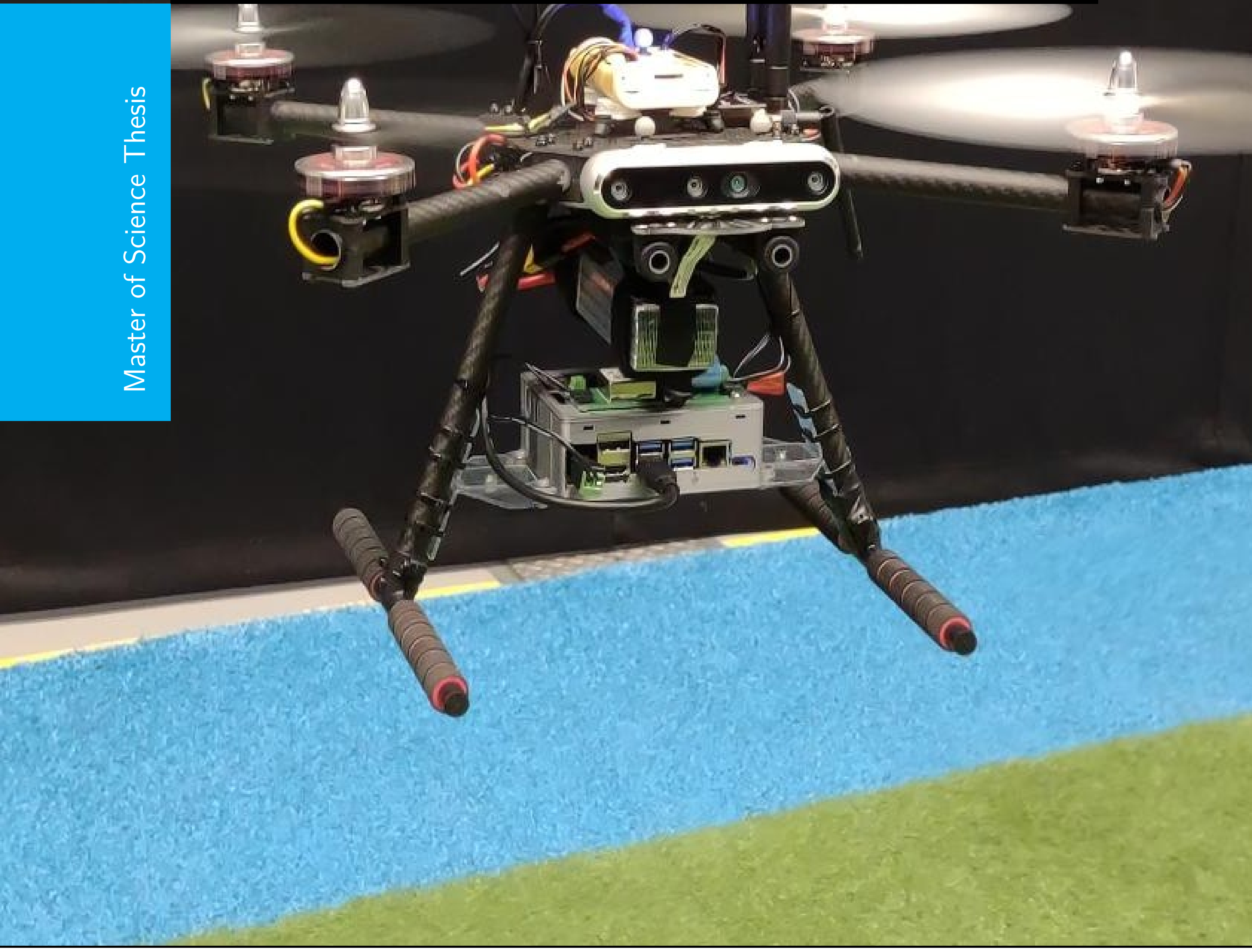
Data-Driven Optimal Control for Safe Quadrotor Navigation in Windy Environments

Johanna Probst

Master of Science Thesis



Delft Center for Systems and Control

Data-Driven Optimal Control for Safe Quadrotor Navigation in Windy Environments

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Johanna Probst

March 5, 2023

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Creating autonomous Micro Aerial Vehicles for executing complex missions poses various challenges, including safe navigation in the presence of external wind disturbances. Most current navigation methods handle external wind disturbances through real-time estimation and rejection algorithms in the control stage, but lack safety guarantees in strong winds. Recent robust methods provide safety guarantees but can be overly conservative. With the availability of more powerful computing devices, data-driven control algorithms are becoming increasingly feasible. Combining Gaussian Process models with Model Predictive Control has shown to enhance safety and performance in various control applications. Moreover, Model Predictive Control has been extended to solve more complex optimization algorithms that combine trajectory generation and tracking, preventing reference trajectories that are risky and challenging to control in the presence of disturbances.

This research aims to improve quadrotor navigation in windy environments by using Gaussian Processes to model wind disturbances. The Gaussian Process model is integrated with a Model Predictive Contouring Control formulation that combines trajectory generation and control into one optimization problem. A nominal model of the quadrotor is derived and the Gaussian Process disturbance model is trained with the quadrotor position as input and wind disturbance as output. The wind disturbance map, along with the nominal model, is implemented in the Model Predictive Contouring Controller to consider both the mean and uncertainty of the resulting probabilistic model.

This study validates the use of Gaussian Processes to model complex wind disturbances in quadrotor navigation. The wind disturbance map is trained from available state information, with data collected using an optimal exploration design to minimize uncertainty and reduce exploration times. The hyperparameters involved in training the Gaussian Process model are discussed and the implementation of sparse Gaussian Processes is outlined to make it real-time feasible for the Model Predictive Contouring Control formulation. Including the prediction model by incorporating chance constraints results in improved tracking and increased robustness in cluttered environments compared to the nominal Model Predictive Contouring Control formulation. The proposed algorithm is shown to outperform state-of-the-art methods for safe quadrotor navigation in windy and cluttered environments, being able to handle more complex wind fields than existing methods while also being less conservative.

Table of Contents

List of Figures

List of Tables

Acknowledgements

My special thanks goes to my daily supervisor Dennis Benders for his assistance throughout my entire Master's thesis. Thanks for the weekly updates, the valuable input and the assistance during lab sessions, which were a lot of fun. Even during busy times, you always had a listening ear for my questions and offered the right advice at the right moment.

I would also like to express my gratitude my supervisors Professor Tamas Keviczky and Professor Laura Ferranti for collaborating with me on this project. Thank you for your valuable inputs during our meetings and research updates, helping me to set the right priorities.

Additionally, I would like to extend my thanks to the scientific staff of the Department of Cognitive Robotics for their support. Special thanks to Thijs Nietsen and Clarence Chen, for help with the physical platform and software, Giovanni Franzese and Lorenzo Lyons for help with Gaussian Processes and Oscar de Groot for assistance with the solver.

Special thanks also goes to my fellow students and friends. Thanks to Michiel for the joint work on the physical drone and late night lab sessions. Also, thanks to Riccardo for the exchange about Gaussian Processes and for becoming a friend, making the process of writing this thesis a bit less lonely. And thanks to everyone that joined us in our study sessions and breaks: Pol, Rosita, Daniel, Daniel and Edoardo. Thanks to Mar and Jasmine for being great friends by distracting me with non-technical topics.

Lastly I would like to thank my family and friends at home who supported me throughout my studies. Thanks for being there for me, listening to my concerns and complains and comforting me in difficult times. And thanks for providing me with the motivation I needed to complete this thesis.

Delft, University of Technology
March 5, 2023

Johanna Probst

Chapter 1

Introduction

Micro Aerial Vehicles (MAV's), such as quadrotors, are becoming increasingly important in our society. Tasks of MAV's include search and rescue, environment monitoring, security surveillance, transportation, and inspection [1]. Developing autonomous quadrotors that are capable of executing complex missions poses various challenges for research.

1-1 State of the art

One of these challenges is safe navigation under real-world conditions where quadrotors are exposed to external wind disturbances. Especially in complex, cluttered environments such as forests or dense urban areas, unforeseen disturbances can cause the quadrotor to deviate significantly from its course and potentially collide with nearby objects. The inclusion of external disturbances for safe trajectory generation and tracking is therefore an important aspect to consider for real-world applications.

For the quadrotor to effectively compensate for disturbances, its controller and planner must take into account the external force. Existing methods mainly include external disturbances in the control stage using algorithms such as real-time wind disturbance estimation and rejection (see e.g. [2], [3]), and robust Model Predictive Control (MPC) methods such as tube-based MPC [4]. Obtaining an aerodynamic model of the wind disturbances is generally too complex and may also render the control problem infeasible. Instead, disturbance estimation and rejection algorithms estimate the wind online and use it to adjust the quadrotor's control inputs in real-time to compensate for the external disturbance, but they lack guarantees on their safety and can fail dramatically in case of tremendous forces. Other control methods that provide theoretical guarantees on safety, such as robust and tube-based MPC, consider bounded wind disturbances, but can be overly conservative as they consider all sources of uncertainty.

The combination of data-driven models and MPC has shown great potential for improving safety and performance in several control applications. Gaussian Process (GP) models [5], in particular, offer a mean and uncertainty estimate of the disturbance model that can be integrated into the controller to enhance tracking while also increasing robustness through the measure of uncertainty. While GP models have been applied to model aerodynamic effects in quadrotor flight [6], their use to model external disturbances in this context is not yet known.

Moreover, navigating the quadrotor through the environment requires motion planning, to generate a feasible trajectory. In many cases, the trajectory is planned with a simplified model, not taking into account uncertainties. This can lead to dangerous situations in which the planned trajectory is safe, but the actual system trajectory enters unsafe regions. Robust trajectory planners have only been active areas of research in recent years, introducing robustness into the planner using reachability theory [7] or MPC methods [8]. Modifications to the standard MPC formulation such as the Model Predictive Contouring Control (MPCC) formulation presented by the researchers in [9] allow to combine motion planning and control into one Optimal Control Problem (OCP), ensuring safe trajectories.

1-2 Research objective

The objective of this study is to enhance the safety and performance of quadrotor navigation by employing GP-based modeling techniques to learn external wind disturbances acting on the quadrotor. Since wind disturbances can be highly variable and non-linear, the aim is to assess the GP's ability to capture these complex patterns in the data. In addition to predicting mean wind conditions, the GP model can quantify the uncertainty in model predictions. This information will be used to not only improve tracking and performance, but also provide safety guarantees. In doing so, this research aims to bridge the gap between wind disturbance estimation methods and robust, model-based methods used to compensate for wind disturbances.

Moreover, this study combines trajectory generation and control into one optimal control problem, using a MPCC formulation. The modified formulation ensures that unsafe trajectories are avoided as a reference to the controller, and that robustness is maintained during both planning and control stages. Similar to existing robust control methods, the uncertainty of the disturbances is considered in the controller, only that now the uncertain bounds are provided by the model. Furthermore, it will be explored how to further reduce conservatism by utilizing the probabilistic information provided by the GP.

The research objectives can be summarized into the following research questions:

1. How well do Gaussian Processes perform to model external wind disturbances using the quadrotor state information?
2. By how much can the robustness of the MPCC controller be increased by including the learned Gaussian Process model into the MPCC formulation?

As this is the first attempt on modelling external wind disturbances with GP with the system states as input and combining it with a robust control scheme, several simplifying assumptions are made, which should be eliminated gradually in future research:

- The wind field is assumed to be spatially-varying and time-invariant. A map of the wind disturbances, represented by a GP, can be created by the quadrotor by exploring the environment beforehand, learning the disturbances offline.

- A mathematical description of the obstacle position and geometry is known beforehand.
- A simplified simulation is used, in which the only disturbance is the wind disturbance. This means that the MPCC controller has perfect system knowledge, except for the wind disturbances.
- The dimensionality of the quadrotor environment is reduced from 3D to 2.5D.
- The attitude controller is given, therefore the focus is on trajectory planning and position control of the quadrotor.

1-3 Thesis contributions

The research discussed in this thesis has the following main contributions:

System identification of the attitude dynamics model of the HoverGames platform This research is one of the first to work with the newly designed HoverGames [10] platform, which is optimized for running computationally heavy algorithms on board of the quadrotor. Utilizing the attitude controller provided by the flight controller onboard, the attitude dynamics model is identified. As the HoverGames platform will be used in future research at the department, a contribution is made by making the attitude dynamics model available to other researchers.

Validation of GP's to model a complex wind disturbance map based on quadrotor state information and optimal data collection It is demonstrated that GP's are capable of modeling complex wind disturbances, by training a wind disturbance map based on available state information. Additionally, an optimal experiment design is provided to save time during data collection while generating a more accurate wind disturbance map.

Extension of MPCC algorithm with the GP model for improved tracking and increased robustness in quadrotor platforms The existing MPCC algorithm in [9] is extended with the uncertain model including both the mean and uncertainty of the prediction model, by incorporating the chance constraints in [11]. The improved tracking and increased robustness of the GP-based MPCC method compared to the standard formulation is shown.

Advanced trajectory generation and control algorithm for complex wind fields and cluttered environments By benchmarking the GP-based MPCC algorithm against existing state-of-the-art methods for quadrotor navigation in windy and cluttered environments, it is demonstrated that it outperforms the state of the art by being capable of handling more complex wind fields while also being less conservative, which is crucial given the limited space.

1-4 Thesis outline

The report is structured as follows:

- Chapter 2 gives a more detailed overview of the state of the art of trajectory planning and control of quadrotors under wind and presents the problem formulation of this thesis.

- Chapter 3 presents the nominal quadrotor model used in the motion planning and control algorithm together with system identification performed to find relevant parameters of the attitude dynamics model.
- Chapter 4 describes training of the Gaussian Process disturbance map together with a background on Gaussian Processes and an optimal experiment design.
- Chapter 5 introduces the proposed motion planning and control algorithm, integrating the information provided by the Gaussian Process model.
- Chapter 6 presents a comparison of the proposed algorithm with the current state of the art.
- Chapter 7 gives a summary of this thesis and recommendations for future work.

Chapter 2

Trajectory planning and control of quadrotors under wind

Research on quadrotor navigation in windy outdoor environments is ongoing. It involves combining existing quadrotor navigation methods with techniques for rejecting wind disturbances. This chapter provides an overview of current research, highlighting areas for improvement and presenting the problem formulation used to address existing limitations.

2-1 Background

This section presents an overview of current motion planners and quadrotor control techniques with a specific focus on methods for handling wind disturbances and uncertainty. This includes data-driven and machine learning methods. Following the overview, the problem statement and the ways in which this thesis aims to expand upon existing literature will be discussed.

2-1-1 Standard flight trajectory planning and control

A quadrotor is the most common type of Micro Aerial Vehicle (MAV) that consists of a rigid body with four rotors. By varying the rotational velocity of the rotors, the quadrotor can perform various flight maneuvers, such as hovering, ascending, descending, and rotating about its axes. Navigating a quadrotor involves both controlling its motion and planning a feasible sequence of movements. The motion planner precedes the quadrotor controller, and ideally both consider the quadrotor's dynamics and external disturbances.

Quadrotor motion planning The goal of the motion planner is to find a valid motion sequence for the quadrotor from start to goal position without collisions. Motion planners can be divided into *global* planners and *local* planners. Global planners plan the entire motion using a map of the environment while local planners update waypoints as the quadrotor moves, taking into account dynamic obstacles and vehicle constraints. Another distinction is made between *path* planners and *trajectory* planners. Path planners ignore the dynamics of the quadrotor and other differential constraints. They focus primarily on the translations and rotations required to move the robot. Common path planners are search-based methods, sampling-based methods and potential field methods [12].

Trajectory planning However, if the robot is subject to dynamics:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{2-1}$$

with state space $\mathbf{x} \in \mathbb{R}^n$, input space $\mathbf{u} \in \mathbb{R}^m$ and mechanical constraints on the state space $\mathbf{x} \in \mathcal{X}$ and input space $\mathbf{u} \in \mathcal{U}$, these must be considered in the motion planning problem. This is called the trajectory planning problem, where the trajectory describes the evolution of vehicle configurations over time. Usually, the path planner precedes the trajectory planner, which then time-parameterizes the solutions of the path planner [13]. Some trajectory planners also solve both problems simultaneously. Existing approaches for quadrotor trajectory planning include polynomial, discrete-time search, sampling-based, and optimization methods [14].

Quadrotor control The quadrotor's motion is typically controlled in a cascaded control structure as shown in Figure 2-1. It consists of an outer loop position controller and an inner loop attitude controller. The position controller computes the desired attitude angles of the quadrotor to reach a certain position $\mathbf{p}_\mathrm{c} \in \mathbb{R}^3$ and thrust T_c needed to keep the quadrotor in the air. The attitude controller tracks the desired roll ϕ_c, pitch θ_c and yaw ψ_c by controlling the angular velocities $\dot{\phi}_\mathrm{c}$, $\dot{\theta}_\mathrm{c}$, $\dot{\psi}_\mathrm{c}$. The mixer is in place to translate the angular velocities and thrust command into individual rotor commands.

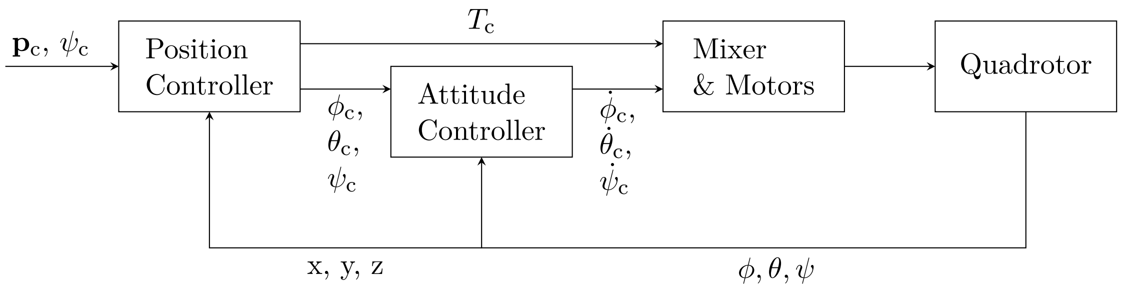

$\mathbf{p}_\mathrm{c}$, ψ_c
Position Controller
T_c
Attitude Controller
ϕ_c, θ_c, ψ_c
Mixer & Motors
$\dot{\phi}_\mathrm{c}$, $\dot{\theta}_\mathrm{c}$, $\dot{\psi}_\mathrm{c}$
Quadrotor
x, y, z
ϕ, θ, ψ

Figure 2-1: Cascaded control architecture of a common quadrotor controller.

Control techniques Due to the increased research interest in quadrotors in the last decades, numerous control techniques have been implemented for the quadrotor, ranging from linear to nonlinear control methods, model-based versus model-free controllers, robust, adaptive, and intelligent controllers (e.g.. [15], [16], [17], [18], [19]). Linear controllers are suitable when the quadrotor does not deviate greatly from the hovering state, but for extreme maneuvers or attitude control, nonlinear methods are typically used [20].

Model predictive control Model-based methods, such as Model Predictive Control (MPC) [21], can incorporate system dynamics into the controller design and facilitate parameter tuning when a model of the quadrotor dynamics is available. MPC solves an on-line optimization problem at each sampling instant, taking the current state as the initial state, predicting future evolution of states over a finite horizon and optimizing a sequence of control inputs with a certain cost. Only the first control input is applied to the system and the process is repeated at the next time-step. One of the main features of MPC is the use of constraints, which make it possible to incorporate physical or operational limits into the control strategy.

Model predictive contouring control With optimization-based methods, it is also possible to combine trajectory planning and trajectory tracking in one optimization problem. Given a path by the planner, the optimization problem solves for the optimal system states and system inputs. The solution to this problem lies in using a modified MPC algorithm that solves the trajectory generation problem simultaneously. Model Predictive Contouring Control (MPCC) [22] is a popular approach for this purpose. MPCC has been applied for static and dynamic obstacle avoidance with ground vehicles [23] and autonomous ground robots [9]. In the context of quadrotor navigation it has found an application in drone cinematography [24] and quadrotor racing [25].

2-1-2 External wind disturbances

In many practical applications, the quadrotor is subject to external wind disturbances. The wind field may vary in time and space, which must be considered in the motion planning and control design.

Wind disturbance estimation and rejection There are several control algorithms that observe the disturbance online and reject it by calculating a compensating control input. If the quadrotor is equipped with airspeed sensors such as Pitot tubes, a local estimate of the wind can be determined directly [26]. In most cases, however, these sensors are not present on board of the quadrotor. Instead, one could use the model of the quadrotor to estimate the wind based on the difference between the desired and measured velocity or acceleration. This idea is used, e.g. for a nonlinear inversion-based position controller in [2]. The authors in [27] use an observer design to estimate the external force vectors. However, this method only provides a deterministic estimate of the disturbance. More information about the disturbance can be obtained using stochastic estimators such as the extended or unscented Kalman filter [3]. In addition to the nominal disturbance force or torque, these estimators also provide an uncertainty of the estimate that can be incorporated into the controller design.

Wind disturbance model In motion planning and predictive control it is advantageous to know the evolution of the disturbance over future states. One way to accomplish this is to have a comprehensive aerodynamic model to describe wind disturbances [28]. A drawback is that determining the model parameters requires extensive experimentation, and incorporating the complex model into a predictive controller increases the computational cost. In addition, robust methods have been developed [29] that assume a constant disturbance bound for all future states, and guarantee performance for all disturbances within this bound.

Robust motion planners In particular, robust motion planning algorithms have been active areas of research in the past few years [7]. A way to introduce robustness into the motion planner are reachability-based methods, taking into account all possible states that the quadrotor can reach, given information about its uncertainty. The authors in [30] precompute a library of safety funnels around trajectories, which are pieced together at runtime to generate feasible and safe trajectories. Furthermore, Hamilton-Jacobi (HJ) reachability analysis is developed for robust offline trajectory planning in fully known environments and independent of the motion planner, called FasTrack [31]. FasTrack computes the tracking error as a function of control inputs and generates a feedback controller to compensate for it. The work presented by the authors in [7] uses Forward Reachable Set (FRS)'s that are computed offline and bound the tracking error. During online planning, the FRS are used to map obstacles to parameterized trajectories so that a safe trajectory can be selected. The authors in [8] use FRS to compute outer bounding ellipsoids that approximate the position of the quadrotor plus its uncertainty, which are then used in an Nonlinear Model Predictive Control (NMPC) controller.

Robust and stochastic nonlinear model predictive control Robust motion planning and control algorithms that can handle both set-bounded and probabilistic uncertainties can be achieved through the use of NMPC methods. For example, robust Model Predictive Control (MPC) formulations have been used to solve the robust collision avoidance problem as demonstrated in [32]. Additionally, the authors in [33] build upon the concept of robust MPC and employ control contraction metrics to create a feedback control law and a tube. Stochastic MPC formulations for collision and obstacle avoidance are first introduced by [34], where polytopic chance constraints are reformulated into deterministic constraints. This work is further advanced with the addition of quadratic obstacle avoidance constraints for dynamic collision avoidance as presented in [11].

2-1-3 Data-driven disturbance model

With the availability of more powerful computing devices, data-driven modelling techniques are becoming increasingly popular. Techniques for data-driven modeling have been developed to enhance existing system models and estimate disturbances for various applications.

Set-membership identification A prominent method for learning a robust representation of the model uncertainty is set-membership identification [35], [36]. A practical implementation of set-membership identification together with model predictive control on a quadrotor is presented by the authors in [37]. Using set-membership identification, the quadrotor is able to adapt parameters in uncertain wind conditions, while satisfying the constraints.

Neural networks The authors in [38] use meta-learning with a deep neural network to model the residual dynamics associated with wind disturbances on a quadrotor platform.

Gaussian Processes Gaussian Process (GP) [5] models are used learn a stochastic model of the dynamics. In the context of quadrotor control, GP's have been used to describe the nonlinear dynamics and aerodynamic effects during aggressive flight maneuvers [6] and to guarantee robust obstacle avoidance [39]. Furthermore, GP's are used to model wind uncertainties in [40]. Three independent GP's are trained based on the observed disturbances. Therefore, the approach can only respond to disturbances if they have been observed. The work in [41] creates a wind map based on the quadrotor position fusing Inertial Measurement Unit (IMU) and airflow measurements. Moreover, GP's have been used successfully to model residual dynamics for mobile robots [42], race cars [43] and a robotic manipulator [44].

2-2 Breakdown of the proposed solution

Creating safe trajectories in complex and dynamic environments where quadrotors are subject to external disturbances is a critical concern. Existing robust control methods that can track a given trajectory and compensate for disturbances may struggle with handling large instantaneous forces or produce overly conservative trajectories that account for all sources of uncertainty. Furthermore, not considering external forces during trajectory planning can result in potentially hazardous reference trajectories.

Proposed solution To avoid the risk of dangerous trajectories and control failures, this approach uses a formulation of MPCC that combines trajectory generation and control into a single optimal control problem, similar to MPC which is well established for quadrotor control. In addition, data-driven modeling techniques are employed to construct a model of the wind disturbances that not only accounts for instantaneous forces but also predicts disturbances for future states. Specifically, a GP model is used which provides a mean and stochastic uncertainty of the disturbance, which can be incorporated into the optimal control problem to increase robustness without being overly conservative.

The problem is divided into three distinct steps as described below.

2-2-1 Nominal system model identification

As a baseline, a nominal system model is required. MPCC already uses a nonlinear Optimal Control Problem (OCP) formulation, so a nonlinear model is chosen to describe the quadrotor's nominal dynamics. This research focuses on navigation, thus the onboard attitude controller will be used and priority will be given to position control. This also introduces a separation layer to keep critical code running even if there is a failure in the high-level computer [32]. The attitude dynamics are modeled with a linear first-order approximation derived through system identification.

2-2-2 Data-driven wind disturbance model

As there are no additional wind sensors mounted on the quadrotor, the wind disturbance is estimated based on the available sensor data. Starting from a nominal quadrotor model and assuming that there are no other disturbances acting on the quadrotor, the momentary wind is attributed to the difference between the predicted velocity and the measured velocity. With the measured error as the training objective and the quadrotor states as the training input, a GP model is learned.

Wind disturbance map To keep the initial problem simple while demonstrating the validity of this approach, spatially varying, time-invariant wind is assumed. Thus, a wind disturbance map is trained offline, reducing the training input to the quadrotor's position $\mathbf{p}$. This map can be created during an initial mapping of the environment or when the quadrotor traverses the environment several times. Later iterations should investigate the suitability of this approach to also learn time-varying wind disturbances online.

Practical considerations of the GP model This research primarily aims to examine the effectiveness of using Gaussian Process Regression (GPR) to learn wind disturbances based on available data. Additionally, it addresses practical concerns such as reducing computational complexity through the use of sparse GP and how to propagate uncertain system states.

2-2-3 Motion planning and control formulation

The probabilistic nature of the wind disturbance model leads to a mean and uncertain state description, which is incorporated into the MPCC framework in [9], assuming a static map of obstacles. By using a simple reference path and reference velocity, the MPCC's optimal control formulation guarantees obstacle avoidance while maintaining the reference velocity. Taking inspiration from work on robust and stochastic model predictive control, this is achieved by tightening the obstacle avoidance constraints based on the uncertain region around the quadrotor trajectory. Since the uncertainty information is probabilistic in nature, an ellipsoidal and a chance-constrained formulation are employed and compared to ensure safe obstacle avoidance. In this scenario, a global map of the obstacles is known beforehand.

2-3 Problem formulation

An overview of the problem formulation is given in Figure 2-2. The center of the proposed method is the MPCC controller which takes into account the static obstacle map, the given reference path and the quadrotor model. The quadrotor model consists of the nominal model and the added GP model.

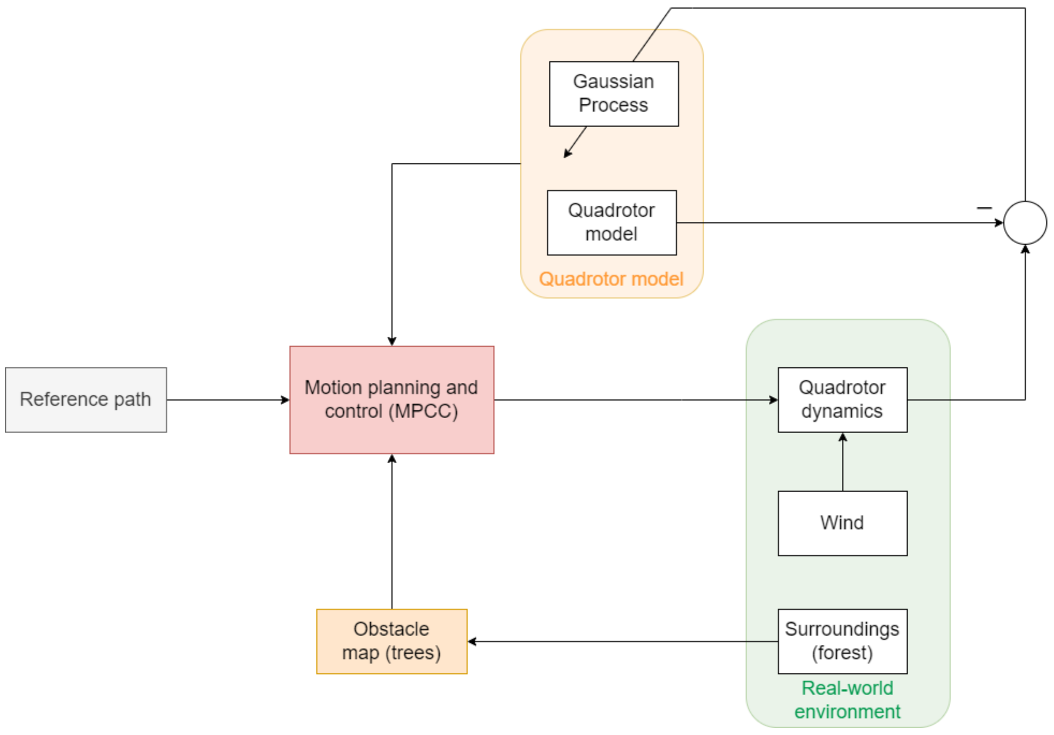


Figure 2-2: Overview of the data-driven motion planning and control method.

Given this information, the MPCC solves an OCP for the control input that is sent to the quadrotor. If the quadrotor is subject to any wind disturbances, the actual velocity of the quadrotor will deviate from the velocity predicted by the nominal quadrotor model. This information can be used to train the GP.

2-4 Summary and discussion

The existing methods for navigation of quadrotors in wind disturbances have certain shortcomings. One limitation is that wind disturbances are often only rejected after they have been observed, making it difficult to cope with strong winds due to the physical limitations of the platform. Additionally, the task of identifying a feasible path and following it is commonly approached separately, with wind disturbances only taken into account in one of these steps. This leads to either a planned trajectory that is risky or motion planning and control methods that are robust but overly conservative.

This work proposes to use a data-driven approach to model wind disturbances using a GP model to predict disturbances in the future while being less conservative by using a stochastic description of the uncertain bounds. This model is included into a MPCC formulation combining both motion planning and control into one step, eliminating over-conservative or risky reference trajectories.

In the coming chapters, the proposed method is explained in more detail and the validity of the approach is shown.

Chapter 3

Nominal quadrotor model

This chapter discusses the derivation of the nominal quadrotor Equations of Motion (EOM) based on first-principle models, and how the inner-loop position controller can be approximated and integrated into the dynamic model of the quadrotor. The second part of this chapter describes the system identification to determine the coefficients of the attitude dynamics model.

3-1 First principle quadrotor model

The nominal quadrotor model can be found using first principle modeling techniques based on simplifying assumptions of the real-world dynamics. Newton-Euler EOM are used to describe the quadrotor dynamics.

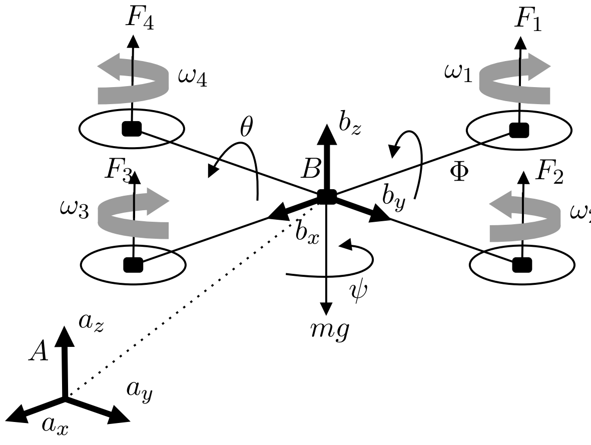


Figure 3-1: Quadrotor configuration with world frame A and body frame B.

Quadrotor configuration The quadrotor consists of a rigid body cross-frame and four fixed-pitch rotors at each end. Each rotor's speed can be varied individually to control the output states of the quadrotor. The output states are the Cartesian position states x, y, z, and the three Euler angles roll, pitch, and yaw, denoted by ϕ, θ, and ψ, respectively. Having four inputs and six outputs, the quadrotor is an under-actuated system. The quadrotor is treated as a rigid body with mass m and inertia $I \in \mathbb{R}^{3\times3}$. Two reference frames are defined as shown in Figure 3-1 to describe the quadrotor motion: The position $\mathbf{p} = [x, y, z]$, and velocity $\mathbf{v}$=[v_x, v_y, v_z] $\in \mathbb{R}^3$ are expressed in the world frame $\{A\}$. The orientation of the quadrotor body frame $\{B\}$ with respect to $\{A\}$ is specified by the Euler angles.

Rotation matrix Different Euler angle representations can be used to describe the rotation from the world frame to the body frame of the quadrotor $\{B\}$. Using a Z - X - Y Euler angle configuration the rotation matrix R is described as:

$$R = R_{B\to A} = R_Z(\psi)R_X(\phi)R_Y(\theta) \tag{3-1a}$$

$$= \begin{bmatrix} c\psi c\theta & c\psi s\phi s\theta - c\phi s\psi & s\psi s\phi + c\psi c\phi s\theta \\ c\theta s\psi & c\phi c\psi + s\psi s\phi s\theta & c\phi s\theta s\psi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\phi c\theta \end{bmatrix} \tag{3-1b}$$

with c denoting the cosine function and s denoting the sine function.

Newton-Euler equations The longitudinal dynamics are then described through Newton's EOM as:

$$\dot{\mathbf{p}} = \mathbf{v} \tag{3-2a}$$

$$m\dot{\mathbf{v}} = -mg\mathbf{a}_3 + R\mathbf{T} \tag{3-2b}$$

where g is the gravitation constant, $\mathbf{a}_3 = [0, 0, 1]^T \in \{A\}$ is the unit vector pointing in the z axis of the world frame, and $\mathbf{T} = [0, 0, T]^T \in \{B\}$ is the generated thrust. Expanding (3-2b) results in:

$$m\dot{v}_x = T(\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi) \tag{3-3a}$$

$$m\dot{v}_y = T(-\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi) \tag{3-3b}$$

$$m\dot{v}_z = T(\cos\phi\cos\theta) - mg. \tag{3-3c}$$

Cascaded control approach In practice, the control is often split such that a low-level attitude controller is present as an inner-loop, while a model-based trajectory tracking controller is running as an outer-loop [45]. The inner-loop attitude controller is stabilizing the system, which can avoid numerical problems in optimization software and allows for a lower sampling rate of the position controller [46]. As an inner-loop attitude controller is also provided by the platform of this research, is it used, focusing on motion planning and position control of the outer loop.

Attitude dynamics To achieve accurate trajectory tracking, the high-level position controller must consider the inner loop system dynamics. These dynamics are approximated by a simple linear model of the form:

$$\dot{\phi} = \frac{1}{\tau_\phi}\left(k_\phi\phi_\mathrm{c} - \phi\right) \tag{3-4a}$$

$$\dot{\theta} = \frac{1}{\tau_\theta}\left(k_\theta\theta_\mathrm{c} - \theta\right) \tag{3-4b}$$

$$\dot{\psi} = \dot{\psi}_\mathrm{c} \tag{3-4c}$$

$$T = T_\mathrm{c} \tag{3-4d}$$

with system inputs $\mathbf{u} = [\phi_{\mathrm{c}}, \theta_{\mathrm{c}}, \psi_{\mathrm{c}}, T_{\mathrm{c}}]$. The coefficients of the first-order model are found using system identification. This is described in section 3-2.

Aerodynamic effects on the quadrotor can be included in the mathematical quadrotor model. The two main effects when flying are blade flapping and induced drag [47]. One can account for these effects by adding an external drag force proportional to the system velocity:

$$m\dot{\mathbf{v}} = -mg\mathbf{a}_3 + R\mathbf{T} - D\mathbf{v} \tag{3-5}$$

where $D = \operatorname{diag}(k_D, k_D, k_D)$ and k_D is the drag coefficient. The drag coefficient for a specific system can be determined through system identification, with values ranging from $k_D = 0.02$ to $k_D = 0.2$ in related literature. Hereafter, literature values will be used for the model. If values are not accurate enough, they should be determined through system identification in future work.

Derived quadrotor model The resulting quadrotor model is summarized below:

$$\dot{\mathbf{p}} = \mathbf{v} \tag{3-6a}$$
$$m\dot{v}_x = T_{\mathrm{c}}\left(\sin(\psi)\sin(\phi) + \cos(\phi)\sin(\theta)\cos(\psi)\right) - k_D v_x \tag{3-6b}$$
$$m\dot{v}_y = T_{\mathrm{c}}\left(-\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi)\right) - k_D v_y \tag{3-6c}$$
$$m\dot{v}_z = T_{\mathrm{c}}\left(\cos(\phi)\cos(\theta)\right) - mg - k_D v_z \tag{3-6d}$$
$$\dot{\phi} = \frac{1}{\tau_\phi}(k_\phi \phi_{\mathrm{c}} - \phi) \tag{3-6e}$$
$$\dot{\theta} = \frac{1}{\tau_\theta}(k_\theta \theta_{\mathrm{c}} - \theta) \tag{3-6f}$$
$$\dot{\psi} = \dot{\psi}_{\mathrm{c}} \tag{3-6g}$$

with state vector $\mathbf{x} = \left[\mathbf{p}^T, \mathbf{v}^T, \phi, \theta, \psi\right]^T$ and input vector $\mathbf{u} = \left[\phi_{\mathrm{c}}, \theta_c, \dot{\psi}_{\mathrm{c}}, T_{\mathrm{c}}\right]^T$.

Linear quadrotor model Assuming small attitude angles, the quadrotor dynamics model can be linearized around its hovering condition with velocity in z-direction $\dot{v}_z = 0$. Furthermore, the vehicle heading is aligned with the inertial frame x-axis, such that the yaw angle $\psi = 0$. The linearized system dynamics are:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -k_D* & 0 & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 0 & -k_D* & 0 & -g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -k_D* & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\phi} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\theta} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{A} \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ \phi \\ \theta \\ \psi \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k_\phi}{\tau_\phi} & 0 & 0 & 0 \\ 0 & \frac{k_\theta}{\tau_\theta} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{B} \begin{bmatrix} \phi_{\mathrm{c}} \\ \theta_{\mathrm{c}} \\ \dot{\psi}_{\mathrm{c}} \\ T_{\mathrm{c}} \end{bmatrix} \tag{3-7}$$

with the mass-normalized drag coefficient $k_D* = \frac{k_D}{m}$ [48].

3-2 System identification of the attitude dynamics

System identification experiments are performed to identify the coefficients of the attitude dynamics model described in Eq. (3-4). Additionally, a thrust model is identified that translates the thrust control inputs to the desired acceleration of the quadrotor.

3-2-1 Attitude dynamics and thrust model

The model of the attitude and thrust dynamics can take different forms depending on if a basic model is considered or if real-world effects are taken into account.

First order attitude model The attitude dynamics in the roll and pitch direction of the quadrotor are approximated by a first order model:

$$\dot{\phi} = \frac{1}{\tau_\phi} \left(k_\phi \phi_\mathrm{c} - \phi \right) \tag{3-8a}$$

$$\dot{\theta} = \frac{1}{\tau_\theta} \left(k_\theta \theta_\mathrm{c} - \theta \right). \tag{3-8b}$$

Here k_ϕ and k_θ are the steady-state gains of the roll and pitch dynamics, respectively and τ_ϕ and τ_θ are the time constants of the roll and pitch dynamics, respectively. These parameters are to be identified for the attitude dynamics model.

First order attitude model with time delay For the physical system delays arise between the generation and execution of the control command. This is taken into account by adding a time delay to the first order model, resulting in dynamics of the form:

$$\dot{\phi} = \frac{1}{\tau\,\phi} \left(k_\phi (t - \beta) \phi_\mathrm{c} - \phi \right) \tag{3-9a}$$

$$\dot{\theta} = \frac{1}{\tau\,\theta} \left(k_\theta (t - \beta) \theta_\mathrm{c} - \theta \right). \tag{3-9b}$$

where β is the time delay. It is assumed to be identical for pitch and roll control and identified as part of the model to get more accurate coefficients. However, for the final model used in the controller it will not be considered to reduce model complexity.

Thrust dynamics model The attitude controller on-board of the quadrotor is designed to take a thrust command $T_\mathrm{PWM} = 0$ ... 1 that is translated into a Pulse-Width Modulation (PWM) signal to the motors, resulting in a desired rotor speed and thrust of the quadrotor. However, for the model in Eq. (3-6), the thrust command T_c corresponds to a force. T_c is divided by the mass of the quadrotor to obtain an acceleration $T_\mathrm{c}^* = \frac{T_\mathrm{c}}{m}$ that can be accurately measured with the on-board sensors. The relation between the PWM thrust command and the accelerative thrust T_c^* of the quadrotor can be described by a linear relation:

$$T_\mathrm{PWM} = a_T \cdot T_\mathrm{c}^* + T_\mathrm{h}. \tag{3-10}$$

The linear scaling a_T and an offset T_h, corresponding to the hovering thrust, are to be identified. For the real, physical quadrotor the thrust relation is furthermore a function of the battery voltage U. A third therm b_T is added to the thrust relation to account for that effect. It is linearized around the average battery voltage at $\bar{U} = 14.5V$, resulting in the thrust dynamics model:

$$T_{\mathrm{PWM}} = a_T \cdot T_{\mathrm{c}}^{*} + b_T \cdot (U - 14.5V) + T_{\mathrm{h}}. \tag{3-11}$$

In the next section, it is described how the experiments are performed to find the model parameters.

3-2-2 Experimental overview

System identification is performed for two types of environments. The first environment is the Gazebo simulation environment, the second environment is the physical quadrotor which is controlled in a laboratory. In both cases, control inputs specifically designed for system identification are sent to the quadrotor. The measured outputs are processed such that they can be used for system identification.

physical platform and software setup

Physical experiments are performed on the HoverGames [10] quadrotor provided as a development toolkit by NXP. It is equipped with the RDDRONE-FMUK66 flight management unit at its base. The software used by the flight controller is the PX4 Autopilot software [49]. The PX4 software handles the control of the quadrotor and the interface with other devices. It provides a flexible, modular framework that allows the integration of an external control scheme for the quadrotor position control while using the attitude controller provided by PX4 to maintain stability. PX4 furthermore provides a built-in state estimation, that fuses the sensor data obtained by the Inertial Measurement Unit (IMU) and magnetometer with the external pose information from the OptiTrack motion capture system to provide an estimate of the system's full odometry data.

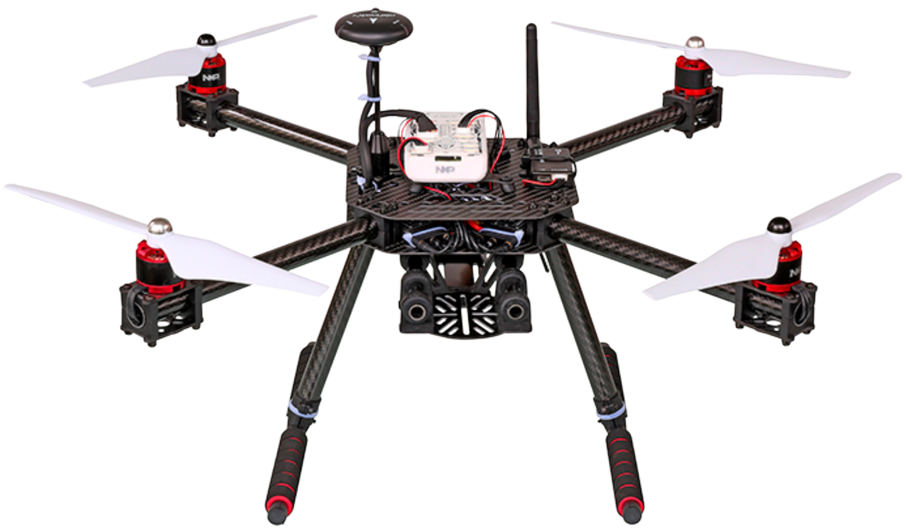


Figure 3-2: The HoverGames platform used for this research [10].

The HoverGames quadrotor platform is moreover equipped with an Nvidia Jetson Xavier on-board processor that sends control commands to the physical quadrotor using the Robot Operating System (ROS) software library. PX4 supports MAVROS messages as a bridge between the flight controller with MAVLink communication protocol and ROS. The MAVROS mesages are sent via serial connection to the PX4 flight controller. A remote WiFi connection allows accessing the on-board computer from the ground station.

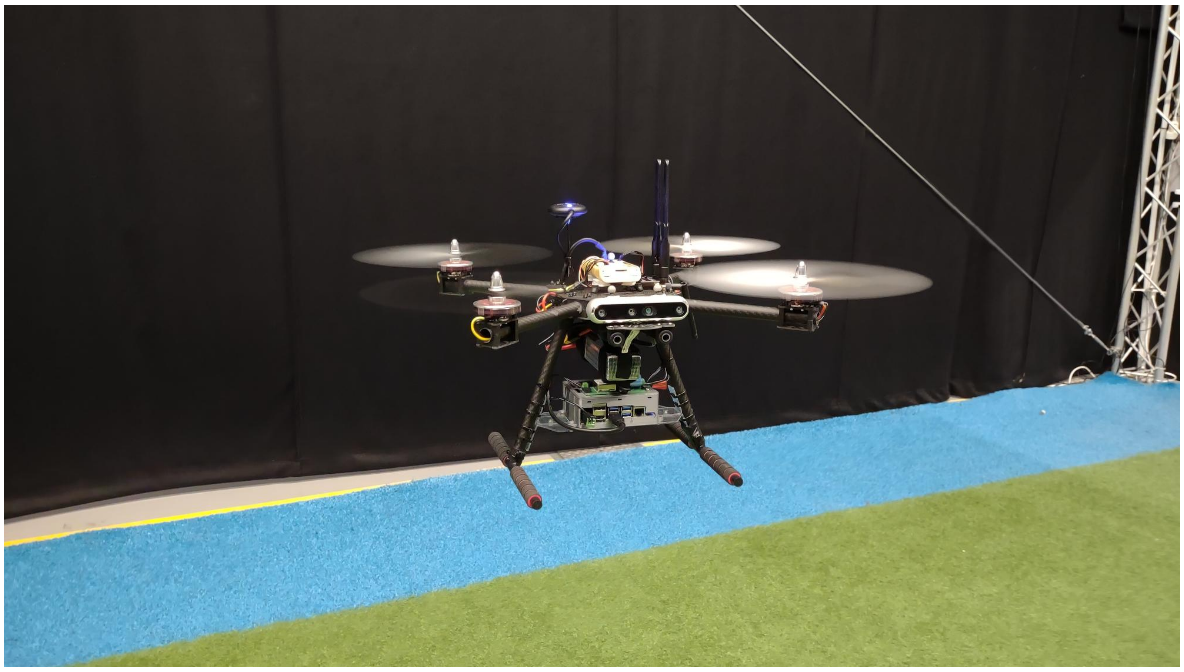


Figure 3-3: HoverGames quadrotor flying in the lab environment.

Simulation environment

Before the code is tested on the actual quadrotor, Software in the Loop (SITL) tests are performed in Gazebo, a 3D simulation environment for autonomous robots as shown in Figure 3-4. The SITL simulation environment is provided by PX4. It is adopted to reflect the dynamics of the HoverGames quadrotor. In the simulation environment, the quadrotor is controlled via MAVROS commands. If the simulation dynamics match the physical dynamics, the controller code can therefore directly be transferred from the simulation to the quadrotor.

Control inputs

The control inputs that are sent to the quadrotor are the roll and pitch command, ϕ_c and θ_c and the thrust command T_{PWM}. The yaw ψ and yaw rate $\dot{\psi}$ are kept at zero. Different types of input signals are selected to excite the system.

Roll and pitch control commands As the system to be identified is first order, most of the relevant coefficients can be identified from a simple step response of the system. To generate more data, the step input is sent as a block wave varying between +1 and -1 with a

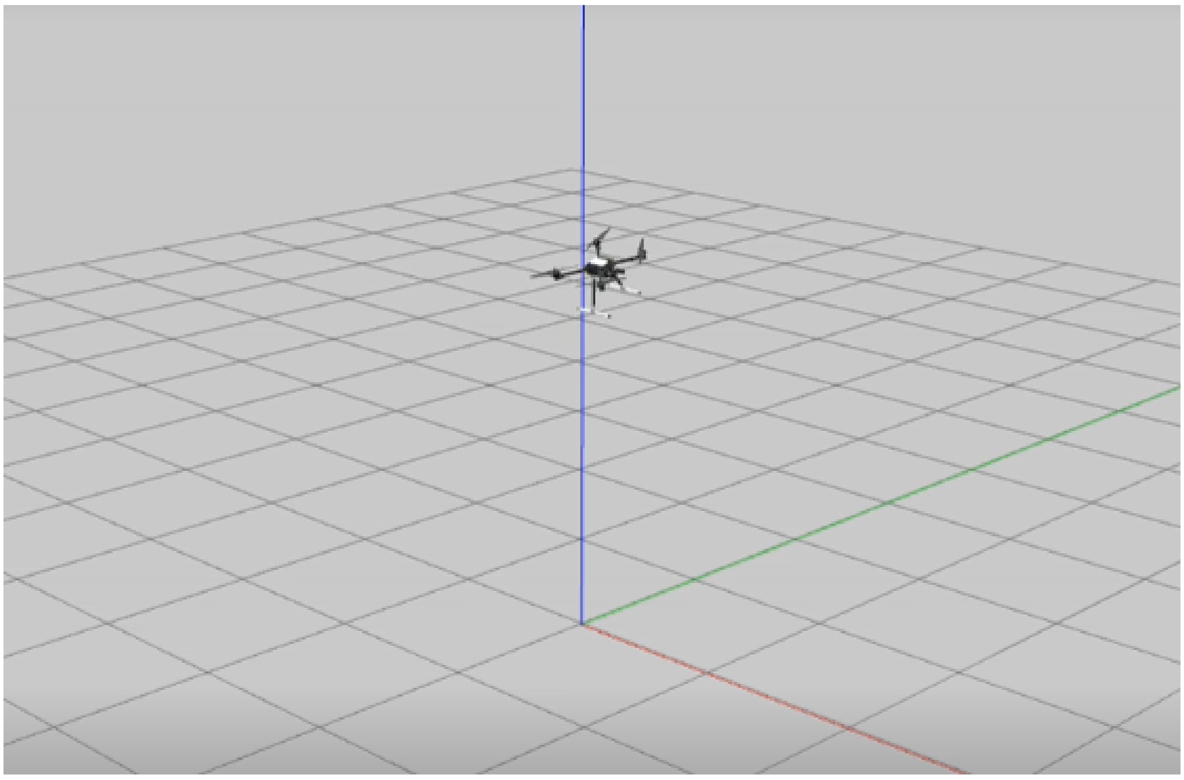


Figure 3-4: HoverGames quadrotor flying in the Gazebo simulation environment.

constant frequency, which is determined by the environment bounds. The block wave is scaled accordingly to track different input angles between 12 degrees and 24 degrees of the quadrotor, assuming that this represents the operating range of the quadrotor. To furthermore excite the system at different frequencies, a Random Binary Signal (RBS) is sent to the quadrotor, varying the frequency of the blockwave randomly within a set frequency range. The upper frequency is set to 0.5Hz which is calculated from the bandwidth of the excited input. The lower frequency is determined by the lab bounds. The roll and pitch directions are excited separately due to limited space in the laboratory. As there is no dependence between the roll and pitch direction, this reflects the behavior of the quadrotor also if the control commands are sent simultaneously. The remaining attitude input commands are calculated by a LQR controller to stabilize the quadrotor. This model-based controller could be implemented using some initial identification data without stabilization. The thrust command is calculated by a PID controller stabilizing the quadrotor in z-direction, as no model is derived for the z-direction.

Thrust control command In simulation, a blockwave is sent to the quadrotor varying the PWM control command T_{PWM} between zero and one and the resulting acceleration of the quadrotor is measured to derive the linear relation. However, in the laboratory these kind of experiments are too risky given the limited space. Instead, the quadrotor hovers with different weights attached to it. The weights are increased in steps of 100g until a total of 1kg. Additionally, the nominal weight is reduced by removing the on-board processor. The quadrotor is kept hovering at different weights, starting from a full battery, until the battery voltage is critically low. The thrust command T_{PMW} at which the drone hovers is recorded. It is automatically adjusted by the PX4 software in response to changes in weight and decreasing voltage.

Data processing

The control commands and outputs are recorded from the MAVROS topics at a frequency of 50Hz. The recorded data is processed using MATLAB.

Roll and pitch identification data The recorded data are the roll and pitch input commands and the roll and pitch system response from ROS topics. As the messages are not precisely sent at each sampling interval, the data is interpolated at equal timestamps of 0.02s, matching the sampling frequency. The input and output are stored as MATLAB identification data (`iddata`) object.

Thrust identification data In simulation, the recorded input is the thrust command and the output is the acceleration of the quadrotor in z-direction. No further pre-processing is required. For the physical experiments, the thrust command and battery level are recorded. The respective acceleration, derived from the mass of the quadrotor for each experiment is:

$$T_\mathrm{c}^* = \frac{m_n}{m_t} g \tag{3-12}$$

with the nominal mass of the quadrotor m_n and m_t the total mass, given the added or removed weight.

3-2-3 Parameter estimation and validation

From the recorded data, the attitude dynamics and thrust model are derived by identifying the model parameters. The derived models are compared against previously unseen data to validate the model.

Parameter estimation

Given the processed data, the attitude models are identified making use of the MATLAB system identification toolbox. The thrust model can be derived by fitting a linear model to the recorded data.

Roll and pitch model parameters The roll and pitch parameters are identified using the transfer function estimation function in MATLAB `tfest`. Given the identification data object and the desired numbers of poles, it estimates the continuous-time transfer function of the roll and pitch dynamics. The steady-state gain and time constant can be extracted from the coefficients of the transfer function model. Multiple datasets are combined and passed on to the transfer function estimation function to cover a wide range of scenarios.

Time delay To estimate the time delay, the MATLAB function for delay estimation `delayest` is used. This function estimates the number of time steps it takes for the output data to respond to the input data. The time delay is passed on to the transfer function estimation method and is taken into account when estimating the continuous-time transfer function model. The delay is estimated from simple step responses.

Thrust model parameters The linear thrust dynamics model is identified fitting a first degree polynomial to the data. For the real-world dynamics, two first-degree polynomial functions are fitted. One describing the relation between T_{PWM} and the acceleration T_{c}^* and the other one describing the relation between T_{PWM} and the battery voltage U. The linear models are combined to find the coefficients of the model in Eq. (3-11).

Model validation

The models are validated against previously unseen system data to determine how well the derived models reflect the actual system behavior. This is described by the goodness of fit, corresponding to $(1 - \text{NRMSE})100\%$, where NRMSE is the Normalized Root Mean Squared Error between the predicted and measured output of the model.

3-2-4 Identification results

The identified model parameters for the first order attitude model including a time delay and the thrust dynamics model for the simulation are listed in Table 3-1 and the parameters of the real-world experiment are listed in Table 3-2.

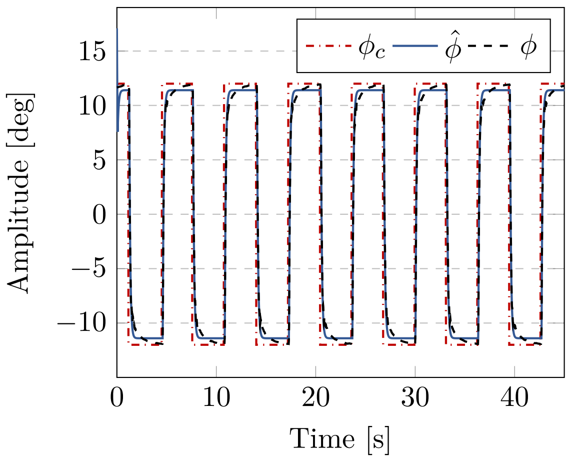
Identified parameters in simulation In simulation, the quadrotor is perfectly symmetric with equal tuning parameters of the attitude controller, such that the behavior in roll and pitch direction is identical. Therefore, the identified parameter values for the attitude dynamics are equal. A time delay of $0.8s$ is identified, which corresponds to 4 sampling intervals. The quadrotor hovers at a PWM command of 0.657, corresponding to a thrust acceleration of $T_{\text{c}}^* = 0$. The large scaling factor a_T implies that even small changes of T_{PWM}, result in large accelerations of the quadrotor.

Table 3-1: Identified parameter values for the attitude and thrust dynamics in simulation.

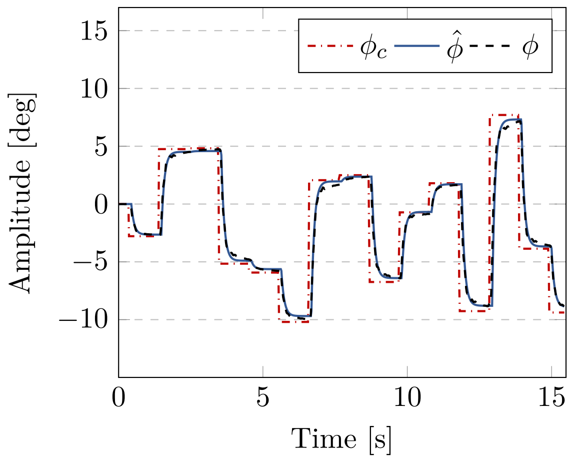
Parameter	$k_\phi = k_\theta$	$\tau_\phi = \tau_\theta$	β [s]	$a_T[\text{s}^2\,\text{m}^{-1}]$	T_h
Value	0.963	0.104	0.08	23.258	0.675

Input response comparison in simulation The response to the input command for the roll dynamics and the behavior of the fitted model are shown in Figure 3-5. Figure 3-5a shows the behavior of the system for one of the training data sets while Figure 3-5b shows the input response to previously unseen validation data. A small offset between the identified model and the peak value of the measured data can be noted, specifically in Figure 3-5a. This could be due to higher-order dynamics in the underlying system that are not captured by the first-order model. Nevertheless, the training data reaches a goodness of fit of 95.12%, the validation data has a fit of 91.16%. These high goodness of fit values indicate that the underlying attitude dynamics are well approximated by the identified first order model. The results for the pitch dynamics are identical and therefore not discussed in more detail.

Thrust dynamics comparison in simulation For the thrust dynamics model in simulation, the input command T_{PWM} between zero and one is scaled according to the identified hovering thrust and linear scaling factor to find the normalized thrust command $\hat{T_{\text{c}}}^*$. It is compared to the measured acceleration of the system $\dot{v}_z$, which corresponds to T_{c}^*, as the roll



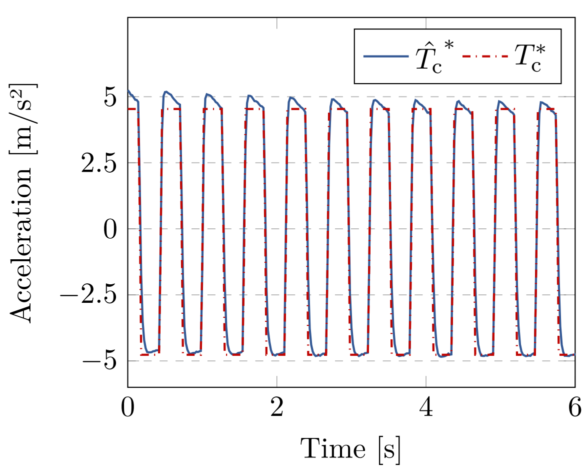
(a) Training data fit: 95.12%



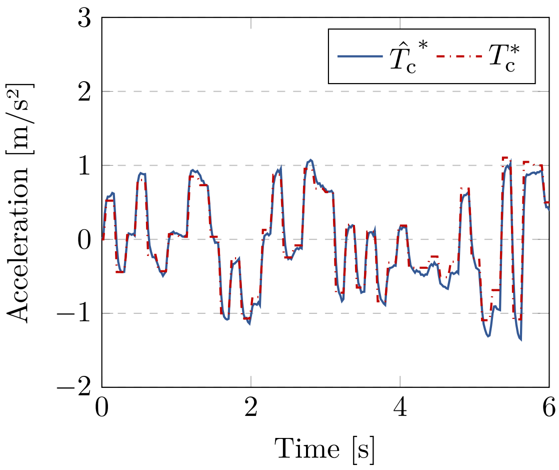
(b) Validation data fit: 91.16%

Figure 3-5: Input response and model fit of the roll dynamics in simulation.

and pitch are zero during these experiments. This comparison is shown in Figure 3-6. Again, the training data in Figure 3-6a is compared with a validation data in Figure 3-6b. The goodness of fit for the simulation is 84.59% for the training data and 57.86% for the validation data. These lower fit values can be explained by the fact that the thrust relation has its own dynamics that are not accounted for by the linear model. Especially when sending more complex thrust commands, as is the case for the validation data, the simple thrust model cannot reflect the actual system behavior as well anymore. However, these fit values are still relatively good, such that the model is expect to perform sufficiently in an Model Predictive Control (MPC)-like controller that does correct for some model mismatch by design.



(a) Training data fit: 84.59%



(b) Validation data fit: 57.86%

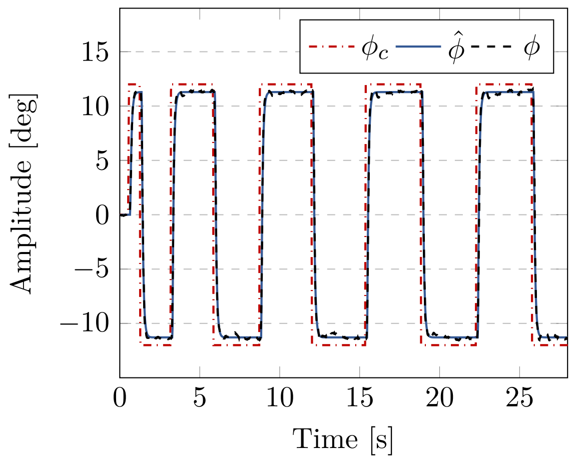
Figure 3-6: Comparison of the PWM commmand which is scaled according to the linear model with the measured acceleration in z-direction.

Identified parameters for the real-world dynamics For the real-world dynamics, the behavior in roll and pitch direction differ slightly, caused by a different weight distribution on the quadrotor due to on-board processors and sensors and therefore also different tuning parameters of the attitude controllers. This is also reflected by the obtained parameter values for roll and pitch dynamics documented in Table 3-2. While the gain constants are almost equal to the identified simulation parameters, the time constants for the real-world system are smaller, meaning a slower system response to the step input, most likely due to higher inertia of the quadrotor. For the real-world scenario, the pitch dynamics are faster than the roll dynamics. Also, the delay is slightly larger with 0.1s, most likely caused by the physical connections.

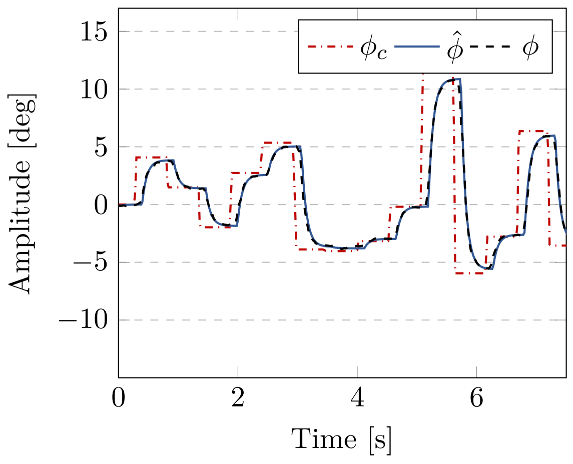
Table 3-2: Identified parameter values for the attitude and thrust dynamics on the physical quadrotor.

Parameter	k_ϕ	k_θ	τ_ϕ	τ_θ	β [s]	a_T [s^2 m^{-1}]	b_T [V^{-1}]	T_h
Value	0.954	0.950	0.065	0.074	0.1 s	0.0329	-0.0456	0.2911

Input response comparison for the real-world dynamics The input responses in roll and pitch direction on the real quadrotor are shown in Figure 3-7 and Figure 3-8. Again, high values are reached for the goodness of fit, with 95.89% and 92.37% for training and validation of the roll dynamics and 95.76% and 87.08% for training and validation of the pitch dynamics. This confirms that the underlying attitude dynamics are well approximated by the identified first order model, even in the real-world scenario.



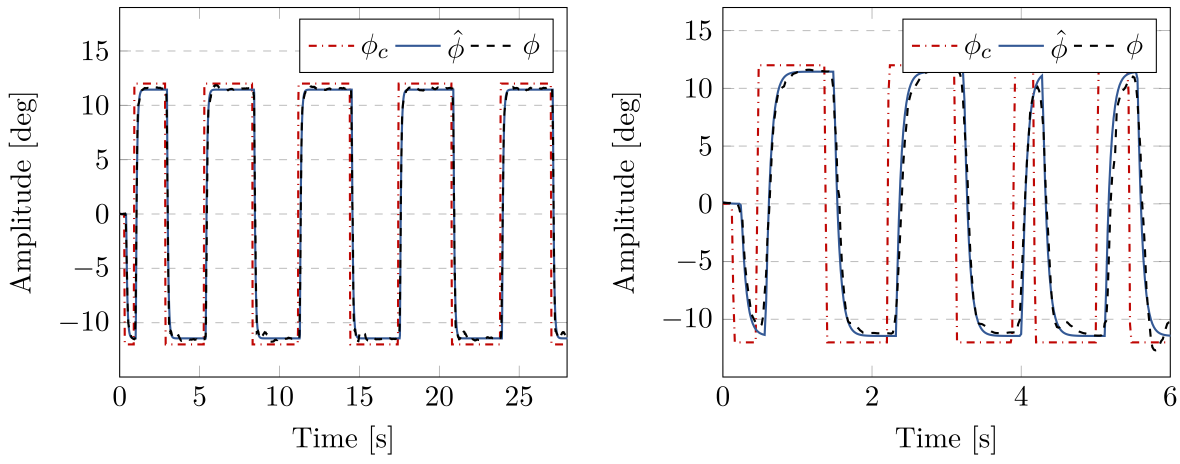
(a) Training data fit: 95.89%



(b) Validation data fit: 92.37%

Figure 3-7: Input response and model fit of the real-world roll dynamics.

Real-world thrust dynamics As the experiments for identifying the thrust dynamics are performed slightly different, the fitted curves are also shown in a different way. Figure 3-9 displays the measured curves of the hover thrust command T_PWM over the battery voltage for different weights, i.e. accelerations. The required thrust command increases as voltage decreases and with an increase in weight or acceleration. In this case $T^*_{\mathrm{c},1}$ indicates the thrust curve without any added weight. For $T^*_{\mathrm{c},2}...T^*_{\mathrm{c},6}$, weight is added in steps of 200g, resulting in a maximum added weight of 1kg. To this behavior, the linear model in Eq. (3-11) is fitted, resulting in the solid curves displayed in Figure 3-9. The goodness of fit of this model is 65.85%.



(a) Training data fit: 95.76%

(b) Validation data fit: 87.08%

Figure 3-8: Input response and model fit of the real-world pitch dynamics.

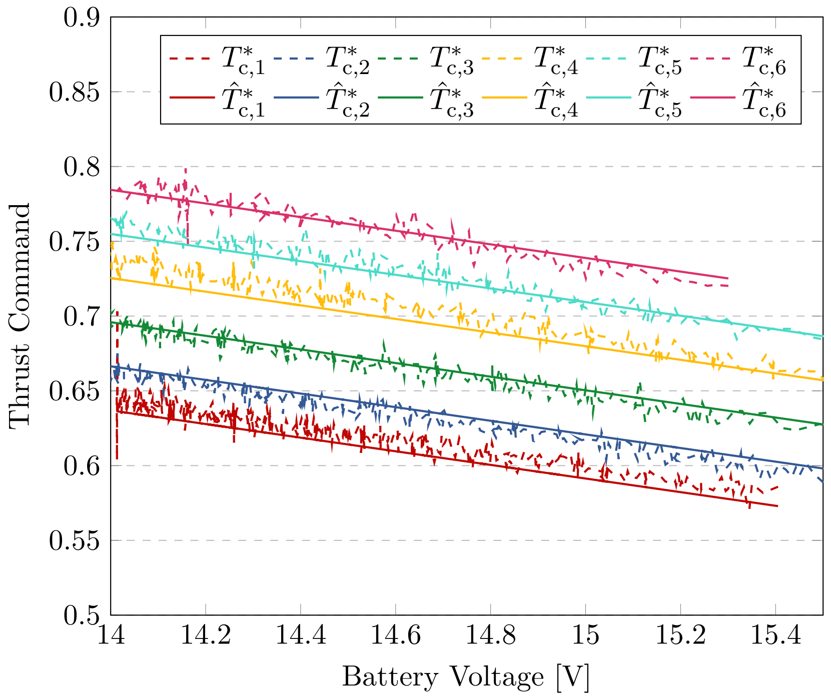


Figure 3-9: Comparison of the real-world thrust dynamics and the fitted linear model for training data. The training data is shown as dashed lines and the fitted model as solid lines. The training data fit is 65.85%.

Real-world thrust dynamics validation However, when comparing the model to validation data that is collected on a different day, the model fails, as the slope of the acceleration curve changes. This behavior is shown in Figure 3-10. This might be caused by different propeller models used on different days due to some braking during a crash, or it might depend on internal computations of the PX4 controller translating the thrust input command into PWM values in another way. This behavior is something that needs to be investigated further. However, as this thesis does not focus on the dynamics in z-direction, the identified model is found good enough, as long as is does not show unstable or undesired behavior during experiments.

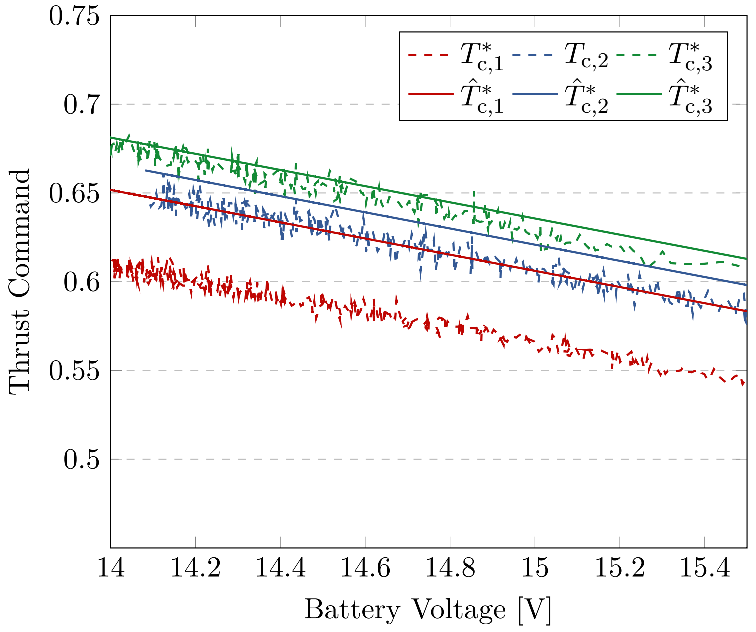


Figure 3-10: Comparison of the real-world thrust dynamics and the fitted linear model for validation data. The validation data is shown as dashed lines and the fitted model as solid lines. The validation data fit is 10.82%.

The performance for all identified models is summarized in Table 3-3.

Table 3-3: Goodness of fit for the conducted experiments to identify the parameters of the roll, pitch and thrust dynamics model.

	Roll		**Pitch**		**Thrust**	
	Simulation	Experiment	Simulation	Experiment	Simulation	Experiment
Training	95.12%	95.89%	95.12 %	95.76%	84.59%	65.85%
Validation	91.16%	92.37%	91.16%	87.08%	57.86%	10.82%

3-3 Summary and discussion

The system dynamics nominal model can be obtained through first-principle modeling, leading to linear and nonlinear approximations of quadrotor dynamics. Focusing on motion planning and position control, the inner-loop attitude controller already available for the drone platform is used, which stabilizes the quadrotor. The inner loop attitude dynamics are approximated by a first-order linear model for the pitch and roll direction of the quadrotor. The coefficients for this linear model are determined by sending step input commands to the quadrotor and fitting the output to the model.

The attitude dynamics model is successfully identified with validation data goodness of fit values over 90% in simulation and 85% on the real quadrotor. This indicates that the attitude dynamics can be accurately approximated with a first-order model, even in real-world environments. The difference between real-world experiments and simulation lies in the time constants, meaning that the physical system response is slower. A time delay of four to five sampling intervals is also identified, though it is neglected in subsequent sections for simplicity in the simplified simulation model. However, in more complex simulations or physical environments, accounting for the time delay could be critical.

Identifying a thrust dynamics model is also necessary, as the input to the controller is a PWM command. This identification is simple in simulation by sending step input PWM commands and fitting a linear model to the measured acceleration. The results are good, but with lower goodness of fit values compared to the attitude dynamics, as the linear model can't account for the more complex thrust dynamics.

The real-world thrust dynamics identification results are worse. In these experiments, the quadrotor is loaded with varying weights that can be converted into an acceleration. While a model can be fitted to the training data, it fails for much of the validation data. Further research is necessary to properly validate the thrust model with the physical quadrotor.

Chapter 4

Gaussian Process disturbance map

The nominal model is extended by a Gaussian Process (GP) model to capture the external wind disturbances. Starting from the quadrotor position as input and the external wind disturbances as output, a wind disturbance map is trained. The background of GP's is first explained before the details of training the GP map are covered. This includes an optimal experimental design and choosing the correct hyperparameters for training. Specifically, the wind disturbance map is trained for two types of scenarios that will be used in the Model Predictive Contouring Control (MPCC) motion planner and controller.

4-1 Gaussian Process Regression

The goal of Gaussian Process Regression (GPR) [5] is to obtain an approximation of the nonlinear function that describes the behavior of wind disturbances with uncertainty. This section introduces the mathematical foundation of GP, including how it is used to train and predict the nonlinear model. It then extends to the idea of sparse GP to reduce computational costs for the model to be used in the MPCC controller, before going into the specific GP disturbance model used in this research.

4-1-1 Mathematical background

The basic building block of the GP is the Gaussian distribution. Particularly, the multivariate Gaussian distribution, where each random variable is distributed normally and the joint probability is also Gaussian. Assuming that the function values of the nonlinear function $f(\mathbf{x})$ follow a multivariate Gaussian distribution, GP regression uses a Bayesian approach to determine a predictive distribution of the function value $f(\mathbf{x}_*)$ at unseen test locations $\mathbf{x}_*$ [5]. The Bayesian approach works by specifying a prior on the nonlinear function and shifting probabilities based on the observed data.

Gaussian Process prior Prior knowledge can be incorporated into the GP through the selection of its mean $\mu(\mathbf{x})$ and covariance or kernel function $k(\mathbf{x}, \mathbf{x}')$:

$$\mu(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \tag{4-1a}$$
$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))\left(f(\mathbf{x}') - m(\mathbf{x}')\right)\right]. \tag{4-1b}$$

such that the nonlinear function is described by a GP, according to:

$$f(\mathbf{x}) \sim \mathcal{GP}\left(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right). \tag{4-2}$$

Dataset To train the nonlinear function, a collection of inputs and respective outputs is required. The dataset $\mathcal{D}$ of n observations, $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$, is assumed to be generated according to:

$$y_i = f(\mathbf{x}_i) + \varepsilon_i \tag{4-3}$$

where $f : \mathbb{R}^D \to \mathbb{R}$ is the nonlinear function to be trained and $\varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ is Gaussian measurement noise present in the observations with zero mean and variance σ_ε [50]. Noise in the data can be added to the covariance function. The prior on the measured function values is then defined as $p(\mathbf{y} \mid X) = \mathcal{N}(0, K(X,X) + \sigma_\varepsilon^2 I)$. Here, $K(X,X)$ denotes the $n \times n$ matrix of the covariance evaluated at pairs of the input points. From the GP prior, the available system data and the desired test locations, one can write the joint distribution of the observed values $\mathbf{y}$ and function values $\mathbf{f}_*$ as:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) + \sigma_\varepsilon^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right). \tag{4-4}$$

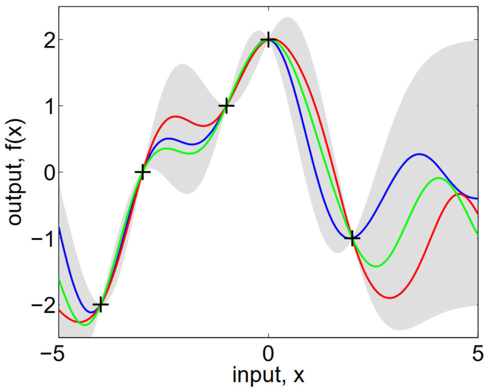


Figure 4-1: GP posterior distribution conditioned on five, noise free observations. The grey area represents the mean $\pm$ two times the standard deviation. In colour are three random samples drawn from the posterior distribution [5].

Inference Conditioning the joint Gaussian prior distribution on the observations, one can derive the posterior distribution:

$$p(\mathbf{f}_* \mid X, \mathbf{y}, X_*) \sim \mathcal{N}(\boldsymbol{\mu}_*, \Sigma_*), \text{ where} \tag{4-5a}$$
$$\boldsymbol{\mu}_* = K(X_*, X)\left[K(X,X) + \sigma_\varepsilon^2 I\right]^{-1} \mathbf{y}, \tag{4-5b}$$
$$\Sigma_* = K(X_*, X_*) - K(X_*, X)\left[K(X,X) + \sigma_\varepsilon^2 I\right]^{-1} K(X, X_*). \tag{4-5c}$$

In the case that there is only one test point x_* the equations reduce to:

$$\mu_* = \boldsymbol{\mu}(x_*) + \mathbf{k}_*^\top \left(K(X,X) + \sigma_\varepsilon^2 I\right)^{-1} \mathbf{y} \tag{4-6a}$$

$$\Sigma_* = k\left(x_*, x_*\right) - \mathbf{k}_*^\top \left(K(X,X) + \sigma_\varepsilon^2 I\right)^{-1} \mathbf{k}_* \tag{4-6b}$$

with $\mathbf{k}_* = K(x_*, X)$ and $\mathbf{k}_*^T = K(X, x_*)$. A GP posterior distribution for noise-free data is shown in Figure 4-1.

Model selection The mean and covariance function of the GP prior must be chosen to describe the underlying system model and form a critical part of the prediction process. The mean of the GP prior is chosen to be zero. This is typically done if no other information on the prior is available. The covariance is specified by a kernel function. The choice of kernel or covariance function expresses the similarity between function values. It specifies certain properties of the GP such as smoothness or periodicity [51]. For this thesis, two types of kernels are explored. The most commonly used kernel is the Squared Exponential (SE) kernel:

$$k_{\text{SE}}\left(\mathbf{x}, \mathbf{x}'\right) = \sigma_f^2 \exp\left(-\frac{1}{2} d\left(\mathbf{x}, \mathbf{x}'\right)\right) \tag{4-7}$$

where $d\left(\mathbf{x}, \mathbf{x}'\right) = \sum_{d=1}^{D} \left(x_d - x_d'\right)^2 l_d^{-2}$ is the scaled, squared Euclidean distance. Another relevant kernel is the Matern kernel:

$$k_{\text{MAT}}\left(\mathbf{x}, \mathbf{x}'\right) = \sigma_f^2 \frac{1}{\Gamma(\nu) 2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{l} d\left(\mathbf{x}, \mathbf{x}'\right)\right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}}{l} d\left(\mathbf{x}, \mathbf{x}'\right)\right) \tag{4-8}$$

where $K_\nu(\cdot)$ is a modified Bessel function and $\Gamma(\cdot)$ is the gamma function. The class of Matern kernels is a generalization of the SE kernel.

Model parameters Each kernel function has free parameters which define the properties of the GP. Common parameters of the SE and Matern kernel are the horizontal length scale for each input dimension l_d and the output variance σ_f. The horizontal length scale determines how correlated neighboring inputs are. The Matern kernel moreover has an additional parameter ν which controls the smoothness of the resulting function. The smaller ν, the less smooth the approximated function is. As $\nu \to \infty$, the kernel becomes equivalent to the SE kernel. When $\nu = 1/2$, the Matern kernel becomes identical to the absolute exponential kernel. Important intermediate values are $\nu = 1.5$ and $\nu = 2.5$. The output variance σ_f determines the average distance of the function away from its mean. In addition, the noise covariance σ_ε can be considered as a parameter of the kernel. The parameters do not have to be determined beforehand but are trained as part of the GP prior using the measured data $\mathcal{D} = \{(X, \mathbf{y})\}$.

Model parameter training The model parameters are found by maximizing the marginal likelihood (or evidence) of the prior, which is conditioned on the parameters $\boldsymbol{\theta}$ through the kernel function $K_\theta = K(\mathbf{x}, \mathbf{x})(\boldsymbol{\theta})$:

$$\mathcal{L} = \log p(\mathbf{y} \mid X, \boldsymbol{\theta}) = -\frac{1}{2} \mathbf{y}^\top \left(K_\theta + \sigma_\varepsilon^2 I\right)^{-1} \mathbf{y} - \frac{1}{2} \log \left|K_\theta + \sigma_\varepsilon^2 I\right| - \frac{n}{2} \log(2\pi). \tag{4-9}$$

The parameter vector:

$$\hat{\boldsymbol{\theta}} \in \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{y} \mid X, \boldsymbol{\theta}) \tag{4-10}$$

is the maximum likelihood estimate of the parameters, which trades off model complexity versus the data fit [50]. Maximizing the likelihood is a non-convex, nonlinear optimization problem which can be solved by hand or using available libraries such as GPyTorch [52] or sklearn [53].

Computational complexity Training the GP with a training set of size n requires $\mathcal{O}\left(n^3\right)$ per gradient step due to the inversion of the Kernel matrix $K(X,X)$ which makes training very expensive [5]. To make predictions, the inversion of the kernel matrix demands most operations and results in a computational complexity of $\mathcal{O}\left(n^3\right)$ per prediction. Cholesky factorization can help to reduce the computational complexity of the matrix inversion to $\mathcal{O}\left(n^3/6\right)$. Sparse GP's can further reduce the complexity, which motivates the use of it.

4-1-2 Sparse Gaussian Process Regression

To achieve a real-time feasible MPCC controller, the GP model has to be fast in making predictions. Sparse GP's reduce the complexity of training the GP model and making predictions. Instead of using all n training inputs, a smaller training set of size m, also referred to as inducing inputs is used to form the covariance matrix $K(X,X)$. The complexity of inverting the covariance matrix is thereby reduced to $\mathcal{O}\left(nm^2\right)$ - a significant speed up.

Choosing inducing inputs Using fewer inputs to train and predict GP's also reduces the accuracy and expressiveness of the GP. Selecting the inducing inputs can be challenging. If it is selected greedily from the available data, it becomes difficult to include all the necessary information. In this thesis, a variational approach [54] is used which is a popular method to train sparse GP's as it optimally selects the inducing points.

Variational Inference Using a variational approach, the inducing points are note selected from the available dataset, but insetad are pseudo data points $\mathbf{u}$, that are optimized for in the course of training. The posterior distribution in Eq. (4-5a) is approximate as a multivariate Gaussian $q(\mathbf{u})$ over the inducing variables $\mathbf{u}$ with variance $\boldsymbol{\mu}_\mathbf{u}$ and covariance $\Sigma_\mathbf{u}$. The inducing points, mean and variance are the variational parameters which are selected by minimizing the Kullback-Leibler divergence between the variational distribution and the exact posterior GP. The Kullback-Leibler divergence is minimized using stochastic gradient descent methods, where the gradients are estimated using samples from the approximating distribution. This is done together with the minimization of the marginal likelihood in Eq. (4-10) as part of the model parameter training.

Practical use of sparse GP Some parametrization tricks come in handy for practical applications of sparse GP prediction. Instead of defining the random variable $q(\mathbf{u})$, one can define the inducing variable $\mathbf{w} := \operatorname{chol} K_\mathbf{u}^{-1}\mathbf{v}$, where $\mathbf{v} = \mathbf{u} - \boldsymbol{\mu}_\mathbf{u}$ and chol refers to the Cholesky decomposition. The variational parameters are the mean vector $\boldsymbol{\mu}_\mathbf{w}$ and covariance matrix $\Sigma_\mathbf{w}$. From this, the posterior sparse GP formulation can be computed as follows:

$$f(\mathbf{x}) \sim \mathcal{GP}\left(\boldsymbol{\mu}(\mathbf{x}) + \mathbf{k}_{\mathbf{ux}} L_\mathbf{u}^{-\top} \boldsymbol{\mu}_\mathbf{w}, k\left(\mathbf{x},\mathbf{x}\right) - \mathbf{k}_{\mathbf{xu}} L_\mathbf{u}^{-\top}\left(I - \Sigma_\mathbf{w}\right) L_\mathbf{u}^{-1}\mathbf{k}_{\mathbf{ux}}\right) \tag{4-11}$$

denoting $\operatorname{chol}\left(K_\mathbf{u}\right)$ as $L_\mathbf{u}$ and making use of the identity $K_\mathbf{u}^{-1} = L_\mathbf{u}^{-\top} L_\mathbf{u}^{-1}$. This parametrization of the inducing variables can facilitate the optimization and speed up the prediction. The method has been implemented in GPyTorch [52] which provides a library for sparse GP inference.

4-1-3 Disturbance model

After introducing the mathematical background on GP, this section presents the specifics of how sparse GPs are employed in this study to model wind disturbances affecting the quadrotor in the x- and y-directions. The input and output data necessary for this approach, as well as the technique for handling a multivariate training target, are described.

Training dataset The quadrotor does not have any on-board sensors to directly measure the wind disturbance. Instead, the wind will be estimated using available state information. Starting from a nominal quadrotor model and assuming that there are no other disturbances acting on the quadrotor, the momentary wind is attributed to the difference between the predicted velocity $\hat{\mathbf{v}}_k$ by the nominal model and the measured velocity at the same time instance $\mathbf{v}_k$. The disturbance at the quadrotor position $\mathbf{p}_k = [x_k, y_k]^T$ can be calculated as follows:

$$d_{v_x,k}(\mathbf{p}_k) = \hat{v}_{x,k} - v_{x,k} \tag{4-12a}$$
$$d_{v_y,k}(\mathbf{p}_k) = \hat{v}_{y,k} - v_{y,k}. \tag{4-12b}$$

With this information, the GP model is trained for the disturbance vector $\mathbf{d}(\mathbf{p}) = [d_{v_x}, d_{v_y}]^T$.

Multivariate GP model Classic GP's, as described in section 4-1, are only capable of modelling nonlinear dynamics with one output. As the disturbance output of the GP model is a multivariate target, i.e. $\mathbf{d} \in \mathbb{R}^2$, each output is trained independently. This is a simplification of the problem, assuming that the two wind directions are computationally independent, resulting in a diagonal covariance matrix. In a real-world scenario, the wind directions may be correlated, leading to information loss when training models independently. In that case, it might be beneficial to use multi-output Gaussian processes [55]. However, it also makes GP training more complex. Therefore, the performance of independent GP's will be tested first in this research.

Independent GP model Two GP's are trained for the two disturbance directions using the same quadrotor position as input, but only one of the disturbances as training target. The data collection method and hyperparameters of the training discussed hereafter will be the same. However, training the GP with different training targets will result in different trained model parameters. The resulting disturbance vector is:

$$\mathbf{d}(\mathbf{p}) = \begin{bmatrix} d_{v_x}(\mathbf{p}) \sim \mathcal{N}\left(\mu^{d_{v_x}}(\mathbf{p}), \Sigma^{d_{v_x}}(\mathbf{p})\right) \\ d_{v_y}(\mathbf{p}) \sim \mathcal{N}\left(\mu^{d_{v_y}}(\mathbf{p}), \Sigma^{d_{v_y}}(\mathbf{p})\right) \end{bmatrix}. \tag{4-13}$$

The predictive distribution is given by:

$$\boldsymbol{\mu}_d = [\mu_{v_x}, \mu_{v_y}]^\top \tag{4-14a}$$
$$\Sigma_d = \text{diag}\left(\begin{bmatrix} \Sigma^{d_{v_x}} & \Sigma^{d_{v_y}} \end{bmatrix}\right) \tag{4-14b}$$

As the training inputs and training targets are sampled in discrete time, the resulting GP model will also be of discrete nature.

4-2 Data collection

To train the GP disturbance map, data of the wind disturbance in the environment has to be collected. This section discusses the experiment setup and generation of the wind force as well as an optimal Active Learning (AL) strategy to explore the environment.

4-2-1 Active Learning

Active Learning [56] is an iterative algorithm to query data points in an optimal way. It is very useful for experiment design, especially when the training samples are limited because they are expensive, time-consuming or difficult to obtain. The key components of the AL algorithm are the model, the uncertainty measure and the querying strategy.

Active Learning strategy The advantage of using AL with GP's is that the GP model directly provides the uncertainty. The query strategy is chosen such that the most uncertain points are sampled at each iteration. Afterwards, the GP is retrained with the newly obtained data samples. Querying and retraining the model is repeated until the model is found to be accurate enough. The general workflow of AL is shown in Figure 4-2.

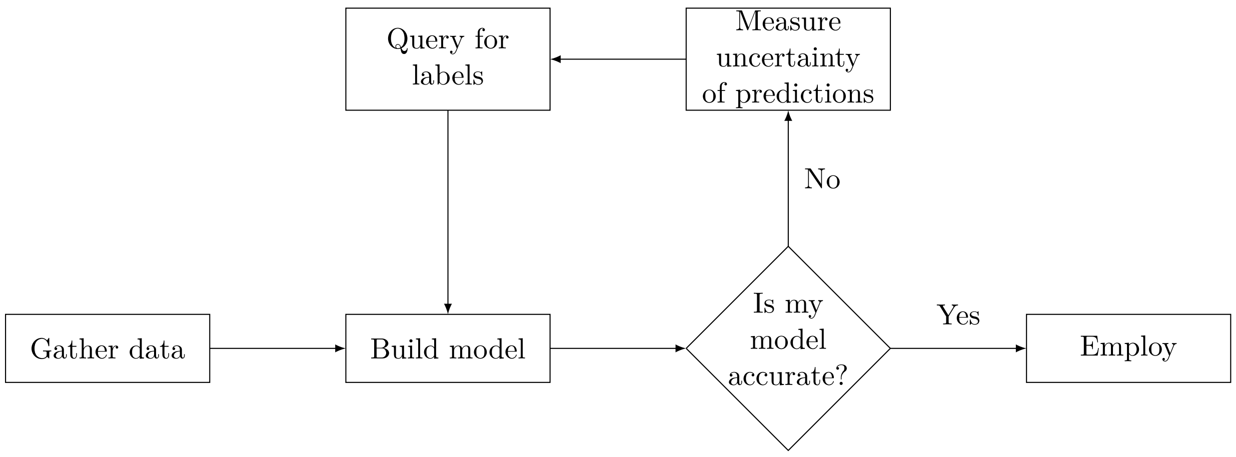


Figure 4-2: Active Learning workflow [56].

4-2-2 Quadrotor environment

The quadrotor is navigated in a windy environment. The wind varies in x- and y-direction and is assumed constant over time, resulting in a spatially varying wind field, that can be learned by the quadrotor.

Simulation environment

The initial experiments are performed in a simple simulation environment, where the quadrotor dynamics perfectly match the derived quadrotor model in Chapter 3, including the identified simulation model parameters in 3-1 and a normalized drag coefficient of $k_D^* = 0.02$. The environment is programmed using a Python client library for Robot Operating System (ROS), to keep the quadrotor control in the ROS environment.

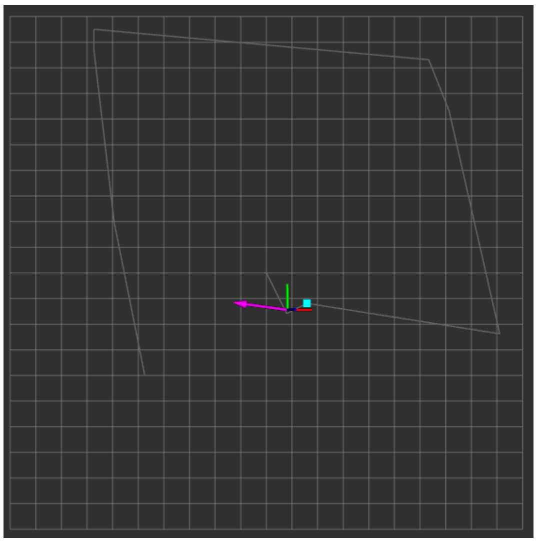


Figure 4-3: Simplified simulation environment of the quadrotor, visualized in RViZ. The quadrotor is represented by the 3D coordinate system indicating it's position and orientation, the pink arrow shows the wind acting on the quadrotor at any given position, the grey path shows the reference path and the cyan point indicates the next reference point.

Dynamics update The quadrotor is controlled by sending attitude commands, i.e. $\mathbf{u}_c = \left[\phi_c, \theta_c, \dot{\psi}_c, T_c\right]^T$. The simple simulation environment uses Runge-Kutta 4th (RK4) method to integrate the quadrotor dynamics and returns the odometry data of the quadrotor. This happens at a rate of 20Hz.

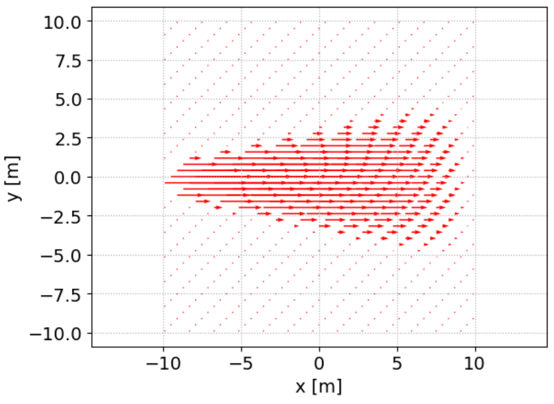
Simplified simulation environment The quadrotor is assumed to move in a grid of $20\,\mathrm{m} \times 20\,\mathrm{m}$, in which the wind acts on the quadrotor. During exploration, there are no obstacles present in the environment. The control and dynamics in z-direction are kept at the same altitude, reducing the problem to 2.5 dimensions. The environment and quadrotor are visualized using RViz, a visualization software tool for ROS. The simulation scenario is shown in Figure 4-3.

Generated wind fields

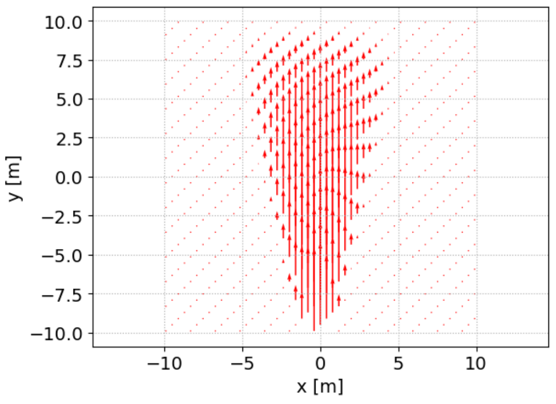
The wind field is simulated by adding a force to the quadrotor dynamics in v_x and v_y, changing the speed of the quadrotor.

Constant wind The simple simulation supports adding a constant wind in both directions, which is equal across the entire environment. That force can moreover be altered with Gaussian noise, simulating randomness of the wind.

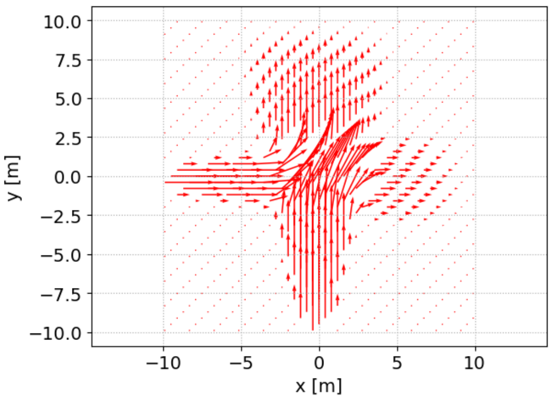
Wind plugin Next to that, a self-written wind plugin is added to the simulation that allows adding any custom, static wind field to the simulation. The wind is defined in terms of a grid, which is given by the resolution of the grid and dimension of the environment. The wind force is defined at each grid point and interpolated using a 2D-interpolation scheme to



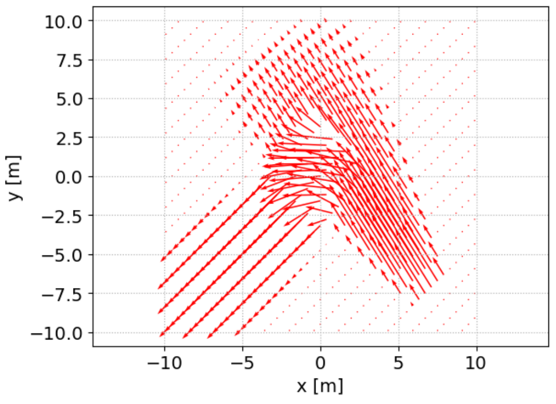
(a) Fan in x-direction with an initial speed of $5\,\mathrm{m\,s^{-1}}$.



(b) Fan in y-direction with an initial speed of $5\,\mathrm{m\,s^{-1}}$.



(c) Crossing fans in x- and y-direction.



(d) Crossing fans in random directions.

Figure 4-4: Generated wind fields.

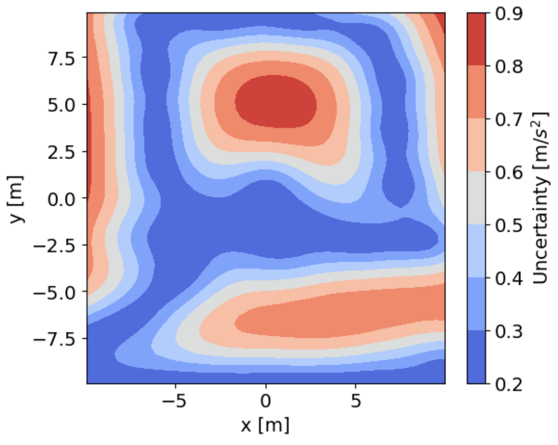
calculate the force at any point in the environment. For now, the wind fields are generated in 2D, but the plugin can easily be extended to a 3D-environment.

Custom wind fields The wind plugin is used to generate wind fields that can easily be replicated in a lab environment. The wind fields are created to resemble fans distributed in the environment, with a high velocity close to the fan and decreasing velocity with a spread of the wind field further away from the fan. Simple cases are generated where only one fan points in the x- or y-direction, but more complex cases are also experimented with where two fans are crossing. Again, noise can be added to the scenarios. A visualization of the different wind fields is shown in Figure 4-4. More information on the wind plugin and generation of the wind fields can be found in Appendix A.

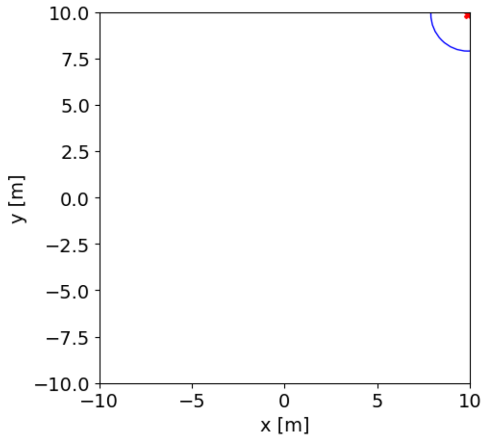
4-2-3 Exploration of the environment

To train the GP model, data on the wind disturbance map has to be gathered. One way of doing this is by maneuvering the quadrotor inside the environment with a predefined path. However, this is either very time-expensive or does not guarantee that all relevant points needed to train an accurate GP model are visited. Instead, the exploration experiment is designed in an optimal way using the concept of AL.

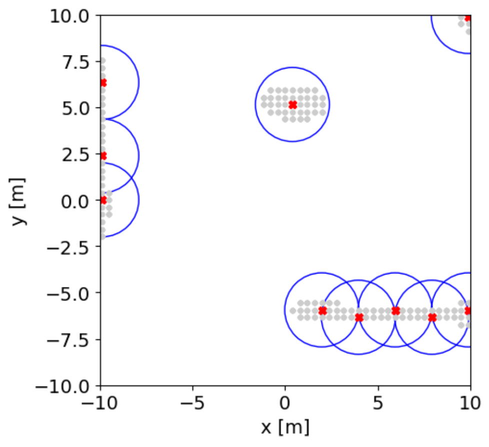
Active Learning experiment As the active learning experiment needs an initial model, N points in the environment are randomly selected and explored. An initial GP model is trained to describe the disturbance map. From this model, the points with the maximum uncertainty are determined from a predefined grid. As the points of maximum uncertainty tend to appear in lager clusters in the map, additionally a threshold radius $r=$ 2m is defined. If a point is selected as most uncertain point, all other points within the radius of that point are discarded to avoid exploring only in one corner of the environment. In total, another N uncertain points are selected this way. At each iteration the selected points are visited in an optimal way by solving the Traveling Salesman Problem (TSP). Details on the implementation of the TSP can be found in Appendix B. Sampling the most uncertain points and retraining the model is repeated until the maximum uncertainty is lower than a specified threshold $\bar{\sigma}$.



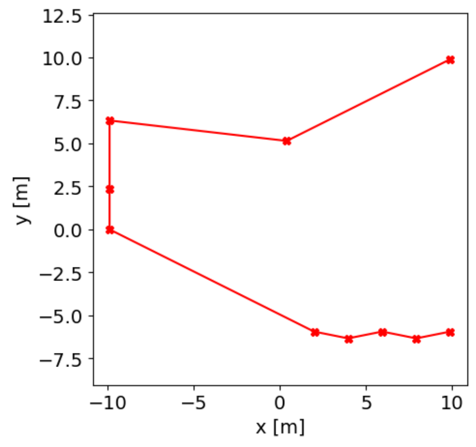
(a) Uncertainty of the disturbance map after two iterations.



(b) The most uncertain data point and threshold radius.



(c) Sampled data points and threshold radii. Discarded points are shown in gray.



(d) Generated path connecting data points solving the TSP.

Figure 4-5: Steps of the Active Learning experiment.

Number of points visited The number of points sampled at each iteration is selected by conducting a few experiments and tracking the time of the quadrotor path and training results. For small N, the quadrotor path is too random, resulting in long experiment duration. On the other hand, if N is too large, training of the GP is repeated not frequent enough, such that the points of maximum uncertainty are no longer relevant to explore while conducting the experiments. A number of $N = 10$ is found to be a good intermediate value.

Active Learning illustration The procedure of the AL experiment design is shown in Figure 4-5. Figure 4-5a shows the uncertainty disturbance map of the entire grid after two iterations, such that the quadrotor has already collected some data. The certainty is high in areas where disturbance data is collected, while uncertainty is high in areas that have not yet been visited. From the information on the uncertainty, the most uncertain point is sampled as shown in Figure 4-5b. It also illustrates the threshold radius. Within the threshold radius all subsequent sampled points are discarded, until the next most uncertain point outside of that radius is found. Repeating this for ten points results in Figure 4-5c. The optimal sequence of points computed with the TSP is shown in Figure 4-5d, which is sent as a path reference to the quadrotor.

Quadrotor control and data collection The quadrotor is navigated in the environment using a PID position controller. The data is collected along the entire quadrotor path at a frequency of 20Hz. As the quadrotor is travelling at a low speed through the environment, sampling the data at a frequency of 20Hz results in large datasets with data points sampled very close to each other. For GP training, the data is therefore down-sampled to 4Hz to avoid large datasets and training times, without a loss of information on the disturbance map.

4-3 Training the Gaussian Process disturbance map

With the collected data samples, the GP model can be trained to find the parameters of the GP kernel function. In addition, different types of kernel functions are explored and the hyperparameters of the GP training algorithm are selected. During training, the disturbance vector $\mathbf{d}(\mathbf{p})$ is scaled by the sampling interval T_s, for easier comparison with the original wind fields.

4-3-1 Model and hyperparameters

The GP is trained to find the model parameters using stochastic variational GPR in GPyTorch [52], which is the sparse GP regression method explained in section 4-1-2 with the Cholesky form of the variational distribution. As explained there, the sparse GP regression method uses stochastic optimization techniques. The optimization loops over a number of training iterations (epochs) and minibatches of the given data, maximizing the variational Evidence-Lower Bound (ELBO) between the variational distribution and the exact posterior GP. Next to the model parameters, that can be found during training, relevant hyperparameters of the training need to be selected beforehand. Those are the number of inducing points, epoch and batch size and the type of kernel function.

Number of inducing points The sparse GP is initialized with 30 inducing points. This number is selected based on related literature (see. e.g. [6], [44], [43]) and is a balance between the speed of the prediction and the accuracy of the GP.

Epoch and batch size Choosing the optimal combination of epochs and batch size is a crucial step in training the GP dynamics model and precedes the model parameter optimization.

Kernel function The GP kernel function must also be selected as the GP prior. Two kernel functions, the SE kernel and Matern kernel, are considered to vary kernel smoothness.

4-3-2 Hyperparameter selection

The training hyperparameters that need to be explored are the epoch, batch size and type of kernel function. To find the optimal model parameters, the model is trained with the data collected during the AL experiment. The results are then validated by comparing the predicted wind field over the entire grid with the original wind field, to make sure that the trained model generalizes to previously unseen data points.

Epoch and Batch size

To investigate the influence of the epoch and batch size on the accuracy of the trained model, grid search is performed. The data used for this investigation is generated by exploring the environment according to the experiment design in 4-2-3 for a total of four iterations. This is done for the SE and Matern kernel for a simple scenario with the wind generated from a fan pointing in x-direction as shown in Figure 4-4a. As it is found during the experiments that the choice of kernel function minimally influences the optimal epoch and batch size, the results here are discussed for the SE kernel.

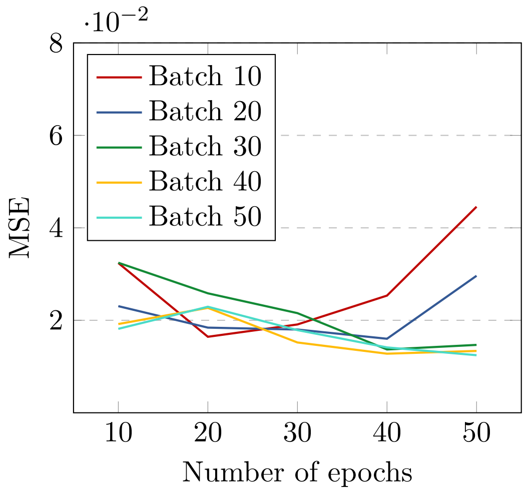
Grid search During GP training the epoch and batch size are each varied in steps of 10, between 10 and 50 and each pair of epoch and batch size is combined for training. The result of the training is compared by computing the Mean Squared Error (MSE) of the prediction of the trained GP model and the ground truth wind field over the entire environment. Simultaneously, the training time is tracked and compared for the combinations of epoch and batch size. The results are shown in Figure 4-6.

Trade-off between accuracy and training times The most accurate results with a MSE of less than 0.02 can be achieved for a large number of epochs. However, if the size of the batch is too small, the quality of the prediction decreases again. In that case, the GP is most likely overfitting the data. While the training time scales almost linearly with the number of epochs and size of the data batch, training already becomes accurate with a batch size around 40, if the number of epochs is not chosen too low. It is found that the optimal combination of accuracy and training time is reached at 20 epochs.

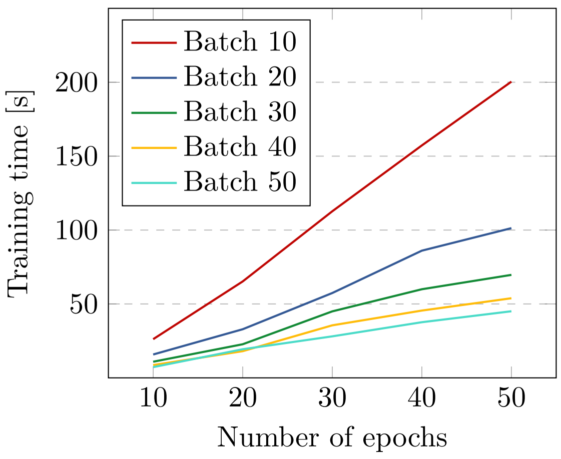
Size of training data and batch Performing this step for less AL experiment iterations, i.e. with less data, it is furthermore found that the optimal batch size depends on the size of the training data. If the training dataset is smaller, the batch size has to be lowered as well in order to achieve similar results for the accuracy. The resulting choice for the batch size is therefore $i \times 10$, where i is the number of experiment iterations. The number of training epochs is fixed at 20.

Kernel function

To find the optimal kernel function the SE kernel is compared to a Matern kernel with a medium smoothness parameter of $\nu = 1.5$. The SE kernel function is chosen as it is the most



(a) MSE.



(b) Training times.

Figure 4-6: Mean squared error and training times for different combinations of epoch and batch size.

commonly used kernel functions that works for most applications. It is compared against the Matern kernel, which is less smooth and might fit complex wind fields better than the smooth SE kernel. The comparison is made for the more complex wind field with two fans crossing as shown in Figure 4-4c, assuming that the results will generalize to simpler wind fields. The two kernels are compared in terms of the generated path, the MSE and the mean and maximum uncertainty of the wind, corresponding the the variance of the trained GP, averaged for both directions. In total, five AL iterations are performed.

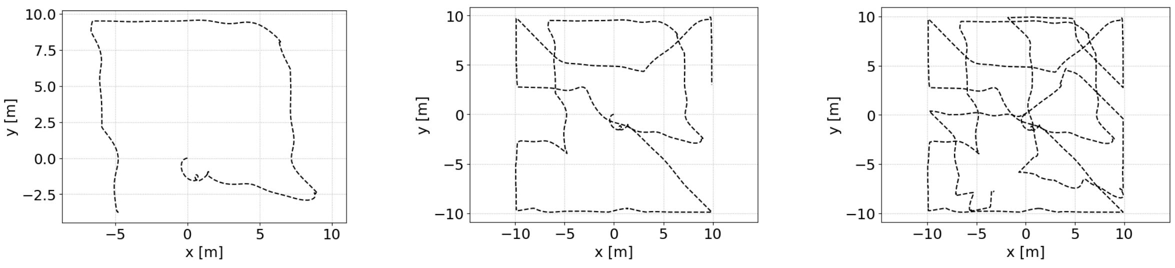


Figure 4-7: Generated path after one, three, and five Active Learning iterations with the squared exponential kernel.

Influence on the data collection Figures 4-7 and 4-8 show the generated path and collected wind data for the SE and Matern kernel, respectively. The generated paths by the two different kernel types do not differ much, such that after five iterations, almost the entire space is covered with similar exploration times.

Accuracy and uncertainty Comparing the MSE shown in Figure 4-9a the uncertainties shown in Figure 4-9b, the training results are similar. The SE kernel is slightly more accurate for more training iterations, but also has a slightly higher uncertainty. Again, there is no significant difference between the two types of kernel functions.

Effect of smoothness on the learning result In general, the SE kernel is smoother than

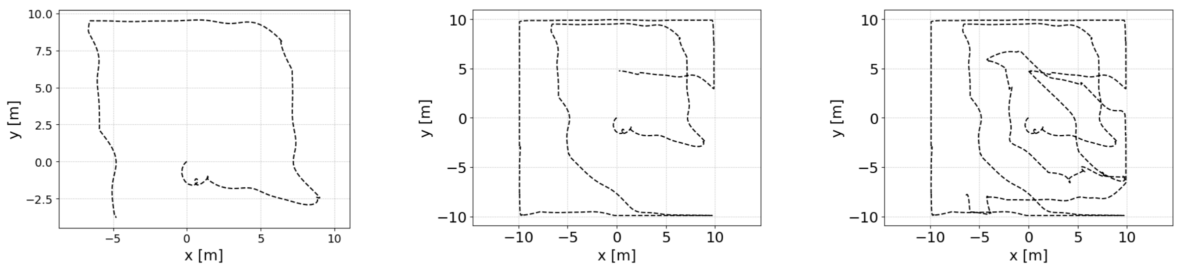
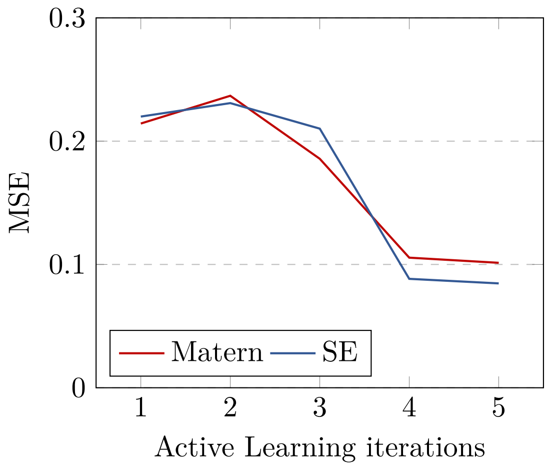
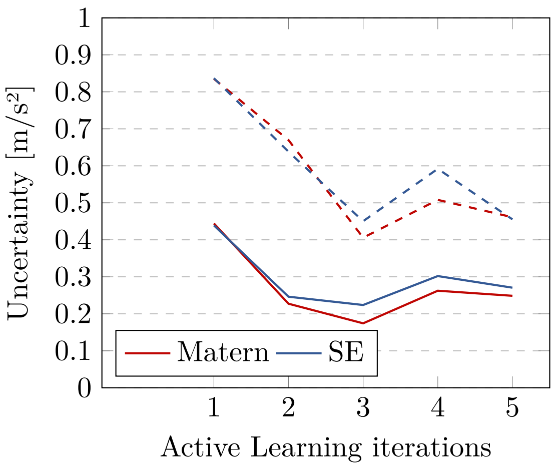


Figure 4-8: Generated path after after one, three, and five Active Learning iterations with the Matern kernel.



(a) MSE over five AL iterations.



(b) Mean (solid) and maximum (dashed) uncertainty over five AL iterations.

Figure 4-9: MSE and uncertainty of the trained GP model for the SE and Matern kernel.

the Matern kernel. It therefore extrapolates the information better into regions where no data is available. Since the generated wind fields in simulation are also smooth, this combination works well for the training data resulting in accurate models with less training data and less uncertainty. However, this might also be a pitfall if the wind fields are less smooth, which could be the case in a real-world scenario. In that case, the SE kernel might extrapolate the data too much not capturing the disturbance model and the less smooth Matern kernel could perform better.

Chosen kernel type The results of the experiments are similar, so the simpler SE kernel is chosen for subsequent hyperparameter training. The Matern kernel should still be considered in case the SE kernel doesn't perform well in real-world conditions where wind fields are likely less smooth.

4-4 Results

After deciding on the disturbance model, experiment setup and the training and model parameters, the GP model can be trained and evaluated in terms of its performance. First, collecting the data using AL is compared to other methods for data collection. Second, the sparse GP approach is compared to training the GP with all available data. Third, the disturbance maps for two types of wind fields are trained, that are going to be used in the following chapters.

4-4-1 Data collection with Active Learning

The data collection method is compared to two other exploration paths. In one scenario, the data is collected by exploring the environment in a structured way, to cover as much area as possible. The resulting collected data is shown in Figure 4-10a. In the second scenario, the environment is explored by flying a lemniscate trajectory, with collected data shown in Figure 4-10b. The collected data using the AL approach with four iterations is shown in Figure 4-10c. All paths are tracked under the same conditions, with the wind acting in both x- and y- directions as shown in Figure 4-4c to ensure the generality of the comparison. The three exploration methods are compared in terms of the exploration time and collected data samples, the training time and the goodness of the prediction in terms of the accuracy and uncertainty. The results are summarized in Table 4-1.

Table 4-1: Comparison of collecting data using the AL exploration method versus predefined paths.

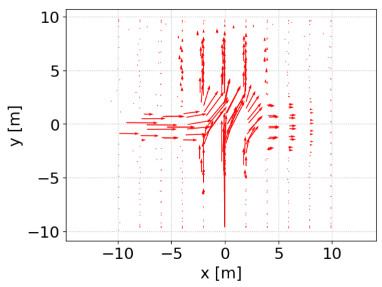
Path	**Structured**	**Lemniscate**	**Active Learning**
Exploration time [s]	471	173	394
Data samples	1827	767	1321
MSE	0.033	0.274	0.031
Mean uncertainty [m/s^2]	0.662	1.072	0.474
Maximum uncertainty [m/s^2]	0.984	1.571	0.722

Numerical comparison of the data collection methods Both the structured and AL path learn the distribution of the wind field well with similar MSE. However, tracking the structured path takes almost one-third more time, also increasing the amount of training data. Moreover, comparing the maximum uncertainty, it is lower for the AL approach as the environment is covered in such a way to minimize the uncertainty. The lemniscate path is tracked faster than the rest, however it covers only part of the environment such that it fails to generate accurate predictions over the entire grid.

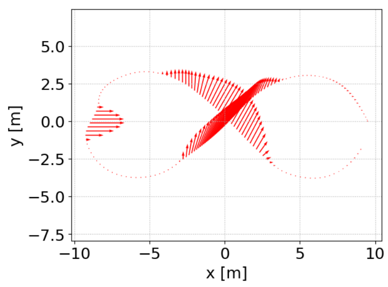
Visual comparison of the data collection methods This is also reflected in the second and third row of Figure 4-10 where the predicted wind fields and uncertainties are shown. The structured path and AL path can reconstruct the wind field over the entire grid, while the prediction for the lemniscate path is only accurate near areas covered by the quadrotor. The uncertainty is large where the environment is not covered by the lemniscate reference path and more evenly distributed for the structured and AL path.

4-4-2 Sparse versus full Gaussian Process model

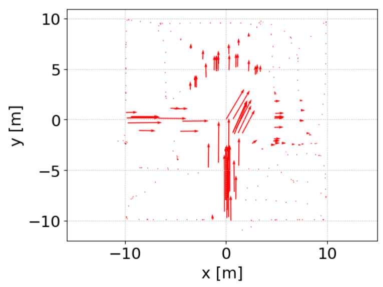
The sparse GP training and prediction is compared to full GP training and prediction in terms of training and prediction time and the accuracy of the model. The reference data



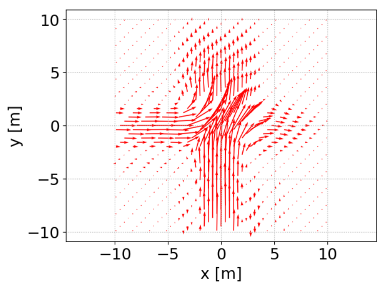
(a) Data collected along the structured path.



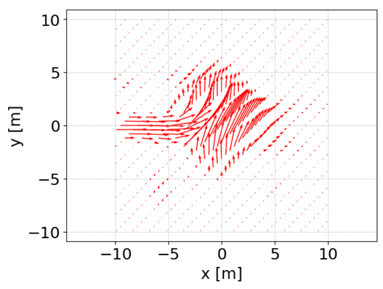
(b) Data collected along the lemniscate path.



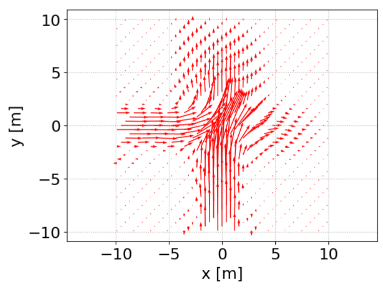
(c) Data collected along the AL path.



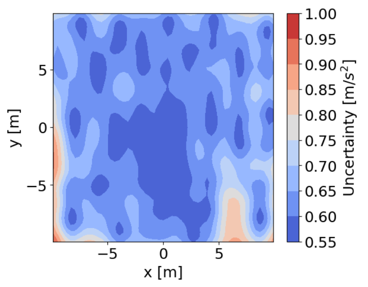
(d) Wind prediction from structured path data.



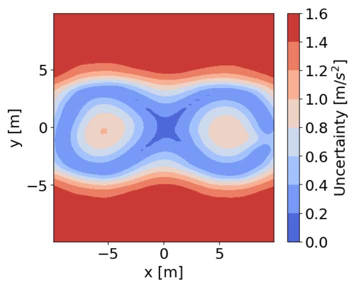
(e) Wind prediction from lemniscate path data.



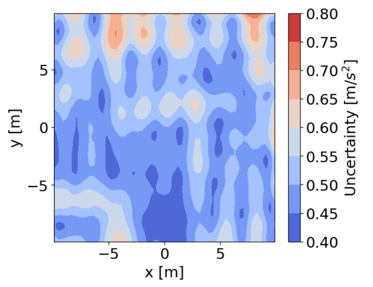
(f) Wind prediction from AL path data.



(g) Uncertainty of the prediction with structured path data.



(h) Uncertainty of the prediction with lemniscate path data.



(i) Uncertainty of the prediction with AL path data.

Figure 4-10: The collected data, predictions of wind fields, and predicted uncertainties for the structured path, lemniscate path, and Active Learning path.

for this comparison is generated with a simple wind field pointing in x-direction for four AL iterations. This simple scenario is chosen as the superiority of sparse GP is already visible here and generalizes to more complex scenarios.

Full versus sparse GP The sparse GP approach is compared to the full GP approach in terms of the accuracy and uncertainty, and the training and prediction times. The results are summarized in Table 4-2. The results on the accuracy and uncertainty show that the performance of the GP prediction is slightly worse when using the sparse approach. The MSE is about five times higher and the uncertainty is doubled. This is to be expected, as

Table 4-2: Comparison of full versus sparse GP.

GP	Full	Sparse
MSE	0.0045	0.029
Mean uncertainty [m/s^2]	0.112	0.221
Maximum uncertainty [m/s^2]	0.248	0.501
Training time [s]	5.6	15
Prediction time [s]	9.224	0.059

the number of data points used to express the model is significantly decreased. However, despite the lower MSE, the model is still able to replicate the wind fields for the largest part. This will also be shown when discussing the final trained wind fields in section 4-4-3. Also, the information on the higher uncertainty will be accounted for in the controller design. The advantage of using sparse GP for this application is clear when comparing training and prediction times, as prediction is around 150 times faster. While the training time is longer due to the stochastic training and learning the predictive distribution along with the kernel parameters, it is not a concern as the GP parameters are not trained online. The prediction time will be even faster for a single data point.

4-4-3 Trained disturbance models

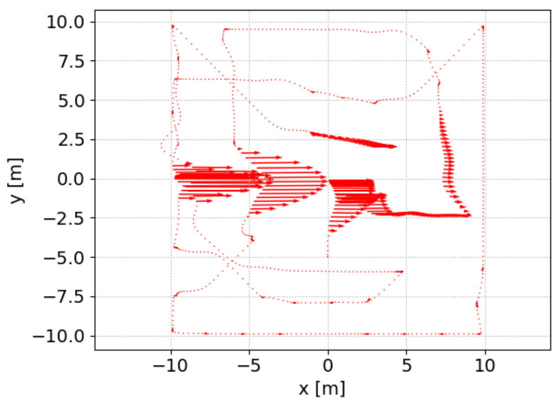
Two disturbance models are trained to be used in subsequent sections. The two scenarios are the wind field pointing in x-direction (see Figure 4-4a) and the crossing wind fields (see Figure 4-4d). Summarizing the findings from the previous sections, the training conditions used to train the disturbance models are listed in Table 4-3. The wind fields are trained using a sparse GP with 30 inducing points and a SE kernel. For both wind fields, four AL iterations are preformed. The generated paths and collected data are shown in the top row of Figure 4-11.

Table 4-3: Final training conditions.

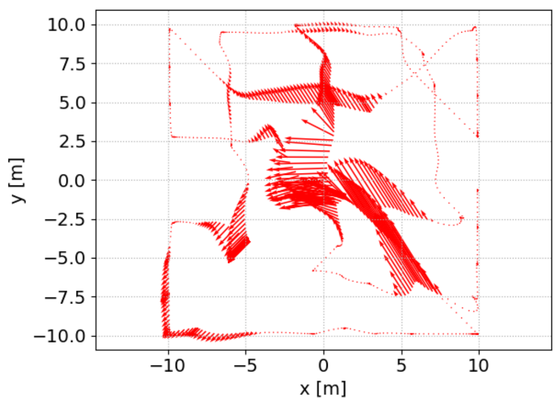
Training hyperparameter	Setting
Kernel	SE
Number of inducing points	30
Number of AL iterations	4
Epochs	20
Batch size	40

Trained model parameters The final trained model parameters of the SE kernel in Eq. 4-7 are summarized in Table 4-4. The length scales of the wind field in x are generally larger, indicating that the wind field is smoother than the wind field with crossing fans. The output variance in y-direction is very small for the wind field pointing only in x-direction, while all other output variances are in a similar range.

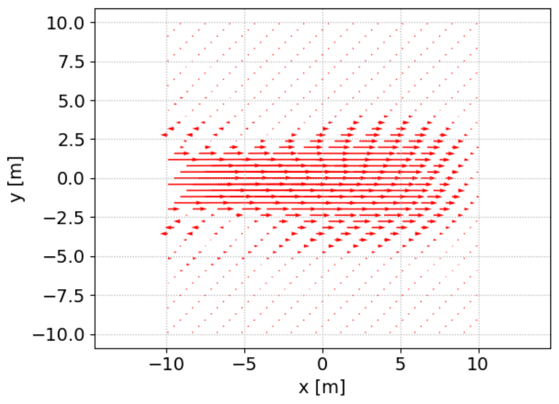
Performance metrics The final performance metrics of the trained disturbance models is summarized in Table 4-5. The GP model is able to train the wind field with an MSE of



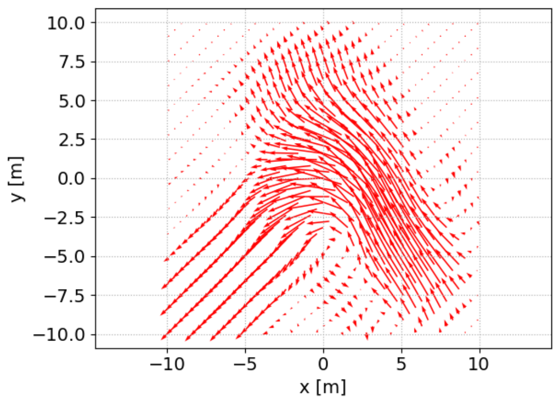
(a) Collected wind disturbance data for the wind field in x-direction.



(b) Collected wind disturbance data for the wind field with crossing fans.



(c) Trained wind disturbance map for the wind field in x-direction.



(d) Trained wind disturbance map for the wind field with crossing fans.

Figure 4-11: Collected training data and trained disturbance maps.

Table 4-4: Final model parameters of the two trained wind disturbance models.

Wind field	**Wind in x-direction**		**Crossing fans**	
Disturbance direction	d_{v_x}	d_{v_y}	d_{v_x}	d_{v_y}
Length scale l_1[m]	7.875	7.93	2.27	1.675
Length scale l_2 [m]	1.009	11.21	2.66	2.597
Output covariance σ_f^2[m^2]	0.096	2.8e-6	0.050	0.076
Noise covariance σ_ε^2 [m^2]	0.032	2.4e-4	0.037	0.043

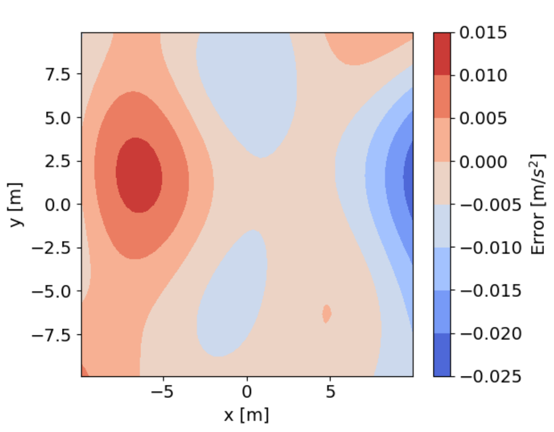
0.019 and 0.060, respectively. With similar training times, the accuracy is lower for the more complex wind field. Also, the uncertainties are higher. However, both trained models can capture most of the underlying wind dynamics which is also visible in the bottom row of Figure 4-11 which shows the trained wind disturbance maps over the entire grid.

Comparison to ground truth wind field From the results in Figure 4-11, it can already be seen that the mean of the trained wind disturbance map is not accurate everywhere. Figure

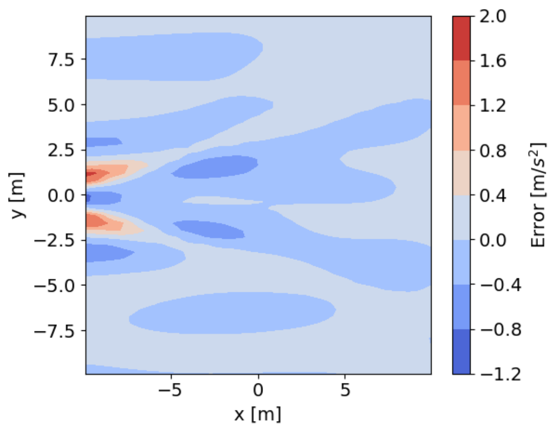
Table 4-5: Final performance metrics of the two trained wind disturbance maps.

Wind field	Wind in x-direction	Crossing fans
Training time [s]	15.07	14.81
Prediction time [s]	0.069	0.067
MSE	0.019	0.060
Mean uncertainty [m/s^2]	0.248	0.498
Maximum uncertainty [m/s^2]	0.3575	0.639

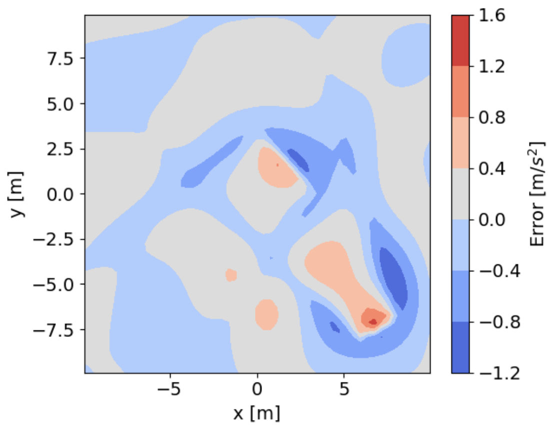
4-12 therefore shows the difference between the trained mean $\hat{d}_{v.}$ and the actual mean $d_{v.}$ for the x- and y-direction independently, again for both trained wind fields. Specifically, near the fans, the prediction is less accurate. This is likely due to the rapid drop in wind near the fans, which cannot be accurately captured by the smooth GP model, especially since the expressiveness is reduced due to the sparse GP approach.



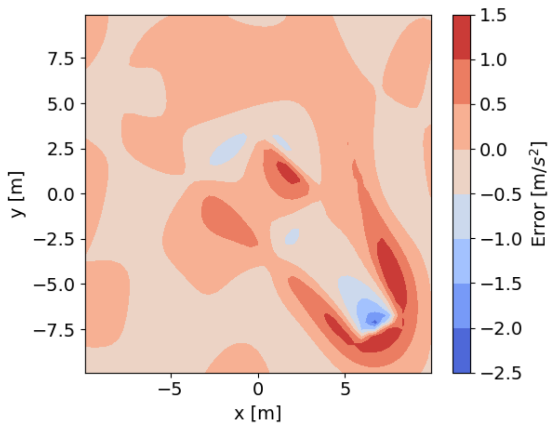
(a) Training error $\hat{d}_{v_x} - d_{v_x}$ for fan pointing in x-direction.



(b) Training error $\hat{d}_{v_y} - d_{v_y}$ for fan pointing in x-direction.

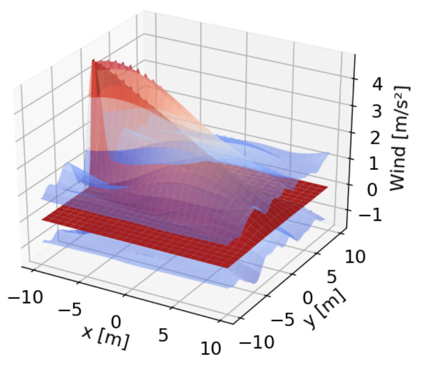


(c) Training error $\hat{d}_{v_x} - d_{v_x}$ for crossing fans.

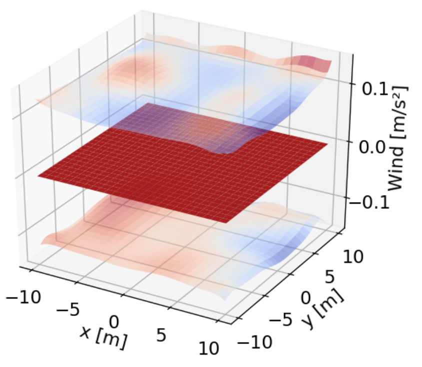


(d) Training error $\hat{d}_{v_y} - d_{v_y}$ for crossing fans.

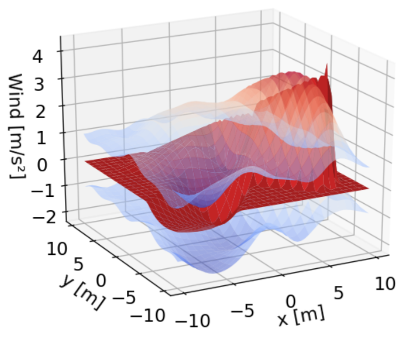
Figure 4-12: Difference between the trained mean disturbance and the mean disturbance of the original wind fields.



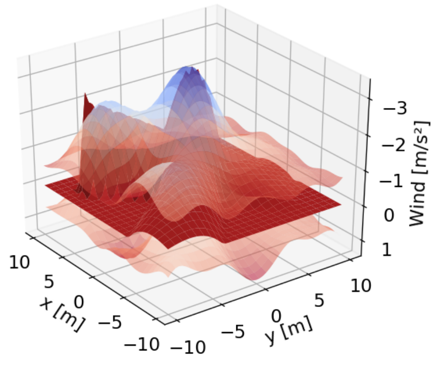
(a) Disturbance bound of the trained wind wind field pointing in x-direction in x.



(b) Disturbance bound of the trained wind wind field pointing in x-direction in y.



(c) Disturbance bound of the trained wind wind field for crossing fans in x.



(d) Disturbance bound of the trained wind wind field for crossing fans in y.

Figure 4-13: Confidence bounds of the trained wind fields compared to the mean of the original wind fields. The original wind fields are shown in red, the disturbance bounds corresponding to a confidence interval of 99.5% are represented by the transparent surfaces.

Uncertainties of the trained wind fields Finally, the ground truth wind fields are compared to the trained models in terms of the uncertainties to see if the uncertain region covers the actual disturbance despite some model mismatch. Figure 4-13 shows the actual mean disturbance the uncertain regions above and below the trained mean within 2.807 standard deviations. This uncertain region, also referred to as the 2.807σ confidence bound, corresponds to a 99.5% confidence interval, meaning that 99.5% of data points, generated according to the trained GP model will lie in this region. For this choice of confidence interval, the actual disturbance lies within the trained wind fields for the largest part. Despite some mismatch in the mean prediction, the designed motion planner and controller should therefore be robust enough, taking in account the uncertainty with a confidence interval of 99.5%. However for large spikes of the disturbance, the GP model just fails to capture the disturbances within the uncertain bounds, as can be seen in Figure 4-13c and 4-13d. After some investigation on the influence of the training parameters, the sparse learning approach is found as a cause. Again, it is likely that reducing the number of data points to 30, reduces the expressiveness of the GP, which fails to capture the large spikes and their proper uncertainties during training. Moreover, the sparse GP approach complicates the training by using a stochastic approach. It is advised, to investigate this behavior in follow-up research to see if there is a way to train the sparse GP where it also captures the spikes in the data. Moreover, this also motivates the use of online GP in which the data would only capture the environment near the quadrotor and not the entire grid. For now, a way to work around the problem is to increase the confidence bounds to make sure the true wind field lies within the uncertain bounds. However, as in this case the spikes in the disturbance only barely exceed the confidence bounds, the confidence interval of 99.5% is kept so that the controller is less conservative.

4-5 Summary and discussion

GP's are a powerful machine learning tool that can be used to model the wind disturbance maps with little prior knowledge. In order to use GP for this application, several aspects of GP training have to be considered.

Despite the model parameters that are optimized during training, the type of kernel function has to be chosen as the model prior. Comparing a Matern and SE for this application, no significant difference is found so the SE kernel is chosen for its simpler form.

Moreover, the use of AL for optimal exploration is introduced. By collecting data with AL, the data can be gathered optimally and adjusted to the disturbance map. This results in faster exploration times and lower uncertainty in the trained GP compared to pre-designed exploration paths.

The use of sparse GP methods is implemented for compatibility with the MPCC formulation. While this reduces the accuracy of the GP model and increases uncertainty in predictions, it also speeds up predictions by 150 times, making GP feasible for use. The decrease in model performance is captured by the increased uncertainty and is into consideration by the controller design.

Two types of wind maps are trained for the upcoming MPCC design. Although the trained models do not have perfect mean predictions, the model mismatch is largely contained within the uncertain bounds with a confidence interval of 99.5%. Reducing the number of training points during the sparsification process, however, may result in inaccurate predictions of high spikes in the training data. Therefore, exploring online learning methods that don't require mapping the whole environment, especially for real-world scenarios with changing wind conditions, is recommended for future research.

Chapter 5

Data-driven MPCC motion planner

This chapter covers the derivation of the controller formulation of the Model Predictive Contouring Control (MPCC) controller for navigating the quadrotor through windy and cluttered environments using a simple reference path and taking into account the information of the wind disturbance map. Then, the implementation of the MPCC controller and its first results in simple scenarios are discussed to validate that the data-driven controller formulation is functioning properly.

5-1 Data-driven controller formulation

The resulting data-driven controller formulation consists of several building blocks that are familiar from the classical Model Predictive Control (MPC) formulation: the system dynamics, the cost function, and the constraints. All of these components are discussed for the problem at hand before moving on to the resulting controller formulation.

5-1-1 System dynamics

The system dynamics are a combination of the nominal quadrotor model and the disturbance map. As the disturbance map is represented by a Gaussian Process (GP), the resulting system dynamics become probabilistic. Subsequently, the propagation of the model across multiple states will be discussed given the probabilistic formulation.

Data-driven quadrotor model

The system equations of the quadrotor are given as a combination of the nominal model and the GP model. Due to the discrete nature of the GP model, as is trained in discrete time, the system dynamics are represented in discrete form, according to:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) + B_d \left(\mathbf{d}(\mathbf{p}_k) + \mathbf{w}_k \right). \tag{5-1}$$

As the nominal, nonlinear dynamics $f(\mathbf{x}_k, \mathbf{u}_k)$, that follow the model derived in Eq. (3-6), are given in continuous time, they are discretized using a Runge-Kutta 4th (RK4) integration scheme to obtain fully discrete dynamics. As the trained GP disturbance map:

$$\mathbf{d}(\mathbf{p}_k) = \begin{bmatrix} d_{v_x}(\mathbf{p}_k) \sim \mathcal{N}\left(\mu^{d_{v_x}}(\mathbf{p}_k), \Sigma^{d_{v_x}}(\mathbf{p}_k)\right) \\ d_{v_y}(\mathbf{p}_k) \sim \mathcal{N}\left(\mu^{d_{v_y}}(\mathbf{p}_k), \Sigma^{d_{v_y}}(\mathbf{p}_k)\right) \end{bmatrix} \tag{5-2}$$

is only defined for the velocities v_x and v_y, the disturbance vector $\mathbf{d}(\mathbf{p}_k) \in \mathbb{R}^2$ is mapped to the correct states with $B_d \in \mathbb{R}^{9\times 2}$. Additional uncertainty in the system is accounted for by the process noise $\mathbf{w}_k \sim \mathcal{N}(0, \Sigma_k^{\mathbf{w}})$.

State and uncertainty propagation

Due to the representation of the nonlinear disturbance map by a Gaussian Process, the predicted states are given as stochastic distributions. Evaluating the posterior of the GP at the next uncertain state input, however, is computationally intractable. Instead, the posterior distribution is approximated by a Gaussian distribution, i.e.

$$\mathbf{x}_{k+1} \sim \mathcal{N}(\boldsymbol{\mu}_{k+1}^{\mathbf{x}}, \Sigma_{k+1}^{\mathbf{x}}) \tag{5-3}$$

with approximate inference using linearization. For more information on the uncertainty propagation the reader is referred to Appendix C.

Propagation of the mean With the next state estimate given by Eq. (5-1) the mean of the next state estimate can be computed by passing the mean values through the system model. This gives:

$$\boldsymbol{\mu}_{k+1}^{\mathbf{x}} = f(\boldsymbol{\mu}_k^{\mathbf{x}}, \mathbf{u}_k) + B_d\left(\boldsymbol{\mu}_k^{\mathbf{d}}(\boldsymbol{\mu}_k^{\mathbf{p}}) + \mathbf{w}_k\right) \tag{5-4}$$

for the update of the state mean. Here, $\boldsymbol{\mu}_k^{\mathbf{d}}(\boldsymbol{\mu}_k^{\mathbf{p}})$, is the mean of the GP disturbance map evaluated at the mean of the uncertain position vector $\mathbf{p}_k$.

Propagating the uncertainty For the propagation of the uncertainty, the next state estimate is approximated by its first-order Taylor approximation:

$$\mathbf{x}_{k+1} \approx \mathbf{x}_k^0 + \left(\frac{df}{d\mathbf{x}} f(\mathbf{x}_k, \mathbf{u}_k)\Big|_{\mathbf{x}=\boldsymbol{\mu}_x} + \frac{df}{d\mathbf{x}} B_d \mathbf{d}(\mathbf{p}_k)\Big|_{\mathbf{x}=\boldsymbol{\mu}_x}\right)\mathbf{x}_k \tag{5-5a}$$

$$\mathbf{x}_{k+1} \approx \mathbf{x}_k^0 + \left(\nabla_{\mathbf{x}} f(\boldsymbol{\mu}_k^{\mathbf{x}}, \mathbf{u}_k) + B_d \nabla_{\mathbf{x}} \boldsymbol{\mu}_k^{\mathbf{d}}(\boldsymbol{\mu}_k^{\mathbf{p}})\right)\mathbf{x}_k \tag{5-5b}$$

where $\nabla_{\mathbf{x}} \boldsymbol{\mu}_k^{\mathbf{d}}(\boldsymbol{\mu}_k^{\mathbf{p}})$ is the derivative of the GP with respect to its state vector evaluated at the mean of the position vector. Given the function linearization, one can propagate the uncertainty as for the linear case:

$$\begin{aligned}\Sigma_{k+1}^{\mathbf{x}} = [\nabla_{\mathbf{x}} f(\boldsymbol{\mu}_k^{\mathbf{x}}, \mathbf{u}_k) + B_d \nabla_{\mathbf{x}} \boldsymbol{\mu}_k^{\mathbf{d}}(\boldsymbol{\mu}_k^{\mathbf{p}})]\Sigma_k[\nabla_{\mathbf{x}} f(\boldsymbol{\mu}_k^{\mathbf{x}}, \mathbf{u}_k) + B_d \nabla_{\mathbf{x}} \boldsymbol{\mu}_k^{\mathbf{d}}(\boldsymbol{\mu}_k^{\mathbf{p}})]^T \\ + B_d\left(\Sigma_k^{\mathbf{d}} + \Sigma_k^{\mathbf{w}}\right) B_d^T\end{aligned} \tag{5-6}$$

with $B_d\left(\Sigma_k^{\mathbf{d}}+\Sigma_k^{\mathbf{w}}\right)B_d^T$ being the propagation of the GP uncertainty $\Sigma_k^{\mathbf{d}}$ at the k-th state plus the uncertainty from the added noise $\mathbf{w}_k$.

Rewriting the expression results in the final update equation of the uncertainty:

$$\Sigma_{k+1}^{\mathbf{x}}=\left[\nabla_{\mathbf{x}}f\left(\boldsymbol{\mu}_k^{\mathbf{x}},\mathbf{u}_k\right)\,B_d\right]\left[\begin{array}{cc}\Sigma_k^{\mathbf{x}} & \Sigma_k^{\mathbf{xd}}\\ \Sigma_k^{\mathbf{dx}} & \Sigma_k^{\mathbf{d}}+\Sigma_k^{\mathbf{w}}\end{array}\right]\left[\nabla_{\mathbf{x}}f\left(\boldsymbol{\mu}_k^{\mathbf{x}},\mathbf{u}_k\right)\,B_d\right]^T \tag{5-7}$$

with $\Sigma_k^{\mathbf{dx}}=\nabla_{\mathbf{x}}\boldsymbol{\mu}_k^{\mathbf{d}}\left(\boldsymbol{\mu}_k^{\mathbf{p}}\right)\Sigma_k^{\mathbf{x}}$ and $\Sigma_k^{\mathbf{xd}}=\Sigma_k^{\mathbf{dx}T}$.

State and uncertainty update Replacing the nonlinear nominal function dynamics with the linear model in Eq. (3-7) the resulting mean and uncertainty propagation are given by:

$$\boldsymbol{\mu}_{k+1}^{\mathbf{x}}=f(\boldsymbol{\mu}_k^{\mathbf{x}},\mathbf{u}_k)+B_d\left(\boldsymbol{\mu}_k^{\mathbf{d}}(\boldsymbol{\mu}_k^{\mathbf{p}})+\mathbf{w}_k\right) \tag{5-8a}$$

$$\Sigma_{k+1}^{\mathbf{x}}=\left[A\,B_d\right]\left[\begin{array}{cc}\Sigma_k^{\mathbf{x}} & \Sigma_k^{\mathbf{xd}}\\ \Sigma_k^{\mathbf{dx}} & \Sigma_k^{\mathbf{d}}+\Sigma_k^{\mathbf{w}}\end{array}\right]\left[A\,B_d\right]^T. \tag{5-8b}$$

The update equations of the state mean and state uncertainty are used in the cost function and obstacle avoidance constraints, derived in the following.

5-1-2 Model Predictive Contouring Control and resulting cost function

The MPCC controller follows the reference path by tracking a given velocity while minimizing the the distance to the path. This objective has to be translated into the cost function of the MPCC controller. The formulation discussed hereafter is taken from the research in [9] and slightly modified for the application of this thesis.

Reference path The reference path of the MPCC controller is given by a set of reference points determining by the position and yaw angle of the quadrotor. The reference points are translated into a reference path using spline interpolation.

Progress along the path The progress along the path is described by a variable s_k. The desired path is parameterised by s. For a given longitudinal vehicle speed $v_k=\|\mathbf{v}_k\|_2$ at time step k, the approximate progress along the reference path can be described by:

$$s_{k+1}=s_k+v_kT_s. \tag{5-9}$$

with sampling time T_s. To make progress along the path and track a desired speed, a cost is defined that penalizes the deviation of the quadrotor speed v_k from a reference velocity v_{ref}, i.e.,

$$\mathcal{J}_{\text{speed}}\left(\mathbf{x}_k\right)=Q_v\left(v_{\text{ref}}-v_k\right)^2 \tag{5-10}$$

with Q_v a design weight.

Tracking error For tracking of the reference path, a contour error e^c and lag error e^l are defined and combined in an error vector:

$$\mathbf{e}_k=\left[\begin{array}{c}\hat{e}^l\left(\mathbf{x}_k,s_k\right)\\ \hat{e}^c\left(\mathbf{x}_k,s_k\right)\end{array}\right]. \tag{5-11}$$

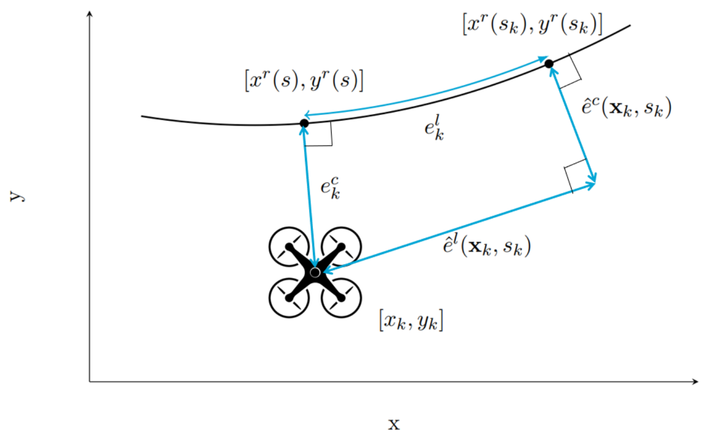


Figure 5-1: Contour and lag error and their approximations in 2D, inspired by [22].

The contour and lag error are approximated to define the distance to the reference path, given the progress along the path s_k as shown in Figure 5-1. The tracking cost is designed to minimize the distance to the path:

$$\mathcal{J}_{\text{tracking}}(\mathbf{x}_k, s_k) = \mathbf{e}_k^T Q_\epsilon \mathbf{e}_k, \tag{5-12}$$

where Q_ϵ is a design weight.

Penalizing the inputs Additionally, the inputs are penalized to discourage large control inputs with

$$\mathcal{J}_{\text{input}}\ (\mathbf{u}_k) = \mathbf{u}_k^T Q_u \mathbf{u}_k, \tag{5-13}$$

where Q_u is a design weight.

Total cost The total stage cost of the MPCC is

$$\mathcal{J}_{\text{MPCC}}\left(\mathbf{x}_k, \mathbf{u}_k, s_k\right) := \mathcal{J}_{\text{tracking}}\ \left(\mathbf{x}_k, s_k\right) + \mathcal{J}_{\text{speed}}\ \left(\mathbf{x}_k\right) + \mathcal{J}_{\text{input}}\ \left(\mathbf{u}_k\right). \tag{5-14}$$

The stage cost is evaluated at the mean of the Gaussian state distribution.

5-1-3 Obstacle avoidance constraints

To guarantee that the generated trajectory is feasible, given the obstacle space, obstacle avoidance constraints have to be enforced such that:

$$\mathcal{B}\left(\mathbf{x}_k, \Sigma_k^{\mathbf{x}}\right) \cap \mathcal{C}_o = \emptyset \tag{5-15}$$

where $\mathcal{B}(\mathbf{x}_k, \Sigma_k^{\mathbf{x}})$ represents the occupancy volume of the quadrotor and $\mathcal{C}_o$ denotes the collision region with surrounding obstacles. As the dynamics model provides an uncertainty of the state distribution, this uncertainty is taken into account in the constraint formulation to guarantee more robust obstacle avoidance in contrast to using only the nominal state. For obstacle avoidance, the relevant uncertainties are the uncertainties in the positional states:

$$\Sigma_k^{\mathbf{p}} = \begin{bmatrix} \sigma_x & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix}. \tag{5-16}$$

The uncertainty of the position $\Sigma_k^{\mathbf{p}}$ is extracted from the propagated uncertainty at each state k. Two types of constraints are considered in this work and compared in terms of their performance: ellipsoidal constraints and Gaussian constraints. Both types of constraints assume static, circular obstacles.

Ellipsoidal constraints

The uncertainty in the position can be used to construct an uncertain ellipsoidal bound around the quadrotor. The major axis a and minor axis b of the ellipsoid and its rotation ψ can be found using a singular value decomposition of $\Sigma_k^{\mathbf{p}}$ [57]. To guarantee a certain collision probability, the ellipsoid is scaled given the confidence interval χ. The obstacle avoidance constraint at each state is [57]:

$$c_k^o(\mathbf{x}_k) = \begin{bmatrix} \Delta x_k^o \\ \Delta y_k^o \end{bmatrix}^{\mathrm{T}} R(\psi)^{\mathrm{T}} \begin{bmatrix} \frac{1}{\alpha^2} & 0 \\ 0 & \frac{1}{\beta^2} \end{bmatrix} R(\psi) \begin{bmatrix} \Delta x_k^o \\ \Delta y_k^o \end{bmatrix} > 1, \forall o \in \mathcal{I}_o \tag{5-17}$$

where $\Delta x_k^o$and Δy_k^o is the distance between the quadrotor and the obstacle in x and y, respectively, for each obstacle in the obstacle space $\mathcal{I}_o$. The rotation matrix $R(\psi)$ is describing the rotation of the uncertain ellipsoid. The quadrotor radius r_{quad} is enlarged by the uncertain ellipsoid such that the total clearances α and β, are given by:

$$\alpha = \sqrt{\chi} \cdot a + r_o + r_{\text{quad}} \tag{5-18a}$$

$$\beta = \sqrt{\chi} \cdot b + r_o + r_{\text{quad}} \tag{5-18b}$$

with obstacle radius r_o.

Chance constraints

Instead of converting the Gaussian uncertainties into hard ellipsoidal boundaries using sigma confidence intervals, the Gaussian distribution of the uncertainty can be directly transformed into constraints using a chance constraint formulation which is derived by the authors in [11]. The chance constraints are defined as a collision probability:

$$\Pr\left(\mathbf{x}_k \notin \mathcal{C}_o\right) \geq 1 - \delta_o, \forall o \in \mathcal{I}_o \tag{5-19}$$

$$\Pr\left(\mathbf{a}_o^T\left(\mathbf{p} - \mathbf{p}_o\right) \leq b\right) \tag{5-20}$$

with collision probability δ_o and collision region with each obstacle $\mathcal{C}_o$. The collision region is given as an enlarged half space of the actual collision region:

$$\tilde{\mathcal{C}}_o := \left\{\mathbf{x} \mid \mathbf{a}_o^T\left(\mathbf{p} - \mathbf{p}_o\right) \leq b\right\}, \tag{5-21}$$

where $\mathbf{a}_o = \left(\hat{\mathbf{p}} - \mathbf{p}_o\right) / \left\|\hat{\mathbf{p}} - \mathbf{p}_o\right\|$ with uncertain quadrotor position

$$\mathbf{p} \sim \mathcal{N}\left(\hat{\mathbf{p}}, \Sigma_k^{\mathbf{p}}\right), \tag{5-22}$$

the deterministic obstacle position $\mathbf{p}_o$ and nominal distance between the obstacle and quadrotor $b = r_{\text{quad}} + r_o$. The chance constraints can be reformulated into deterministic ones using the following relation in [34]:

$$\Pr\left(\mathbf{a}^T\mathbf{x} \leq b\right) = \frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{b - \mathbf{a}^T\hat{\mathbf{x}}}{\sqrt{2\mathbf{a}^T \sum \mathbf{a}}}\right) \tag{5-23}$$

where erf denotes the error function:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt. \tag{5-24}$$

This deterministic chance constraints take the form:

$$\mathbf{a}_o^T (\hat{\mathbf{p}} - \mathbf{p}_o) - b \geq \mathrm{erf}^{-1}(1 - 2\delta_o) \sqrt{2\mathbf{a}_o^T (\Sigma_k^{\mathbf{p}}) \mathbf{a}_o}, \forall o \in \mathcal{I}_o. \tag{5-25}$$

Soft constraints

To ensure feasibility of the optimization problem, the obstacle avoidance constraints are formulated as soft constraints. A slack variable $\xi \geq 0$ is added to the constraints. Using a linear quadratic soft constraint formulation, an additional cost $\mathcal{J}_\xi = \|\xi\|_{q_\xi}^2 + c_\xi \xi$ for the slack variable is defined. For sufficiently large c_ξ the soft constraint formulation is exact, if feasible [58].

5-1-4 Resulting controller formulation

Combining the dynamics model, the cost function and obstacle avoidance constraints with the standard MPC formulation, the following controller formulation is implemented:

$$\begin{aligned}
\mathcal{J}^*_{\mathrm{MPCC}} = & \min_{\mathbf{x},\mathbf{u},s} \mathcal{J}_{\mathrm{MPCC}}(\boldsymbol{\mu}^{\mathbf{x}}, \mathbf{u}, s) && \text{(5-26a)}\\
\text{s.t. } & \boldsymbol{\mu}_0^{\mathbf{x}} = \boldsymbol{\mu}^{\mathbf{x}}(0), && \text{(5-26b)}\\
& s_0 = s(0), && \text{(5-26c)}\\
& \boldsymbol{\mu}_{k+1}^{\mathbf{x}} = f(\boldsymbol{\mu}_k^{\mathbf{x}}, \mathbf{u}_k) + B_d \left(\boldsymbol{\mu}_k^{\mathbf{d}}(\boldsymbol{\mu}_k^{\mathbf{p}}) + \mathbf{w}_k\right) && \text{(5-26d)}\\
& \Sigma_{k+1}^{\mathbf{x}} = [A\, B_d] \begin{bmatrix} \Sigma_k^{\mathbf{x}} & \Sigma_k^{\mathbf{xd}}, \\ \Sigma_k^{\mathbf{dx}} & \Sigma_k^{\mathbf{d}} + \Sigma_k^{\mathbf{w}} \end{bmatrix} [A\, B_d]^T, && \text{(5-26e)}\\
& s_{k+1} = s_k + v_k T_s, && \text{(5-26f)}\\
& \mathcal{B}(\mathbf{x}_k, \Sigma_k^{\mathbf{x}}) \cap \mathcal{C}_o = \emptyset, && \text{(5-26g)}\\
& \boldsymbol{\mu}_k^{\mathbf{x}} \in \mathcal{X}, && \text{(5-26h)}\\
& \mathbf{u}_k \in \mathcal{U}, && \text{(5-26i)}\\
& 0 \leq s_k \leq L && \text{(5-26j)}\\
& k = 0, \ldots, N_p - 1 && \text{(5-26k)}
\end{aligned}$$

with cost function

$$\mathcal{J}_{\mathrm{MPCC}}(\mathbf{x}, \mathbf{u}, s) = \sum_{k=0}^{N_p-1} \mathcal{J}_{\mathrm{MPCC}}(\boldsymbol{\mu}_k^{\mathbf{x}}, \mathbf{u}_k, s_k) + \mathcal{J}_{\mathrm{MPCC}}(\boldsymbol{\mu}_{N_p}^{\mathbf{x}}, \mathbf{u}_{N_p}, s_{N_p}). \tag{5-27}$$

In addition to the obstacle avoidance constraints, this formulation also sets input and state constraints as well as a path constraint. Details on the constraints and tuning parameters are given in the next part of this section, describing the controller implementation.

5-2 Data-driven controller implementation

With the resulting data-driven MPCC formulation, an optimization problem is solved to find the control inputs to the quadrotor. This section discusses the implementation of the optimal control problem, the tuned parameters and chosen state constraints and the system interface to control the quadrotor in simulation.

5-2-1 Parameters of the controller implementation

The data-driven MPCC problem is implemented with a prediction horizon of N_p= 20 at a sampling rate of $T_s = 50$ms, resulting in a 1s look-ahead. The underlying optimization problem is solved with the interior-point-based Nonlinear Programming (NLP) solver from FORCES PRO [59]. The maximum number of solver iterations is set to 300.

Adding the GP model to the solver As FORCES PRO does not support an interface with the GPyTorch library yet, the GP model prediction is implemented by hand, extracting the variational parameters $\mathbf{u}$, $\boldsymbol{\mu}_{\mathbf{u}}$, and $\Sigma_{\mathbf{u}}$ from the trained model in GPyTorch. The mean GP prediction is added to the nominal system dynamics. The state uncertainty is propagated outside of the solver, based on the predicted value at the previous MPCC iteration, and set as a fixed parameter. This choice is made to reduce the optimization variables of the solver and to save computation time. Furthermore, numerical issues appear in the solver when having the state uncertainty as an optimization variable inside the constraints. The solver lacks transparency into its internal processes, leading to speculation about the cause of the issue. It is recommended to investigate this further using a more insightful open-source solver or studying the research by [60], which suggests a specific formulation of the optimization problem that incorporates the uncertainty within the solver.

Input and state constraints The input and state constraints are given by:

$$\mathcal{U} = \left\{ \mathbf{u} \in \mathbb{R}^m \;\middle|\; \begin{array}{l} (\phi_c, \theta_c) \in [-40, 40]\text{deg} \\ \dot{\psi}_c \in [-10, 10]\text{deg/s} \\ T_c \in [5, 15]\text{m/s}^2 \end{array} \right\}, \tag{5-28}$$

$$\mathcal{X} = \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{l} (x, y) \in [-\infty, \infty]\text{m} \\ z \in [-1, 1]\text{m} \\ (v_x, v_y, v_z) \in [-10, 10]\text{m/s} \\ (\phi, \theta) \in [-50, 50]\text{deg} \\ \dot{\psi} \in [-15, 15]\text{deg/s} \end{array} \right\}. \tag{5-29}$$

The path constraint is dependent on the scenario and can therefore be set to a high value:

$$\mathcal{U} = \{s \in \mathbb{R} \mid s \in [0, 1e4]\text{m}\} \tag{5-30}$$

Furthermore, the controller parameters listed in Table 5-1 are used.

5-2-2 Simulation interface

The output of the optimal control problem is interfaced with the simple simulation environment described in section 4-2-2, visualized in RViz. The control commands generated by

Table 5-1: MPCC parameters.

Name	**Value**
Roll and pitch input weight ϵ_ϕ and ϵ_θ	0.5
Yaw rate input weight $\epsilon_{\dot{\psi}}$	0.01
Thrust input weight ϵ_T	0.01
Lag error weight ϵ_l	3.0
Contour error weight ϵ_c	1.0
Reference velocity weight ϵ_v	1.0
Reference velocity v_{ref}	2.0
Slack weight q_ξ	1000
Slack weight c_ξ	100

the solver are translated into Robot Operating System (ROS) commands. The quadrotor is subject to the same wind disturbances that are trained previously. In addition, the environment is now populated with static obstacles whose locations are assumed to be known. The quadrotor in simulation has an outer radius of $r_{\text{quad}} = 0.325$m. To demonstrate the validity of the approach the quadrotor is flying a lemniscate trajectory with one circular obstacle placed at $p_o = [0, 0]$ with a radius $r_{\text{obst}} = 0.5$m. This scenario is shown in Figure 5-2a.

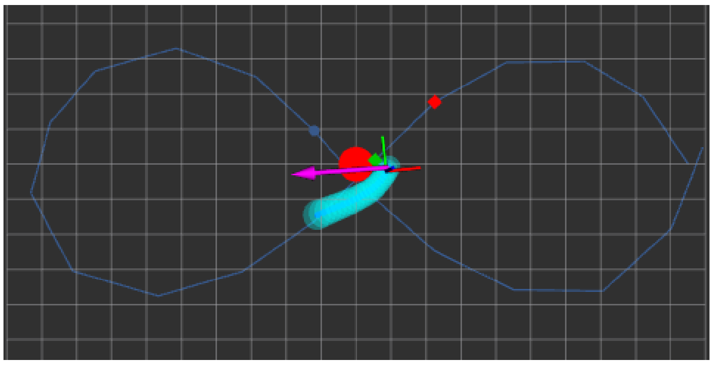
5-3 Data-driven controller validation

To show the validity of the data driven MPCC motion planner, several scenarios are tested for the described simulation environment. First, the GP mean is added to the nominal model to assess its impact on tracking performance. Second, the uncertainty is added to the model together with ellipsoidal and chance constraints for obstacle avoidance. A comparison for both types of constraints is presented. Third, robustness of the controller formulation is validated.

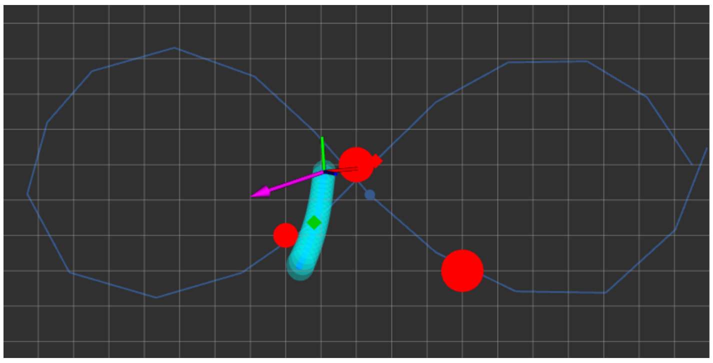
5-3-1 Tracking performance

The lemniscate trajectory is tracked by giving the controller path way points along the trajectory as a reference. Then, the quadrotor is navigated through the environment three times. First, without any wind in the environment and using only the nominal model of the quadrotor. Second, with wind acting on the quadrotor but without considering the GP model, and, third, with wind acting on the quadrotor and propagating the mean of the learned wind disturbance map within the controller formulation. These scenarios are tested for the two types of wind fields trained in the previous chapter.

Visual comparison The results are shown in Figure 5-3 for the wind field pointing in x-direction and in Figure 5-4 for the two crossing fans. If there is no wind acting on the quadrotor, it is perfectly capable of tracking the reference path, due to the perfectly matching models. When wind is acting on the quadrotor and it is not considered in the model, tracking the reference path becomes significantly worse, particularly in regions of high wind speeds. When adding the GP model, the quadrotor tracks the reference more accurately again, however worse than in the case without any present winds. This can be explained



(a) Simple obstacle scenario with one obstacle.



(b) Complex obstacle scenario with three obstacles.

Figure 5-2: Quadrotor following a lemniscate reference path while avoiding circular obstacles visualized in red. The quadrotor is represented by the 3D coordinate system indicating it's position and orientation, the pink arrow shows the wind acting on the quadrotor at any given position. The uncertainty of the quadrotor position is propagated for the training horizon and shown in cyan.

by the fact that the mean of the trained wind disturbance map and the actual wind in the environment do not match at all locations. Where the difference is higher, as shown in Figure 4-12, tracking of the reference is less accurate.

Table 5-2: Distance between the reference path and nearest point on the trajectory flown by the quadrotor with and without the Gaussian process model.

	Wind in x-direction	**Crossing fans**
Wind - no GP	0.126	0.150
Wind - GP	0.070	0.053

Numerical comparison Improved tracking is also validated numerically by computing the distance between the reference path and nearest point on the trajectory flown by the quadrotor, summarized in Table 5-2. By adding the mean of the GP model, tracking becomes around two to three times as accurate, showing superiority of the data-driven model.

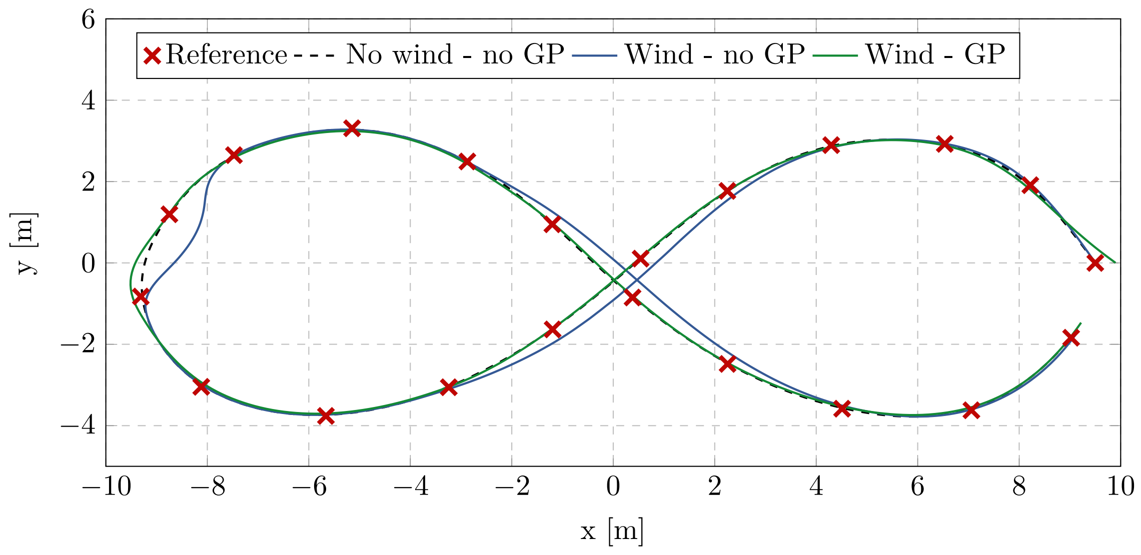


Figure 5-3: Tracking of the lemniscate reference path with and without the GP model for the wind field pointing in x-direction.

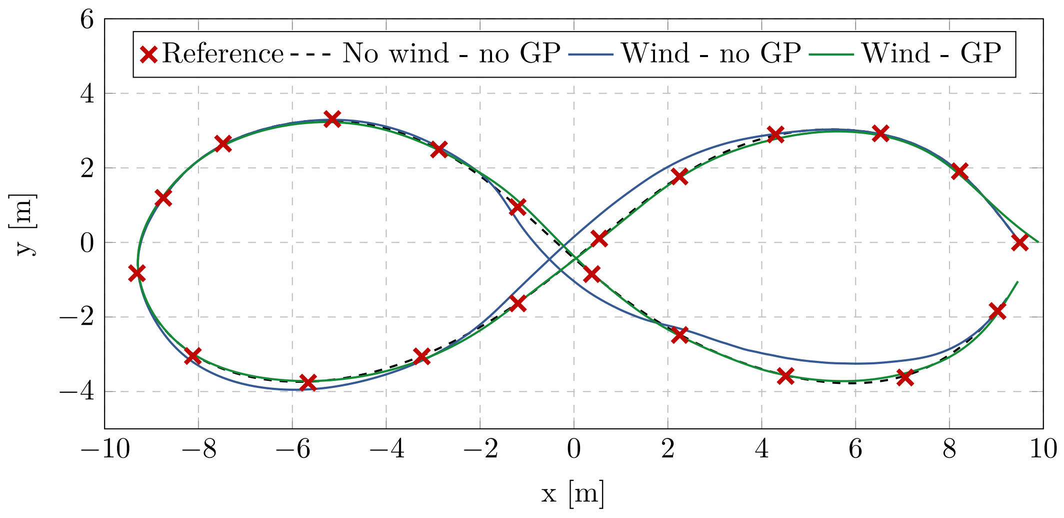
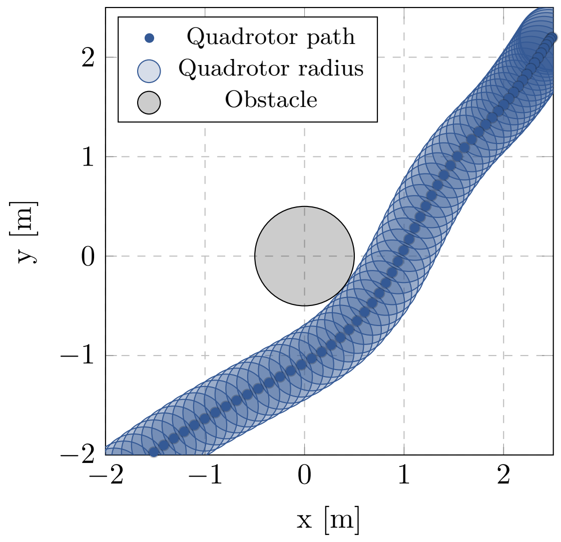


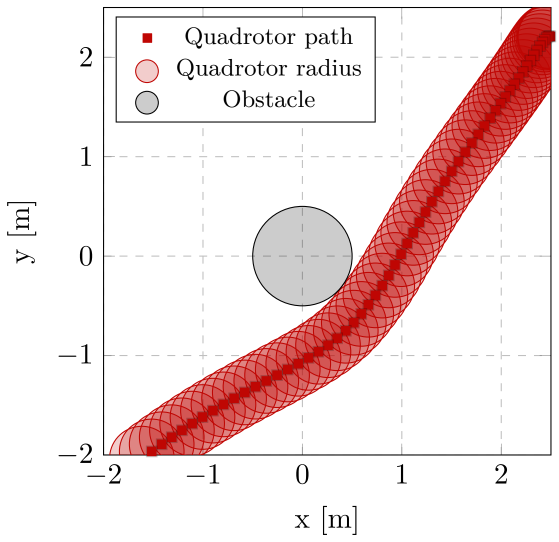
Figure 5-4: Tracking of the lemniscate reference path with and without the GP model for the wind field with two crossing fans.

5-3-2 Ellipsoidal versus chance constraints

Apart from including the mean information to make tracking more accurate, the uncertainty information of the GP model can be used to make obstacle avoidance more robust. In the following, the ellipsoidal and chance constraints for obstacle avoidance discussed in section 5-1-3 are compared for the more complex wind field with the crossing fans. In both cases, the collision probability is set to 0.5%, matching the confidence interval of 99.5%, discussed in section 4-4-3. The quadrotor is navigated around the obstacle at $\mathbf{p} = (0, 0)$, for which the relevant path of the quadrotor is shown for ellipsoidal constraints in Figure 5-5a and chance constraints in Figure 5-5b.



(a) Ellipsoidal constraints. The quadrotor path and outer radius are shown in blue.



(b) Chance constraints. The quadrotor path and outer radius are shown in red.

Figure 5-5: Quadrotor navigating around the obstacle with the two types of constraints active. The circular obstacle is shown in gray.

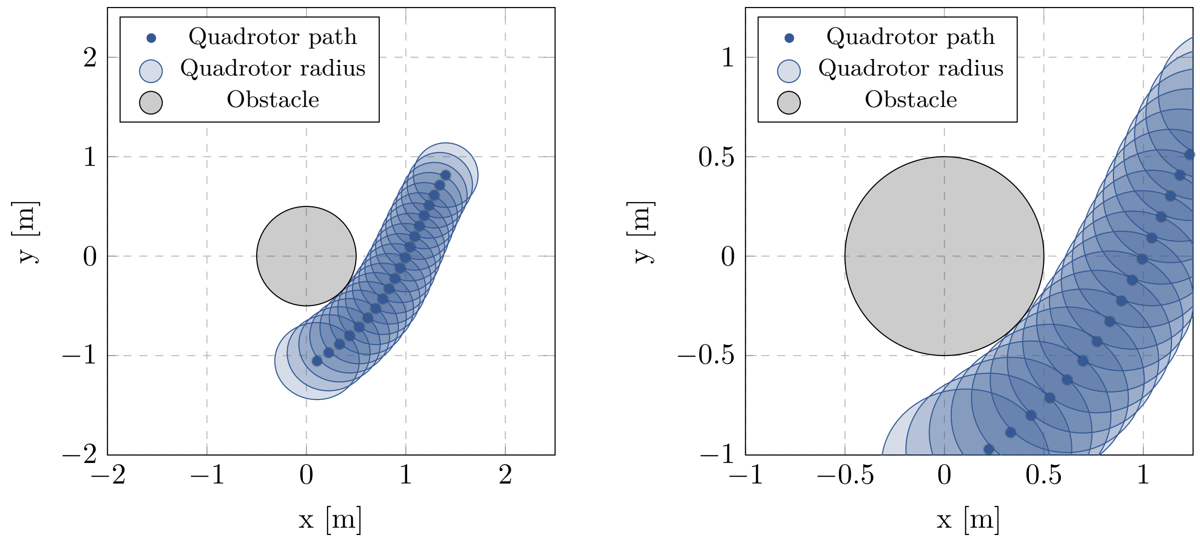


Figure 5-6: Propagation of the quadrotor path and uncertain ellipsoidal bound around the quadrotor for one solver iteration over the prediction horizon with ellipsoidal constraints and a collision probability of 0.5%. The quadrotor path and outer radius are shown in blue, the obstacle is shown in gray.

Conservatism of the constraints Taking a closer look at the propagated uncertainty for one solver iteration over the entire horizon of $N_p = 20$, a difference between the two types of constraints can be noted. Both Figures 5-6 and 5-7 show the propagated bound around the quadrotor, resulting from the sum of the quadrotor radius and the uncertain ellipsoid for a collision probability of 0.5%. With the ellipsoidal constraints in Figure 5-6, the uncertain bounds are exactly tangent to the obstacle, as a hard constraint bound is used. However, with the chance constraints in Figure 5-7, the uncertain ellipsoidal bounds are intersecting with the obstacle. While this seems to be a break of the constraint at first, it can be shown that in both cases no constraints are violated over the horizon, guaranteeing the same theoretic collision probability. This is because the chance constraints provide a probabilistic guarantee rather than a strict requirement, which makes them less conservative than the ellipsoidal constraints.

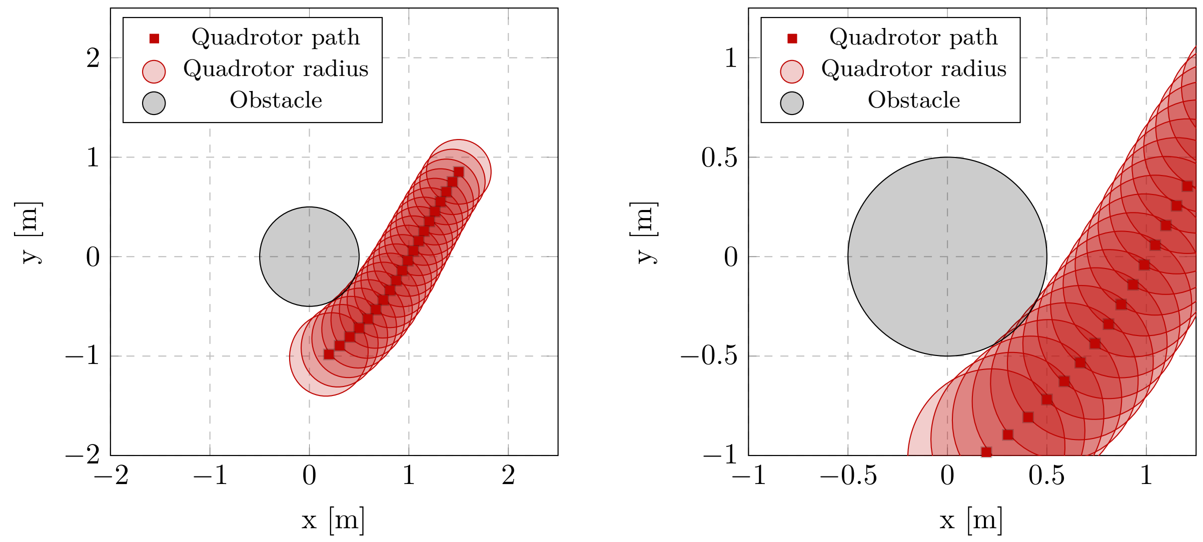


Figure 5-7: Propagation of the quadrotor path and uncertain ellipsoidal bound around the quadrotor for one solver iteration over the prediction horizon with chance constraints. As the stochastic uncertainty cannot be visualized the uncertain ellipsoidal bound for a collision probability of 0.5% is shown. The quadrotor path and outer radius are shown in red, the obstacle is shown in gray.

Chosen constraint type As the ultimate goal of this project is to navigate the quadrotor in crowded environments, having less conservative constraints is crucial for finding a feasible path. In subsequent sections, the solver is therefore always implemented with chance constraints.

5-3-3 Robustness

The robustness of the MPCC controller is evaluated by comparing its nominal and GP-based versions in a more complex scenario with additional obstacles, as shown in Figure 5-2b for the wind field with the crossing fans. The robustness of the controllers is measured by examining the slack ξ, which represents the degree of constraint violation with respect to the obstacles.

Nominal MPCC The path of the quadrotor with the nominal MPCC controller is shown in Figure 5-8. Computing the mean of the slack variable, it is $\xi = 3.13e - 4$, indicating several constraint violations. When checking for the number of constraint violations at the initial state, there are a total of seven constraint violations. In a real scenario, this would result in several crashes of the quadrotor with obstacles in the environment.

GP-based MPCC The path of the quadrotor with the GP-based MPCC controller is shown in Figure 5-9. It can be noted, that the quadrotor chose to take a different path taking into

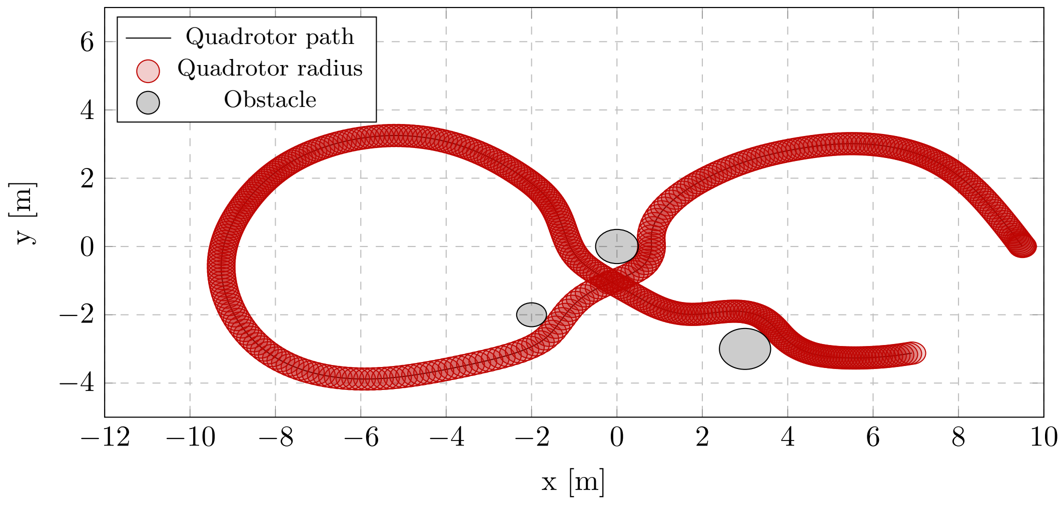


Figure 5-8: Quadrotor navigating around multiple obstacles with the nominal MPCC controller.

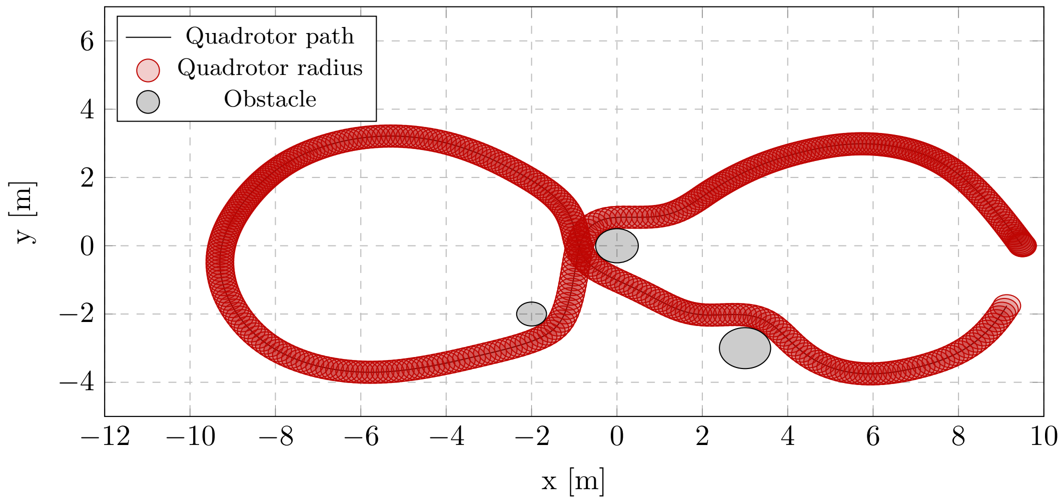


Figure 5-9: Quadrotor navigating around multiple obstacles with the GP-based MPCC controller.

account the mean and uncertainty of the GP model. As the quadrotor flies past the obstacle on the side where the wind is blowing, it needs less control input, which likely explains the solver's choice to pass on this side. The mean of the slack variable in this case is $\xi = 2.91e{-}12$. As this is within orders of the numerical accuracy of the solver, no constraints are violated. With the GP model and chance constraints in place the quadrotor therefore manages to successfully navigate around the obstacles without any crashes.

5-3-4 Solver times

To discuss the real-time capabilities of the GP based controller, the solving times of the optimization and control loop are compared for the two experiments performed in the previous section: one with the MPCC formulation including the GP model with uncertainty and one with the MPCC formulation using only the nominal model. Here, the control loop refers to the total time that passes between receiving the next state and sending the computed control command. The optimization loops refers to the time that FORCES PRO needs to compute the optimal control sequence. The simulations run on an Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz processor with 16GB ram and a Mesa Intel(R) UHD Graphics card without GPU acceleration. A summary of the timing results is given in Table 5-3.

Nominal MPCC For the nominal MPCC controller, the control and optimization loop run at similar sampling times. The optimization loop has a mean solving time of 83.2ms and the control loop runs at a mean sampling frequency of 83.6ms. Here, the solver takes up most of the total time needed to run the entire control loop. With this setup, the controller is not yet capable to run in real-time at a desired sampling frequency of 20Hz. However, as the design of the nominal MPCC controller is not the focus of this thesis, a comparison of the solving times with the added GP model is presented, showing by how much the added GP model is slowing down the computations.

GP-based MPCC Adding the mean of the GP model to the solver, its solving times are in fact decreased to 61.2ms. This is most likely due to the reduced model mismatch, such that the initial guess of the solver, which is based on the previous prediction, is closer to the output of the solver. The added GP model does not seem to negatively effect the solving times here. Nevertheless, the total control loop takes longer to compute with a mean solving time of 108.9ms. This results from the propagation of the uncertainty which is done in the control loop, based on the previous solver output. Currently, this is implemented by calling an external Python-based node with the previous positional states, which then sends back the propagation of the uncertainty to the MPCC implementation in C++. The service call is what takes up most of the computation time and can be avoided by implementing the uncertainty propagation together with the MPCC in C++. The uncertainty is then set as a parameter of the solver before the optimization loop.

Table 5-3: Solving times of the optimization and control loop of the MPCC controller with and without the GP model.

	Nominal MPCC	GP-based MPCC
Optimization loop	83.2ms	61.2ms
Control loop	83.6ms	108.9ms

Discussion Overall, for this setup the solver is not capable yet of running real-time at a sampling frequency of 50ms. However, a large part of this is caused by the nominal MPCC implementation being too slow. Adding the mean of the GP model, for the current solving times, does not negatively affect the optimization loop. However, propagating the uncertainty outside of the solver still causes a slow down of the solver of around 40ms.

Speed-up of the computations Most importantly, the nominal computation times need to be sped up. As the real quadrotor supports GPU acceleration, this might already speed up the computation times. The other bottleneck of the computations is the propagation of the uncertainties. To speed up the compuation, it is recommended to implement the propagation of uncertainty inside the MPCC implementation in C++. Moreover, the researchers in [60] suggest an optimal way to solve the problem of propagating the uncertainty inside the solver. They make use of the acados [61] solver, which also supports a Python interface to directly include the trained GP model from GPyTorch, unlike FORCES PRO.

5-4 Summary and discussion

MPCC as presented by the authors in [9], offers a way to follow a reference path in an optimal way, solving the trajectory generation and tracking problem simultaneously. The motion planner and controller requires a system model. If the model is not accurate, such as when the quadrotor is subject to wind, the controller performance will be poor and it may fail in the presence of obstacles, leading to a crash in a real-world scenario.

By incorporating a data-driven approach, using a GP model of the wind that is integrated into the MPCC controller, it is demonstrated that the quadrotor not only exhibits improved tracking but also increased robustness when avoiding obstacles in the environment.

Extending the MPCC controller with the GP model of the wind derived in chapter 4 poses various challenges. These challenges include propagating the uncertain model over future states, calculating the cost of the MPCC controller using a probabilistic model and incorporating uncertainty for obstacle avoidance. Solutions to these challenges are provided in this chapter. Additionally, it is demonstrated that a chance constraint formulation is less conservative for obstacle avoidance and effectively utilizes the uncertainty information provided by the model, making the implementation particularly attractive for cluttered environments.

However, the real-time capability of the approach has not yet been fully validated. It is found that incorporating the mean propagation of the GP model does not slow down the solver, however, even the solving times of the nominal MPCC are currently too large. Including uncertainty in the model further slows down the computations. Improving computation speed is recommended for future research to validate this approach on a physical system. Methods to improve computation times are discussed in section 5-3-4.

Chapter 6

Performance comparison

After validating the data-driven Model Predictive Contouring Control (MPCC) algorithm, it is compared to the external forces resilient safe motion planning algorithm by the authors in [8] to benchmark the developed algorithm in existing literature.

6-1 Comparative method

The data-driven MPCC algorithm aims to solve two main issues in quadrotor flight: First, ensuring safe flight by considering external forces and their uncertainty, and second, navigating the quadrotor through a populated environment given the information on the external force. For the comparison, a method is chosen that is solving the same two issues: The external forces resilient safe motion planning algorithm by the authors in [8].

Motion planning and control The comparative method [8] solves the motion planning and control problem in two separate steps. Given a goal point, a front-end A* path finding algorithm, searches for a feasible trajectory, considering a nominal, external force and simplified quadrotor model. In a second step, the reference trajectory is tracked by a robust Nonlinear Model Predictive Control (NMPC) controller, that is also considering the current, nominal force and a constant, outer bound of the uncertainty. Using feedback control, the authors in [8] construct an uncertain ellipsoid around the quadrotor using forward reachable sets.

Force estimation and re-planning The external force is obtained by VID-Fusion [62], and therefore updated online. When the variance of the external force is larger than the allowed bound, the force is too fierce for the NMPC to find a solution, given the current initial state and reference trajectory. In that case, the front-end path finder re-plans the trajectory with the new, current force estimate. Details on the formulation can be found in [8]. A system overview diagram of the re-planning framework is shown in Figure 6-1.

Theoretic comparison A theoretic comparison of the data-driven MPCC method of this thesis and the method by the authors in [8] is summarized in Table 6-1. The main differences concern the way the information on the external force is obtained and processed in the algorithm and the (separate) motion planning and controller formulation. While the external

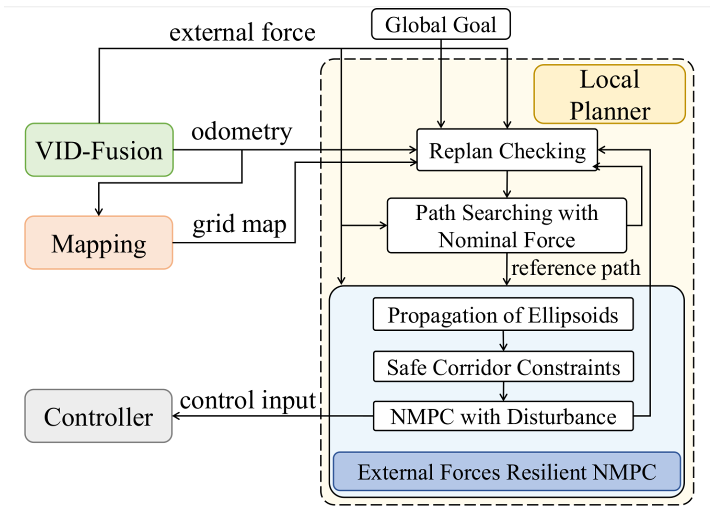


Figure 6-1: System overview diagram of the re-planning framework of the external forces resilient motion planning algorithm [8].

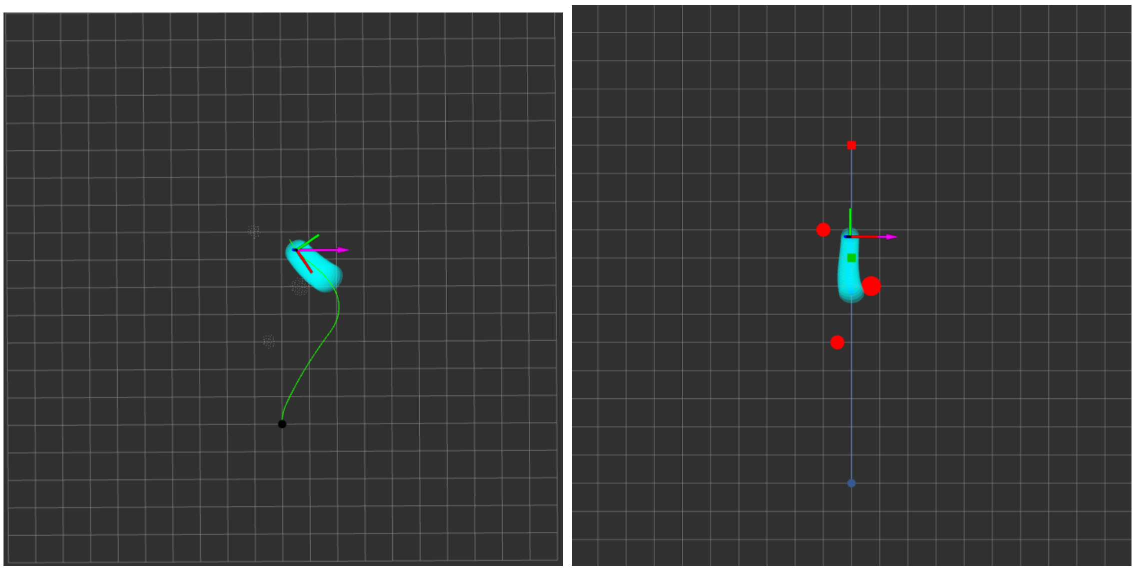
forces resilient motion planner obtains information on the force online, and uses it instantly, the data-driven MPCC controller gathers information on the force and processes it into a model offline. This in turn influences the way the force information is integrated into the controller. As the Gaussian Process (GP) model provides a stochastic uncertain bound, this information can be used in the motion planner and controller formulating chance constraints. The external forces resilient method, on the other hand, only has the nominal information on the force available, with the uncertain bound constructed by hand, resulting in an ellipsoidal constraint formulation. Moreover, the representation of the obstacles is different for both planners. While the external forces resilient planner uses a point cloud representation of the obstacles, the GP-based MPCC has a geometric shape of the obstacles available. Having a point-cloud information is in this case beneficial for real-world experiments where the exact shape of the obstacle might not be known.

Table 6-1: Theoretic comparison of the external forces resilient planner and the GP-based MPCC.

External Forces	**Data-driven MPCC**
Online force estimation	Offline force estimation
No external force model	External force model
Force estimation from VID-fusion	Force estimation from model mismatch
Hard, constant noise bound	Stochastic, dynamic noise bound
Feedback in controller	No feedback in controller
Ellipsoidal constraints	Chance constraints
Trajectory re-plan only when force too high	Constant re-planning
Separate trajectory generation and control	Trajectory generation and control combined
Point cloud representation of obstacles	Geometric description of obstacles

6-2 Scenario

The two methods are compared for the same scenario in the simple simulation environment, displayed in RViz in Figure 6-2. The quadrotor is given goal points between $(0, -5)$ and $(0, 5)$, such that it is always traveling in $y-$direction. The environment on the way is populated with three, circular obstacles, at $\mathbf{p}_1 = (-0.5, 2)$, $\mathbf{p}_2 = (0.7, 0)$ and $\mathbf{p}_3 = (-1, 2)$ with a radius of $r_1 = 0.25$, $r_2 = 0.35$ and $r_3 = 0.25$, respectively. Additionally, a wind force is acting on the quadrotor. The comparative method and the GP-based MPCC are tested with wind pointing in x-direction and the two crossing fans.



(a) External forces resilient planner. The obstacles are represented by a point cloud shown in white.

(b) GP-based MPCC. The obstacles are represented by a geometric shape shown in red.

Figure 6-2: Quadrotor traveling between the two goal points with wind acting on it and obstacles populating the environment for the two types of navigation algorithms. The quadrotor is represented by the 3D coordinate system indicating it's position and orientation, the pink arrow shows the wind acting on the quadrotor at any given position. The uncertainty of the quadrotor position is propagated for the training horizon and shown in cyan.

Modifications to the external forces resilient planner The external forces resilient planner is provided with its own Gazebo simulation environment. To benchmark it against the proposed algorithm, which uses a simplified environment with perfectly matching models, the simulation interface is changed to the simple RViz environment with likewise perfectly matching models. This also reduces the problem from 3D to 2.5D. Moreover, the force estimation step with VID-fusion is skipped and a perfect force estimation is directly provided for the planner. The force estimate is updated at the same sampling frequency as the simulation at 50ms. The obstacles are provided as a global point cloud. The quadrotor radius is set at $r_{\text{quad}} = 0.27\text{m}$ with an inflate ratio of 1.2 such that the resulting radius equals the radius of the quadrotor used in the GP-based controller.

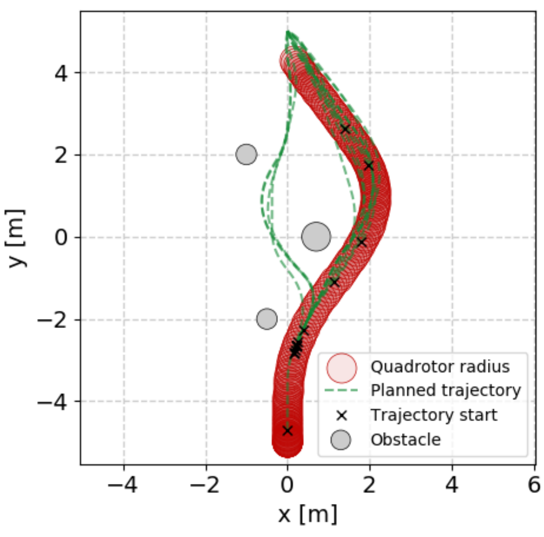
6-3 Experiment results

With the scenarios described above, several experiments are performed to compare the external forces resilient planner against the proposed algorithm. The two methods are evaluated in terms of the generated trajectory, the conservatism and reliability.

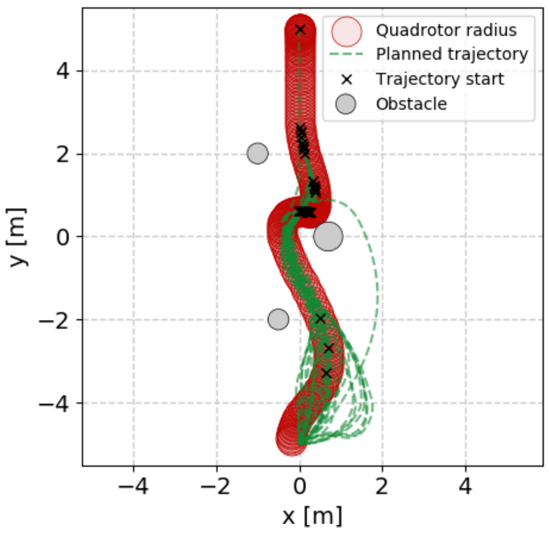
6-3-1 External forces resilient safe motion planner

First, the external forces resilient safe motion planner is tested with the wind pointing in x-direction. From the described algorithm, it is expected that the path is re-planned several times, when the magnitude of the wind disturbance exceeds the predefined bound of $0.5\mathrm{m\,s^{-1}}$. Moreover, by arranging the obstacles in the chosen way, the conservatism of the algorithm is tested.

Quadrotor path In Figure 6-3, the resulting quadrotor trajectory including the radius of the quadrotor is shown for the quadrotor travelling in both positive and negative y-direction.



(a) Quadrotor travelling in positive y-direction.

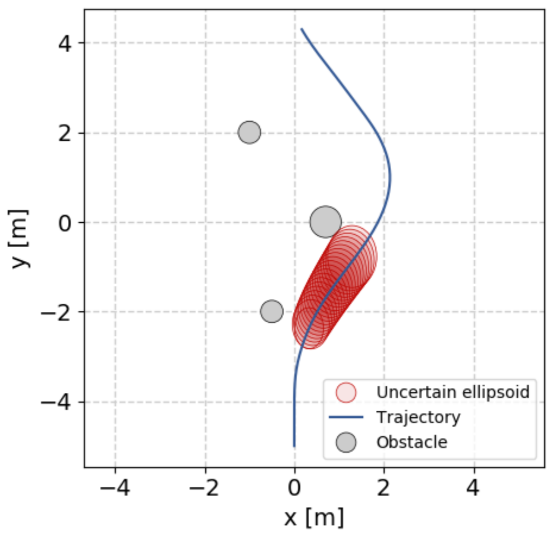


(b) Quadrotor travelling in negative y-direction.

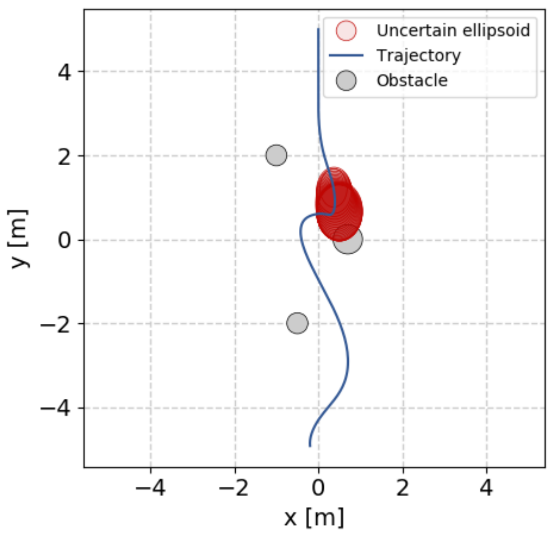
Figure 6-3: Trajectory generated by the external forces resilient safe motion planner for the quadrotor travelling from the start point to the end goal. The dotted green lines show the planned path by the front-end planner. Each black mark indicates the point along the trajectory, where the path is re-planned and also marks the start of the newly planned path.

Travelling in positive y-direction Initially, the front-end path planner is planning a path through the obstacles, which is the shortest way to reach the goal. However, when travelling along the path, the quadrotor encounters the wind disturbance which cause a re-planning of the path. Moreover, around the bottom left obstacle, the path is re-planned multiple times as the planned quadrotor trajectory collides, when moving through the obstacles. The resulting quadrotor trajectory is planned around the obstacles, leading to a safe but more conservative way to reach the goal point.

Travelling in negative y-direction Traveling in the other direction, the algorithm chooses the shorter way through the obstacles. However, when the wind disturbances start acting on the quadrotor this is nearly resulting in a crash. A lot of re-planning iterations are needed to find a feasible path to the goal point, significantly slowing down the quadrotor near the top right obstacle. Finally, the quadrotor finds a feasible path, reaching the goal point.



(a) Quadrotor ravelling in positive y-direction.



(b) Quadrotor travelling in negative y-direction.

Figure 6-4: Propagated uncertain ellipsoids over the planning horizon by the external forces resilient safe motion planner at one point of the total trajectory shown in blue.

Conservatism and uncertainty The reason for the quadrotor choosing the more conservative path to the goal and multiple re-planning iterations in the second case can be explained by the uncertain bounds shown in Figure 6-4. Propagating the uncertain bound results in large uncertainties along the predicted trajectory over one iteration, making it more complicated for the NMPC controller to find a feasible trajectory.

More complex wind fields Moreover, the external forces resilient safe motion planner is tested for the force field with two crossing fans. In that case, the change in force is too large, such that the re-planning is only triggered once the quadrotor is already too close to the obstacle in order to find a feasible path. The algorithm fails, getting stuck in the re-planning loop, as shown in Figure 6-5.

6-3-2 Data-driven MPCC

The same experiments are repeated with the data-driven MPCC approach.

Quadrotor path As the data-driven MPCC controller solves the problem of trajectory planning and control simultaneously, and both the obstacle location and wind field are known beforehand, the trajectory is planned online as part of the optimization problem. The resulting quadrotor trajectories are shown in Figure 6-6. In both travelling directions the algorithm chooses the straight path to the goal, without any collisions.

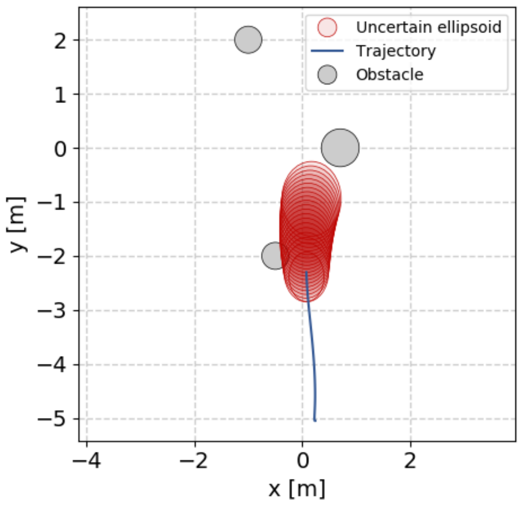
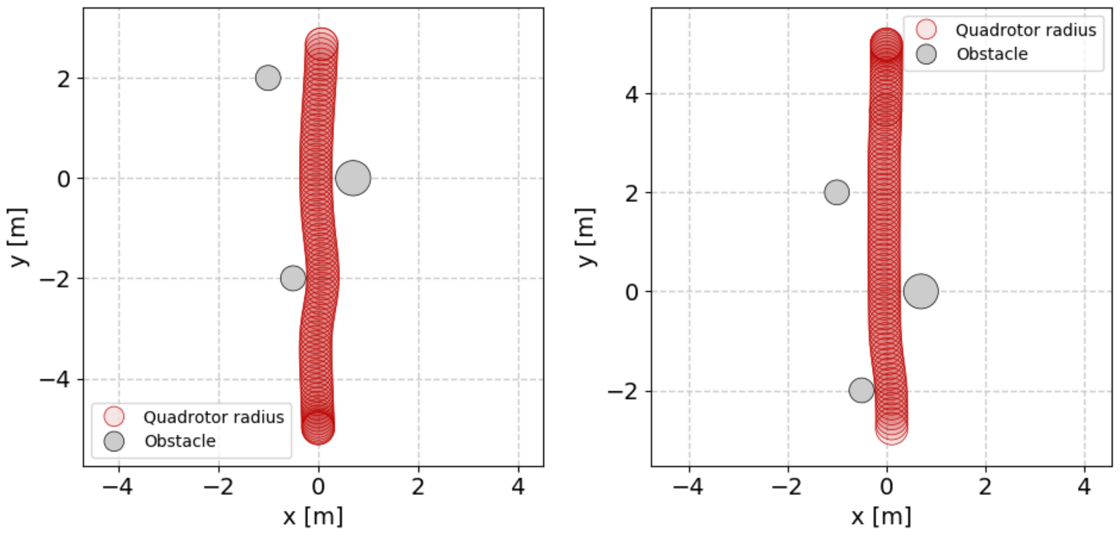


Figure 6-5: Quadrotor getting stuck in the re-planning framework with the wind field generated by two crossing fans.



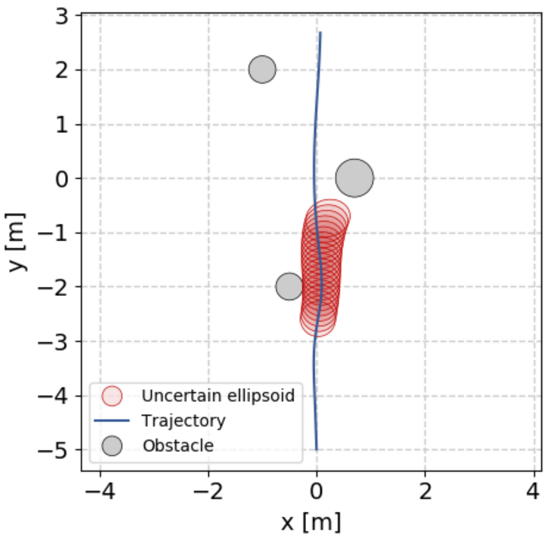
(a) Quadrotor travelling in positive y-direction.

(b) Quadrotor travelling in negative y-direction.

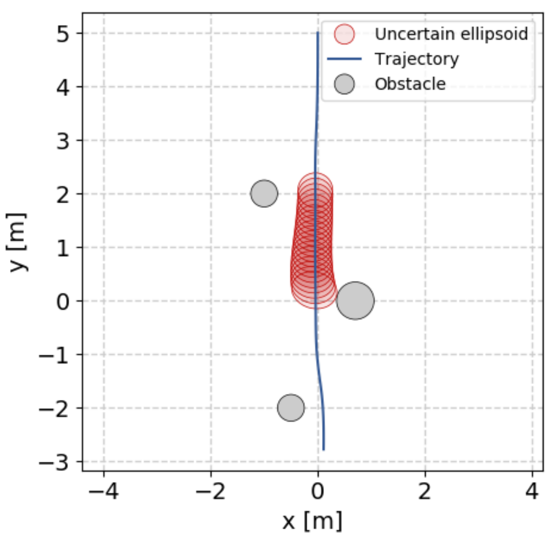
Figure 6-6: Trajectory generated by the GP-based MPCC controller for the quadrotor travelling from the start point to the end goal. As the trajectory is not planned prior only the quadrotor trajectory and radius are shown.

Conservatism As the GP describes the uncertainty of the wind field in a probabilistic way, the uncertain bounds are significantly smaller than for the forces resilient planner (see Figure 6-7). This in combination with the use of a chance constraint formulation to generate an optimal trajectory through the obstacles, results in a less conservative trajectory through the obstacles without having to avoid the obstacles by a large margin.

Other wind fields Repeating the experiment for the wind field with crossing fans yields similar results. The force of the wind and its uncertainty are known beforehand, such that



(a) Quadrotor travelling in positive y-direction.



(b) Quadrotor travelling in negative y-direction.

Figure 6-7: Propagated uncertain ellipsoids over one planning horizon by the GP-based MPCC controller at one point of the total trajectory shown in blue.

the quadrotor is able to find a feasible trajectory without any crashes.

6-3-3 Experimental comparison

The experiments validate the theoretical comparison. As the external forces resilient planner only reacts to instantaneous forces, the situation occurs where this information is obtained too late, leading to infeasibility of the approach. The GP-based motion planner, on the other hand, has the information on the wind available beforehand, allowing it to take the evolution of the force acting on the quadrotor into account during planning and avoiding a deadlock. However, this comparison also has to be taken with caution for two reasons. On one hand, the force estimate is currently only updated at a rate of 20Hz, which is much slower than in the Gazebo simulation environment proposed in the original paper [8]. Failure of the forces resilient planner may therefore not only result from too fierce changes in force but also the decreased update frequency of the force. On the other hand, the GP-based MPCC does an exploration of the environment beforehand, providing it information that would not be available in case of online exploration. The online capability is one of the major advantages of the forces-resilient planner. For a fair comparison, it is recommended to extend the GP-based MPCC to an online learning setup. Still, one would anticipate that the online GP-based MPCC would outperform the forces resilient planner, given that it offers a force model that enables predicting into the future. Moreover, for now perfect force information is assumed for the forces resilient planner, which will not be the case when using VID-fusion measurements, that decrease the accuracy of the planner.

Conservatism Moreover, it is validated experimentally that the GP-based MPCC is less conservative than the forces resilient planner despite the missing feedback in the control law that could help contain the uncertainty even more. It is one of the main advantages of the

GP-model that it provides and estimate of the uncertainty, which is generally less conservative than a hand-designed uncertain bound. It is also already shown in the previous chapter that the chance constraint formulation is less conservative than the ellipsoidal constraints used in the forces resilient planner.

Complex wind fields Finally, it is also shown that combining the motion planner and controller into one optimization problem avoids the re-planning framework which can cause the quadrotor to get stuck or requires a lot of re-planning iterations, as is the case for the forces resilient planner. Nevertheless, the current formulation of the Optimal Control Problem (OCP) also causes the quadrotor to come very close to the obstacles. A sufficient safety margin is therefore recommended in a real-world setup.

6-4 Summary and discussion

Comparing the proposed GP-based motion planner to the current state-of-the-art external forces resilient safe motion planner by [8] shows the potential and areas for improvement of the proposed GP method.

It is shown that providing a GP model of the wind can outperform methods that only react to the force once it has been observed, especially in case of more rapidly changing winds. However, for a fair comparison it is crucial to extent the current method to online learning. The GP-based motion planner has the potential to be less conservative, which is a huge benefit in crowded environments, while avoiding constant re-planning. It even has the potential to become less conservative by including feedback into the controller. Further research should also include validation of the method in more complex simulation environments and real-world scenarios.

Chapter 7

Conclusions and future work

This report concludes by summarizing the results and drawing main findings and conclusions from the discussed results. The limitations of the current implementation are highlighted and recommendations for improvement are provided for future work.

7-1 Summary

One of the main challenges in developing autonomous quadrotors that can execute complex missions is ensuring their safe navigation in real-world conditions, particularly in cluttered and complex environments like forests or urban areas where they are exposed to external wind disturbances. This research aims to improve the state of the art in safe quadrotor navigation in windy environments by using a Gaussian Process (GP) model to model wind disturbances and incorporating the learned model into a state-of-the-art Model Predictive Contouring Control (MPCC) [9] framework. The research is novel in that it is the first to validate the use of GP's to model external wind disturbances using quadrotor state information. Additionally, by incorporating the trained GP model into the MPCC controller, it is possible to improve the performance and robustness of the controller, resulting in improved trajectory generation and tracking.

The MPCC controller is based on a model of the quadrotor, and the GP only captures differences in the model caused by external disturbances. Therefore, a nominal model of the quadrotor is derived using first principle modeling techniques. This research is the first to work with the Hovergames platform and make use of the attitude controller on board of the quadrotor. A first-order approximation of the attitude dynamics model is identified by sending step signals to the quadrotor and using the data to identify the model parameters for the Gazebo simulation and the real-world platform. Additionally, a linear thrust dynamics model is identified to convert the Pulse-Width Modulation (PWM) signal to the quadrotor into an acceleration. The attitude dynamics model has very good fit results, verifying that a first-order model is sufficient to describe the attitude dynamics. However, the thrust dynamics model fails to capture the underlying dynamics particularly in real-world experiments. While currently there is no focus on the z-axis and the MPCC can compensate for some model mismatch, future investigations should be conducted.

Based on the derived nominal model, the GP disturbance model is trained. Specifically, a wind disturbance map is trained with the quadrotor position as input and wind disturbance as output. The wind disturbance is estimated from the difference between the model's prediction at the next state and the actual next state. In a simple simulation with perfect matching models, the model mismatch can directly be attributed to the measured disturbance. Collecting the data through an optimal experiment design using Active Learning (AL), exploration times are reduced and overall uncertainty of the model is decreased. To reduce computation times, a sparse GP approach is used, making it feasible for use in subsequent controllers. The relevant training parameters such as the kernel, batch, and epoch size are discussed, and the trained disturbance map for two wind scenarios is provided for use in subsequent sections. The trained disturbance map is able to capture the actual wind disturbances with slight model mismatch in areas of rapidly changing wind. Despite this, all disturbances are captured within the uncertain bounds, with the exception of some extreme spikes in the data. Using a sparse GP approach can decrease computation times and make it usable for future controllers, but it may also limit the expressiveness of the model in capturing all spikes in the data. These extreme spikes, however, are unlikely to occur in real-world scenarios.

The trained wind disturbance map, along with the nominal model, is implemented in the MPCC to consider both the mean and uncertainty of the resulting probabilistic model. The uncertainty is approximately propagated using successive linearization and included into the MPCC controller through the use of ellipsoidal and chance constraints. Chance constraints are chosen to continue with for this research as they are shown to be less conservative. This implementation of the GP-based MPCC is shown to improve tracking and increases robustness of the motion planner and controller during obstacle avoidance. However, it should be noted that several assumptions or simplifications are made within the entire implementation, such as using a discrete model, propagating the uncertainty using linearization and propagating the uncertainty outside of the solver, which introduce other sources of uncertainty into the model. These should be considered during real-world applications by including a safety margin or increasing the process noise. Unfortunately, the real-time feasibility of the approach on the machine used to run the experiments has not yet been proven. This is mainly due to the original MPCC implementation not being real-time feasible, and also because propagating the uncertainty adds significant computation times to the controller.

The proposed GP-based MPCC implementation is compared to the external forces resilient safe motion planner proposed in [8] which is the state of the art for safe quadrotor navigation in windy and cluttered environments. The external forces resilient safe motion planner uses an external force estimator to estimate instantaneous forces acting on the quadrotor and combines motion planning and robust Model Predictive Control (MPC) in a re-planning framework to generate safe trajectories given information on the external force. When both navigation algorithms are tested in the same scenario, it is observed that the external forces resilient safe planner struggles with rapidly changing wind conditions. In contrast, the GP-based MPCC controller, which uses a trained disturbance map, performs well and does not show any difficulties. However, for a fair comparison, the GP-based MPCC controller should also be implemented online. If the GP-based MPCC controller is implemented online, it is still expected to perform better than the external forces resilient planner as it has access to a model of the disturbance and information about it before it can be encountered. Additionally, the resilient planner, despite its feedback in the controller, is more conservative, creating larger safety margins and stricter constraints, resulting in the controller taking a safer, longer route to reach the goal points in cluttered environments, rather than the direct route. This highlights the superior performance of the GP-based MPCC approach in cluttered environments.

7-2 Conclusions

This research shows that Gaussian Processes (GP) can effectively model the external wind disturbance map, even in the presence of complex wind conditions, using a simple simulation. Additionally, the optimal experiment design developed for gathering data on the wind disturbance map may have other potential applications. It is also found that incorporating the GP model into the controller does not significantly increase computation time, making the nominal GP model feasible for real-time implementation. The thesis also provides an explanation of how GP can be integrated into the MPCC controller to improve tracking and increase robustness during obstacle avoidance, and includes validation of the theoretical design. Initial comparisons with state-of-the-art methods suggest that the GP-based MPCC controller has the potential to handle more complex wind disturbances while being less conservative.

Nevertheless, this research also has some drawbacks to consider. One limitation is that the simulation used is very simple and does not fully reflect real-world scenarios. As noted earlier, the identification of the thrust model suggests that a perfect match with the real system cannot be expected. Also, the wind fields in the real-world will likely be more complex and may also change over time. The research has also made a number of simplifying assumptions, which could affect the results when applied to real-world scenarios, particularly when it comes to strict obstacle avoidance. Another issue to keep in mind is that the long computation times of the MPCC with uncertainty can be a challenge, particularly when uncertainty is included. Additionally, this research uses offline learning methods, which are not fair to compare with online learning.

The current implementation of this research serves as a first step towards the use of a GP model for wind disturbances in quadrotor flight. It is a feasibility study that shows that it is generally possible to use GP to model wind disturbances for quadrotor flight, and it also discusses various aspects that are relevant to successfully train the disturbance map and include it into the controller formulation. The potential of this method to improve the performance and safety of quadrotor flight is significant. However, it should be noted that the current implementation is very basic and not feasible for online exploration. Future work should build on this research, with a more detailed and advanced implementation. This is discussed hereafter.

7-3 Recommendations for future work

The following points are recommended as a follow-up on the research presented in this thesis.

Test in more complex simulation and real-world scenarios The proposed method for training a wind disturbance map and testing the GP-based MPCC has only been validated in a simple simulation scenario with perfectly matching models. It is recommended to test the method in more complex scenarios, both in simulation and on a real quadrotor platform, where model mismatch and noise are present. Initial tests in more complex simulation environments and real-world experiments to collect wind disturbance data indicate that the drag of the system also needs to be identified. Additionally, it must be confirmed that estimating external forces from state information is feasible on a physical platform, otherwise alternative force estimators such as VID-fusion [62] should be considered. Furthermore, the algorithm should be tested with real wind to determine if it can capture the stochastic behavior of real wind in a model.

Ensure real-time feasibility Several steps are proposed to make the algorithm real-time feasible. The first step is to speed up the computation of the basic MPCC by identifying the cause of the slow computation. Secondly, the way that the uncertainty is propagated can be sped up by integrating the uncertainty propagation code in C++. Lastly, it is recommended to look into the algorithm presented by [60] as it provides a way to propagate the uncertainty insight the solver while ensuring real-time feasibility. Additionally, it may be beneficial to consider using a different solver than FORCES PRO, as the researchers in the cited paper used the acados solver with Python.

Revisit MPCC algorithm In addition to improving the computation times of the MPCC controller, other improvements are also recommended. To ensure the performance of the MPCC in more realistic scenarios, it is suggested to extend the obstacle avoidance constraints to other shapes of obstacles. Furthermore, incorporating feedback into the controller can reduce uncertainty and further decrease conservatism. The current setup of the controller finds the closest distance to the obstacle and barely avoids it when the path crosses through an obstacle, which can be risky, especially if there is model mismatch. To address this, a prior path planner can be used to construct a more realistic path, and then the GP-based MPCC controller can be used to ensure performance and safety.

Extension to online learning methods Incorporating online learning of the GP model into the MPCC controller would allow for the exploration of new environments and provide a more realistic comparison to other quadrotor navigation algorithms such as the external forces resilient planner [8]. It could also help with the reduced expressiveness of the sparse GP model when covering less of the environment. Online learning has been implemented for other applications such as race cars and robotic arms, see e.g. [44], [63]. Additionally, this extension would also enable the controller to adapt to time-varying wind fields.

Explore other applications The primary objective of this research is to enhance the safety of quadrotor navigation in windy and cluttered environments by incorporating a GP model of the wind. Initially, a wind disturbance map is trained to demonstrate the feasibility of the approach. However, the ultimate goal is to extend the method to include online learning, allowing the controller to adapt to time-varying wind fields during exploration. However, as discussed earlier, there are still several potential issues that may arise in the real-world implementation of the proposed algorithm. Therefore, it is important to consider other applications where the current setup of training a wind disturbance map could be useful, such as in quadrotor swarm applications for the maintenance of wind parks [64] or firefighting [65]. In such scenarios, one or more quadrotors could explore the environment and make the wind disturbance map available to other quadrotors, thus improving their performance and safety when performing their task.

Appendix A

Custom wind plugin

The wind plugin for the simple simulation is inspired by the wind plugin for the RotorS simulator, whose description can be found here.

Grid format and interpolation

The wind data is defined on a specified grid in 3D, where each grid point specifies the magnitude of the wind in x-, y- and z-direction. The data is interpolated using a 3D interpolation scheme as described in the wind plugin for the RotorS simulator.

Wind file text format

To specify a custom wind field, a text file is loaded that contains information about the grid geometry and wind values. The data needed in the text file consists of:

- The smallest x-coordinate of the grid `min_x`
- The smallest y-coordinate of the grid `min_x`
- The number of grid points in x-direction `n_x`
- The number of grid points in y-direction `n_y`
- The resolution of the grid in x-direction `res_x`
- The resolution of the grid in y-direction `res_y`
- The (n_z)-dimensional array of `vertical_spacing_factors` in z-direction. These are float values between 0 for the lowest and 1 for the highest point.
- The altitude of each grid point contained in the lower x-y plane `bottom_z`

- The altitude of each grid point contained in the upper x-y plane `top_z`
- The $(n_x \times n_y \times n_z)$-dimensional array of wind speed for each grid point in x-direction `u`, in y-direction `v`, and in z-direction `w`.

Code to generate fan scenario

The code to generate the fan scenario was custom written to imitate fans distributed in the environment. It is not based on any scientific wind properties as this would be out of the scope of this research but rather aims to imitate the fans which have a certain initial wind speed, that decreases further away from the fan while also spreading out from the initial dimensions of the fan. The code written supports to add as many fans as desired, by specifying the fan properties in a list as shown in the code below. For each fan, one has to specify the position of the fan, the size of the fan, the wind direction in x and y, the initial strength, how much the fan can spread and also how fast the initial speed is decreased.

```
fans = {
    'fan1': {
        "x_pos": -10,
        "y_pos": 0,
        "size": 2,
        "x_dir": 0,
        "y_dir": 1,
        "strength": 5,
        "spread": 0.25,
        "decrease": 0.25,
    }
    'fan2': {
            "x_pos": -10,
            "y_pos": 0,
            "size": 2,
            "x_dir": 0,
            "y_dir": 1,
            "strength": 3,
            "spread": 0.15,
            "decrease": 0.15,
        }
}
```

With that information, for each fan of the list, the grid information is generated as follows (also see the code hereafter): for the initial speed and size of the fan, a quadratic polynomial is fitted such that it has the maximum strength at the center of the fan and is zero or negative outside of the defined fan region. For each grid dimension (here in y, but it would equally work for x), the polynomial is fitted and negative values are corrected to zero. Afterward, the start and end point of the fan are increased according to the defined spread and the maximum strength is lowered according to the defined decrease. The process of fitting a polynomial to the new data row is repeated, until the whole x-y-grid is filled. Then, this matrix is rotated and shifted to place the fan at the desired location with the desired rotation. For this application the z-dimension is one, but the code could be extended to also allow rotation and shift in z. The data is placed in the wind speed fields u, v, and w and stored to a text file together with the other relevant information discussed above.

```
start = - fan_parameters["size"] / 2
end = fan_parameters["size"] / 2
max_strength = fan_parameters["strength"]
# For one row of the grid
for i in range(n_y):
    # Fit a quadratic polynomial to the start and end point
    # with the maximum at the maximum strength value
    x_fit = np.array([start, 0, end])
    y_fit = np.array([0, max_strength, 0])
    z = np.polyfit(x_fit, y_fit, 2)
    p = np.poly1d(z)
    # Define grid row and fit the data
    y_in = np.arange(min_y + res_y / 2, min_y + dim_y + res_y / 2, res_y)
    y_out = p(y_in)
    # Correct negative values to zero
    y_out[y_out < 0] = 0
    # Append data row to grid
    mat = np.append(mat, y_out)
    # Spread the fan for next row
    start = start - res_x * fan_parameters["spread"]
    end = end + res_x * fan_parameters["spread"]
    # Decrease the strength for next row
    max_strength = max_strength - res_x * fan_parameters["decrease"]
# Reshape to marix format
wind_mat = np.reshape(mat, (n_x, n_y))
# Rotate matrix to have the correct direction of the fan
rot = math.atan2(fan_parameters["y_dir"],fan_parameters["x_dir"]) * 180 /
     math.pi
wind_mat_rot = ndimage.rotate(wind_mat, rot, reshape=False)
# Shift to the correct starting position
max_value = np.where(wind_mat_rot == np.amax(wind_mat_rot))
x_shift = math.floor(fan_parameters["y_pos"] - (max_value[0] * res_x +
    min_x))/res_x
y_shift = math.floor(fan_parameters["x_pos"] - (max_value[1] * res_y +
    min_y))/res_y
wind_mat_shift = ndimage.shift(wind_mat_rot, (x_shift, y_shift))
```

Loading the wind field

The generated text file can be passed on the the simple simulator, which reads the file making use of the custom written wind plugin. For future projects the wind plugin was also implemented and texted for the Gazebo environment which reads the same wind field files.

Appendix B

Travelling salesman problem

The Traveling Salesman Problem (TSP) is an algorithmic problem tasked with finding the shortest route between a set of points and locations that must be visited. In this thesis, it is used to find the most efficient route for the quadrotor to travel between different points in the environment. The TSP is NP-hard, meaning that it cannot be solved in polynomial time, but several solutions have been proposed to solve it.

In this thesis, Dynamic programming is used to solve the TSP to find the optimal route connecting the ten optimal points to visit to maximize the uncertainty of the Gaussian Process (GP). Specifically, `pyhton-tsp` is used, which is a library written in Python for solving typical TSP problems.

The code to solve the TSP problem is shown below. First, the distance between all 10 points has to be computed and stored it in a matrix, which is done by the `create_distance_matrix` function. Then, the library solves the dynamic programming problem to find the optimal permutation of points. The points are then rearranged according to the permutation and returned.

```python
from itertools import permutations
from math import dist
import numpy as np
from python_tsp.exact import solve_tsp_dynamic_programming
from hovergames_simplesim_control import helpers

def create_distance_matrix(X_path):
    distance_matrix = np.zeros((X_path.shape[0],X_path.shape[0]))
    for i in range(X_path.shape[0]):
        for j in range(X_path.shape[0]):
            distance = helpers.compute_coordinate_distance(X_path[i],
                X_path[j])
            distance_matrix[i,j] = distance
    return distance_matrix

def compute_permutation(X_path):
    distance_matrix = create_distance_matrix(X_path)
    permutation, distance = solve_tsp_dynamic_programming(distance_matrix
        )
    return permutation

def solve_tsp(X_path, X_last):
    X = np.vstack((X_last, X_path)) #Add the current last point to TSP
    permutation = compute_permutation(X)
    X_tsp = X[permutation,:]
    X_tsp = X_tsp[1:,:] #Exclude the first point from list again
    return X_tsp
```

Appendix C

Uncertainty propagation for nonlinear systems

Inspired by the derivations in the appendix [66] the mean and uncertainty can be propagated using the law of iterated expectations and the law of total variance, which are repeated here for the readers information.

The system model is

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) + B_d \left(\mathbf{d}(\mathbf{p}_k) + \mathbf{w}_k\right). \tag{C-1}$$

and the mean and variance are denoted by $\boldsymbol{\mu}_k^{\mathbf{x}}$ and $\Sigma_k^{\mathbf{x}}$, respectively. Using the law of iterated expectation, we have:

$$\boldsymbol{\mu}_{k+1}^{\mathbf{x}} = \mathbb{E}_{\mathbf{x}_k}\Big(\mathbb{E}_{\mathbf{d}|\mathbf{x}_k}\left(\mathbf{x}_{k+1}\right)\Big) \tag{C-2a}$$

$$\boldsymbol{\mu}_{k+1}^{\mathbf{x}} = \mathbb{E}_{\mathbf{x}_k}\Big(f(\mathbf{x}_k, \mathbf{u}_k) + B_d\boldsymbol{\mu}_k^{\mathbf{d}}(\mathbf{p}_k)\Big) \tag{C-2b}$$

Similarly, for the law of total variance, we have:

$$\Sigma_{k+1}^{\mathbf{x}} = \mathbb{E}_{\mathbf{x}_k}\Big(\mathrm{var}_{\mathbf{d}|\mathbf{x}_k}\left(\mathbf{x}_{k+1}\right)\Big) + \mathrm{var}_{\mathbf{x}_k}\Big(\mathbb{E}_{\mathbf{d}|\mathbf{x}_k}\left(\mathbf{x}_{k+1}\right)\Big) \tag{C-3a}$$

$$\Sigma_{k+1}^{\mathbf{x}} = \mathbb{E}_{\mathbf{x}_k}\Big(B_d\Sigma_k^{\mathbf{d}}(\mathbf{p}_k)B_d^T\Big) + \mathrm{var}_{\mathbf{x}_k}\Big(f(\mathbf{x}_k, \mathbf{u}_k) + B_d\boldsymbol{\mu}_k^{\mathbf{d}}(\mathbf{p}_k)\Big) \tag{C-3b}$$

Bibliography

[1] G. Loianno and D. Scaramuzza, "Special issue on future challenges and opportunities in vision-based drone navigation," *Journal of Field Robotics*, vol. 37, no. 4, pp. 495–496, 2020.

[2] E. J. Smeur, G. C. de Croon, and Q. Chu, "Cascaded incremental nonlinear dynamic inversion for MAV disturbance rejection," *Control Engineering Practice*, vol. 73, pp. 79–90, 2018.

[3] D. Hentzen, T. Stastny, R. Siegwart, and R. Brockers, "Disturbance estimation and rejection for high-precision multirotor position control," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2797–2804, IEEE, 2019.

[4] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control*, pp. 207–226, Springer, 1999.

[5] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, vol. 2. MIT press Cambridge, MA, 2006.

[6] G. Torrente, E. Kaufmann, P. Föhn, and D. Scaramuzza, "Data-driven MPC for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3769–3776, 2021.

[7] S. Kousik, S. Vaskov, F. Bu, M. Johnson-Roberson, and R. Vasudevan, "Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots," *The International Journal of Robotics Research*, vol. 39, no. 12, pp. 1419–1469, 2020.

[8] Y. Wu, Z. Ding, C. Xu, and F. Gao, "External forces resilient safe motion planning for quadrotor," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 8506–8513, 2021.

[9] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, "Model predictive contouring control for collision avoidance in unstructured dynamic environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4459–4466, 2019.

[10] HoverGames, "https://www.hovergames.com/."

[11] H. Zhu and J. Alonso-Mora, “Chance-constrained collision avoidance for MAVs in dynamic environments,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, 2019.

[12] L. Quan, L. Han, B. Zhou, S. Shen, and F. Gao, “Survey of UAV motion planning,” *IET Cyber-systems and Robotics*, vol. 2, no. 1, pp. 14–21, 2020.

[13] C. Goerzen, Z. Kong, and B. Mettler, “A survey of motion planning algorithms from the perspective of autonomous UAV guidance,” *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1, pp. 65–100, 2010.

[14] R. Penicka and D. Scaramuzza, “Minimum-time quadrotor waypoint flight in cluttered environments,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5719–5726, 2022.

[15] T. P. Nascimento and M. Saska, “Position and attitude control of multi-rotor aerial vehicles: A survey,” *Annual Reviews in Control*, vol. 48, pp. 129–146, 2019.

[16] A. Zulu and S. John, “A review of control algorithms for autonomous quadrotors,” *arXiv preprint arXiv:1602.02622*, 2016.

[17] B. J. Emran and H. Najjaran, “A review of quadrotor: An underactuated mechanical system,” *Annual Reviews in Control*, vol. 46, pp. 165–180, 2018.

[18] L. Li, L. Sun, and J. Jin, “Survey of advances in control algorithms of quadrotor unmanned aerial vehicle,” in *2015 IEEE 16th international conference on communication technology (ICCT)*, pp. 107–111, IEEE, 2015.

[19] F. Kendoul, “Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems,” *Journal of Field Robotics*, vol. 29, no. 2, pp. 315–378, 2012.

[20] H. Nguyen, M. Kamel, K. Alexis, and R. Siegwart, “Model predictive control for micro aerial vehicles: A survey,” in *2021 European Control Conference (ECC)*, pp. 1556–1563, IEEE, 2021.

[21] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*, vol. 2. Nob Hill Publishing Madison, 2017.

[22] D. Lam, C. Manzie, and M. Good, “Model predictive contouring control,” in *49th IEEE Conference on Decision and Control (CDC)*, pp. 6137–6142, IEEE, 2010.

[23] W. Schwarting, J. Alonso-Mora, L. Paull, S. Karaman, and D. Rus, “Safe nonlinear trajectory generation for parallel autonomy with a dynamic vehicle model,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 9, pp. 2994–3008, 2017.

[24] T. Nägeli, L. Meier, A. Domahidi, J. Alonso-Mora, and O. Hilliges, “Real-time planning for automated multi-view drone cinematography,” *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 1–10, 2017.

[25] A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, “Model predictive contouring control for near-time-optimal quadrotor flight,” *arXiv preprint arXiv:2108.13205*, 2021.

[26] N. Sydney, B. Smyth, and D. A. Paley, "Dynamic control of autonomous quadrotor flight in an estimated wind field," in *52nd IEEE Conference on Decision and Control*, pp. 3609–3616, IEEE, 2013.

[27] T. Tomić and S. Haddadin, "A unified framework for external wrench estimation, interaction control and collision reflexes for flying robots," in *2014 IEEE/RSJ international conference on intelligent robots and systems*, pp. 4197–4204, IEEE, 2014.

[28] S. Waslander and C. Wang, "Wind disturbance estimation and rejection for quadrotor position control," in *AIAA Infotech@ Aerospace conference and AIAA unmanned... Unlimited conference*, p. 1983, 2009.

[29] C. T. Ton and W. Mackunis, "Robust attitude tracking control of a quadrotor helicopter in the presence of uncertainty," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 937–942, IEEE, 2012.

[30] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.

[31] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "FaSTrack: A modular framework for fast and guaranteed safe motion planning," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 1517–1522, IEEE, 2017.

[32] M. Kamel, M. Burri, and R. Siegwart, "Linear vs nonlinear MPC for trajectory tracking applied to rotary wing micro aerial vehicles," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3463–3469, 2017.

[33] S. Singh, B. Landry, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust feedback motion planning via contraction theory," *The International Journal of Robotics Research*, 2019.

[34] L. Blackmore, M. Ono, and B. C. Williams, "Chance-constrained optimal path planning with obstacles," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1080–1094, 2011.

[35] M. Lorenzen, M. Cannon, and F. Allgöwer, "Robust MPC with recursive model update," *Automatica*, vol. 103, pp. 461–471, 2019.

[36] A. Didier, K. P. Wabersich, and M. N. Zeilinger, "Adaptive model predictive safety certification for learning-based control," in *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 809–815, IEEE, 2021.

[37] A. Didier, A. Parsi, J. Coulson, and R. S. Smith, "Robust adaptive model predictive control of quadrotors," in *2021 European Control Conference (ECC)*, pp. 657–662, IEEE, 2021.

[38] M. O'Connell, G. Shi, X. Shi, and S.-J. Chung, "Meta-learning-based robust adaptive flight control under uncertain wind conditions," *arXiv preprint arXiv:2103.01932*, 2021.

[39] G. Garimella, M. Sheckells, and M. Kobilarov, "Robust obstacle avoidance for aerial platforms using adaptive model predictive control," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5876–5882, IEEE, 2017.

[40] M. Mehndiratta and E. Kayacan, “Gaussian process-based learning control of aerial robots for precise visualization of geological outcrops,” in *2020 European Control Conference (ECC)*, pp. 10–16, IEEE, 2020.

[41] A. Tagliabue and J. P. How, “Airflow-inertial odometry for resilient state estimation on multirotors,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5736–5743, IEEE, 2021.

[42] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot, “Robust constrained learning-based NMPC enabling reliable mobile robot path tracking,” *The International Journal of Robotics Research*, vol. 35, no. 13, pp. 1547–1563, 2016.

[43] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, “Learning-based model predictive control for autonomous racing,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[44] A. Carron, E. Arcari, M. Wermelinger, L. Hewing, M. Hutter, and M. N. Zeilinger, “Data-driven model predictive control for trajectory tracking with a robotic arm,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3758–3765, 2019.

[45] R. Amin, L. Aijun, and S. Shamshirband, “A review of quadrotor UAV: control methodologies and performance evaluation,” *International Journal of Automation and Control*, vol. 10, no. 2, pp. 87–103, 2016.

[46] T. Keviczky and G. J. Balas, “Receding horizon control of an F-16 aircraft: A comparative study,” *Control Engineering Practice*, vol. 14, no. 9, pp. 1023–1033, 2006.

[47] R. Mahony, V. Kumar, and P. Corke, “Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor,” *IEEE Robotics and Automation magazine*, vol. 19, no. 3, pp. 20–32, 2012.

[48] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart, “Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system,” in *Robot operating system (ROS)*, pp. 3–39, Springer, 2017.

[49] P. Autopilot, “https://px4.io/.”

[50] M. P. Deisenroth, *Efficient reinforcement learning using Gaussian processes*, vol. 9. KIT Scientific Publishing, 2010.

[51] M. Liu, G. Chowdhary, B. C. Da Silva, S.-Y. Liu, and J. P. How, “Gaussian processes for learning and control: A tutorial with examples,” *IEEE Control Systems Magazine*, vol. 38, no. 5, pp. 53–86, 2018.

[52] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson, “GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration,” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 7587–7597, 2018.

[53] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,

M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[54] M. Titsias, "Variational learning of inducing variables in sparse Gaussian processes," in *Artificial intelligence and statistics*, pp. 567–574, PMLR, 2009.

[55] H. Liu, J. Cai, and Y.-S. Ong, "Remarks on multi-output Gaussian process regression," *Knowledge-Based Systems*, vol. 144, pp. 102–121, 2018.

[56] K.-H. Huang, "DeepAL: Deep active learning in Python," *arXiv preprint arXiv:2111.15258*, 2021.

[57] L. Ferranti, B. Brito, E. Pool, Y. Zheng, R. M. Ensing, R. Happee, B. Shyrokau, J. F. Kooij, J. Alonso-Mora, and D. M. Gavrila, "Safevru: A research platform for the interaction of self-driving vehicles with vulnerable road users," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1660–1666, IEEE, 2019.

[58] E. C. Kerrigan and J. M. Maciejowski, "Soft constraints and exact penalty functions in model predictive control," 2000.

[59] A. Domahidi and J. Jerez, "FORCES professional, embotech GmbH (http://embotech.com/forces-pro)," Jul. 2014.

[60] A. Lahr, A. Zanelli, A. Carron, and M. N. Zeilinger, "Zero-order optimization for Gaussian process-based model predictive control," *arXiv preprint arXiv:2211.15522*, 2022.

[61] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, 2021.

[62] Z. Ding, T. Yang, K. Zhang, C. Xu, and F. Gao, "Vid-fusion: Robust visual-inertial-dynamics odometry for accurate external force estimation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14469–14475, IEEE, 2021.

[63] L. Hewing and M. N. Zeilinger, "Stochastic model predictive control for linear systems using probabilistic reachable sets," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 5182–5188, IEEE, 2018.

[64] M. H. Nordin, S. Sharma, A. Khan, M. Gianni, S. Rajendran, and R. Sutton, "Collaborative unmanned vehicles for inspection, maintenance, and repairs of offshore wind turbines," *Drones*, vol. 6, no. 6, p. 137, 2022.

[65] J. J. Roldán-Gómez, E. González-Gironda, and A. Barrientos, "A survey on robotic technologies for forest firefighting: Applying drone swarms to improve firefighters' efficiency and safety," *Applied Sciences*, vol. 11, no. 1, p. 363, 2021.

[66] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using Gaussian process regression," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.

Glossary

List of Acronyms

EOM	Equations of Motion
PWM	Pulse-Width Modulation
ROS	Robot Operating System
SITL	Software in the Loop
RBS	Random Binary Signal
MAV	Micro Aerial Vehicle
MPC	Model Predictive Control
OCP	Optimal Control Problem
MPCC	Model Predictive Contouring Control
FRS	Forward Reachable Set
NMPC	Nonlinear Model Predictive Control
GP	Gaussian Process
GPR	Gaussian Process Regression
IMU	Inertial Measurement Unit
SE	Squared Exponential
TSP	Traveling Salesman Problem
ELBO	Evidence-Lower Bound
MSE	Mean Squared Error
AL	Active Learning
NLP	Nonlinear Programming
RK4	Runge-Kutta 4th

List of Symbols

β	Time delay
$\boldsymbol{\theta}$	Hyperparameter vector
χ	Confidence interval
$\dot{\phi}$	Roll rate
$\dot{\psi}$	Yaw rate
$\dot{\theta}$	Pitch rate
$\Gamma(\cdot)$	Gamma function
$\mathbf{d}$	Disturbance vector
$\mathbf{d}$	Disturbance vector
$\mathbf{d}(\mathbf{p}_k)$	Disturbance vector
$\mathbf{p}$	Position
$\mathbf{u}$	Inducing points
$\mathbf{u}$	Input space
$\mathbf{v}$	Speed
$\mathbf{w}_k$	Process noise
$\mathbf{x}$	State space
$\mathcal{B}$	Occupancy volume
$\mathcal{C}_o$	Collision region
$\mathcal{D}$	Dataset of observations
$\mathcal{I}_o$	Obstacle space
$\mathcal{U}$	Input constraint set
$\mathcal{X}$	State constraint set
$\mu(\mathbf{x})$	Mean
ν	Smoothness parameter of the Matern kernel
ϕ	Roll
ψ	Ellipsoid rotation
ψ	Yaw
θ	Pitch
ξ	Slack variable
$\{A\}$	World frame
$\{B\}$	Body frame
a	Major ellipsoid axis
a_T	Linear scaling
b	Minor ellipsoid axis
B_d	Mapping of the uncertainty
b_T	Linear scaling
$f(\mathbf{x})$	Nonlinear function
g	Gravitation constant

I	Inertia
k	Steady-state gain
k	Time step
$K(X, X)$	Kernel matrix
$k(\mathbf{x}, \mathbf{x}')$	Kernel function
$K_\nu(\cdot)$	Modified Bessel function
k_D*	Normalized drag coefficient
k_D	Drag coefficient
l_d	Horizontal length scale for output dimension d
m	Mass
N	Sampling points
n	Size of dataset
N_p	Prediction horizon
R	Rotation matrix
r	Radius
s	Path variable
T	Thrust
T_c	Thrust command
T_h	Hover thrust
T_s	Sampling time
U	Battery voltage
v_x	Velocity in x
v_y	Velocity in y
v_z	Velocity in z
v_ref	Reference velocity
v_k	Normalized velocity
x	Position in x
y	Position in y
z	Position in z
σ_ε	Noise covariance
σ_f	Output variance
τ	Time constant
e^c	Contour error
e^l	Lag error