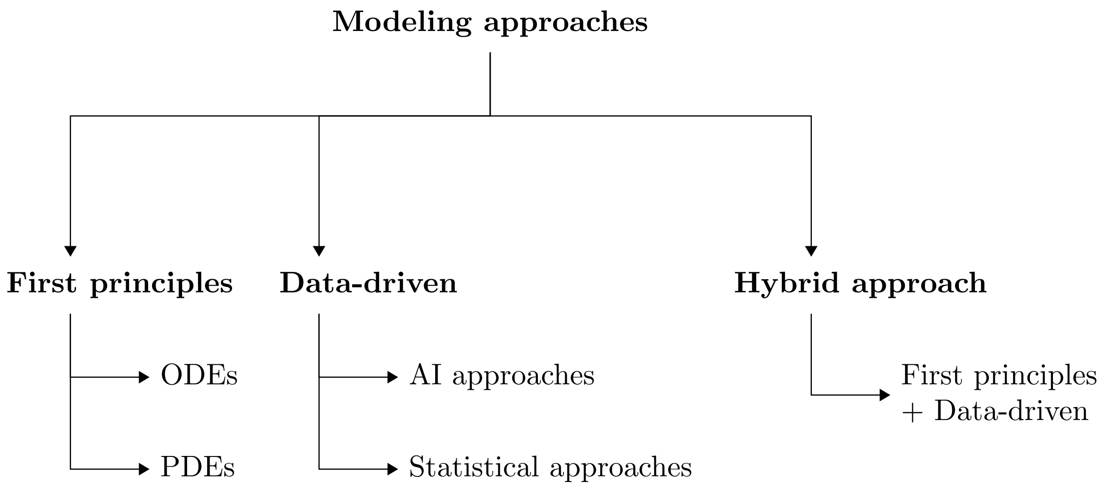


Figure 2.1: Taxonomy of the three modeling approaches discussed in this thesis

MODELS are abstract representations of reality, created using reasoning about said reality, that are used to explain and predict phenomena. They can take the form of mathematical equations, logical statements, computer programs. Using a model allows one to consider only those aspects of reality that are relevant to the problem at hand, and to ignore the rest.

First principles modeling and *data-driven* modeling are two different approaches to modeling physical systems. First-principles modeling involves using the fundamental laws and principles of physics to develop a mathematical model that describes the behavior of the system. This approach is based on a deep understanding of the underlying physics of the system and has the distinct advantage in that its generally transparent and such can be understood easier by users [24], which is why they tend toward the white side of the white-grey-black box scale discussed in section 2.1.

On the other hand, data-driven modeling involves acquiring input-output data from the system and using it to develop a model that can make predictions about the system's behavior. This approach does not require a deep understanding of the underlying physics of the system, but instead relies on data to uncover relationships between inputs and outputs, which leads to opaque models, falling into the category of black-box models, which represents a roadblock in their widespread application [38].

In general, first principles modeling tend to be more accurate than data-driven modeling, but at the cost of being more computationally expensive.

In summary, first-principles modeling is based on a deep understanding of the underlying physics of the system, while data-driven modeling relies on data to uncover relationships between inputs and outputs.

2.1 White Box, Black Box

All models can be assigned position on the *white-grey-black box* scale according to the amount of a priori information about the system that is encoded into the model structure. Black box models are constructed without any a priori information, leaving the structure free to fit the data. They are not concerned with the internal workings of the system, only with its input-output relationship. In contrast to this, white box models are constructed in such a way as to incorporate all the a priori information into the model structure, fully describing the system's interior. All models lie on a spectrum between these two extremes as seen in Fig. 2.2.

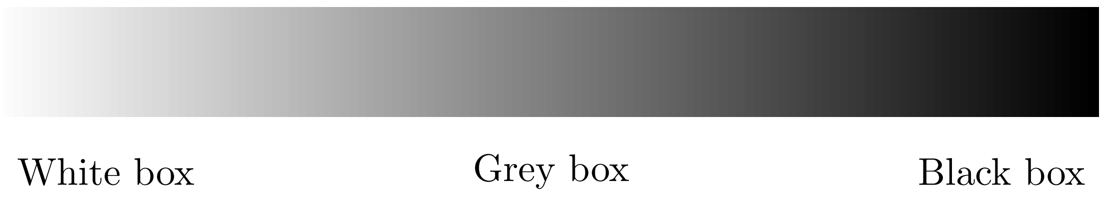


Figure 2.2: Different modeling approaches lie on the white-black box spectrum.

There is a distinction to be made between the a priori information available and the data available. Where the a priori information refers information about the model structure, the data available is only the observations made of the modeled system.

The available a priori information can be in terms of the type of function capable of capturing the behavior of the system, for example a second order linear ordinary differential equation. Since such a model's parameters still need to be identified we would not classify it as a strictly white box model. When no a priori information is available, a general model structure can be chosen, yielding an opaque, black box model.

Note that where a model lies on the white-black box spectrum is not directly related to the model's usefulness or validity. An unwieldy, opaque, black box model can be very useful in practice, while a white box model can be too complex to be of any use.

2.2 First Principles Modeling

Modeling from first principles involves using the fundamental laws and principles of physics to develop a mathematical model that describes the behavior of the system. Such a model is usually in the form of a set of differential equations that describe the evolution of the system over time, as is discussed in this thesis. A model created from first principles most often is not fully set, but rather contains parameters (mass, inertia, etc.) that need to be identified. This identification can be done either offline (gather data $\rightarrow$ identification) or online (real-time, recursive identification). Generally, as model complexity increases, the identification process becomes more difficult.

Identification Identification of model parameters can be either a complex or a simple process. If the model designer is able to measure the model parameters directly using sensors, then the identification process is simple. For example, the mass of a body can be measured using a scale. Many other parameters can be measured in this way, such as the dimensions, the density, the viscosity, etc. However, in many cases, it is not possible to measure the parameters directly. For example, the air resistance parameter of a body can be measured using wind tunnel experiments, but this can be very expensive and time consuming. In such cases, the parameters are estimated using indirect measurements using statistical methods such as by finding the parameters minimizing the least squares criterion [23], using predictive error minimization methods (PEM), the Kalman filter (KF) [22], the extended Kalman filter (EKF) [4], the particle filter (PF) [34] and the unscented Kalman filter (UKF) [20].

2.3 Data-driven Modeling

Data-driven models are made up of a general structure, that is restructured, or learned, using observed data from a system to build a model that can be used to predict the behavior of the system. The models employed do not require a deep understanding of the underlying physics of the system, but instead use mathematical models which can be fit to the observed data [2]. Often, they rely on minimizing the model prediction error in various ways, which can lead to potentially *greater* accuracy than their first principles counterparts, however, they are prone to overfitting to the observed data, leading to loss of generalization outside of the observed domain.

They can be generally divided into two categories: (1) machine learning approaches, such as neural networks [31] (NN) and (2) statistical approaches like Gaussian process (GP) regression [29] and recursive Gaussian process (RGP) regression [18] discussed in chapter 4.

2.4 Hybrid Models

The two modeling approaches described above can be combined, hybridized to create a model that is able to take advantage of the strengths of both approaches. Given a first principles model f_{phys} and a data-driven model f_{data} we can combine them to create a hybrid model f_{hybrid}.

The most general hybridization of two mathematical models is using an arbitrary function h to combine the two models like so:

General Model Hybridization

$$\left.\begin{aligned} \dot{x} &= f_{\text{phys}}(x) \\ \dot{x} &= f_{\text{data}}(x), \end{aligned}\right\} \quad \dot{x} = f_{\text{hybrid}}(x) = h(f_{\text{phys}}(x), f_{\text{data}}(x))$$

where x is the current state of the model, $\dot{x}$ is the state's time derivative.

The most straightforward approach is to choose h in additive form as

$$\dot{x} = f_{\text{hybrid}}(x) = f_{\text{phys}}(x) + f_{\text{data}}(x). \tag{2.1}$$

This choice is has many advantages, one of which is that it conserves the units of both submodels.

In the chapter 6, we develop a hybrid model of a quadrotor using this approach with f_{phys} describing the rigid body dynamics and f_{data} describing the body's drag.

Discrepancy Modeling Discrepancy modeling [10] is an approach for developing a data-driven model to augment an known first principles model. The idea is to use the first principles model of the system's dynamics and the data-driven model as a model of the measured discrepancy between the idealized model and the actual system as illustrated in figure 2.3. The discrepancy model is then

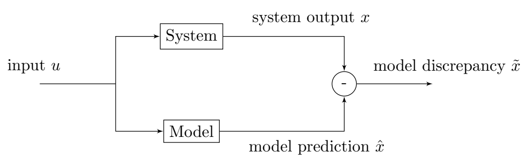


Figure 2.3: Discrepancy modeling allows for accuracy when the system under study has both the idealized known physics and hard to model disturbances.

used to correct the idealized model to obtain a more accurate model of the system's dynamics.

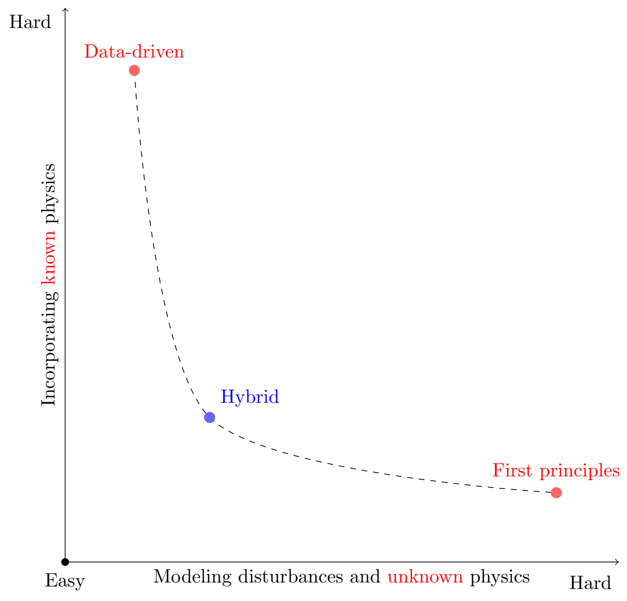

Hard
Data-driven
Incorporating known physics
Hybrid
First principles
Easy
Modeling disturbances and unknown physics
Hard

Figure 2.4: Discrepancy modeling allows for accuracy when the system under study has both the idealized known physics and hard to model disturbances.

Oftentimes the model designer has access to an idealized a priori physics model f_{phys}, that fails to capture some of the behavior of the system. The hard-to-model disturbances present can be prohibitively complex to model using the first principles approach. Therefore the data-driven approach can be used to create a comparatively simple model f_{data} of the discrepancy between the idealized model and the actual system. Ideally, we imagine the modeling problem to lie on a knee-shaped graph as shown in fig. 2.4, where we optimize the model to lie on the knee.

The physics model f_{phys} can be derived from first principles as described in section 2.2 while discrepancy model model f_{data} can be derived using the data-driven approach described in section 2.3.

There are many systems that can benefit from discrepancy modeling. For example, a robotic arm can measure the discrepancy resulting from the friction in the arm joints. This friction is difficult to model, since it changes due to the deterioration of the arm's components as a result of aging and changing environmental conditions.

Another example can be a discrepancy model of the air resistance on a moving body, supplementing a model of the aerodynamics of the body. Air resistance is a complex phenomenon that is difficult to model, usually requiring wind tunnel experiments to model. Using a machine learning model to fit the air resistance from the data can be a more efficient approach. This example is explored in chapter 6.

Another use is a discrepancy model of bearing chatter in rotating machinery. Bearing chatter is a complex phenomenon that is difficult to model, usually requiring expensive and time consuming experiments to model. Using a machine learning model to fit the bearing chatter from the data can be a more efficient approach.

2.5 A Short History of the Two Approaches

The split between the first-principle and data-driven modeling approaches is not new. It has been present in the scientific discourse for centuries. In this thesis, we examine two historical clashes between the antecedents of these approaches, namely from the discourse on the Heliocentric and Geocentric models of the solar system, as well as on the models of gravity and celestial bodies.

2.5.1 Kepler v. Ptolemy

The difference in the first principles and the data-driven modeling approaches can be illustrated with a historical excursion into the discourse on Heliocentrism and Geocentrism.

During Kepler's lifetime, the heliocentric model of the solar system was less accurate than the geocentric model, because of the presence of unmeasured disturbances, since many significant bodies were yet to be discovered: Uranus, Pluto, Saturn's moons, etc [10].

These unmeasured disturbances, seen in both the "correct" heliocentric and the "incorrect", were not as significant when seen from the perspective of the geocentric coordinate frame. The greater accuracy of the "incorrect" geocentric model was essentially a result of **overfitting** to the measured data.

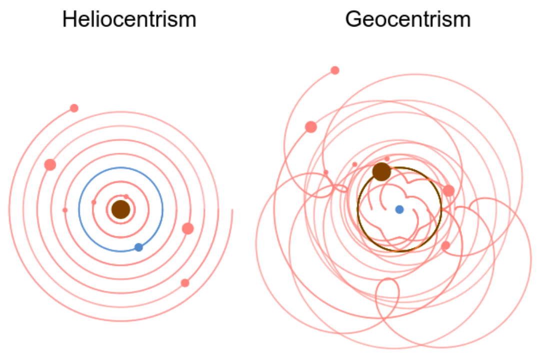


Figure 2.5: Orbits of the planets as seen from the Heliocentric and Geocentric coordinate frames [9]. The heliocentric model offers a simpler view of the phenomena, even if it was, at the time of its inception, less accurate than its geocentric counterpart.

Here, the "correct" heliocentric model was the first principles model, while the "incorrect" geocentric model was the data-driven model fitting better to the available data. The difference between these is illustrated in fig. 2.5. This historical discourse illustrates the roots of both approaches and the trade-off between accuracy and generalizability. Hybrid models of the solar system are not in use today, since the PBM model used nowadays, General Relativity, provides discrepancy-less accuracy.

2.5.2 Galileo v. Aristotle

In physics folklore, Galileo's ball drop experiment from the leaning tower of Pisa is often used to illustrate that the acceleration experienced by a falling body is constant, thus disproving Aristotle's theory of gravity, that states that objects fall at different speeds depending on their mass.

However, if we were to perform the experiment, we would find that the acceleration of the ball is not constant, but rather depends on the mass of the ball, much closer to Aristotle's theory. This is so, because the air resistance experienced by the ball changes with its Reynolds number, which depends on the mass of the ball.

Figure 2.6: Artistic depiction of Galileo's ball drop experiment. The experiment was most likely a thought experiment, and not an actual experiment

Galileo's most significant contribution was not the measurement and fitting of the data, but instead the position of the idea of a fixed acceleration, which was a conclusion that would have been exceptionally difficult to come to from such measurement data alone [10].

Both examples, i.e., *Kepler v. Ptolemy* and *Galileo v. Aristotle* illustrate the difference between the first principles and the data-driven modeling approaches. Where the first principles approach attempts to find *parsimonious* models that explain the data, the data-driven approach attempts to find models that fit the data as well as possible, without trying to understand the underlying principles. Aristotle's model fits the data well, but tells us little about the underlying physics. Galileo's model does not fit the data as well, but it does tell us something about the underlying physics.

The parsimony of the first principles approach is critical when using the philosophy of Occam's razor [11]: The simplest explanation is the most likely explanation; the simplest set of explanatory variables is the most suitable.

2.6 All Models Are Wrong

The phrase “All Models Are Wrong” originates from a 1976 aphorism by the late American statistician George Box [6]. The full quote is:

> All models are wrong, but some are useful.

This aphorism stresses that even though statistical models are fundamentally wrong, they can still be useful in practice. The search for the "correct" model is a futile one, as there is no such thing as a "correct" model. The model designer should instead focus on choosing a model based on two criteria: *parsimony* and *worrying selectively.*

Parsimony stresses the importance of Occam’s razor in model selection. The model designer should choose the simplest model that is able to explain the natural phenomena without excessive elaboration. Worrying selectively concerns the awareness of the limitations of the model. Since all models are *wrong*, the model designer must be aware of what is *importantly wrong.*

The idea of the aphorism predates Box’s work, it was articulated decades earlier both by scientists such as John von Neumann and by artists:

> We all know that art is not truth. Art is a lie that makes us realize truth, at least the truth that is given us to understand. The artist must know the manner whereby to convince others of the truthfulness of his lies.
> — Pablo Picasso

Chapter 3

Mathematical Optimization

Mathematical optimization deals with finding the best solution from a set of all feasible solutions. It consists of maximizing or minimizing a criterion (objective) function $f : \mathbb{X} \to \mathbb{R}$ by systematically choosing an input $\boldsymbol{x}$ from within a allowed set $\mathbb{X}$ and determining the value of the criterion.

In this thesis we deal with optimizing a continuous function evaluating the state of a model, but there exists a separate branch of optimization dealing with discrete functions. This branch is called combinatorial optimization and is not covered in this thesis.

We can further divide the optimization problems into two categories: constrained and unconstrained. In constrained optimization, the set $\mathbb{X} \subset \mathbb{R}^n$ and is defined by a set of constraints. In unconstrained optimization, the set $\mathbb{X}$ is the whole $\mathbb{R}^n$. We will focus on *constrained* optimization in this thesis which can be formulated as

where $\boldsymbol{x} \in \mathbb{X}$ is the vector of variables, $f : \mathbb{X} \to \mathbb{R}$ is the objective function, $h_i : \mathbb{R}^n \to \mathbb{R}$ are the equality constraints and $g_j : \mathbb{R}^n \to \mathbb{R}$ are the inequality constraints which constrain $\boldsymbol{x}$ to the allowed inputs $\mathbb{X}$.

3.1 Quadratic Programming

Quadratic programming (QP) is the process solving optimization problems where the criterion function f is *quadratic*, subject to linear equality and inequality constraints defined as

Quadratic Optimization Problem

$$\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \tag{3.1}$$

$$\text{subject to } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \tag{3.2}$$

where $\boldsymbol{Q}$ is a symmetric matrix representing the quadratic weights, $\boldsymbol{c}$ is a vector representing the linear weights, $\boldsymbol{A}$ is a matrix and $\boldsymbol{b}$ is a vector representing the linear inequality constraints.

IPOPT and HPIPM Solving QP problems numerically is a non-trivial task. Several algorithms have been developed to solve QP problems, such as the interior-point method, active-set method, and the augmented Lagrangian method. The interior-point method is the most popular method for solving QP problems. The interior-point method is implemented in the IPOPT[1] [36] package, instead the `acados` package uses the HPIPM solver[2] [12], since it is better suited to solving optimal control problems.

3.1.1 Nonlinear Programming

A nonlinear constrained optimization problem is an optimization problem, where the objective function and/or the constraints are nonlinear. The process of solving a nonlinear constrained optimization problem is called nonlinear programming (NLP). A constrained optimization problem can be posed in the following form

Nonlinear Constrained Optimization Problem

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x}), \tag{3.3}$$

$$\text{subject to } h_i(\boldsymbol{x}) = 0 \quad i = 1, \dots n_h, \tag{3.4}$$

$$\text{and } g_j(\boldsymbol{x}) \leq 0 \quad j = 1, \dots n_g, \tag{3.5}$$

where f is the nonlinear objective function, h_i are the equality constraints, and g_j are the inequality constraints.

Transformation into an unconstrained optimization problem To move from the constrained optimization problem to the unconstrained optimization problem, we transform the constraints to the objective function. This is done by introducing a linear penalty function P, that is large when the constraints are violated defined as

$$P = \sum_i \lambda_i h_i(\boldsymbol{x}) + \sum_j \mu_j g_j(\boldsymbol{x}) = \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\mu}^T \boldsymbol{g}(\boldsymbol{x}), \tag{3.6}$$

where $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are the penalty weights (also called the Lagrange multipliers) for the equality and inequality constraints respectively. The constraints h_i and g_j can be

[1]IPOPT source code: https://github.com/coin-or/Ipopt
[2]HPIPM source code: https://github.com/giaf/hpipm

aggregated into a single vector $\boldsymbol{h}$ and $\boldsymbol{g}$ as

$$\boldsymbol{h}(\boldsymbol{x}) = \begin{bmatrix} h_1(\boldsymbol{x}) \\ h_2(\boldsymbol{x}) \\ \vdots \\ h_{n_h}(\boldsymbol{x}) \end{bmatrix}, \quad \boldsymbol{g}(\boldsymbol{x}) = \begin{bmatrix} g_1(\boldsymbol{x}) \\ g_2(\boldsymbol{x}) \\ \vdots \\ g_{n_g}(\boldsymbol{x}) \end{bmatrix}. \tag{3.7}$$

By combining the objective function f and the penalty function $\boldsymbol{P}$ we obtain the Lagrangian L of the constrained optimization problem

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}\boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\mu}\boldsymbol{g}(\boldsymbol{x}). \tag{3.8}$$

The penalty function is then added to the objective function and we find the optimum by first maximizing over the Lagrange multipliers $\boldsymbol{\lambda}, \boldsymbol{\mu}$ and then minimizing over $\boldsymbol{x}$ while keeping the multipliers non-negative to get

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathbb{R}^n}, \ \max_{\boldsymbol{\lambda},\boldsymbol{\mu}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \\ &\text{subject to } \boldsymbol{\lambda} \geqq 0, \\ &\text{and } \boldsymbol{\mu} \geqq 0. \end{aligned} \tag{3.9}$$

The optimization problem as defined in (3.9) is called the *primal* problem. The *dual* problem is obtained by switching the order of the maximization and minimization.

$$\begin{aligned} &\max_{\boldsymbol{\lambda},\boldsymbol{\mu}} \ \min_{\boldsymbol{x}\in\mathbb{R}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}), \\ &\text{subject to } \boldsymbol{\lambda} \geq 0, \\ &\text{and } \boldsymbol{\mu} \geq 0. \end{aligned} \tag{3.10}$$

In general, the optimal solutions to a *primal* problem is not equal to the equivalent *dual* problem solution, instead being separated by the *duality gap*. As such, the *dual* problem (3.10) gives us the lower bound on the optimal value of the *primal* (3.9) problem. However, under specific conditions the *duality gap* is zero, referred to as *strong duality*.

Karush–Kuhn–Tucker conditions The Karush–Kuhn–Tucker (KKT) conditions are the necessary conditions that for $\boldsymbol{x}$ to be a critical point. Since the Lagrangian function combines all the information about the optimization problem into a single function using the Lagrange multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, the Lagrangian function L can be optimized by finding critical points where its gradient is zero. Since the Lagrangian function is a function of $\boldsymbol{x}$, $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, the gradient is a vector of partial derivatives with respect to each of these variables. The critical point of this problem must necessarily satisfy the KKT conditions defined as

KKT conditions

$$\nabla L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \begin{bmatrix} \frac{\partial L}{\partial \boldsymbol{x}} \\ \frac{\partial L}{\partial \boldsymbol{\lambda}} \\ \frac{\partial L}{\partial \boldsymbol{\mu}} \end{bmatrix} = \begin{bmatrix} \nabla f + \boldsymbol{\lambda} \nabla \boldsymbol{h} + \boldsymbol{\mu} \nabla \boldsymbol{g}^* \\ \boldsymbol{h} \\ \boldsymbol{g}^* \end{bmatrix} = \boldsymbol{0}. \tag{3.11}$$

Since the inequality constraints are only active when $\boldsymbol{g}(\boldsymbol{x}) \to 0$ we only need to consider the active inequality constraints, since those that are inactive do not participate in the solution and $\boldsymbol{\mu} = 0$ for them. The active set of inequality constraints is denoted by $\boldsymbol{g}^*$.

The active set method The active set method involves solving the KKT conditions while guessing which inequality constraints are active (and are hence are in the *active set*) if the solution. They typically start by assuming that all inequality constraints are inactive, finding a solution in $\boldsymbol{x}$ and then checking for feasibility. If the solution is feasible, then the solution is optimal. If the solution is not feasible, then the active set is updated by adding the violated constraints to the active set and repeating the process. This is repeated until the solution is feasible.

3.2 Model Predictive Control

Model Predictive Control (MPC) [16, 25] is a control scheme used to control a system described by a system of ordinary differential equations $\dot{\boldsymbol{x}} = \boldsymbol{f}_C(\boldsymbol{x}, \boldsymbol{u})$ around a desired state $\boldsymbol{x}^*$ using a computed control input $\boldsymbol{u}^*$. The state of the system is a vector $\boldsymbol{x} \in \mathbb{X}$ and the control input is a vector $\boldsymbol{u} \in \mathbb{U}$.

To quantify the optimality of a solution we define a cost function $\mathcal{L}$ that penalizes deviations from the desired state $\boldsymbol{x}^*$. In addition, it is often desirable to penalize the $\boldsymbol{u}$, for computational reason, since a penalty on $\boldsymbol{u}$ may make the problem easier to solve, but also for reasons of modeling, since we want to avoid values of $\boldsymbol{u}$ that result in excessive usage of energy. For these reasons, we define the cost function as : $\mathcal{L} : \mathbb{X} \times \mathbb{U} \to \mathbb{R}_0^+$.

We require of this cost function to be 0 when the system is in the desired state $\boldsymbol{x}^*$ and $\boldsymbol{u}^*$, i.e. $\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{u}^*) = 0$ and to be positive otherwise. We also require that the cost function is continuous and differentiable. A common choice of cost function is the quadratic cost function defined as

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x} - \boldsymbol{x}^*)^T \boldsymbol{Q} (\boldsymbol{x} - \boldsymbol{x}^*) + (\boldsymbol{u} - \boldsymbol{u}^*)^T \boldsymbol{R} (\boldsymbol{u} - \boldsymbol{u}^*). \tag{3.12}$$

Given a cost function $\mathcal{L}$ and a prediction horizon length $N \geq 2$, we can define a optimal control problem (OCP_N) as

OCP_N problem

$$J_N(\boldsymbol{x}_0) \triangleq \min_{\boldsymbol{u}_{0:N-1}} \sum_{k=0}^{N-1} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{3.13}$$

$$\text{with respect to } \boldsymbol{u}_{0:N-1} \in \mathbb{U}^N, \quad \boldsymbol{x}_{0:N-1} \in \mathbb{X}^N, \tag{3.14}$$

$$\text{subject to } \boldsymbol{x}_{k+1} = \boldsymbol{f}_\mathrm{D}(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{3.15}$$

where $\mathbb{U}^N \triangleq \cup_{k=0}^N \mathbb{U}$ is a set of control inputs to optimize over, $\mathbb{X}^N \triangleq \cup_{k=0}^N \mathbb{X}$ is a set of admissible states to optimize over and $\boldsymbol{f}_\mathrm{D}$ is the function $\boldsymbol{f}_\mathrm{C}$ discretized, for example using the equation (6.4). Solving this OCP allows us to find the optimal control input $\boldsymbol{u}^*_{0:N-1}$ for the next N time steps. This, however, is not enough to control the system effectively, since it does not take into account the state measurements of the system. To extend the OCP_N with feedback we define a MPC feedback control law μ_N that uses the current state of the system to compute the optimal control input $\boldsymbol{u}^*_{0:N-1}$ for the next N time steps at each control interval. This feedback control is defined as:

Algorithm 1 Basic MPC algorithm for constant reference $\boldsymbol{x}^\mathrm{ref} = \boldsymbol{x}^*$ for time samples $t_n; n = 0, 1, 2, \ldots$

```
1: Measure the state $\boldsymbol{x}(t_n) \in \mathbb{X}$
2: Set $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}(t_n)$
3: Solve the $\mathrm{OCP}_N$ and denote the obtained solution as $\boldsymbol{u}^*_{0:N-1} \in \mathbb{U}^N$
4: Define the MPC feedback value $\mu_N(\boldsymbol{x}(t_n)) \triangleq \boldsymbol{u}^*_0$
```

Running the algorithm 1 in a loop allows one to control the controlled plant in a receding horizon fashion. MPC has been historically used primarily in the process and chemical industries, where the system dynamics are slow enough to make the MPC computations feasible in real time. However, with the advent of faster, more available compute and automatic differential tools like `CasADi`, MPC has become more popular in the field of robotics and in recent years have been used in the automotive industry, for instance in gasoline engines mass produced by General Motors as of 2018 [5].

3.3 Programming Methods

To solve the optimization problem, we need to find the optimal value of the decision variables x. This can be done using a variety of methods. The most common methods we will discuss here are the *Newton's method* and *sequential quadratic programming*.

3.3.1 Newton's Method

Newton's method, also known as Newton–Raphson method, is a iterative method for *unconstrained* optimization. The method can be used to find critical points of a fuction $\phi : \mathbb{R} \rightarrow \mathbb{R}$ of x by finding the roots of its derivative $\phi' = 0$. In comparison to other methods, particularly gradient descent, Newton's method converges faster, since it uses the second derivative of ϕ to take a more direct route to the minimum as illustrated in figure 3.1. However, the second derivative ϕ'' needs to exist and if it does, its computation is often expensive.

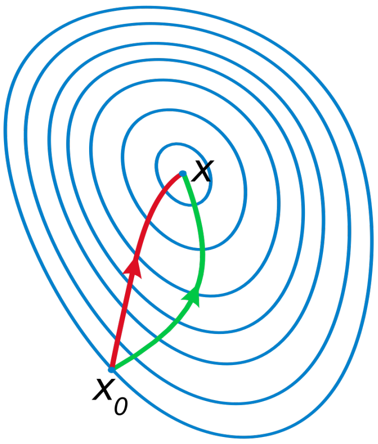


Figure 3.1: A comparison of gradient descent (green) and Newton's method (red) for minimizing a function (with small step sizes). Newton's method uses curvature information using the second derivative of ϕ to take a more direct route.

Given a twice-differentiable function ϕ, we seek to solve the optimization problem

$$\min_{x} \phi(x), \tag{3.16}$$

by constructing a sequence of iterates $\{x_k\}_0^N$ that converge to a minimizer x^* of f. At each iteration k, the Taylor series expansion of ϕ about x_k is constructed as

$$\phi(x_k + t) \approx \phi(x_k) + \phi'(x_k)t + \frac{1}{2}\phi''(x_k)t^2. \tag{3.17}$$

To find the step t that minimizes $\phi(x_k + t)$, we set the derivative of the Taylor expansion to zero and solve for t as follows

$$\frac{d}{dt}\phi(x_k + t) = 0 \tag{3.18}$$

$$\frac{d}{dt}\left(\phi(x_k) + \phi'(x_k)t + \frac{1}{2}\phi''(x_k)t^2\right) = 0 \tag{3.19}$$

$$\phi'(x_k) + \phi''(x_k)t = 0. \tag{3.20}$$

The t that minimizes the Taylor expansion is given by

$$t = -\frac{\phi'(x_k)}{\phi''(x_k)}, \tag{3.21}$$

and the next iterate is then given by

$$x_{k+1} = x_k - \frac{\phi'(x_k)}{\phi''(x_k)}. \tag{3.22}$$

The above formulation deals with the univariate case of $x \in \mathbb{R}$. In the multivariate case, ie. $\boldsymbol{x} \in \mathbb{R}^n$, the Taylor expansion is given by

$$\phi(\boldsymbol{x}_k + \boldsymbol{t}) \approx \phi(\boldsymbol{x}_k) + \nabla\phi(\boldsymbol{x}_k)^T \boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T \nabla^2 \phi(\boldsymbol{x}_k)\boldsymbol{t}, \tag{3.23}$$

and the method can be extended to the multivariate case in (3.24) by simply replacing the derivative $\phi'(x)$ with the gradient $\nabla\phi(\boldsymbol{x})$ and the second derivative $\phi''(x)$ with the Hessian matrix $\nabla^2\phi(\boldsymbol{x})$.

Newton's Method Iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left(\nabla^2 \phi(\boldsymbol{x}_k)\right)^{-1} \nabla\phi(\boldsymbol{x}_k) \tag{3.24}$$

3.3.2 Sequential Quadratic Programming

One of the most effective methods for solving nonlinear constrained optimization problems from section 3.1.1 is sequential quadratic programming [14] (SQP). This method works by generating and solving quadratic subproblems [27], which it uses to iteratively converge to the solution of the parent problem. The critical points of the objective function $f(\boldsymbol{x})$ are also the critical points of its Lagrangian $L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$, since $f(\boldsymbol{x}) = L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ at the KKT point ($\boldsymbol{\lambda} = 0$ and $\boldsymbol{\mu} = 0$), therefore we can use Newton's method to find the critical points of the Lagrangian $L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ like so:

$$\begin{bmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{\lambda}_{k+1} \\ \boldsymbol{\mu}_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_k \\ \boldsymbol{\lambda}_k \\ \boldsymbol{\mu}_k \end{bmatrix} - \underbrace{\begin{bmatrix} \nabla^2_{\boldsymbol{xx}} L & \nabla \boldsymbol{h} & \nabla \boldsymbol{g}^* \\ \nabla \boldsymbol{h}^T & 0 & 0 \\ \nabla \boldsymbol{g}^{*T} & 0 & 0 \end{bmatrix}^{-1}}_{\nabla^2 L} \underbrace{\begin{bmatrix} \nabla f + \boldsymbol{\lambda}\nabla \boldsymbol{h} + \boldsymbol{\mu}\nabla \boldsymbol{g}^* \\ \boldsymbol{h} \\ \boldsymbol{g}^* \end{bmatrix}}_{\nabla L}. \tag{3.25}$$

The Newton step $\boldsymbol{d} = \left[d_{\boldsymbol{x}}, d_{\boldsymbol{\lambda}}, d_{\boldsymbol{\mu}}\right]^T$ from the iterate $(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)$ to the next iterate $(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1})$ is obtained by solving the linear system

$$\begin{bmatrix} \nabla^2_{\boldsymbol{xx}} L & \nabla \boldsymbol{h} & \nabla \boldsymbol{g}^* \\ \nabla \boldsymbol{h}^T & 0 & 0 \\ \nabla \boldsymbol{g}^{*T} & 0 & 0 \end{bmatrix} \begin{bmatrix} d_{\boldsymbol{x}} \\ d_{\boldsymbol{\lambda}} \\ d_{\boldsymbol{\mu}} \end{bmatrix} = \begin{bmatrix} \nabla f + \boldsymbol{\lambda}\nabla \boldsymbol{h} + \boldsymbol{\mu}\nabla \boldsymbol{g}^* \\ \boldsymbol{h} \\ \boldsymbol{g}^* \end{bmatrix}. \tag{3.26}$$

However, since the Hessian of the Lagrangian $\nabla^2 L$ is likely to be non-invertible, the Newton step $\boldsymbol{d}$ is to be found indirectly, by solving a *quadratic* minimization subproblem defined as

QP subproblem

$$\min_{\boldsymbol{d}} \; f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \nabla^2_{\boldsymbol{xx}} L(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)\boldsymbol{d} \tag{3.27}$$

$$\text{subject to } \nabla \boldsymbol{h}(\boldsymbol{x}_k)\boldsymbol{d} + \boldsymbol{h}(\boldsymbol{x}_k) = 0 \tag{3.28}$$

$$\text{and } \nabla \boldsymbol{g}(\boldsymbol{x}_k)\boldsymbol{d} + \boldsymbol{g}(\boldsymbol{x}_k) = 0. \tag{3.29}$$

The QP subproblem is still a non-linear optimization problem (solved for example using quadratic programming), however, it is much easier to solve than the general nonlinear programming parent problem. There exists many varieties of the SQP algorithm, making use of specifin assumptions about the problem structure, however, the general idea is the same. One can use various solvers to solve the QP subproblem, for example, the interior-point solver `fmincon` in MATLAB or solvers specialized for QP problems, such as the HPIPM solver discussed in section 3.1. The SQP algorithm is summarized in Algorithm 2 and its schematic representation is visualized in fig. 3.2.

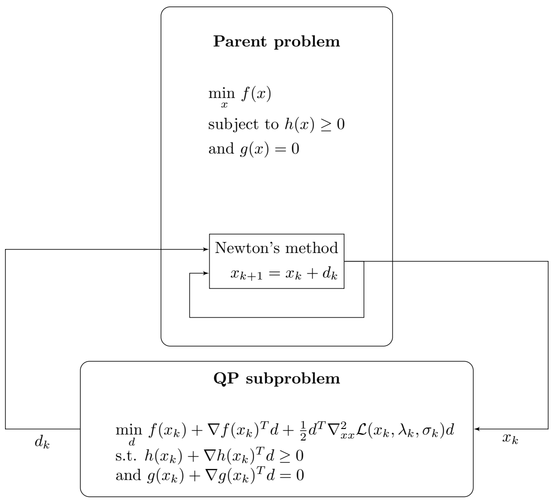

Parent problem
$\min_x f(x)$
subject to $h(x) \geq 0$
and $g(x) = 0$
Newton's method
$x_{k+1} = x_k + d_k$
QP subproblem
$\min_d f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k, \sigma_k) d$
s.t. $h(x_k) + \nabla h(x_k)^T d \geq 0$
and $g(x_k) + \nabla g(x_k)^T d = 0$
d_k
x_k

Figure 3.2: Schematic representation of the SQP algorithm.

Algorithm 2 Local SQP Algorithm

```
1: Choose initial iterate $(\boldsymbol{x}_0, \boldsymbol{\lambda}_0, \boldsymbol{\mu}_0)$
2: for $k = 0, 1, \ldots$ do
3:     Solve (3.27) to obtain $\boldsymbol{d}_k$
4:     Update iterate $(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1}) \leftarrow (\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k) + \boldsymbol{d}_k$
5:     if Convergence test satisfied then
6:         $\boldsymbol{x}^* \leftarrow \boldsymbol{x}_k$
7:         Stop
8:     end if
9: end for
```

Real-Time Iteration (RTI) SQP Solving each OCP$_N$ step (6.18) in the MPC framework is computationally expensive, and the computational burden increases with the prediction horizon. The `SQP-RTI` scheme [35] is a variant of the SQP algorithm that exploits the sequential similarity of the OCP$_N$ problem. At each sampling instant i, the NMPC solution is updated using a full Newton step, instead of performing the SQP algorithm to full convergence [15]. The Newton step is taken on the NMPC solution obtained at the previous time instant $i - 1$. This approach is computationally efficient, since it does not require the full convergence of the SQP algorithm at each sampling instant.

3.3.3 `CasADi` and `acados`

`CasADi` and `acados` are two software frameworks we use in this thesis to construct and solve optimization problems. They are both open-source and written in C++ with interfaces to Python and MATLAB.

`CasADi` [3] is an open-source tool[3] for nonlinear optimization and algorithmic differentiation. Using an internal symbolic representation of expression, it facilitates the efficient computation of symbolic differentiation using algorithmic differentiation required to solve optimization problems.

This framework allows the user to code a wide variety of nonlinear optimization problem (NLP) formulations. It also includes a suite of NLP solvers, including SQP, `IPOPT` and more.

`acados` [35] is a software package[4] of solvers designed for nonlinear OCPs. This is in contrast with `CasADi` which only solves general NLP problems. `acados` implements SQP type solvers tailored to OCP structured NLPs, which aim to solve those problems very fast. The solution time of acados for typical MPC problems is expected to be orders of magnitude faster compared to using `IPOPT` in `CasADi`.

[3]Source code: `https://web.casadi.org/`
[4]Source code: `https://docs.acados.org/`

Chapter 4

Gaussian Process Regression

4.1 Gaussian Process Regression

GAUSSIAN process regression (GP) [29] is a non-parametric method that generalizes Gaussian probability *distribution*. Instead of describing the probability over a scalar (or vector) space, stochastic *processes* describe probability over function space.

Gaussian process does not draw a function as-is, instead it only evaluates the function at a finite number of points, sidestepping the issue of evaluating an infinite-dimensional vector. Every finite collection of these function values has a multivariate normal distribution. The Gaussian process is the joint distribution of all of those function values, and is thus a distribution over functions with continuous domain. Gaussian processes can be used for regression problems as well as for classification. Classification problems are concerned with providing outputs as discrete labels, whereas in regression, the outputs are continuous variables.

The GP is a non-parametric method which, when used for regression tries to regress observations $\boldsymbol{o}_k = \{\boldsymbol{x}_k^\bullet, \boldsymbol{y}_k^\bullet\}$, where the symbol "$\bullet$" signifies a observation, generated from a noisy process

$$\boldsymbol{y}_k^\bullet = g(\boldsymbol{x}_k^\bullet) + \epsilon_k, \tag{4.1}$$

where $\epsilon_k \sim \mathcal{N}(0, \sigma_n^2)$ is the process noise.

For the GP to make predictions, we need to provide it with access to a training dataset of n observations $\mathcal{D} = \{\boldsymbol{o}_i | i = 1, ..., n\}$. The GP is then conditioned on the training data and can be used to make predictions $\boldsymbol{y}$ on new data points $\boldsymbol{x}$.

The covariance function (also called a kernel) used changes the possible behavior of the GP. An example of a kernel is the *squared exponential kernel* $\boldsymbol{K}$ given by

Source code available at `https://github.com/smidmatej/Gaussian-process`

Squared exponential kernel

$$K((\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')\boldsymbol{L}^{-1}(\boldsymbol{x} - \boldsymbol{x}')^T) + \sigma_n^2, \tag{4.2}$$

where σ_f is the signal variance, σ_n is the noise variance and $\boldsymbol{L}$ is the length scale. These three parameters form the hyperparameters $\eta = \left(l, \sigma_f, \sigma_n\right)$ of the GP.

A Gaussian process is completely specified by its mean function $\boldsymbol{m}$, its covariance function $\boldsymbol{K}$ and the available dataset $\mathcal{D}$. The mean function is usually set to zero for simplicity, but not necessarily.

We define mean function $\boldsymbol{m}$ and the covariance function $\boldsymbol{K}$ of a estimated process $\hat{g}$ as

$$\begin{aligned} m(\boldsymbol{x}) &= \mathbb{E}[\hat{g}(\boldsymbol{x})], \\ K(\boldsymbol{x}, \boldsymbol{x}') &= \mathbb{E}[(\hat{g}(\boldsymbol{x}) - m(\boldsymbol{x}))(\hat{g}(\boldsymbol{x}') - m(\boldsymbol{x}'))], \end{aligned} \tag{4.3}$$

and will write the Gaussian process as

$$\hat{g}(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), K(\boldsymbol{x}, \boldsymbol{x}')). \tag{4.4}$$

When provided no data, i.e., $\mathcal{D} = \emptyset$, the GP is said to be *unconditioned* and the prior distribution is given by a normal distribution with mean $m(\boldsymbol{x})$ and covariance $K(\boldsymbol{x}, \boldsymbol{x}')$. For $\eta = \{1, 1, 1\}$ with $m(\boldsymbol{x}) = 0$ the distribution for each $\boldsymbol{x}$ is then $\mathcal{N}(0, 1)$ as seen in fig 4.1.

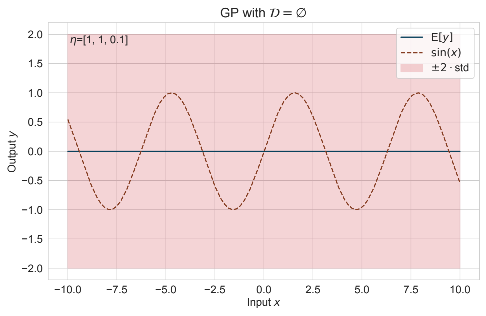
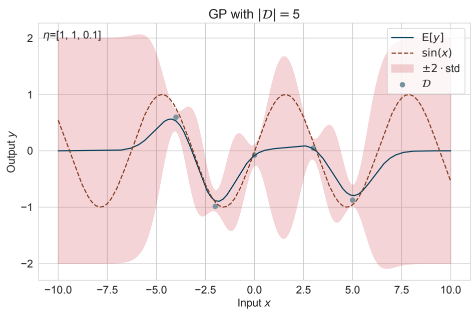


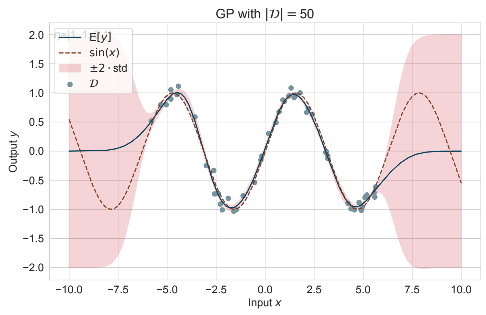
Figure 4.1: A Gaussian process in $\mathbb{R}$ with no training data. For each value of x the GP output is a normal distribution $\mathcal{N}(0, 1)$

Adding training data $\mathcal{D}$ to the GP, the posterior distribution changes to accommodate the new data. In fig. 4.2a we see the posterior distribution of a GP with 5 training samples. The GP conforms to the data, where available, but it is plainly visible that the GP does not generalize well outside the training data.

Adding more data, i.e. increasing the size of $\mathcal{D}$, the GP is able to fit the data better. In fig. 4.2b we see the posterior distribution of a GP with 50 training samples. The GP is able to fit the data well where it is available, but does not do a good job of generalizing the fit outside $\mathcal{D}$.



(a) GP with $|\mathcal{D}| = 5$



(b) GP with $|\mathcal{D}| = 50$

.

Figure 4.2: The GP regressed to $y = \sin(x)$ with 5 and 50 training samples with no MLE. The probability distribution conforms to the available data to according to its hyperparameters η.

4.1.1 Maximum Likelihood Estimation

Maximum likelihood estimation (MLE) is a method of estimating the parameters of a given probability distribution, given some observed data. This is done by maximizing the likelihood function of the distribution. The likelihood function is the probability of observing the data $\boldsymbol{\xi} = \{\xi_1, \xi_2, \ldots, \xi_{n_{\text{mle}}}\}$ given the parameters θ of the distribution γ. The likelihood function for $\gamma \sim \mathcal{N}(\mu, \sigma)$, i.e., a normal distribution is given by

$$\Omega(\theta; \boldsymbol{\xi}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\xi_i - \mu)^2}{2\sigma^2}\right), \tag{4.5}$$

where $\theta = \{\mu, \sigma\}$. The goal of the MLE is to find the parameters θ that maximize the likelihood function, i.e.,

$$\theta^* = \arg\max_{\theta} \Omega(\theta; \boldsymbol{\xi}). \tag{4.6}$$

For mathematical convenience, the log-likelihood function is often used instead of the likelihood function. The log-likelihood function is given by

$$\omega(\theta; \boldsymbol{\xi}) = \log \Omega(\theta; \boldsymbol{\xi}), \tag{4.7}$$

whose maximum is the same as the maximum of the likelihood function, since the natural logarithm is a monotonic.

MLE for GP hyperparameters To find the maximizing hyperparameters η^* of the GP, we need to find the maximum of its log-likelihood function. The log-likelihood function of a GP with a dataset $\mathcal{D}$ given hyperparameters η can be calculated [29]

(Section 2.2, Algorithm 2.1) using the following algorithm:

Algorithm 3 Log-likelihood function of a GP

```
    Input: $\eta = \left(l, \sigma_f, \sigma_n\right)$, $\mathcal{D} = \{(\boldsymbol{x}^\bullet, \boldsymbol{y}^\bullet)\}$
1:  $\overline{K} = K(\boldsymbol{x}^\bullet, \boldsymbol{x}^\bullet)$
2:  $L = \mathrm{chol}(\overline{K})$
3:  $\boldsymbol{\alpha} := L^T \setminus (L \setminus \boldsymbol{y}^\bullet)$
4:  $\omega(\eta; \mathcal{D}) := -\frac{1}{2}\boldsymbol{y}^{\bullet T}\boldsymbol{\alpha} - \sum_i L_{ii} - \frac{n}{2}\log(2\pi)$
    Output: $\omega(\eta; \mathcal{D})$
```

Using the algorithm 3, we can use a standard numerical optimization algorithm to find the maximizing hyperparameters η^*. In this work we use the Python function `scipy.optimize.minimize` to perform the optimization.

In Fig. 4.3 we see that the GP does a much better job of fitting the data. The mean function touches the samples and all samples are within the ± standard deviation interval. The hyperparameter optimization procedure is called *training* the GP.

Here it is important to note that even though the data points are generated from a sine function, the GP is not aware of it. During training, MLE optimizes the hyperparameters so that the GP fits the data in a maximum likelihood sense. The GP is not aware of the underlying function that generated the data. This is an example of the generalization problem of machine learning, where the learned function has no means to generalize outside the training data, unless guided to, for example, by adding a regularization term to the MLE.

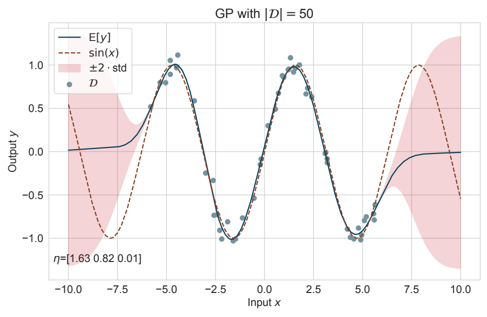
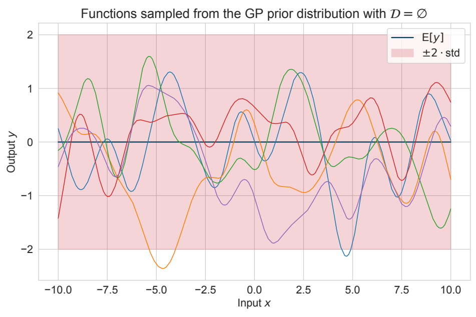
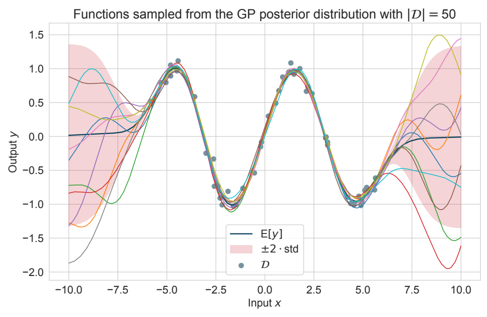


Figure 4.3: A Gaussian process in $\mathbb{R}$ with data after MLE. The data are identical to those in Fig. 4.2b.

Since GP defines a distribution over function space, we can freely draw samples from this distribution, ie. functions as seen in Fig. 4.4. These functions have to be sampled over x in order to display them.



(a) Function sampled from the GP with no training data.



(b) Function sampled from the GP with training data and MLE.

Figure 4.4: The GP represents a probability distribution over function space. We can therefore draw sample functions from this distribution to better understand the GP.

4.2 Recursive Gaussian Process

Recursive Gaussian process (RGP) regression [18] is a method for *online* learning of Gaussian processes. Standard GP regression described in section 4.1 requires the entire training dataset $\mathcal{D}$ to be available before the model can be learned using hyper-parameter optimization. In contrast to this, RGP allows for updating the GP model as new observations $\boldsymbol{o}_k$ become available from the underlying process

$$\boldsymbol{y}_k^{\bullet} = g(\boldsymbol{x}_k^{\bullet}) + \epsilon_k, \qquad \epsilon_k \sim \mathcal{N}(0, \sigma_n^2). \tag{4.8}$$

The RGP is initialized with a set of basis vectors $\mathbf{X}^+ \triangleq [\boldsymbol{x}_1^+, \boldsymbol{x}_2^+, \ldots, \boldsymbol{x}_m^+]$, which hold the local estimates $g^+ \triangleq g(\mathbf{X}^+)$ of the latent function $g(\cdot)$ and is chosen to be a set of m vectors that span the interval of interest of the input space $\boldsymbol{x}$. For example, one can choose to distribute $\mathbf{X}^+$ uniformly inside the interval of interest. The set $\mathbf{Y}^+ \triangleq \left[\boldsymbol{y}_1^+, \boldsymbol{y}_2^+, \ldots, \boldsymbol{y}_m^+\right]$ is the initial mean of the estimates g^+, which can be for simplicity initialized to zero. We refer to the set of basis vectors $\mathbf{X}^+$ and their estimates $\mathbf{Y}^+$ as $\mathcal{B} \triangleq \left\{\mathbf{X}^+, \mathbf{Y}^+\right\}$.

To run inference, we need only to evaluate on the basis vectors $\mathbf{X}^+$, which is a much cheaper operation than evaluating on the entire dataset $\mathcal{D}$. Thus, the basis vectors can be thought of as an active set allowing a sparse GP representation. The datapoints in $\mathcal{D}$ are not saved by the RGP, they only serve to update the estimates g^+ of the latent function, which alleviates the computational requirements, since $m \ll n$.

The initial distribution of the RGP is given by

$$p_0(g^+) = \mathcal{N}(g^+; \boldsymbol{\mu}_0^+, \boldsymbol{C}_0^+), \tag{4.9}$$

Source code available at https://github.com/smidmatej/RGP where $\boldsymbol{\mu}_0^+ \triangleq \mathbf{0}$ is the mean at $\mathbf{X}^+$ and $\boldsymbol{C}_0^+ \triangleq K(\mathbf{X}^+, \mathbf{X}^+)$[1] is the covariance at $\mathbf{X}^+$.

The goal of the RGP method is then to use the new observations $\boldsymbol{o}_k$ as they become available to calculate the posterior distribution $p(g^+|\boldsymbol{o}_{1:k})$, where $\boldsymbol{o}_{1:k} \triangleq \{\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_k\}$, from the prior distribution

$$p(g^+|\boldsymbol{o}_{1:k-1}) \sim \mathcal{N}(\boldsymbol{\mu}_{k-1}^+, \boldsymbol{C}_{k-1}^+). \tag{4.10}$$

The desired posterior distribution $p(g^+|\boldsymbol{o}_{1:k})$ is expanded as

$$p(g^+|\boldsymbol{o}_{1:k}) = \int \underbrace{c_k \cdot p(\boldsymbol{o}_k|g^+, g^\bullet) \cdot \overbrace{p(g^\bullet|g^+) \cdot p(g^+|\boldsymbol{o}_{1:k-1})}^{\text{inference}}}_{\text{update}} \, \mathrm{d}g^+, \tag{4.11}$$

in two steps *inference* and *update*. The inference step is the calculation of the joint prior $p(g^+, g^\bullet|\boldsymbol{o}_{1:k})$, which provides the required correlation between $\boldsymbol{x}_k^\bullet$ and $\mathbf{X}^+$. The update step is the updating of the joint prior distribution with observations $\boldsymbol{y}_k^\bullet$ by applying Bayes' rule and integrating out $g^\bullet = g(\boldsymbol{x}_k^\bullet)$ where c_k is a normalization constant.

Inference

At any point in time, we can run inference to find the distribution of $g^\bullet = g(\boldsymbol{x}^\circ)$, i.e., of the latent function $g(\cdot)$ at an arbitrary point $\boldsymbol{x}^\circ$, by evaluating the joint prior $p(g^+, \boldsymbol{y}^\circ|\boldsymbol{o}_{1:k})$ and marginalizing out the basis vectors g^+ such as

$$\begin{aligned} p(g^+, g^\bullet|\boldsymbol{o}_{1:k}) &= p(g^\bullet|g^+) \cdot p(g^+|\boldsymbol{o}_{1:k}) \\ &= \mathcal{N}(g^\bullet; \boldsymbol{\mu}_k^\circ, \boldsymbol{C}_k^\circ) \cdot \mathcal{N}(g^+; \boldsymbol{\mu}_k^+, \boldsymbol{C}_k^+), \end{aligned} \tag{4.12}$$

where

Recursive Gaussian Process Inference

$$\boldsymbol{\mu}_k^\circ \triangleq m(\boldsymbol{x}^\circ) + \boldsymbol{H}_k \cdot (\boldsymbol{\mu}_k^+ - m(\mathbf{X}^+)), \tag{4.13}$$

$$\boldsymbol{C}_k^\circ \triangleq K(\boldsymbol{x}^\circ, \boldsymbol{x}^\circ) - \boldsymbol{H}_k \cdot K(\boldsymbol{x}^\circ, \mathbf{X}^+), \tag{4.14}$$

$$\boldsymbol{H}_k \triangleq K(\boldsymbol{x}^\circ, \mathbf{X}^+) \cdot K(\mathbf{X}^+, \mathbf{X}^+)^{-1}. \tag{4.15}$$

An example of the initial distribution of the RGP is shown in Fig. 4.5a. The initial distribution is a normal distribution with mean zero and covariance $K(\mathbf{X}^+, \mathbf{X}^+)$.

[1]When applied to vector arguments, the kernel K is applied element-wise to each pair of elements in the arguments, yielding a matrix.

Updating the RGP

To update the RGP we make use of several properties. Firstly, since g^+ is not observed the conditional $p(\boldsymbol{o}_{1:k}|g^+, g^\bullet) = p(\boldsymbol{o}_{1:k}|g^\bullet)$ is independent of g^+. Furthermore, both $p(\boldsymbol{o}_{1:k}|g^\bullet)$ according to 4.8 and $p(g^\bullet|\boldsymbol{o}_{1:k})$ according to 4.12 are Gaussian and therefore g^+ can be easily updated via a *Kalman filter* update step [22].

The Kalman filter update step is yields $p(g^\bullet|\boldsymbol{o}_{1:k}) = \mathcal{N}(g^\bullet; \boldsymbol{\mu}_k^{\text{KF}}, \boldsymbol{C}_k^{\text{KF}})$, where

$$\boldsymbol{\mu}_k^{\text{KF}} \triangleq \boldsymbol{\mu}_k^\bullet + \boldsymbol{G}_k \cdot (\boldsymbol{y}_k^\bullet - \boldsymbol{\mu}_k^\bullet), \tag{4.16}$$

$$\boldsymbol{C}_k^{\text{KF}} \triangleq \boldsymbol{C}_k^\bullet - \boldsymbol{G}_k \boldsymbol{C}_k^\bullet, \tag{4.17}$$

$$\boldsymbol{G}_k \triangleq \boldsymbol{C}_k^\bullet \cdot (\boldsymbol{C}_k^\bullet + \sigma_n^2 \boldsymbol{I})^{-1}. \tag{4.18}$$

However, we are not concerned with updating the distribution of $g^\bullet$ but rather the distribution of g^+. The multiplication of the Gaussians $p(g^\bullet|\boldsymbol{o}_{1:k})$ with $p(g^+|g^\bullet, \boldsymbol{o}_{1:k})$ yields a joint Gaussian distribution of g^+ and $g^\bullet$.with mean and covariance given by

$$\mu_k = \begin{bmatrix} \boldsymbol{\mu}_k^+ \\ \boldsymbol{\mu}_k^{\text{KF}} \end{bmatrix} \quad \text{and} \quad C_k = \begin{bmatrix} \boldsymbol{C}_k^+ & \boldsymbol{L}_k \cdot \boldsymbol{C}_k^{\text{KF}} \\ \boldsymbol{C}_k^{\text{KF}} \cdot \boldsymbol{L}_k^T & \boldsymbol{C}_k^{\text{KF}} \end{bmatrix}, \tag{4.19}$$

where

Recursive Gaussian Process Update

$$\boldsymbol{L}_k \triangleq \boldsymbol{C}_{k-1}^+ \boldsymbol{H}_k^T \left(\boldsymbol{C}_k^\bullet\right)^{-1}, \tag{4.20}$$

$$\tilde{\boldsymbol{G}}_k \triangleq \boldsymbol{L}_k \boldsymbol{G}_k = \boldsymbol{C}_{k-1}^+ \boldsymbol{H}_k^T \cdot (\boldsymbol{C}_k^\bullet + \sigma_n^2 \boldsymbol{I})^{-1}, \tag{4.21}$$

$$\boldsymbol{\mu}_k^+ = \boldsymbol{\mu}_{k-1}^+ + \tilde{\boldsymbol{G}}_k \cdot \left(\boldsymbol{y}_k^\bullet - \boldsymbol{\mu}_k^\bullet\right), \tag{4.22}$$

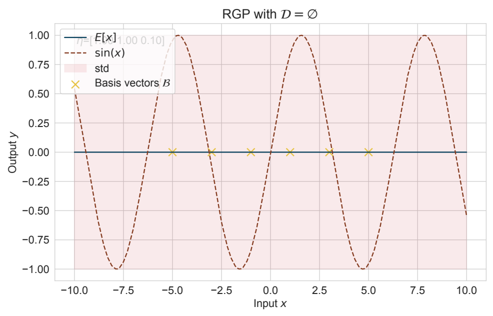
$$\boldsymbol{C}_k^+ = \boldsymbol{C}_{k-1}^+ - \tilde{\boldsymbol{G}}_k \boldsymbol{H}_k \boldsymbol{C}_{k-1}^+. \tag{4.23}$$

In order to keep memory usage and computational complexity low, we need only to update the distribution of g^+ at the basis vectors $\mathbf{X}^+$ while integrating $g^\bullet$ out which, in the case of a Gaussian distribution, is equivalent to neglecting $\boldsymbol{\mu}_k^\bullet, \boldsymbol{C}_k^\bullet$ and the cross-covariances $\boldsymbol{L}_k \cdot \boldsymbol{C}_k^{\text{KF}}$ and $\boldsymbol{C}_k^{\text{KF}} \cdot \boldsymbol{L}_k^T$.

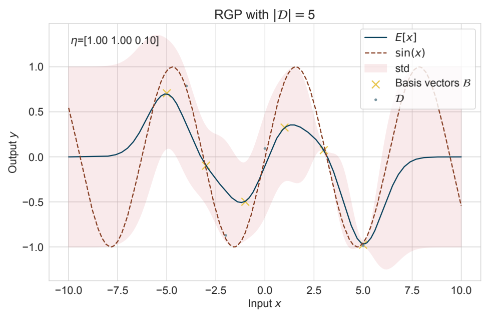
Combining together the *inference* and *update* parts at time steps $k = 1, 2, \ldots$ gives us the algorithm to recursively process observations $\boldsymbol{y}_k^\bullet$ at the inputs $\boldsymbol{x}_k^\bullet$ as described in Algorithm 4.

Algorithm 4 Recursive Gaussian Process

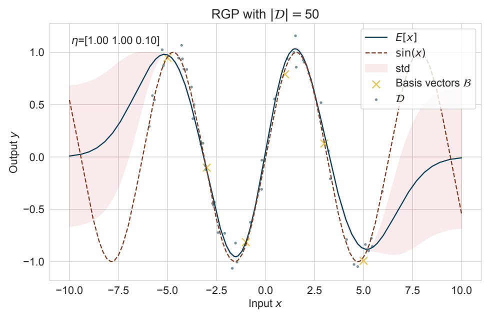
```
1: Inference step
2:     Calculate $\boldsymbol{H}_k$ using (4.15)
3:     Calculate $\boldsymbol{\mu}_k^\bullet$ using (4.13) and $\boldsymbol{C}_k^\bullet$ using (4.14)
4: Update step
5:     Calculate $\tilde{\boldsymbol{G}}_k$ using (4.21)
6:     Calculate $\boldsymbol{\mu}_k^+$ using (4.22) and $\boldsymbol{C}_k^+$ using (4.23)
```



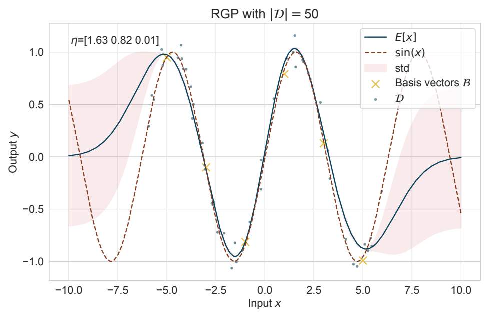
(a) Initial distribution of the RGP with no observations



(b) Posterior distribution with 5 observations



(c) Posterior distribution with 50 observations



(d) Posterior distribution with 50 observations and with hyperparameters chosen by maximum likelihood estimation on the RGP

Figure 4.5: The RGP regressed to a $y = g(x) = \sin(x) + \epsilon$ function with $\epsilon \sim \mathcal{N}(0, 0.1)$ noise. The RGP is initialized with $\eta = [1, 1, 0.1]$ and $\mathbf{X}^+ = [-5, -3, -1, 1, 3, 5]$. Figure 4.5d shows the posterior distribution of a RGP with the same $\mathcal{D}$ and $\mathcal{B}$ as in Fig. 4.5c, but with the hyperparameters chosen by maximum likelihood estimation on the RGP

While allowing for the online regression of a function, the RGP is limited in that it does not easily allow for the online hyperparameter estimation. Figure 4.5d shows the posterior distribution of a RGP with the same $\mathcal{D}$ and $\mathcal{B}$ as in Fig. 4.5b, but with the hyperparameters chosen by maximum likelihood estimation on the GP, as in 4.3. We see that the posterior distribution in Fig. 4.5d fits the data better and is able to both interpolate and extrapolate more accurately.

Chapter 5

Simulation Environments

SIMULATION is an important part of the development of any robotic system. When designing new hardware and software, it is conveneient to be able to test and validate them before building prototypes in the real world. This is especially true for aerial robots, where the cost of failure is high. Simulation environments allows for the testing of new algorithms and software in a safe and controlled environment. There are multiple considerations when choosing a simulation environment, such as the fidelity of the simulation, the ease of use.

In the robotics comunnity, the most popular simulation environments are Gazebo, Webots, and AirSim. However, simulation environments developed for use in machine learning and artificial inteligence contexts, have been emerging, notably from Unity's robotics simulator[1] and NVIDIA's Isaac Sim[2]. Notably, these simulators are often integrated with the Robot Operating System (ROS), which is the most popular middleware for robotics development, meaning experiments performed in simulation are able to brought into the physical world relatively easily and also that the developments are easily communicated within the robotics developent community. In this thesis we used Gazebo, as it is the most popular simulation environment in the ROS community, and it is the most integrated with the Robotic Operating System.

[1]https://unity.com/solutions/automotive-transportation-manufacturing/robotics
[2]https://developer.nvidia.com/isaac-sim

5.1 Robot Operating System

Robot Operating System (ROS)[3] is open-source middleware for robotics, maintained by Open Source Robotics Foundation, Inc., consisting of a set of tools and libraries that help in the development of robot applications.

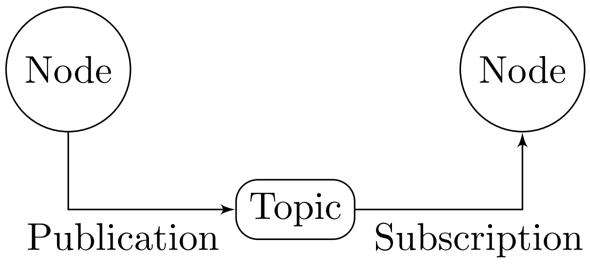


Figure 5.1: Basic ROS communication model.

Usage of ROS has become ubiquitous in both research and industrial contexts, being used by robotics manufacturers and related applications such as ABB, Bosch, BMW and Boeing.

The core of ROS is its message passing system, which allows for standardised communication between different parts of a robot. ROS processes are represented as nodes in a graph structure. These nodes post and receive messages to other nodes using a publish/-subscribe model to a specific topics as shown in Figure 5.1.

ROS also offers a set of tools for developers, including hardware abstraction, launch, debugging, project management and visualisation. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is not a real-time framework, though it is possible to integrate ROS with realtime code.

Figure 5.2: Latest (Release date: 2020) distribution of ROS, Noetic Ninjemys

ROS is primarily designed for Unix-based operating systems, especially for Ubuntu and Debian Linux distributions and macOS, with an experimental Windows branch also being available.

The main draw of ROS is that it is designed to support code reuse in robotics research and development. Being a distributed framework of loosely coupled *nodes*, it is possible to write a node that performs a specific task, such as path planning, and then share it in multiple projects.

[3] https://www.ros.org/

5.2 Gazebo Simulator

Gazebo[4] is a high-fidelity robot simulator, originaly developed by Dr. Andrew Howard and his student Nate Koenig, now maintained by Open Source Robotics Foundation.

Because Gazebo was originally developed to simulate robots in indoor environments, the name Gazebo was chosen as the closest outdoor structure to an indoor stage. Gazebo is a part of, and is seemlessly integrated in ROS. Gazebo allows the user to create a virtual robot and a virtual environment (world) with which the robot can interact. Gazebo's physics engine then allows for the simulation of physics phenomena, e.g., forces, friction, joint interactions, colisions and others. The user can then add functionality to the robot or the environment by adding premade (or creating their own) plugins. The plugins allow for changes in the behaviour of the world, model, sensors, physics, visualisation, the GUI and more. Gazebo was not made for aerial robots, hence it lacks some of the functionality of other simulators, such as `PX4 SITL`. The plugin `LiftDragPlugin` allows for basic aerodynamic simulations in Gazebo.

Figure 5.3: Logo of the Gazebo simulator, one of the most popular in the ROS community.

5.2.1 Physics Engine

At the core of Gazebo is its ability to simulate physical phenomena, the most important of these are the rigid body dynamics and the collision detection. For this purpuse Gazebo allows the user to choose between multiple physics engines. The default physics engine is ODE (Open Dynamics Engine)[5]. Aside from the rigid body dynamics, Gazebo also simulates the interaction of the robot with the environment, such as contacts but also sensing using cameras, laser range finders, etc.

5.2.2 RotorS Simulator

RotorS[6] is a Gazebo plugin that allows for the simulation of micro aerial vehicles (MAVs). It provides multiple multirotor models such as the AscTec Hummingbird, the AscTec Pelican, or the AscTec Firefly. We used the AscTec Hummingbird model to perform all of our simulations in Gazebo.

[4] https://gazebosim.org/
[5] https://www.ode.org/
[6] https://github.com/ethz-asl/rotors_simulator

Chapter 6

Experiment

To present how hybrid discrepancy modeling as described in chapter 2 can improve the performance of a control system, we use a quadrotor as a case study. We model a quadrotor using a combination of first principles modeling and Recursive Gaussian Process Regression, described in section 4.2. We use the model to control the quadrotor using Model Predictive Control as described in subsection 3.2. We compare the performance of the controller with and without the discrepancy model.

The usage of a discrepancy model is motivated by the fact that the quadrotor's air drag characteristics are not known a priori and can change over time. In particular, this changing in time motivates us to use a recursive Gaussian Process Regression model instead of a pretrained Gaussian Process Regression model. This means our model of the quadrotor is a *time-variant system.*

The drag model parameters, such as the coefficients, fluid density, or the quadrotor-related areas, and their spatial and time variability, are, unfortunately, known with a very *limited* accuracy. To improve the accuracy, the model of the total drag acceleration or its parameters should be identified from measured data, while the physical model (6.3) is respected. This, recently developed concept, is referred to as the *data-augmented physics-based* (DAPB) modeling [19].

Previous works have explored the case where the total acceleration error made up by both drag forces[1] (6.5), (6.6), was modeled using an offline trained Gaussian process (GP) which was then used as an extension to the physical model for a trajectory control [33, 26]. The approach leads to significant improvement of the resulting position accuracy assuming *no* time or spatial variability of the drag-related acceleration, which might not be always fulfilled.

The goal of this experiment is to further extend and simplify the applicability of hybrid modeling approaches by online identification of the drag-related acceleration.

Source code: https://github.com/smidmatej/mpc_quad_ros

[1]Since the relationship between acceleration and force is given by Newton's second law, it is not possible to directly disentangle the effects of the body and rotor drag from each other.

In particular, we

- Use the RGP described in section 4.2 for online drag modeling,
- Design a MPC algorithm capable of working with inherently time-varying DAPB models.

We illustrate the viability of the proposed method using a realistic scenarios generated Gazebo simulator [1] and publicly available source code[2].

The experiment chapter is organized as follows. Section 6.1 describes the first principles quadrotor model and assumptions about its drag characteristics. Section 6.2 describes in detail the proposed method for observing the drag, using those observations to train the RGP regression, augmenting the first principles model with the RGP, and using this augmented model in the MPC framework and finally, section 6.4 describes the simulation environment used for evaluation, the specificities of the conjunction of the MPC with the RGP, and the results.

6.1 First Principles Quadrotor Modeling

In this section, we describe the first principles model of the quadrotor using understanding of Newton's laws of linear and rotational motion. We also describe the assumptions about the drag characteristics of the quadrotor.

Notation To describe the quadrotor, we need to consider 2 reference frames. The world reference frame W is defined by the orthonormal basis $\{\boldsymbol{x}_B, \boldsymbol{y}_B, \boldsymbol{z}_B\}$ and the body reference frame $\{B\}$ by the basis vectors $\{\boldsymbol{x}_W, \boldsymbol{y}_W, \boldsymbol{z}_W\}$. The origin of $\{B\}$ is located in the center of mass of the quadrotor. We denote the vector from coordinate system $\{W\}$ to coordinate system $\{B\}$, as seen from the perspective of reference frame $\{C\}$, as $\boldsymbol{v}^C_{WB}$. In this paper, we only work with the world frame $\{W\}$ and the body frame $\{B\}$, so we omit the subscript "$_{WB}$" and write $\boldsymbol{v}^B$ as a shorthand for $\boldsymbol{v}^B_{WB}$ and $\boldsymbol{v}^W$ as a shorthand for $\boldsymbol{v}^W_{WB}$. The quaternion–vector product is denoted by $\odot$, representing the rotation of $\boldsymbol{v}$ by $\boldsymbol{q}$ is defined as $\boldsymbol{q} \odot \boldsymbol{v} = \boldsymbol{q}\boldsymbol{v}\boldsymbol{q}^*$, where $\boldsymbol{q}^*$ is the quaternion conjugate of $\boldsymbol{q}$. The cross–product of vectors $\mathbf{a}, \mathbf{b}$ is denoted as $\mathbf{a} \times \mathbf{b}$.

6.1.1 Quadrotor Model

We assume a quadrotor can be described as a rigid body with 6-DoF in free space. The state of the quadrotor $\boldsymbol{x}$ is chosen as the quadrotor current position $\boldsymbol{p}^W = [x, y, z]$, rotation quaternion $\boldsymbol{q}^W = \left(q_w, q_x, q_y, q_z\right)$, velocity $\boldsymbol{v}^W = \left[v_x, v_y, v_z\right]$ and angular rate $\boldsymbol{\omega}^W = \left[\dot{\varphi}, \dot{\theta}, \dot{\psi}\right]$.

[2]Source code available at: https://github.com/smidmatej/mpc_quad_ros

To describe rotation we use a quaternion describing the rotation of frame $\{B\}$ relative to frame $\{W\}$. Rotation of a object in $\mathbb{R}^3$ is described fully by 3 angles (such as using Euler angle representation), but the 4-dimensional quaternion with length 1 allows us to avoid the gimbal lock problem present in other representations.

The quadrotor's rotors are collocated on the xy-plane of the $\{B\}$ frame going through the center of mass of the quadrotor. The rotors are numbered 0 to 3 in a clockwise direction when seen from the positive $\boldsymbol{z_B}$ direction. Furthermore, the $\boldsymbol{x_B}$ axis is pointing forward, in between the zeroth and the third rotor as seen in Fig. 6.1.

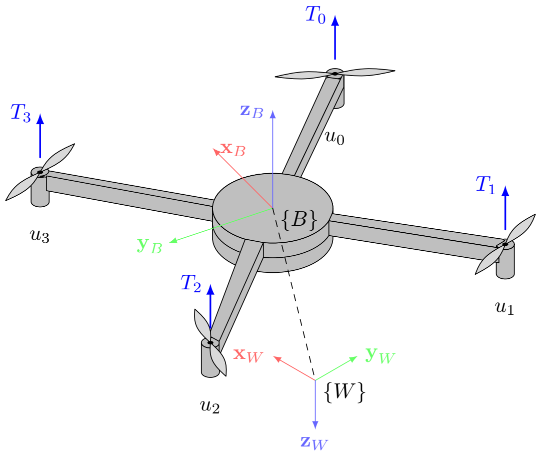


Figure 6.1: Quadrotor schematic. Thrust T_i is generated by individual rotors given inputs u_i The body frame $\{x_B, y_B, z_B\}$ is attached to the center of mass of the quadrotor. The world frame $\{x_W, y_W, z_W\}$ is attached to the inertial frame.

The model is controlled by an input vector $\boldsymbol{u} = [u_1, u_2, u_3, u_4]$ where $u_i \in [0,1]$ for $i = 0,1,2,3$. The input $\boldsymbol{u}$ corresponds to the activation of each individual rotor. Each rotor provides a thrust $T_i = T_{\max} \cdot u_i$. The collective thrust vector $\boldsymbol{T}^B$ and the torque vector $\boldsymbol{\tau}^B$, both expressed in the body frame $\{B\}$ and constituted by the individual thrusts T_i, are given by

$$\boldsymbol{T}^B = \begin{bmatrix} 0 \\ 0 \\ \sum_i T_i \end{bmatrix} \text{ and } \boldsymbol{\tau}^B = \begin{bmatrix} d_y(-T_0 - T_1 + T_2 + T_3) \\ d_x(-T_0 + T_1 + T_2 - T_3) \\ c_\tau(-T_0 + T_1 - T_2 + T_3) \end{bmatrix}. \tag{6.1}$$

We model the nominal quadrotor dynamics using the following ordinary differential equations (ODEs) [13]

$$\begin{aligned} \dot{\boldsymbol{p}}^W &= \boldsymbol{v}^W \\ \dot{\boldsymbol{q}}^W &= \boldsymbol{q}^W \cdot \begin{bmatrix} 0 \\ \boldsymbol{\omega}_B/2 \end{bmatrix} \\ \dot{\boldsymbol{v}}^W &= \boldsymbol{q}^W \odot \left(\frac{1}{m}\boldsymbol{T}_B\right) + \boldsymbol{g}^W \\ \dot{\boldsymbol{\omega}}^W &= \boldsymbol{J}^{-1}(\boldsymbol{\tau}_B - \boldsymbol{\omega}_B \times \boldsymbol{J}\boldsymbol{\omega}_B). \end{aligned} \tag{6.2}$$

The parameters of the model are the mass m, moment of inertia $\boldsymbol{J} \in \mathrm{R}^{3\times3}$, the rotor displacements d_x, d_y, the rotor drag constant c_τ and the gravitational acceleration vector $\boldsymbol{g}_W = [0, 0, -9.81]^T ms^{-2}$.

Together, these ODEs describe the *physical* model of the quadrotor dynamics using a 13-dimensional state vector $\boldsymbol{x}$ using the first principles model as

First Principles Quadrotor Model

$$\dot{\boldsymbol{x}} = \boldsymbol{f}_{\text{PHYS}} = \begin{bmatrix} \boldsymbol{v}_{WB} \\ \boldsymbol{q}_{WB} \cdot \begin{bmatrix} 0 \\ \boldsymbol{\omega}_B/2 \end{bmatrix} \\ \boldsymbol{q}_{WB} \odot \left(\frac{1}{m}\boldsymbol{T}_B\right) + \boldsymbol{g}_W \\ \boldsymbol{J}^{-1}(\boldsymbol{\tau}_B - \boldsymbol{\omega}_B \times \boldsymbol{J}\boldsymbol{\omega}_B) \end{bmatrix}. \tag{6.3}$$

The ODE $\boldsymbol{f}_{\text{PHYS}}$ describes the relationship between a quadcopter state $\boldsymbol{x}$ and its time derivative $\dot{\boldsymbol{x}}$. To solve the ODE numerically, we need to discretize the ODE from continuous time t into discrete timesteps $t_{k+1} = t_k + \Delta t$. To do this, we use the Runge-Kutta 4th order method [28] given by

$$\begin{aligned} k_1 &= \boldsymbol{f}_{\text{PHYS}}(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ k_2 &= \boldsymbol{f}_{\text{PHYS}}(\boldsymbol{x}_k + \frac{1}{2}\Delta t \cdot k_1, \boldsymbol{u}_k) \\ k_3 &= \boldsymbol{f}_{\text{PHYS}}(\boldsymbol{x}_k + \frac{1}{2}\Delta t \cdot k_2, \boldsymbol{u}_k) \\ k_4 &= \boldsymbol{f}_{\text{PHYS}}(\boldsymbol{x}_k + \Delta t \cdot k_3, \boldsymbol{u}_k) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right). \end{aligned} \tag{6.4}$$

6.2 Data-driven Drag Model

The first principles model in (6.3) is a good approximation of the quadrotor dynamics. However, it does not capture the effect of aerodynamic drag. In this section, we propose a data-driven model to model the effects of aerodynamic drag on the quadrotor using RGP regression.

6.2.1 General Drag Model

Drag is a force acting on a body counter to its motion in a fluid. One source of drag is solid's body drag, which is proportional to the velocity of the body and the fluid density. The body drag force acting on the quadrotor body is described as [8]

$$\boldsymbol{F}^B_{bd} = -\frac{1}{2}\rho|\boldsymbol{v}^B|^2 C_D A \cdot \boldsymbol{v}^B, \tag{6.5}$$

where ρ is the fluid density, C_D is the drag coefficient and A is the cross-sectional area of the body. The drag force is applied to the body in the opposite direction of the velocity vector $\boldsymbol{v}^B$.

Another source of drag affecting the quadrotor is the rotor drag affecting each rotor which can be modeled using

$$\boldsymbol{F}_{rd}^B = -\omega C_{rD} \cdot \left(\boldsymbol{v}^B\right)^{\perp}, \tag{6.6}$$

where ω is the angular velocity of the rotor, C_{rD} is the drag coefficient of the rotor and $\left(\boldsymbol{v}^B\right)^{\perp}$ is the orthogonal projection of the quadrotor $\{B\}$ velocity onto the xy-plane.

Since we do not have direct measurement of the drag force, we will use the measured acceleration as a surrogate and work in the acceleration domain for the rest of the paper.

6.2.2 Drag Estimation

To model the total drag force given by (6.5), (6.6), we estimate the drag acceleration $\tilde{\boldsymbol{a}}^B$ acting on the quadrotor in the body frame $\{B\}$ at time t_k as

$$\tilde{\boldsymbol{a}}_k^B = \frac{\boldsymbol{v}_{k+1}^B - \hat{\boldsymbol{v}}_{k+1}^B}{\Delta t_k}, \tag{6.7}$$

where $\hat{\boldsymbol{v}}_{k+1}^B$ is the velocity predicted using the nominal model $\boldsymbol{f}_{\text{PHYS}}$, $\boldsymbol{v}_k^B$ is the measured velocity, $\Delta t_k = t_k - t_{k-1}$, and using the notational shorthand $\boldsymbol{v}_k^B = \boldsymbol{v}^B(t_k)$. The estimated drag acceleration $\tilde{\boldsymbol{a}}_k^B$, together with the current velocity $\boldsymbol{v}_k^B$ then form an observation pair

$$\boldsymbol{o}_k = \left\{\boldsymbol{v}_k^B, \tilde{\boldsymbol{a}}_k^B\right\}, \tag{6.8}$$

that is used in the regression procedure described below. This procedure is in effect a realization of the discrepancy modeling approach described in section 2.3.

6.2.3 Drag Modeling by Recursive Gaussian Process

The GP [29] is a non-parametric method that generalizes Gaussian probability *distributions*. Instead of describing the probability over a scalar (or vector) space, stochastic *processes* (of which GPs are a part of) describe probability over a function space. GPs do not draw a function as-is, instead we can only evaluate the function at a finite number of points. Every finite collection of these function values has a multivariate normal distribution. The GP is the joint distribution of all of those function values, and is thus a distribution over functions with continuous domain.

The standard "offline" GP require access to a training dataset of n_{obs} observations $\boldsymbol{o}_k = \left\{\boldsymbol{v}_k^B, \tilde{\boldsymbol{a}}_k^B\right\}$ forming the dataset $\mathcal{D} = \{\boldsymbol{o}_k \mid k = 0, \dots, n_{\text{obs}}\}$ gathered from the quadrotor during flight. To reduce the complexity of the GP we split the observations into its $d \in \{x, y, z\}$ components, use a separate GP for each component and then aggregate their outputs into an ensemble. The observations $\boldsymbol{o}_k$ forming the dataset $\mathcal{D}$ is generated from the noisy process

$$\tilde{a}_{d,k}^B = g_d(v_{d,k}^B) + \epsilon_{d,k}, \tag{6.9}$$

where g_d is the true, unmodeled drag acceleration for dimension d and $\epsilon_{d,k} \sim \mathcal{N}(0, \sigma_d^2)$ is the noise in the measurement. The GP is used to infer the latent function g_d from $\mathcal{D}$.

The recursive Gaussian process regression [18] is a method for *online* learning of Gaussian processes. Standard GP regression requires the entire training dataset $\mathcal{D}$ to be available before the model can be learned using hyperparameter optimization. In contrast to this, RGP allows for updating the GP model as new observations $\boldsymbol{o}_k$ become available.

The RGP for each dimension $d \in \{x, y, z\}$ is initialized with a set of basis vectors located at $\mathbf{V}_d^+ \triangleq [v_{d,1}^+, v_{d,2}^+, \dots, v_{d,m}^+]$ with the random variable $\tilde{\mathbf{A}}_d^+ \triangleq \left[\tilde{a}_{d,1}^+, \tilde{a}_{d,2}^+, \dots, \tilde{a}_{d,m}^+\right]$, corresponding to output of g_d given a velocity $v_{d,k}^+$. These basis vectors can be thought of as an active set allowing a sparse GP representation. This alleviates the computational requirements immensely, since $m \ll n_{\text{obs}}$. The initial distribution of the RGP is given by

$$p_0(\tilde{\mathbf{A}}_d^+) = \mathcal{N}(\tilde{\mathbf{A}}_d^+; \boldsymbol{\mu}_d^{+,0}, \boldsymbol{C}_d^{+,0}), \tag{6.10}$$

where $\boldsymbol{\mu}_d^{+,0} \triangleq \mathbf{0}$ is the mean at $\mathbf{V}_d^+$ and $\boldsymbol{C}_d^{+,0} \triangleq K(\mathbf{V}_d^+, \mathbf{V}_d^+)$[3] is the covariance at $\mathbf{V}_d^+$.

The goal is then to use the new observations $\boldsymbol{o}_k$ as they become available to calculate the posterior distribution $p(\tilde{\mathbf{A}}_d^+ | \boldsymbol{o}_{1:k})$, where $\boldsymbol{o}_{1:k} \triangleq \{\boldsymbol{o}_1, \boldsymbol{o}_2, \dots, \boldsymbol{o}_k\}$, from the prior distribution

$$p(\tilde{\mathbf{A}}_d^+ | \boldsymbol{o}_{1:k-1}) \sim \mathcal{N}(\boldsymbol{\mu}_d^{+,k-1}, \boldsymbol{C}_d^{+,k-1}). \tag{6.11}$$

At any point in time, we can run inference to find the distribution of $\tilde{a}_d^\bullet = g_d(v_d^\bullet)$, i.e., of the function g_d at an arbitrary point $v_d^\bullet$, in our case corresponding to the drag acceleration given an instantaneous velocity, by evaluating the joint prior $p(\tilde{\mathbf{A}}_d^+, \tilde{a}_d^\bullet | \boldsymbol{o}_{1:k})$

[3] $K(x, x') = \sigma_f^2 \exp\left(-\frac{1}{2}\frac{(x-x')^2}{l}\right) + \sigma_n^2$ is the squared exponential kernel. When applied to vector arguments, the kernel K is applied element-wise to each pair of elements in the arguments, yielding a matrix. The hyperparameters $\eta = (l, \sigma_f, \sigma_n)$ are the only hyperparameters of the GP/RGP regression.

and marginalizing out the basis vectors $\tilde{\mathbf{A}}_d^+$ such as

$$\begin{aligned} p(\tilde{\mathbf{A}}_d^+, \tilde{a}_d^\bullet | \boldsymbol{o}_{1:k}) &= p(\tilde{a}_d^\bullet | \tilde{\mathbf{A}}_d^+) \cdot p(\tilde{\mathbf{A}}_d^+ | \boldsymbol{o}_{1:k}) \\ &= \mathcal{N}(\tilde{a}_d^\bullet; \mu_d^{\bullet,k}, \boldsymbol{C}_d^{\bullet,k}) \cdot \mathcal{N}(\tilde{\mathbf{A}}_d^+; \mu_d^{+,k}, \boldsymbol{C}_d^{+,k}), \end{aligned} \tag{6.12}$$

where

$$\mu_d^{\bullet,k} \triangleq \boldsymbol{H} \cdot \boldsymbol{\mu}_d^{+,k}, \tag{6.13}$$

$$\boldsymbol{C}_d^{\bullet,k} \triangleq K(v_d^\bullet, v_d^\bullet) - \boldsymbol{H} \cdot K(v_d^\bullet, \mathbf{V}_d^+), \tag{6.14}$$

$$\boldsymbol{H} \triangleq K(v_d^\bullet, \mathbf{V}_d^+) \cdot K(\mathbf{V}_d^+, \mathbf{V}_d^+)^{-1}. \tag{6.15}$$

6.2.4 Model Learning, Utilization, and Properties

We use the RGP model to compensate the discrepancy between our nominal model prediction and the measured state of the quadrotor characterized by the drag-related acceleration (6.7).

For each RGP in the ensemble, the basis vectors $\mathbf{V}_d^+$ are initialized by equidistantly sampling the velocity space on $[-v_{max}, v_{max}]$ using m points, setting their corresponding estimates of g_d to 0 and their covariance to $\boldsymbol{C}_d^{+,0} = K(\mathbf{V}_d^+, \mathbf{V}_d^+)$.

At each time step k, we use the current quadrotor state $\boldsymbol{x}_k$ with the RGP model $\boldsymbol{f}_{\text{DATA}}$ to infer the acceleration $\tilde{\boldsymbol{a}}$ at the current velocity $\boldsymbol{v}_k$ using the mean of the marginalized joint distribution in (6.12) as described in (6.16).

Data-Driven Drag Model

$$\boldsymbol{f}_{\text{DATA}}(\boldsymbol{x}_k; \boldsymbol{\mu}^{\bullet,k}) = \begin{bmatrix} \mathbf{0}^{7\times1} \\ \tilde{a}_x \\ \tilde{a}_y \\ \tilde{a}_z \\ \mathbf{0}^{3\times1} \end{bmatrix} = \begin{bmatrix} \mathbf{0}^{7\times1} \\ \mu_x^{\bullet,k}\left(v_{x,k};\ \boldsymbol{\mu}_d^{+,k}\right) \\ \mu_y^{\bullet,k}\left(v_{y,k};\ \boldsymbol{\mu}_y^{+,k}\right) \\ \mu_z^{\bullet,k}\left(v_{z,k};\ \boldsymbol{\mu}_z^{+,k}\right) \\ \mathbf{0}^{3\times1} \end{bmatrix}, \tag{6.16}$$

where $\boldsymbol{\mu}^{\bullet,k} \triangleq \left(\mu_x^{\bullet,k}, \mu_y^{\bullet,k}, \mu_z^{\bullet,k}\right)$. To perform the online update of the RGP distribution $p(\tilde{\mathbf{A}}_d^+ | \boldsymbol{o}_{1:k})$ we keep the covariance matrices $\boldsymbol{C}_d^{+,k-1}$ and the mean $\boldsymbol{\mu}_d^{+,k-1}$ of the prior distribution $p(\tilde{\mathbf{A}}_d^+ | \boldsymbol{o}_{1:k-1})$. However, to run inference using the RGP, only the means $\boldsymbol{\mu}_d^{+,k}$ are needed, while $\boldsymbol{C}_d^{+,k-1}$ is kept in memory only to perform updates.

6.3 Hybridized Quadrotor Model

To complete the hybrid model of the quadrotor dynamics, the physics-based model $\boldsymbol{f}_{\text{PHYS}}$ (6.3) and the online trained RGP model $\boldsymbol{f}_{\text{DATA}}$(6.16) are combined into the single *hybrid* model in the form of

Hybrid Quadrotor Model

$$\boldsymbol{f}_{\text{DAPB}}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) = \boldsymbol{f}_{\text{PHYS}}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) + \boldsymbol{f}_{\text{DATA}}\left(\boldsymbol{x}_k; \boldsymbol{\mu}^{+,k}\right). \tag{6.17}$$

The hybrid model (6.17) provides more accurate description of the underlying system, incorporating both the first principles understanding of motion and the data-driven model of the drag. The hybrid model is used in the MPC controller to generate the control inputs $\boldsymbol{u}_k$ for the quadrotor.

6.3.1 Model Predictive Control for the Hybridized Model

The model predictive control discussed in section 3.2 is a control scheme used to stabilize a system subject to dynamic equations $\dot{\boldsymbol{x}} = \boldsymbol{f}_{\text{DAPB}}(\boldsymbol{x}, \boldsymbol{u})$ given a reference trajectory $\boldsymbol{x}_{\text{ref}}(t), \boldsymbol{u}_{\text{ref}}(t)$ by minimizing a cost function $\mathcal{L}(\boldsymbol{x}, \boldsymbol{x}_{\text{ref}}, \boldsymbol{u}, \boldsymbol{u}_{\text{ref}})$.

We use the MPC to control the movement of a quadrotor in the 3D space. We use the hybrid model (6.17) as the internal MPC dynamics.

We define the OCP as

$$J_N(\boldsymbol{x}_0) \triangleq \min_{\boldsymbol{u}_{0:N-1}} \sum_{k=0}^{N-1} \mathcal{L}(\boldsymbol{x}, \boldsymbol{x}_{\text{ref}}, \boldsymbol{u}, \boldsymbol{u}_{\text{ref}}), \tag{6.18}$$

$$\text{with respect to } h_i(\boldsymbol{x}) = 0 \quad i = 1, \ldots n_h \tag{6.19}$$

$$\text{and } g_j(\boldsymbol{x}) \leq 0 \quad j = 1, \ldots n_g. \tag{6.20}$$

$$\text{subject to } \boldsymbol{x}_{k+1} = \boldsymbol{f}_{\text{DAPB}}(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{6.21}$$

where $\boldsymbol{f}_{\text{DAPB}}$ denotes the hybrid model of the system, $\boldsymbol{x}_{\text{ref}}$ is the tracked trajectory, $\boldsymbol{x}_0$ is the initial state, h_i describes the equality constraints and g_j describes the inequality constraints. The inclusion of h_i and g_j is one of the main advantages of MPC, allowing for solutions satisfying specific constraints. Importantly, we use the inequality constraints g_j to limit the input $\boldsymbol{u}$ to the range $< 0, 1 >$, which represents the normalized thrust of the quadrotor.

Because of the changing nature of the $\boldsymbol{f}_{\text{DATA}}$ model due to online learning, it would be computationally inefficient to reconstruct the OCP at each control time step. Therefore, we use the following simplification [32]: we treat the vector $\boldsymbol{\mu}^{+,k}$ as a parameter of the $\boldsymbol{f}_{\text{DATA}}$ model. In this way, we change the parameters of the nonlinear optimization problem at each control time step k as the RGP ensemble is updated.

The process of computing the control using MPC and updating its RGP model is shown in Algorithm 5. We refer to this approach as RGP−MPC and as GP−MPC when a pre-trained GP is used.

Algorithm 5 Recursive Learning and Control Algorithm Summary

```
 1: initialize $\boldsymbol{f}_{\text{DATA}}$ with $\mathbf{V}_d^+$ and $\tilde{\mathbf{A}}_d^+$
 2: initialize MPC with $\boldsymbol{f}_{\text{DAPB}}$
 3: $\hat{\boldsymbol{x}}_0 \leftarrow$ initial state
 4: $k \leftarrow 0$
 5: while trajectory ready do
 6:     procedure RGP−MPC STEP
 7:         $\boldsymbol{x}_k \leftarrow$ state measurement
 8:         $\boldsymbol{u}_{k,0:0+n_h} \leftarrow$ MPC optimal control
 9:         send control command $\boldsymbol{u}_{k,0}$
10:         $\hat{\boldsymbol{x}}_{k+1} \leftarrow \boldsymbol{f}_{\text{PHYS}}\left(\boldsymbol{x}_k, \boldsymbol{u}_{k,0}\right)$
11:         $\tilde{\boldsymbol{a}}_k^B \leftarrow \frac{\boldsymbol{v}_k^B - \hat{\boldsymbol{v}}_k^B}{\Delta t_k}$
12:         $\boldsymbol{o}_k \leftarrow \left\{\boldsymbol{v}_k^B, \tilde{\boldsymbol{a}}_k^B\right\}$
13:         $p(\tilde{\mathbf{A}}_d^+|\boldsymbol{o}_{1:k}) \leftarrow$ update with $\boldsymbol{o}_k$ from prior
14:         $\boldsymbol{\mu}^{+,k} \leftarrow$ marginalize $p(\tilde{\mathbf{A}}_d^+, \tilde{\boldsymbol{a}}_k^B|\boldsymbol{o}_{1:k})$
15:         update $\boldsymbol{f}_{\text{DAPB}}$ with $\boldsymbol{\mu}^{+,k}$ inside MPC
16:         $k \leftarrow k+1$
17:     end procedure
18: end while
```

The RGP−MPC approach is summarized in the schematic in fig. 6.2. There, the MPC controller is initialized with the hybrid model $\boldsymbol{f}_{\text{DAPB}}$, which is used to compute the optimal control $\boldsymbol{u}_{k,0:0+n_h}$ at each control time step k. The control is then applied to the system, and the state is measured. The measured state is used to update the RGP model, which is then used to update the hybrid model $\boldsymbol{f}_{\text{DAPB}}$ inside the MPC controller. The process is repeated at each control time step.

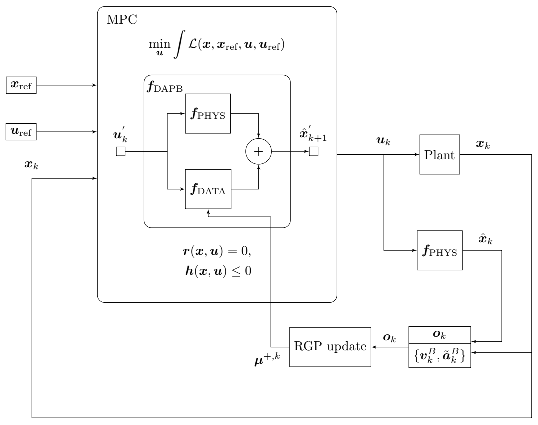

MPC
$\min_{\boldsymbol{u}} \int \mathcal{L}(\boldsymbol{x}, \boldsymbol{x}_{\text{ref}}, \boldsymbol{u}, \boldsymbol{u}_{\text{ref}})$
$\boldsymbol{x}_{\text{ref}}$
$\boldsymbol{u}_{\text{ref}}$
$\boldsymbol{x}_k$
$\boldsymbol{f}_{\text{DAPB}}$
$\boldsymbol{u}_k'$
$\boldsymbol{f}_{\text{PHYS}}$
$+$
$\boldsymbol{f}_{\text{DATA}}$
$\hat{\boldsymbol{x}}_{k+1}'$
$\boldsymbol{r}(\boldsymbol{x}, \boldsymbol{u}) = 0,$
$\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}) \leq 0$
$\boldsymbol{u}_k$
Plant
$\boldsymbol{x}_k$
$\boldsymbol{f}_{\text{PHYS}}$
$\hat{\boldsymbol{x}}_k$
$\boldsymbol{\mu}^{+,k}$
RGP update
$\boldsymbol{o}_k$
$\boldsymbol{o}_k$
$\{\boldsymbol{v}_k^B, \tilde{\boldsymbol{a}}_k^B\}$

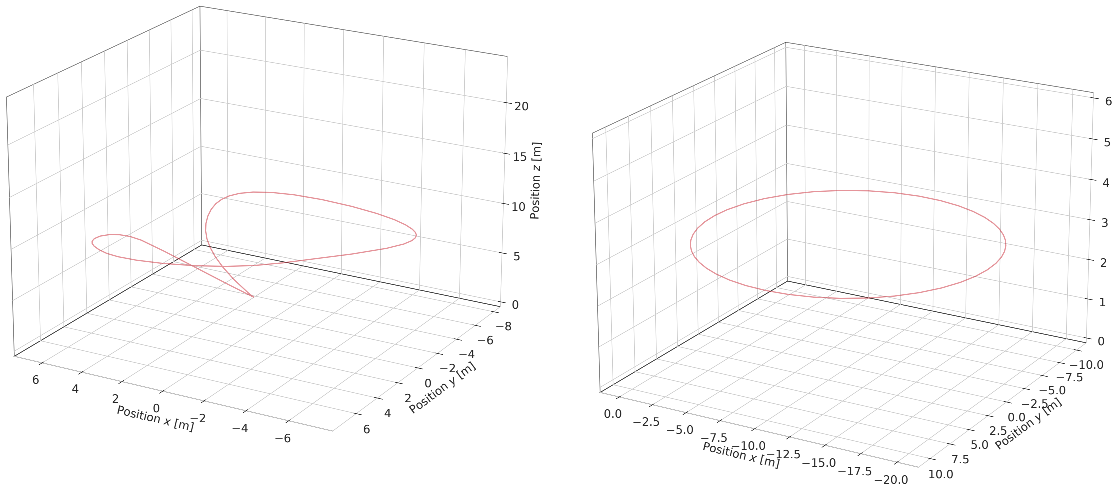
Figure 6.2: Schematic of the RGP−MPC algorithm.

6.4 Simulation

The proposed data-augmented modelling and and predictive control of quadrotor is numerically evaluated on two trajectories using a realistic quadrotor model provided by the Gazebo simulator explained in section 5.2. Details on trajectory generation, MPC implementation, and the results follows. A simplified simulation environment is also provided as described in 6.4.3.

6.4.1 Trajectory Generation

As a tracking reference we generate trajectories illustrated in Fig. 6.3 in $\{W\}$ frame using a trajectory generator[4] [30]. We start by randomly generating waypoints in a 3-dimensional cube of size hsize = 10m raised above the xy ground plane by hsize. Then we provide these waypoints to the `genTrajectory` bash script together with the maximum allowed velocity v_{max} and acceleration a_{max}. The script then generates a trajectory that satisfies the velocity and acceleration constraints as a polynomial which we then sample with the desired control frequency $f_s = \frac{1}{\Delta t}$.



(a) Trajectory created by interpolating between randomly generated waypoints with a polynomial

(b) Circular trajectory with radius $r = 10m$ with velocity increasing linearly from 0 to v_{max}

Figure 6.3: Reference trajectories used for the experiments.

6.4.2 Nonlinear Model Predictive Control Implementation

The RGP−MPC algorithm is implemented using the `Python` interface to `acados` [35], a solver for nonlinear optimal control problems for embedded applications. The system model $\boldsymbol{f}_{\text{DAPB}}$ is implemented as a symbolic `CasADi` [3] function which is used to construct the OCP for the MPC, for which `acados` generates a solver.

[4]https://github.com/whoenig/uav_trajectories

The MPC is parametrized with a desired horizon look–forward time t_h and the number of prediction steps n_h. The length of one prediction step is thus given by $T_h = \frac{t_h}{n_h}$.

To solve the MPC problem at each time step, we use the *Sequential quadratic programming real-time iteration* (`SQP−RTI`) solver type described in section 3.3.2. This solver attempts to solve the optimization problem iteratively, but instead of iterating until a desired optimality condition is satisfied, it performs only a single iteration and then returns the current solution. This non-optimal solution is then used instead of the optimal quadrotor control. At the next time step, the system state is updated and the solver begins to solve the optimization problem again, but this time using the previous solution as an initial guess. The `SQP−RTI` solver scheme allows us to control the quadrotor in real time, but it is not guaranteed to find the optimal solution, especially during the first few control loops.

6.4.3 Simplified `Python` Simulation

In order to evaluate the performance of the RGP−MPC method, we first perform a series of experiments in a simplified simulation, where all the underlying models are perfectly known and the physics-based part coincides with the model used by the MPC. We created a simple simulation environment where we simulate the dynamics of the quadrotor under the influence of a drag force. The quadrotor is simulated using the nominal model (6.3) with the addition of a drag force

$$\dot{\boldsymbol{x}} = \boldsymbol{f}_{\text{PHYS}} + \boldsymbol{f}_{\text{drag}}\left(\boldsymbol{x}\right), \tag{6.22}$$

where the drag force model (6.23) reads

$$\boldsymbol{f}_{\text{drag}}\left(\boldsymbol{x}\right) = C_D \cdot \begin{bmatrix} \mathbf{0}^{7x1} \\ {v_x^B}^2 \cdot \operatorname{sgn}\left(v_x^B\right) \\ {v_y^B}^2 \cdot \operatorname{sgn}\left(v_y^B\right) \\ 5 \cdot {v_z^B}^2 \cdot \operatorname{sgn}\left(v_z^B\right) \\ \mathbf{0}^{3x1} \end{bmatrix}. \tag{6.23}$$

The drag force is applied in the body frame, and the drag coefficient $C_D = 0.01 \text{kg} \cdot \text{m}^{-1}$ is assumed to be constant. We assume the drag in the z body axis more significant as the quadrotor body surface is larger in the xy-plane projection than the side views.

The simulation environment is implemented in `Python`[5] using the fourth-order Runge-Kutta (RK4) integration scheme with the simulation (integration) step $\delta t = 0.001s$. The quadrotor is controlled using the Alg. 5 along a trajectory Fig. 6.3.

[5]Source code: https://github.com/smidmatej/mpc_quad_ros

6.4.4 Gazebo Simulation

Using the `RotorS` [13] package for the Gazebo simulator [1] we simulate the quadrotor in a high fidelity environment. `RotorS` provides a AscTec Hummingbird quadrotor model together with an interface using which we supply the control inputs $\boldsymbol{u}_k$.

Gazebo provides the quadrotor odometry to our MPC controller node with a frequency of 100Hz. The controller node then extracts the state $\boldsymbol{x}_k$ from the odometry and runs the procedure `RGP−MPC STEP` in algorithm 5 to obtain the control action $\boldsymbol{u}_k$. The control action is then sent to the quadrotor model in Gazebo, which then updates the quadrotor state. The procedure is repeated until the trajectory is completed.

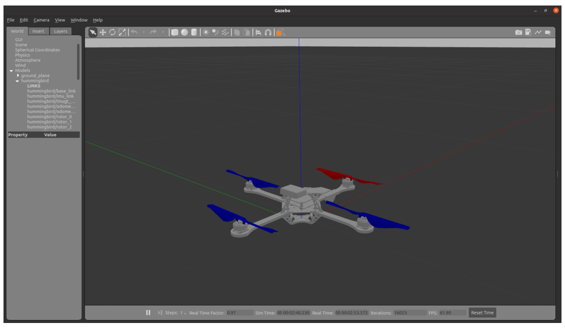
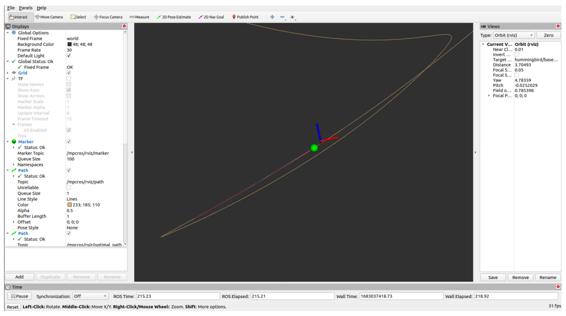


Figure 6.4: Screenshot of the Gazebo simulator showing the AscTech Hummingbird quadrotor model at stand-still.

The Gazebo simulation environment is shown in Fig. 6.4 and the `rviz` visualization tool together with a reference trajectory is shown in Fig. 6.5.

6.4.5 Experimental Setup

The experiments were performed on the circle trajectory shown in Fig. 6.3b and on a randomly pre-generated trajectories shown in Fig. 6.3a. We set the parameters of the MPC control algorithm as follows, horizon length $t_h = 1\text{s}$, $n_h = 5$ and control interval of $\Delta t = 0.01\text{s}$. Both the GP and the RGP models used $m = 20$ inducing points.



```
Global Options
  Fixed Frame          world
  Background Color     48; 48; 48
  Frame Rate           30
  Default Light        ✓
Global Status: Ok
  Fixed Frame          OK
Grid                   ✓
TF
  Show Names
  Show Axes            ✓
  Show Arrows
  Marker Scale         1
  Marker Alpha         1
  Update Interval      0
  Frame Timeout        15
  Frames
    All Enabled        ✓
  Tree
Marker                 ✓
  Status: Ok
  Marker Topic         /mpcros/rviz/marker
  Queue Size           100
  Namespaces
Path                   ✓
  Status: Ok
  Topic                /mpcros/rviz/path
  Unreliable
  Queue Size           1
  Line Style           Lines
  Color                233; 185; 110
  Alpha                0.5
  Buffer Length        1
  Offset               0; 0; 0
  Pose Style           None
Path                   ✓
  Status: Ok
  Topic                /mpcros/rviz/optimal_path

Type: Orbit (rviz)   Zero
Current V...   Orbit (rviz)
  Near Cl...   0.01
  Invert ...
  Target ...   hummingbird/base...
  Distance     3.70493
  Focal S...   0.05
  Focal S...
  Yaw          4.78359
  Pitch        -0.0252029
  Field o...   0.785398
  Focal P...   0; 0; 0

Synchronization: Off   ROS Time: 215.23   ROS Elapsed: 215.21   Wall Time: 1683037418.73   Wall Elapsed: 218.92
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.   31 fps
```

Figure 6.5: Screnshot of the `rviz` visualization tool showing the quadrotor model and the reference trajectory during flight.

The GP in the GP−MPC approach was pre-trained by selecting the m most informative inducing points from a dataset generated in a training run on the *random* trajectory using Gaussian mixture modeling (GMM). The hyperparameters of the GP η were then optimized using the maximum likelihood estimation method [23]. Since the GP−MPC approach is trained on the *random* trajectory it is expected that it will achieve greater reduction of the error for the *random* trajectory than RGP−MPC.

The RGP in the RGP−MPC approach was not pre-trained before performing the trajectory, and was trained online only. Its m basis vectors were uniformly chosen on $[-v_{\max}, v_{\max}]$ with corresponding initial drag acceleration estimates being 0 as shown in Fig. 6.7. Since the RGP does not perform hyperparameter estimation during learning, its hyperparameters were set at the start of the trajectory as $\eta = (1.0, 0.1, 0.1)$. The need to set hyperparameters beforehand limits the RGP–MPC approach, as discussed further.

6.4.6 Results

Having description of the simulation environment, the simulation results of the experiments performed in Gazebo are discussed further.

Trajectory	$v_{\max}$ [ms^{-1}]	**RMSE pos [mm]** Nominal	GP–MPC	RGP–MPC
Random	3	75.9	30.9 **(41%)**	40.6 (53%)
	6	110.1	52.9 **(48%)**	65.1 (59%)
	9	128.5	70.3 **(55%)**	88.2 (69%)
	12	142.9	81.9 **(57%)**	99.6 (69%)
Circle	3	57.5	22.5 **(39%)**	23.6 (41%)
	6	102.7	50.5 (49%)	43.5 **(42%)**
	9	145.1	88.1 (61%)	69.7 **(47%)**
	12	183.9	128.4 (70%)	98.2 **(53%)**
	avg. opt. dt [ms]	0.60	0.66	1.21

Table 6.1: RMSE position error for the circle trajectory generated using Gazebo with both $\boldsymbol{f}_{\text{PHYS}}$ and $\boldsymbol{f}_{\text{DAPB}}$. Both the pretrained GP and the RGP augmented controllers have consistently lower RMSE position error at the cost of increased optimization time.

The results shown in Table 6.1 indicate that both the GP–MPC and RGP–MPC approaches *significantly* reduce the RMSE position tracking error in comparison to the *nominal*, unaugmented physics-based $\boldsymbol{f}_{\text{PHYS}}$ MPC. Performance of MPCs with the augmented models is rather comparable, but the proposed RGP–MPC do *not* rely on any pre-training. On the other hand, both GP–MPC and RGP–MPC increase the optimization time needed to calculate the next MPC output[6]. However, this in-

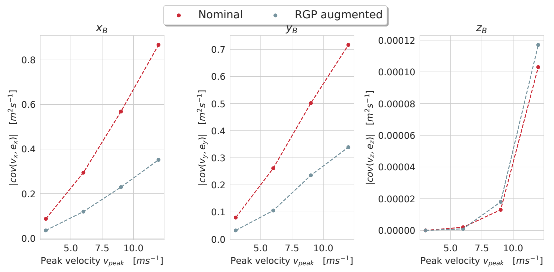


Figure 6.6: Absolute values of covariance between the peak tracking velocity v_{peak} and position tracking error $e_{x,y,z}$ for the circle trajectory captured using the Gazebo simulation.

[6]Simulations performed on Ubuntu 20.04 with an Intel Core i5-8300H CPU and 8GB of RAM.

crease is more significant for RGP−MPC, since it performs additional computations to update the RGP and then to re-parameterize the OCP.

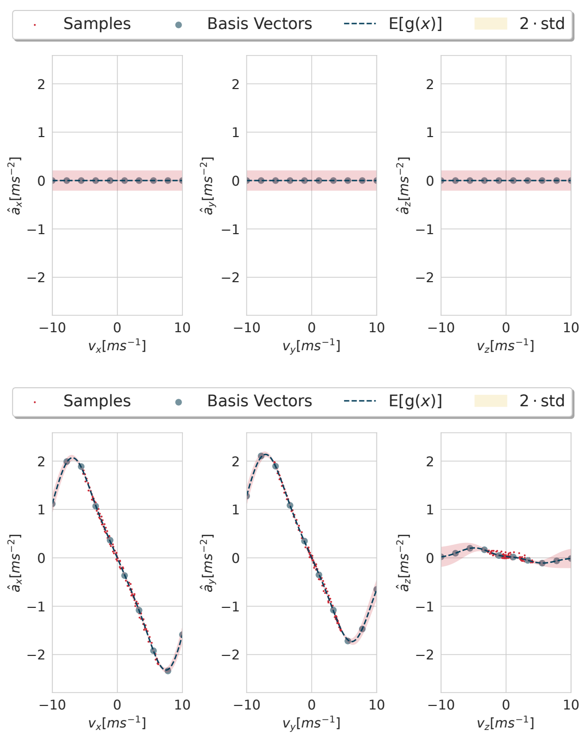


Figure 6.7: RGP inference space at the start (top) and at the end (bottom) of the trajectory using the Gazebo simulator. Since the Gazebo simulator only models rotor drag (6.6), the RGP predicts much lower drag in the z_B direction.

Fig. 6.6 shows the absolute value of covariance between the velocity v_d and the position tracking error e_d for each $d \in \{x, y, z\}$ based on the peak velocity during the simulated trajectory v_{peak}. It can be seen, that the RGP−MPC approach reduces this value w.r.t the *nominal* approach due to the fact that using both $\boldsymbol{f}_{\text{PHYS}}$ and $\boldsymbol{f}_{\text{DATA}}$ leads to improved forecasting than using $\boldsymbol{f}_{\text{PHYS}}$ only.

Fig. 6.7 visualises the RGP inference distribution at the start (upper plot) and the end (bottom plot) of a simulation. The training of the RGP is done in real time is able to fit a distribution to the observations $\boldsymbol{o}_k$ online at the same time as being used to make real-time predictions.

6.4.7 Observations, Notes, and Future Work

Since we are using estimated disturbance acceleration $\tilde{\boldsymbol{a}}_k^B$ calculated in the $\{B\}$ frame of the quadrotor to fit $\boldsymbol{f}_{\text{DATA}}$, we only have a limited ability to distinguish disturbances that are not time-invariant with respect to $\{B\}$. For example, even for a constant wind in $\{W\}$, the disturbance acceleration $\tilde{\boldsymbol{a}}_k^B$ will be time-varying with respect to $\{B\}$ due to the rotation of the quadrotor. The presence of such seemingly time-varying disturbances can induce adaptation that causes increasing $\tilde{\boldsymbol{a}}_k^B$, which might potentially lead to control loop instability.

In the current implementation, we do not perform any hyperparameter optimization in the RGP−MPC approach. We found the hyperparameter estimation method [18] augmenting the RGP state by the hyperparameters η and estimating them using the Kalman filter to be too unstable to be used safely. Thus, the suitable hyperparameters was selected a priori. On-line adaptation can further increase the ability of the proposed RGP−MPC method.

We use the estimated disturbance acceleration (6.7) caused by the drag to fit the

RGP model without any pre-processing. This can lead to unstable control behaviour when the real quadrotor state differs from the predicted state due to faulty or rare-normal behaviour. A possible solution can be based on gathering data in a training run and then selects the data points valid for “nominal” conditions to fit the GP model GP−MPC [33].

Chapter 7

Conclusion

This thesis dealt with the problem of combining the two modeling approaches, first principle and data-driven, to create a hybrid model that balances the properties of both approaches. We have shown that combining the two approaches can lead to better performance than using either of the approaches alone, without significantly increasing the compute required for optimal control using MPC.

We first discussed the theoretical background behind the two modeling approaches, first principles and data-driven. Then we introduced the theory behind mathematical optimization and MPC. The only data-driven models we considered were (recursive) Gaussian Process regression models, which were then used in the Experiment chapter. We also provide some background on the different simulation environments used in the field of robotics, focusing on Gazebo+ROS.

In chapter 6 we develop a method RGP−MPC, which we tested on a quadrotor model described by the data-augmented first principles model. The first principles model is a standard model for quadrotor rigid body dynamics. The data-driven part of the model is realized by a Recursive Gaussian Process regression model that gives an estimate of the air drag force learned online without the need to gather data in a training run. These two models are combined by augmenting the first principles model with the drag force estimate from the RGP model, creating a hybrid model that balances the strengths of each. The *hybrid* model is used by MPC to predict the future position of the quadrotor, while taking into account the drag forces present.

To be able to use the RGP model inside the MPC framework, RGP−MPC includes a parametrization of the MPC plant model, whose parameters are updated online using the RGP model without having to reinitialize the OCP, which is one of the main contributions of the RGP−MPC method.

We show that the data-driven RGP model is able to learn the drag force online, without the need for a training run or any a priori information about the observed dynamics, allowing for predicting the future state of the drag forces present.

Our proposed method, RGP−MPC, is able to quickly adapt to never-before-seen velocities and achieves better position tracking performance than the MPC with the non-augmented (purely physics-based) model.

We test our method in simulation, using the Gazebo simulator, where we show that the RGP−MPC method is able to track the desired trajectory and adapt to the changing drag forces present. This simulation experiment is a proof of concept that the RGP−MPC method is able to improve the performance of the MPC controller in the presence of unknown discrepancies in the model. The source code for the simulation experiment is available at https://github.com/smidmatej/mpc_quad_ros.

Progress in the field of adaptive model approximation allows designing robotic platforms that are able to adapt to changing conditions. This is a significant step towards the transition from the current state of robotics, where robots are designed for a specific task and constrained within a safe and predictable environment, to a future where robots are able move in unstructured environments and use their adaptive capabilities to influence it in intelligent ways.

Bibliography

[1] Design and Use Paradigms for Gazebo, an Open-source Multi-robot Simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE.

[2] Dawn An, Nam H. Kim, and Joo-Ho Choi. Practical Options for Selecting Data-driven or Physics-based Prognostics Algorithms with Reviews. 133:223–236, 2015.

[3] Joel AE Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi: a Software Framework for Nonlinear Optimization and Optimal Control. *Mathematical Programming Computation*, 11:1–36, 2019.

[4] Guangxing Bai, Yunsheng Su, Maliha Maisha Rahman, and Zequn Wang. Prognostics of Lithium-Ion batteries using Knowledge-constrained Machine Learning and Kalman Filtering. 231:108944, 2023.

[5] Alberto Bemporad, Daniele Bernardini, Ruixing Long, and Julian Verdejo. Model Predictive Control of Turbocharged Gasoline Engines for Mass Production. In *WCX World Congress Experience*. SAE International, apr 2018.

[6] George E. P. Box. Science and Statistics. *Journal of the American Statistical Association*, 71(356):791–799, 1976.

[7] Andrea Carron, Elena Arcari, Martin Wermelinger, Lukas Hewing, Marco Hutter, and Melanie N Zeilinger. Data-driven Model Predictive Control for Trajectory Tracking with a Robotic Arm. *IEEE Robotics and Automation Letters*, 4(4):3758–3765, 2019.

[8] Glenn Research Center. Drag Equation, 2022.

[9] Malin Christersson. Heliocentrism and Geocentrism, 2019.

[10] Brian M. de Silva, David M. Higdon, Steven L. Brunton, and J. Nathan Kutz. Discovery of Physics From Data: Universal Laws and Discrepancies. *Frontiers in Artificial Intelligence*, 3, 2020.

[11] Pedro Domingos. The Role of Occam's Razor in Knowledge Discovery. *Data mining and knowledge discovery*, 3:409–425, 1999.

[12] Gianluca Frison and Moritz Diehl. HPIPM: a High-performance Quadratic Programming Framework for Model Predictive Control, 2020.

[13] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham, 2016.

[14] Ben Goodman. Sequential Quadratic Programming, 2016.

[15] Sébastien Gros, Mario Zanon, Rien Quirynen, Alberto Bemporad, and Moritz Diehl. From Linear to Nonlinear MPC: Bridging the Gap via the Real-time Iteration. *International Journal of Control*, 93(1):62–80, 2020.

[16] Lars Grüne and Jürgen Pannek. *Nonlinear Model Predictive Control.* Springer-Verlag, 2017.

[17] Lukas Hewing, Juraj Kabzan, and Melanie N Zeilinger. Cautious Model Predictive Control using Gaussian Process Regression. *IEEE Transactions on Control Systems Technology*, 28(6):2736–2743, 2019.

[18] Marco F Huber. Recursive Gaussian process: On-line Regression and learning. *Pattern Recognition Letters*, 45:85–91, 2014.

[19] Tales Imbiriba, Ahmet Demirkaya, Jindřich Duník, Ondřej Straka, Deniz Erdoğmuş, and Pau Closas. Hybrid Neural Network Augmented Physics-based Models for Nonlinear Filtering. In *2022 25th International Conference on Information Fusion (FUSION)*, pages 1–6, 2022.

[20] S.J. Julier and J.K. Uhlmann. Unscented Filtering and Nonlinear Estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[21] Juraj Kabzan, Lukas Hewing, Alexander Liniger, and Melanie N Zeilinger. Learning-based Model Predictive Control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4):3363–3370, 2019.

[22] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960.

[23] L. Ljung. *System Identification: Theory for the User.* Pearson Education, 1998.

[24] Octavio Loyola-Gonzalez. Black-box vs. White-box: Understanding their Advantages and Weaknesses from a Practical Point of View. *IEEE access*, 7:154096–154113, 2019.

[25] Lalo Magni, Davide Martino Raimondo, and Frank Allgöwer. Nonlinear Model Predictive Control. *Lecture Notes in Control and Information Sciences*, 384, 2009.

[26] Mohit Mehndiratta and Erdal Kayacan. Gaussian Process-based Learning Control of Aerial Robots for Precise Visualization of Geological Outcrops. In *2020 European Control Conference (ECC)*, pages 10–16, 2020.

[27] Jorge Nocedal and Stephen J. Wright. Numerical Optimization. 1999.

[28] William Press. Numerical Recipes in Fortran 77: the Art of Scientific Csomputing. *(No Title)*, 1992.

[29] C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2005.

[30] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments. In *Robotics Research*, pages 649–666. Springer, 2016.

[31] Frank Rosenblatt. The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain. *Psychological review*, 65 6:386–408, 1958.

[32] Matej Smid and Jindrich Dunik. Online Learning and Control for Data-Augmented Quadrotor Model, 2023. Available at: https://arxiv.org/abs/2304.00503.

[33] Guillem Torrente, Elia Kaufmann, Philipp Foehn, and Davide Scaramuzza. Data-Driven MPC for Quadrotors. *IEEE Robotics and Automation Letters*, 2021.

[34] Aditya Tulsyan, R. Bhushan Gopaluni, and Swanand R. Khare. Particle Filtering without Tears: A Primer for Beginners. 95:130–145, 2016. *Computers and Chemical Engineering.*

[35] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. acados—a Modular Open-source Framework for Fast Embedded Optimal Control. Mathematical Programming Computation*, 14(1):147–183, 2022.*

[36] Andreas Wächter and Lorenz T. Biegler. On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming. Math. Program.*, 106(1):25–57, mar 2006.*

[37] Niklas Wahlström and Emre Özkan. Extended target tracking using Gaussian Processes. IEEE Transactions on Signal Processing*, 63(16):4165–4178, 2015.*

[38] Yu Zhang, Peter Tiňo, Aleš Leonardis, and Ke Tang. A Survey on Neural Network Interpretability. IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):726–742, 2021.*

Symbols

$\boldsymbol{f}_{\mathrm{DATA}}$	Data-driven dynamics model using RGP
$\boldsymbol{f}_{\mathrm{PHYS}}$	First principles dynamics model of the quadrotor
$\boldsymbol{f}_{\mathrm{DAPB}}$	Hybrid model of $\boldsymbol{f}_{\mathrm{PHYS}}$ and $\boldsymbol{f}_{\mathrm{DATA}}$
$\boldsymbol{f}_{\mathrm{drag}}$	Drag equation dynamics
$\boldsymbol{f}_{\mathrm{HYB}}$	Generic hybrid model
$\boldsymbol{f}_{\mathrm{C}}$	Generic continuous time model
$\boldsymbol{f}_{\mathrm{D}}$	Generic discrete time model
ϕ	Function to be optimized using Newton's method
t	Newton step
g	Function generating data for GP
$\hat{g}$	Function regressed using GP
K	Kernel function
$\mathbf{X}^{+}$	RGP basis vectors
$\mathbf{Y}^{+}$	Initial estimates at $\mathbf{X}^{+}$
g^{+}	RGP distribution evaluated at $\mathbf{X}^{+}$
$\boldsymbol{\mu}_i^{+}$	Mean of the RGP at $\mathbf{X}^{+}$
$\boldsymbol{C}_i^{+}$	Covariance of the RGP at $\mathbf{X}^{+}$
$\boldsymbol{x}^{\circ}$	Arbitrary RGP input
$\boldsymbol{y}^{\circ}$	RGP output at $\boldsymbol{x}^{\circ}$
$\boldsymbol{\mu}_i^{\circ}$	Mean of the RGP at $\boldsymbol{x}^{\circ}$
$\boldsymbol{C}_i^{\circ}$	Covariance of the RGP at $\boldsymbol{x}^{\circ}$
$\boldsymbol{x}^{\bullet}$	Observation input for the RGP
$\boldsymbol{y}^{\bullet}$	Observation output for the RGP
$\boldsymbol{o}_i$	Observation vector $[\boldsymbol{x}_i^{\bullet}, \boldsymbol{y}_i^{\bullet}]$
$\boldsymbol{o}_{1:i}$	Observations $\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots \boldsymbol{o}_i$
$g^{\bullet}$	RGP distribution evaluated at $\boldsymbol{x}^{\bullet}$
$\boldsymbol{H}_i$	Inference gain of the RGP
$\boldsymbol{\mu}_i^{\mathrm{KF}}$	Mean of the RGP using the Kalman Filter step
$\boldsymbol{C}_i^{\mathrm{KF}}$	Covariance of the RGP using the Kalman Filter step
$\boldsymbol{G}_i$	Kalman gain of the RGP

η	RGP hyperparameters
$\mathcal{D}$	RGP dataset
$\mathcal{B}$	RGP basis vector-pairs
$\boldsymbol{h}$	Equality constraints of a optimization problem
$\boldsymbol{g}$	Equality constraints of a optimization problem
$\boldsymbol{\lambda}$	Lagrange multiplier for $\boldsymbol{h}$
$\boldsymbol{\mu}$	Lagrange multiplier for $\boldsymbol{\mu}$
L	Lagrangian of a optimization problem
$\{B\}$	Quadrotor body referential frame
$\{W\}$	Quadrotor world referential frame
$\boldsymbol{p}$	Quadrotor xyz position $[m]$
$\boldsymbol{v}$	Quadrotor xyz velocity $[m \cdot s^{-1}]$
$\boldsymbol{q}$	Quadrotor rotational unit quaternion
$\boldsymbol{r}$	Quadrotor rotational rate $[\text{rad} \cdot s^{-1}]$