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

Johanna Probst

Master of Science Thesis

TU Delft Delft University of Technology

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

January 27, 2023

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

Abstract

TODO:

- What is the problem? - What are existing methods? - What is my main contribution to this problem? / How am I solving this problem? - What are my key results? - How is it better than existing methods?

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor prof.dr.ir. M.Y. First Reader for his assistance during the writing of this thesis...

By the way, it might make sense to combine the Preface and the Acknowledgements. This is just a matter of taste, of course.

Delft, University of Technology
January 27, 2023

Johanna Probst

Chapter 1

Introduction

Unmanned aerial vehicles (UAVs), such as drones, are becoming increasingly important in our society. Tasks of UAVs include search and rescue, environment monitoring, security surveillance, transportation, and inspection [29]. Developing autonomous drones 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 drones are exposed to external wind disturbances. Especially in complex, cluttered environments such as forests or dense urban areas, unforeseen disturbances can cause the drone to deviate significantly from its course and potentially collide with nearby objects. These applications require the robot to be highly agile and capable of moving through and interacting with its environment while remaining close to obstacles. 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, and robust Model Predictive Control (MPC) methods such as tube-based MPC. 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 drone'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 over-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, 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 drone flight, 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 or MPC methods. Modifications to the standard MPC formulation such as the Local Model Predictive Contouring Control (LMPCC) controller presented by the researchers in [3] allow to combine motion planning and control into one Optimal Control Problem (OCP), ensuring safe trajectories.

1-2 Research objective

This research aims to explore GP-based modeling techniques for external wind disturbances acting on a drone to improve safety and performance for quadrotor navigation. Wind disturbances can be highly variable and non-linear. The aim is to explore how well GP's can capture these complex patterns in the data. The GP model of the wind, is not only capable of predicting future wind conditions but it also provides a way to quantify the uncertainty in the model predictions. By using this information of the wind disturbances, the goal is to improve tracking and performance, and also provide safety guarantees leveraging the information on the uncertainty. This way, it aims to bridge the gap between wind disturbance estimation methods and robust, model-based methods used to compensate for wind disturbances.

By combining trajectory generation and control into one optimal control problem, using a modified MPC formulation, the goal is to avoid unsafe trajectories as a control input and guarantee robustness in the planning and control stage, by including both the mean and uncertainty information of the GP model. Similar to existing robust MPC 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 stochastic MPC methods can further reduce conservatism by utilizing the probabilistic model of uncertainty provided by the GP model.

The following to research question will be answered in this thesis:

1. Are Gaussian processes a suitable method to model external wind disturbances using the quadrotor state information?
2. Is it possible to increase performance and robustness of the MPCC controller in windy environments 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:

- A map of the wind disturbances, represented by a GP, can be created by the quadrotor by exploring the environment beforehand.
- The obstacles in the environment are known a priori.
- A simplified simulation is used, in which the only disturbance is the wind disturbance.
- The dimensionality of the quadrotor environment is reduced from 3D to 2D.
- The attitude controller is given, therefore the focus is on trajectory planning and position control of the quadrotor.

1-3 Thesis contributions

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

System identification of the attitude dynamics model of the Hovergames platform This research is the first to work with the newly designed Hovergames platform, which is optimized for running computationally heavy algorithms on board of the drone. Utilizing the attitude controller provided by the flight controller onboard, the attitude dynamics model was 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 LMPCC algorithm with the GP model for improved tracking and increased robustness in drone platforms The existing LMPCC algorithm in [3] is extended to include the uncertain model including both the mean and uncertainty of the prediction model, by incorporating the chance constraints in [56]. The improved tracking and increased robustness of the GP-based LMPCC method compared to the standard formulation is demonstrated. Additionally, this is the first implementation of LMPCC on a drone platform.

Advanced trajectory generation and control algorithm for complex wind fields and cluttered environments By benchmarking the GP-based LMPCC 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 in this scenario.

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 used for the motion planning and control algorithm together with system identification performed to find relevant constants 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 to address existing drawbacks.

2-1 Background

This section presents an overview of current motion planners and quadrotor control techniques and ways to extend them for wind disturbance and uncertainty handling, including data-driven and machine learning methods. The problem statement and the ways this thesis will expand upon existing literature will be discussed after the overview.

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 power 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 [40].

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 [14]. Some trajectory planners also solve both problems simultaneously. Existing approaches for quadrotor trajectory planning include polynomial, discrete-time search, sampling-based, and optimization methods [39].

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 yaw rotation ψ_c. The attitude controller tracks the desired roll ϕ_c, pitch θ_c and yaw ψ_c by controlling the angular velocities $\dot{\phi}_\mathrm{c}$, $\dot{\theta}_\mathrm{c}$, $\dot{\psi}_\mathrm{c}$ and thrust T_c needed to keep the quadrotor in the air. The mixer is in place to translate the angular velocities and thrust command into individual rotor commands.

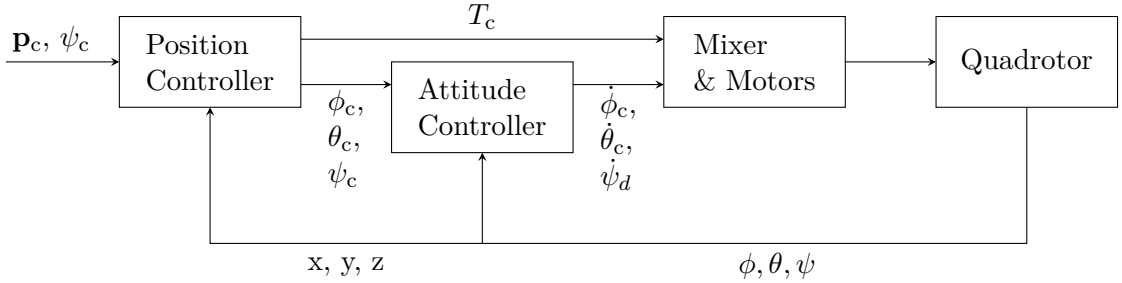


Figure 2-1: Cascaded control architecture of a common quadrotor controller.

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.. [34], [57], [10], [27], [22]). 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 [35].

Model predictive control Model-based methods, such as Model Predictive Control (MPC) [41], 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 advantage of MPC is the ability to impose constraints directly on the system.

Model predictive contouring control With optimization-based methods, it is also possible to combine both 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 key to solving this problem is to use an MPC algorithm that is modified to simultaneously solve the trajectory generation problem. One popular method to achieve this is Model Predictive Contouring Control (MPCC) [26]. MPCC has been applied for static and dynamic obstacle avoidance with ground vehicles [43] and autonomous ground robots [3]. In the context of quadrotor navigation it has found an application in quadrotor racing [42].

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 drone is equipped with airspeed sensors such as Pitot tubes, a local estimate of the wind can be determined directly [46]. In most cases, however, these sensors are not present on board the quadrotor. Instead, one could use the model of the quadrotor to estimate the wind based on the difference between desired and measured velocity or acceleration. This idea was used, e.g. for a nonlinear inversion-based position controller in [45]. The authors in [49] 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 [15]. In addition to the nominal disturbance force or nominal disturbance 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 [53]. 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 [50] 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. [24]. 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 [32] 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 was developed for robust offline trajectory planning in fully known environments and independent of the motion planner, called FasTrack [16]. 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 [24] uses Forward Reachable Set (FRS) that bound the tracking error and are computed offline. During online planning, the FRS are used to map obstacles to parameterized trajectories so that a safe trajectory can be selected. The authors in [55] 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 [20]. Additionally, the authors in [44] build upon the concept of robust MPC and employs control contraction metrics to create a feedback control law and a tube. Stochastic MPC formulations for collision and obstacle avoidance were first introduced by [2], where polytopic chance constraints were reformulated into deterministic constraints. This work was further advanced with the addition of quadratic obstacle avoidance constraints for dynamic collision avoidance as presented in [56].

2-1-3 Data-driven disturbance model

Data-driven disturbance estimation and rejection can be combined with more detailed disturbance models to create data-driven disturbance models. Techniques for data-driven modeling have been developed to enhance existing system models for various applications.

Set-membership identification A prominent method for learning a robust representation of the model uncertainty is set-membership identification [30], [7]. A practical implementation of set-membership identification together with model predictive control on a quadrotor is presented by the authors in [6]. Using set-membership identification, the drone is able to adapt parameters in uncertain wind conditions, while satisfying the constraints [6].

Neural networks The authors in [36] use meta-learning with a deep neural network to model the residual dynamics associated with wind disturbances on a quadrotor platform.

Gaussian processes Furthermore, Gaussian Process (GP) [54] models are used learn a stochastic model of the dynamics. In the context of quadrotor control, GPs have been used to describe the nonlinear dynamics and aerodynamic effects during aggressive flight maneuvers [51] and to guarantee robust obstacle avoidance [13]. Furthermore GPs are used to model wind uncertainties in [33]. Three independent GPs are trained based on the observed disturbances. Therefore, the approach can only respond to disturbances if they have been observed. The work in [47] creates a wind map based on the quadrotor position fusing Inertial Measurement Unit (IMU) and airflow measurements. Moreover, GPs have been used successfully to model residual dynamics for mobile robots [37], race cars [19] and a robotic manipulator [4].

2-2 Problem formulation

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 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 becoming 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. As MPCC is already using a nonlinear formulation for the Optimal Control Problem (OCP), it was decided to use a nonlinear model to describe the nominal dynamics of the quadrotor. Since this research is focused on the navigation domain, the attitude controller already implemented on board the drone will be used, so the priority will be on position control. This also introduces a separation layer to keep critical code running even if there is a failure in the high-level computer [20]. The attitude dynamics are approximated by a linear-first order model. This model is derived using system identification.

2-2-2 Data-driven wind disturbance model

As there are no additional wind sensors mounted on the drone 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.

Research goals This research primarily aims to examine the efficiency 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 wind disturbance model's probabilistic nature results in a mean and uncertain state description, which is integrated into the MPCC framework. The local model predictive contouring control formulation in [3] is enhanced with the uncertain formulation, 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, a chance-constrained formulation is employed to ensure safe obstacle avoidance.

2-3 Problem Formulation

An overview of the problem formulation is given in Figure TODO. The center of the proposed methods is the Local Model Predictive Contouring Control (LMPCC) 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.

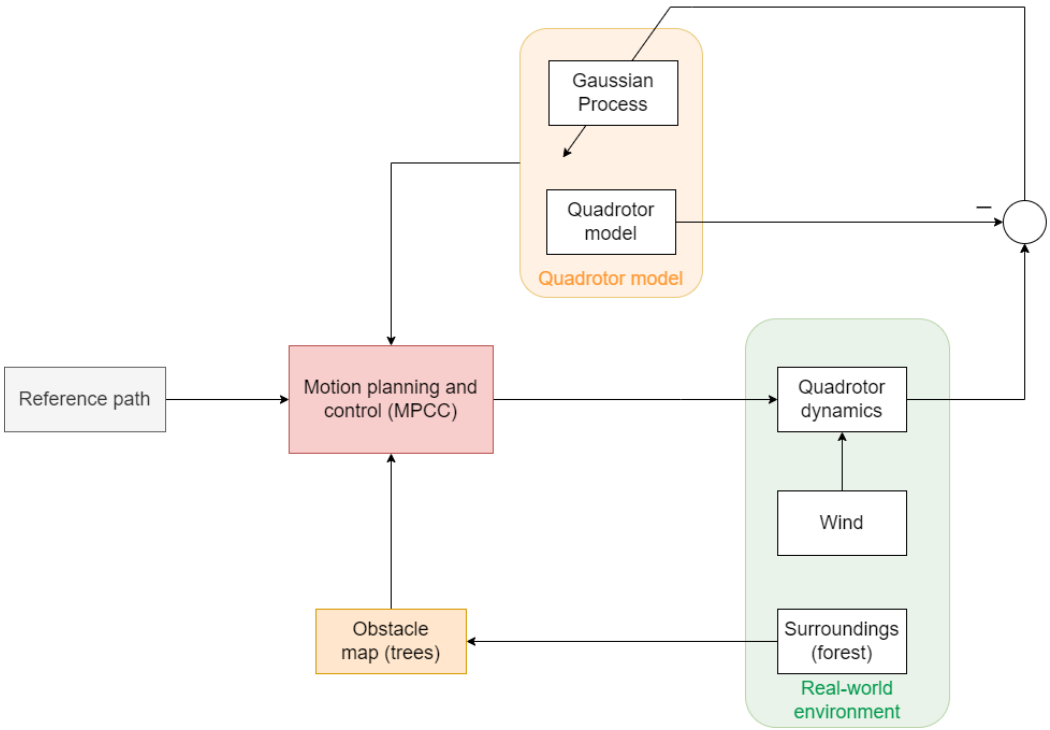


Figure 2-2: Overview of the learning-based motion planning and control method.

Given this information, the LMPCC 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

The following shortcomings were identified in current methods for quadrotor navigation in the presence of wind disturbances: The issue of identifying a feasible path and following that path is frequently addressed separately. When wind disturbances are taken into account, they are often only considered in one of these steps. As a result, existing motion planning and control methods are robust but overly conservative as they assume bounded disturbance or discard disturbances only once they have been observed.

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 LMPCC 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.

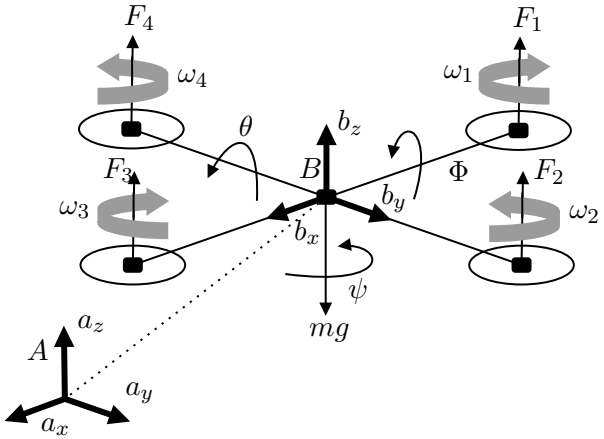


Figure 3-1: Quadrotor configuration with world frame A and body frame B.

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:

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

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

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:

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

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

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

Attitude model 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:

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

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

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

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

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 [31]. 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 $D = \text{diag}\,(k_D, k_D, k_D)$ and k_D is the drag coefficient.

Derived quadrotor model The resulting quadrotor model is summarized below:

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

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

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:

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

with the mass-normalized drag coefficient $k_D* = \frac{k_D}{m}$ [21].

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 drone.

3-2-1 Attitude dynamics and thrust model

The model of the attitude and thrust dynamics can take different forms depending on if the basic model is considered or if real 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 plus time delay For the real, 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.

Thrust dynamics model The attitude controller on board of the drone is designed to take a thrust command $T_\mathrm{PWM} = 0 \ldots 1$ that is translated into a Pulse-Width Modulation (PWM) signal to the motors, resulting in a desired rotor speed and thrust of the drone. However, for the model in 3-6, the thrust command T_c corresponds to an acceleration. The relation between the numeric thrust command and the physical thrust of the drone 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, corresponding to the hovering thrust T_h, are to be identified.

Thrust dynamics model for the physical drone For the real, physical drone the thrust relation is furthermore a function of the battery voltage. A third therm b_T is added to the thrust relation to account for that effect. It is linearized around the average battery voltage U at $\bar{U} = 14.5V$, resulting the thrust dynamics model:

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

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 drone 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.

Figure 3-2: The quadrotor platform used for this research [17].

Experimental setup

Physical experiments are performed on the HoverGames [17] drone provided as a development drone 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 [1]. 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 pose.

The HoverGames drone platform is moreover equipped with an Nvidia Jetson Xavier on-board processor that sends control commands to the physical drone 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.

Simulation environment

Before the code is tested on the actual drone, 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 drone. In the simulation environment, the drone is controlled via MAVROS

Figure 3-3: Hovergames drone flying in the lab environment.

commands. If the simulation dynamics match the physical dynamics, the controller code can therefore directly be transferred from the simulation to the drone.

Control inputs

The control inputs that are sent to the drone are the roll and pitch command, ϕ_c and θ_c and the thrust command T_c. 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 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. 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 drone also if the control commands are sent simultaneously. The remaining attitude input commands are calculated by a LQR controller to stabilize the drone. The thrust command is calculated by a PID controller stabilizing the drone in z-direction.

Thrust control command In simulation, a blockwave is sent to the quadrotor varying the control command between 0 and 1 and the resulting acceleration of the quadrotor is measured

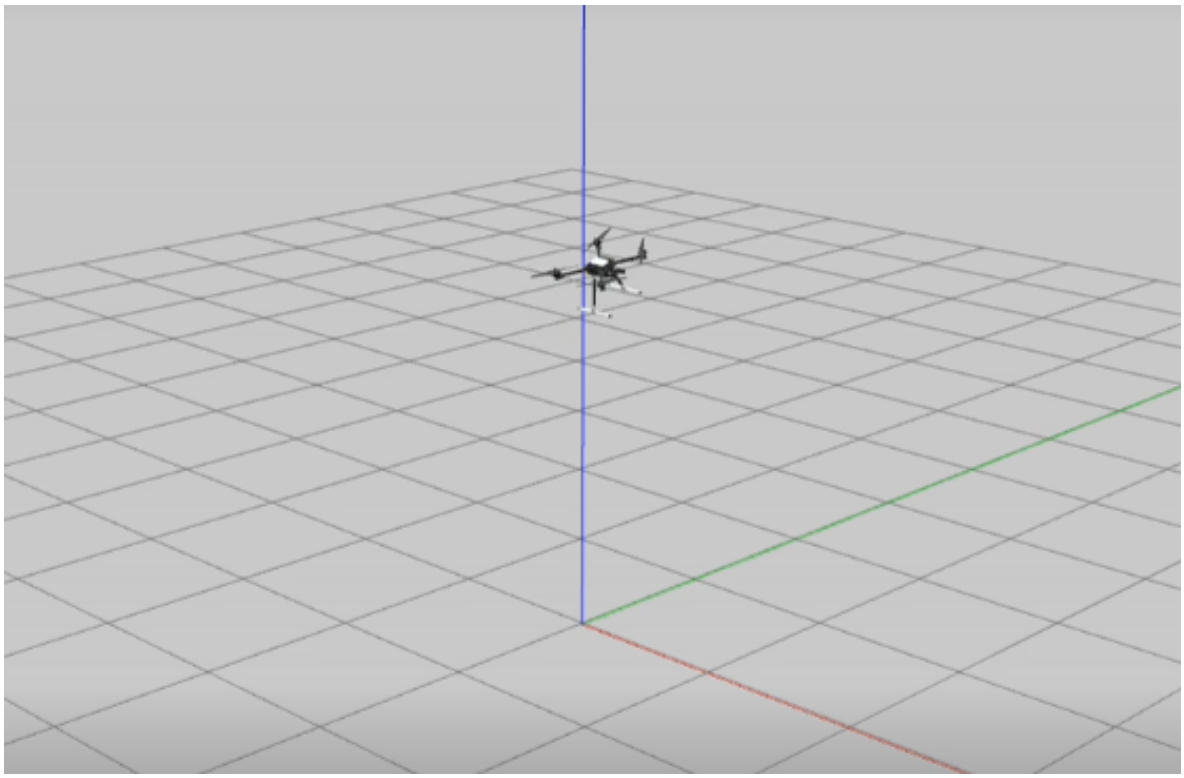


Figure 3-4: Hovergames drone flying in the Gazebo simulation environment.

to derive the linear relation. However, in the laboratory these kind of experiments are too risky given the limited space. Instead, the drone 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 drone is kept hovering, starting from a full battery, until the battery voltage is critically low. The thrust command T_{PMW} is automatically adjusted by the PX4 software to hover at different weights.

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 iddata object.

Thrust identification data In simulation, the recorded input is the thrust command and the output is the acceleration of the drone in z-direction. No further prepossessing is required. From the physical experiments, the thrust command and battery level are recorded and mapped to the respective accelerations, derived from the mass of the drone for each

experiment, where:

$$T_c = \frac{(m_t - m_n)}{m_t} g + g \tag{3-12}$$

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

3-2-3 System identification

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 measured data.

Roll and pitch model parameters The roll and pitch parameters are identified using the transfer function estimation function (tfest) in MATLAB. 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.

Roll and pitch model parameters with 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 dynamics model The linear thrust dynamics models are identified fitting a first degree polynomial to the data. For the real world dynamics, two first degree polynomial functions are fitted for the voltage and acceleration, respectively, and combined to find the model coefficients.

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 reflected by the goodness of fit, corresponding to $(1 - \text{NRMSE})100\%$, where NRMSE is the Normalized Root Mean Squared Error (NRMSE)Normalized Root Mean Squared Error between the predicted and measured output of the model.

3-2-4 Results and Discussion

The identified models for the first order attitude model including a time delay and the thrust dynamics model can be found for the simulation in Table 3-1 and for the real-world experiment in Table 3-2.

Identified simulation parameters In simulation the drone 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 $0.8s$ was identified, which corresponds to 4 sampling intervals. The drone hovers at a PWM command of 0.657, corresponding to a thrust acceleration of $T_c = 0$. The large scaling factor a_T implies that even small changes of T_{PWM}, result in large accelerations of the drone.

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

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

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. The left plot shows the behavior of the system for one of the data sets it was trained with while the right plot shows the input response to previously unseen validation data. 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.

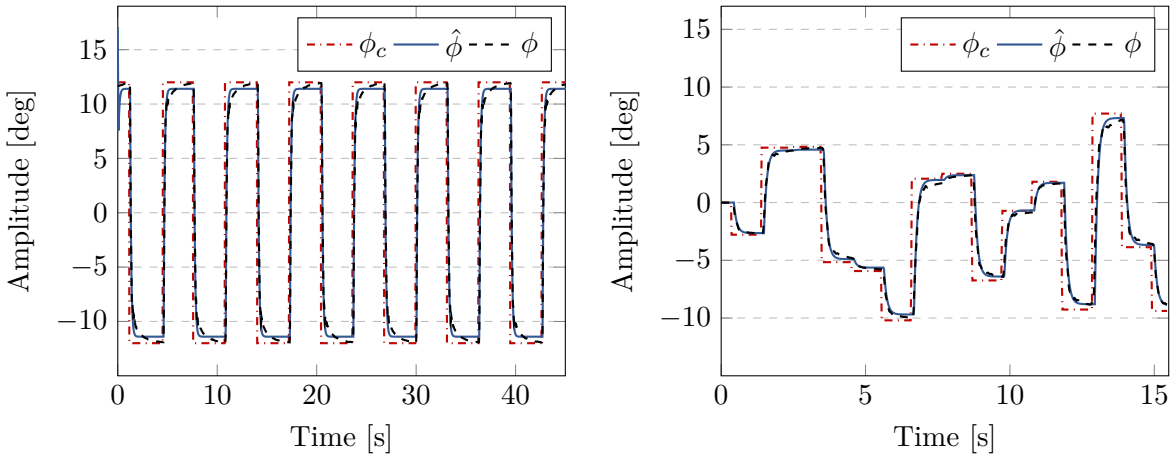


Figure 3-5: Input response of the roll and pitch dynamics in simulation for training data (left) and validation data (right).

Thrust dynamics comparison in simulation For the thrust dynamics model in simulation, the input command T_{PWM} between 0 and 1 is scaled according to the identified hovering thrust and linear scaling factor to find the thrust command $\hat{T}_\text{c}$. It compared to the measured

acceleration of the system $\dot{v}_z$, which corresponds to T_c, as the roll and pitch are zero during these experiments. This is shown in Figure 3-6. Again, the training data on the left is compared with a validation data set on the right. The goodness of fit for the simulation is 84.59% for the validation data and 57.86%. 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 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.

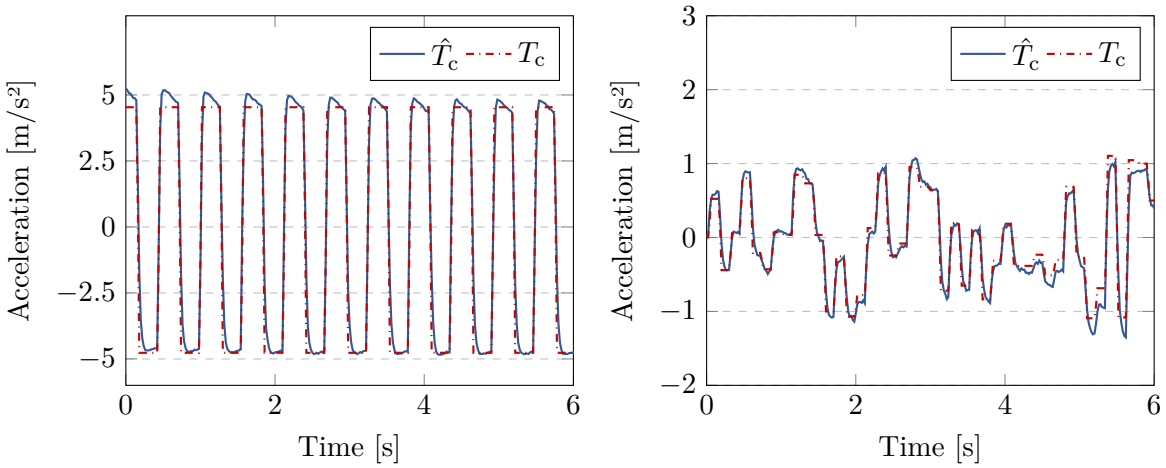


Figure 3-6: Fitted thrust dynamics model in simulation for training data (left) and validation data (right).

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 drone due to on-board sensors and therefore also slightly 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. The parameters differ slightly from each other for roll and pitch and also from the parameter values found in simulation. While the gain constants are almost equal, 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 drone. 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.

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

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

Input response comparison for the real-world dynamics The input responses in roll and pitch direction on the real drone are shown in Figure 3-7 and 3-8, for the training data

on the left and validation data on the right. 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 roll dynamics.

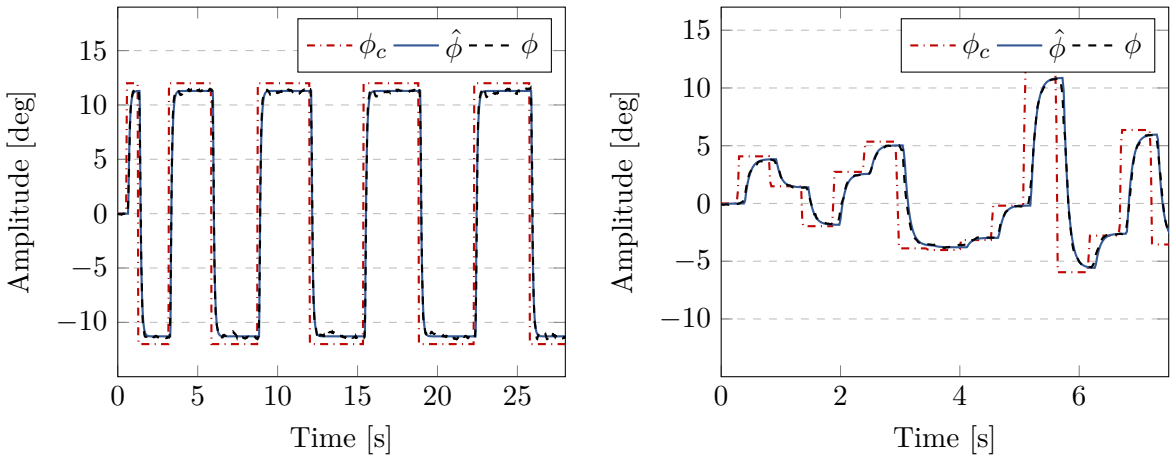


Figure 3-7: Input response of the real-world roll dynamics in simulation for training data (left) and validation data (right).

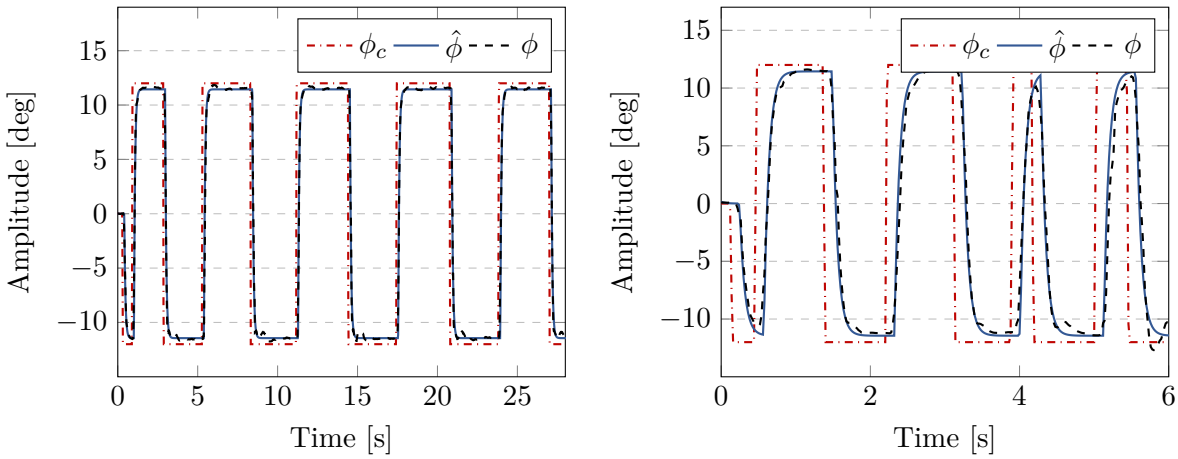


Figure 3-8: Input response of the real-world pitch dynamics in simulation for training data (left) and validation data (right).

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 over the battery voltage for different weights, i.e. accelerations. The lower the voltage, the higher the needed thrust command, and the higher the weight/acceleration, the higher the thrust command. In this case $T_{\mathrm{d},1}$ indicates the thrust curve without any added weight. For $T_{\mathrm{d},2}...T_{\mathrm{d},6}$, 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%.

Real-world thrust dynamics validation However, when comparing the model to valida-

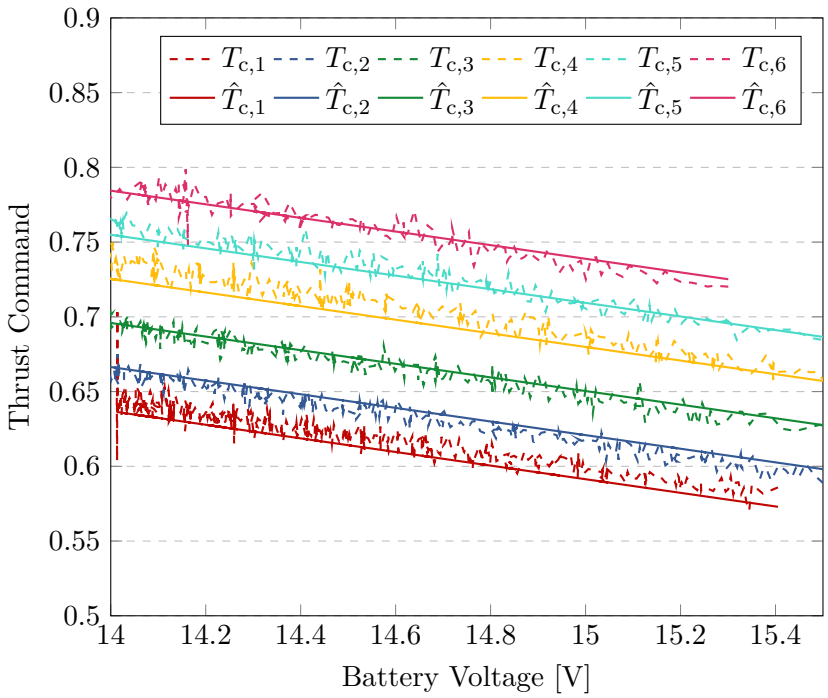


Figure 3-9: Real-world thrust dynamics for training data curves (dashed) and fitted linear thrust dynamics model (solid).

tion data that was 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 was found good enough, as long as is does not show unstable or undesired behavior during experiments.

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

	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%

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

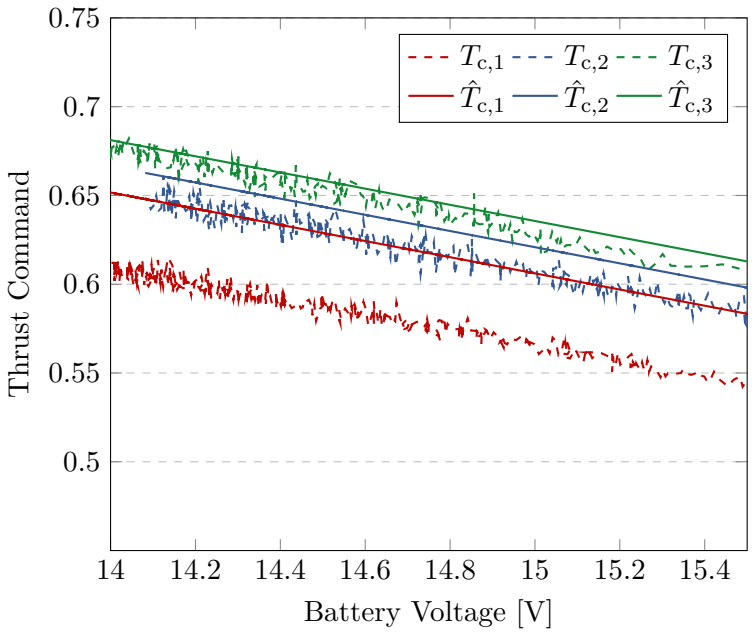


Figure 3-10: Real-world thrust dynamics for validation data curves (dashed) and fitted linear thrust dynamics model (solid).

3-3 Summary and discussion

The nominal model of the system dynamics can be derived using first-principle modeling techniques, resulting in approximate linear and nonlinear models of the quadrotor dynamics. Focusing on motion planning and position control, it is assumed that the attitude controller for the inner loop is already in place and stabilizes the drone. The inner loop attitude dynamics are approximated by a first-order linear model for the pitch and roll direction of the drone. The coefficients of the linear model are identified by sending step-input commands to the drone and fitting the linear model to the recorded output.

In identifying the model of attitude dynamics, very good results are obtained with goodness of fit values for the validation data over 90% in simulation and 85% on the real drone. This confirms that it is sufficiently accurate to approximate the attitude dynamics by a first-order model, even in the real environment. Differences between the real-world experiments and the simulation are in the time constants, meaning that the response of the physical system is slower. Also, a time delay of four to five sampling intervals was identified. While this can be considered in the model as well, it complicates the description of the system model, such that it is neglected for subsequent sections in which the model is used in a simplified simulation. However, when conducting experiments in a more complex simulation or physical environment, accounting for the time delay could be critical.

Additionally, identification of a thrust dynamics model is required since the controller input is a PWM command. In simulation, this identification is straightforward by sending step input PWM commands and fitting a linear model to the measured acceleration. The identification experiment gives good results, but the goodness of fit values are generally lower than for the attitude dynamics because the linear model cannot capture some of the more complex

underlying thrust dynamics.

The identification results of the real-world thrust dynamics is significantly worse. For these experiments, the drone was loaded with different weights that can be converted into an acceleration. While a model can be fitted for the training data, it fails for a large part of the validation data. Since the dynamics in z-direction are not very relevant for training the wind disturbance map, these results are accepted for now. The thrust dynamics model shall be validated during more experiments with the physical drone platform.

Chapter 4

Gaussian process disturbance map

The nominal model is extended by a Gaussian Process (GP) model to capture the external wind disturbances. The GP model was chosen because it can model any nonlinear function with little prior knowledge and can describe not only the nominal external perturbation but also the uncertainty of the trained model. Starting from the quadrotor position as input and the external wind disturbances as output, a wind disturbance map is trained. The background of Gaussian processes is first explained before the details of training the Gaussian process 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 motion planner and controller.

4-1 Gaussian Process Regression

The goal of Gaussian Process Regression (GPR) [54] is to obtain an approximation of the nonlinear function that describes the behavior of wind disturbances with uncertainty. An introduction is given to the mathematical foundation that GP are built on and how this foundation can be used to train and predict the nonlinear model. An explanation of sparse GPs to reduce computational costs and the idea of Active Learning (AL) with GPs to optimally capture the nonlinear perturbation map is also given.

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 $\mathbf{x}_*$ [54]. 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 and covariance functions:

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

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

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

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 generated according to:

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

where $f : \mathbb{R}^D \to \mathbb{R}$ and $\varepsilon_i \sim \mathcal{N}\left(0, \sigma_\varepsilon^2\right)$ is Gaussian measurement noise with zero mean and variance σ_ε [5]. Noise in the data can be added to the covariance function. The prior on the measured function values is defined as $p\left(\mathbf{y} \mid X\right) = \mathcal{N}\left(0, K(X, X) + \sigma_\varepsilon^2 I\right)$. Here, $K(X, X)$ denotes the $n \times n$ matrix of the covariances 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:

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

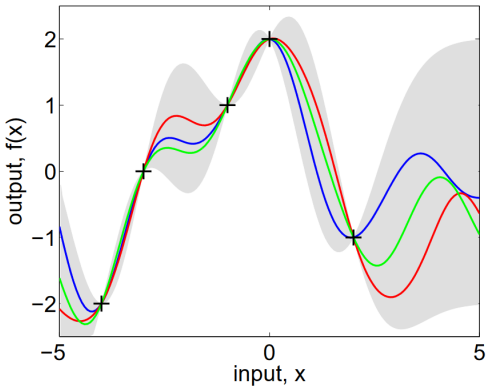


Figure 4-1: GP posterior distribution conditioned on five, noise free observations. The grey area represents the mean ± two times the standard deviation. In colour are three random samples drawn from the posterior distribution [54].

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

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

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

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

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

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 [28]. For this thesis, two types of kernels are explored. The most commonly used kernel is the Squared Exponential (SE) kernel:

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

Another relevant kernel is the Matern kernel:

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

where $d\left(\mathbf{x}, \mathbf{x}'\right) = \sum_{d=1}^{D} \left(x_d - x_d'\right)^2 l_d^{-2}$ is the Euclidean distance, $K_\nu(\cdot)$ is a modified Bessel function and $\Gamma(\cdot)$ is the gamma function. The class of Matern kernels is a generalization of the SE kernel.

Hyperparameters Each kernel function has free parameters which define the properties of the GP. Common hyperparameters 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 hyperparameter of the kernel. The hyperparameters do not have to be determined beforehand but are trained as part of the GP prior using the measured data $\mathcal{D} = \{(X, \mathbf{y})\}$.

Hyperparameter training The hyperparameters are found by maximizing the marginal likelihood (or evidence) of the prior, which is conditioned on the hyperparameters $\boldsymbol{\theta}$ 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 hyperparameter 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 hyperparameters, which trades off model complexity versus the data fit [5]. Maximizing the likelihood is a non-convex, nonlinear optimization problem which can be solved by hand or using available libraries such as GPyTorch [12] or sklearn [38].

Computational complexity Training the GP with a training set of size n requires $\mathcal{O}(n^3)$ per gradient step due to the inversion of the Kernel matrix $K(X, X)$ which makes training very expensive [54]. To make predictions, the inversion of the kernel matrix demands most operations and results in a computational complexity of $\mathcal{O}(n^3)$ per prediction. Cholesky factorization can help reduce the computational complexity of the matrix inversion to $\mathcal{O}(n^3/6)$. Sparse GPs can further reduce the complexity, which motivates the use of it.

4-1-2 Sparse Gaussian process regression

Sparse GPs 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 $\mathcal{O}(nm^2)$ - a significant speed up.

Choosing inducing inputs Using fewer inputs to train and predict GPs 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 [48] is used which provides an objective function for optimizing the inducing points.

Variational Inference Using a variational approach, the inducing points 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 $\boldsymbol{\Sigma}_\mathbf{u}$. 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. This is done together with the minimization of the marginal likelihood in Eq. (4-10) as part of the hyperparameter 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 $\mathbf{w} := \text{chol}\,\mathbf{K}_\mathbf{u}^{-1}\mathbf{v}$, 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 $\boldsymbol{\Sigma}_\mathbf{w}$. From this, the posterior sparse GP formulation can be computed as follows:

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

denoting $\text{chol}(\mathbf{K}_\mathbf{u})$ as $\mathbf{L}_\mathbf{u}$ and making use of the identity $\mathbf{K}_\mathbf{u}^{-1} = \mathbf{L}_\mathbf{u}^{-\top}\mathbf{L}_\mathbf{u}^{-1}$. This parametrization of the inducing variables can facilitate the optimization and speed up the prediction. The method has been implemented in GPyTorch [12] which is a GP library using PyTorch for standard and sparse GP inference.

4-1-3 Active Learning

AL [18] 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 GPs 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.

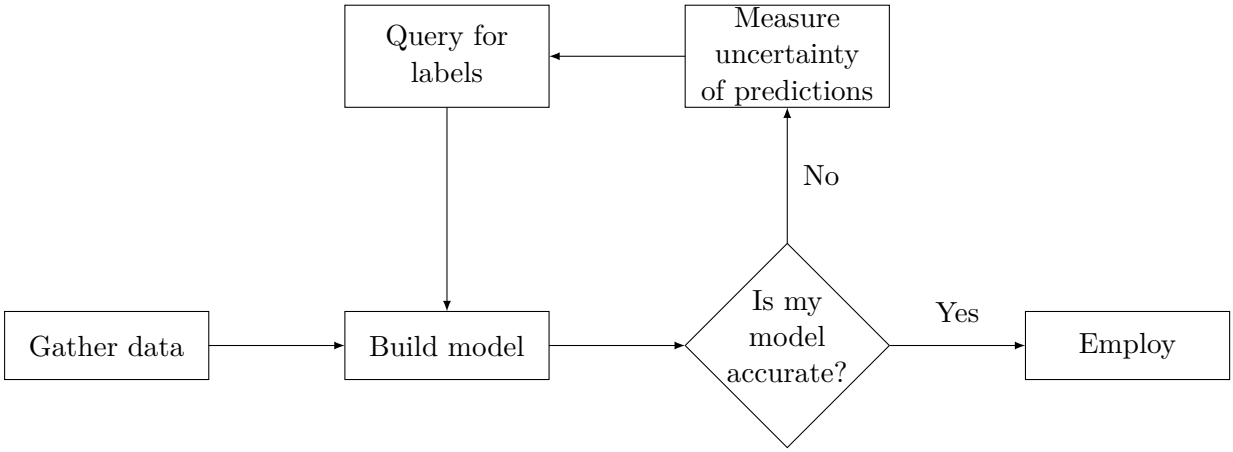


Figure 4-2: Active learning workflow [18].

4-2 Data collection

To train the Gaussian process 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 AL strategy to explore the environment.

4-2-1 Qaudrotor 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. The environment is programmed using a Python client library for Robot Operating System (ROS), to keep the quadrotor control in the ROS environment.

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 an RK4 method to integrate the

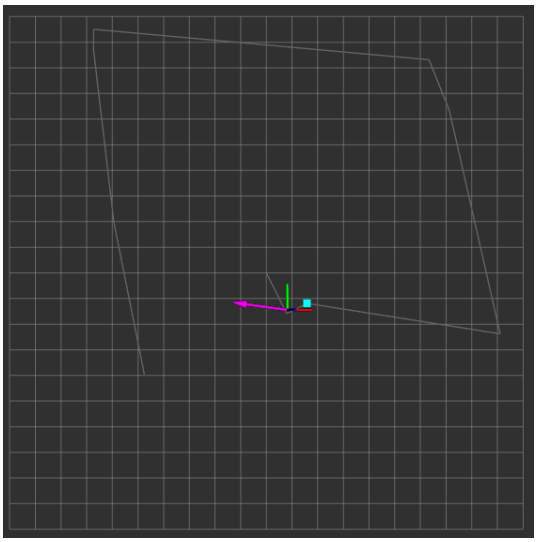


Figure 4-3: RViz simulation environment of the quadrotor. The quadrotor is represented by the 3D coordinate system indicating it's position and orientation, the pink arrow shows the wind acting on the quadrotor at any given position, the grey path shows the reference path and the cyan point indicates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

quadrotor dynamics and returns the odometry data of the quadrotor. This happens at a rate of 20Hz.

RViz 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 ignored in this environment, reducing the problem to two dimensions. The environment and quadrotor are visualized using RViz, a visualization software tool for ROS. The simulation scenario is shown in Figure 4-3.

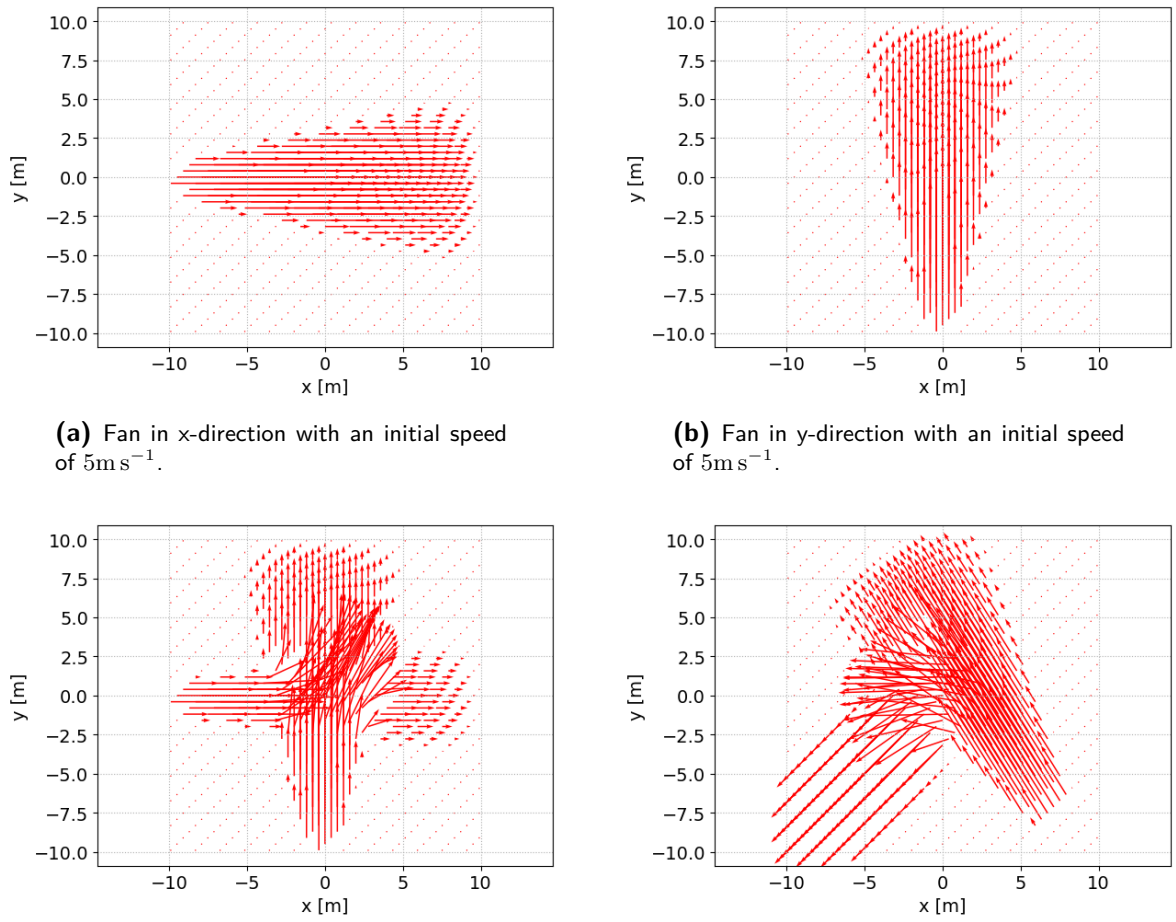
Generated wind fields

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 wind plugin is added to the simulation that allows adding any custom, static windfield 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 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



(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.

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.

4-2-2 Disturbance model

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 GPs, 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. Two GPs are trained for the two disturbance directions using the same quadrotor position as input, but only one of the disturbances as training target. This assumes that the two wind directions are computationally independent. 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:

$$\boldsymbol{\mu}_d = \left[\mu_{v_x}, \mu_{v_y}\right]^\top \tag{4-14a}$$

$$\boldsymbol{\Sigma}_d = \operatorname{diag}\left(\begin{bmatrix} \Sigma^{d_{v_x}} & \Sigma^{d_{v_y}} \end{bmatrix}\right) \tag{4-14b}$$

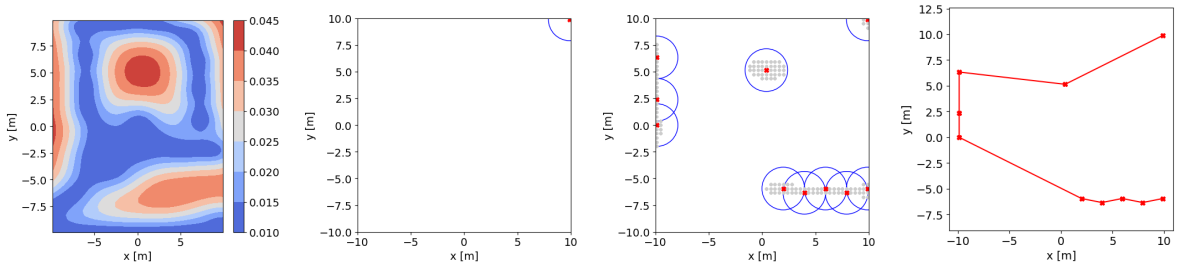
4-2-3 Exploration of the environment

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 were visited. Instead, the exploration experiment is designed in an optimal way using the concept of AL.

Active learning experiment Sampling N random points of the environment, 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 $r = 2$m is defined. If a point was 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). Sampling the most uncertain points and retraining the model is repeated until the maximum uncertainty is lower than a specified threshold $\bar{\sigma}$.

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. It was found that for small N, the quadrotor path is too random, resulting in long experiment durations. On the other hand, if N is too large, training of the GP is repeated not frequent enough, such that the points of maximum of uncertainty are no longer relevant too 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 drone has already collected some data. Those parts of the map, where disturbance data was collected are certain, while points of the map that have not been visited yet



(a) Uncertainty over the disturbance map after 2 iterations.

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

(c) $N = 10$ sampled data points and threshold radii.

(d) Generated path connecting datapoints solving the TSP.

Figure 4-5: Steps of the active learning experiment.

are less certain. 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 10 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 drone.

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. For GP training, the data is down-sampled to 4Hz to avoid too large data sets and training times.

4-3 Training the Gaussian process disturbance map

With the collected data samples, the GP model can be trained to find the hyperparameters of the GP kernel function. In addition, different types of kernel functions are explored and the parameters 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 Gaussian process training

The GP is trained using stochastic variational GPR in GPyTorch [12], which is the sparse GP regression method explained in section 4-1-2 with the Cholesky form of the variational distribution. As this GP regression method is an approximate method, stochastic optimization techniques are used to train the GP. The optimization loops over a number of training iterations (epochs) and minibatches of the given data, mainimizing the variational Evidence-Lower Bound (ELBO) between the variational distribution and the exact posterior GP.

Number of inducing points The sparse GP is initialized with 30 inducing points. This number was selected based on related literature (see. e.g. [51], [4], [19]) and is a balance between the speed of the prediction and accuracy of the GP.

Training and model parameters The optimal combination of epochs and batch size has to be selected prior to the hyperparameter optimization and is part of training the GP dynamics model. Additionally, the GP kernel function has to be selected as the GP prior. Two types of kernel functions are explored: The SE kernel and the Matern kernel to vary the smoothness of the kernel. How to find the optimal combination of kernel and training parameters is discussed next.

4-3-2 Training and model parameter tuning

The model parameters that need to be explored to find the parameters of the GP model training 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 is shown in Figure 4-4a. As it was found that 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 epochs and batch size are 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 time of the training 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 was 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 was 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.

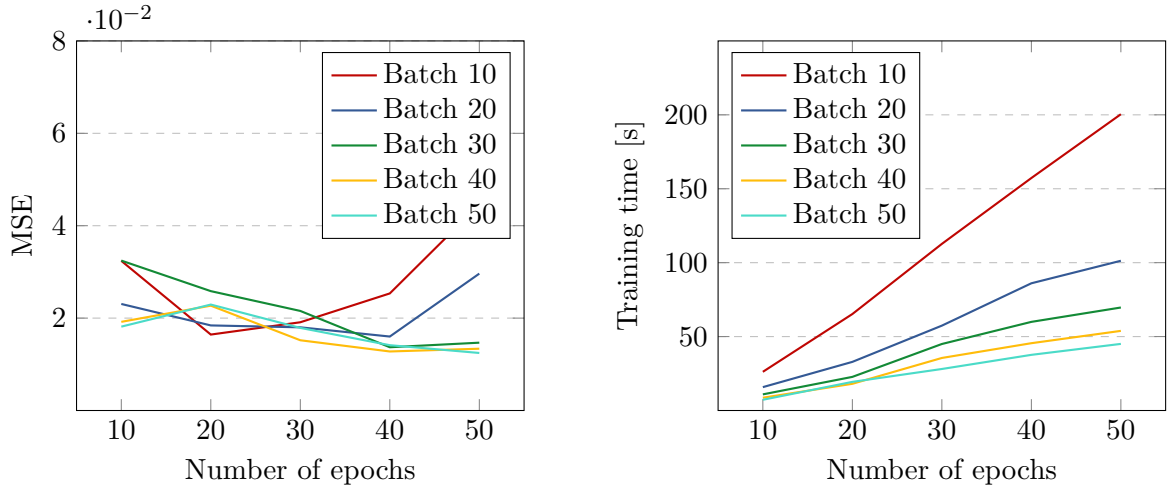


Figure 4-6: MSE (left) and training times (right) for different combinations of epochs and batch size.

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 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, 5 AL iterations are performed.

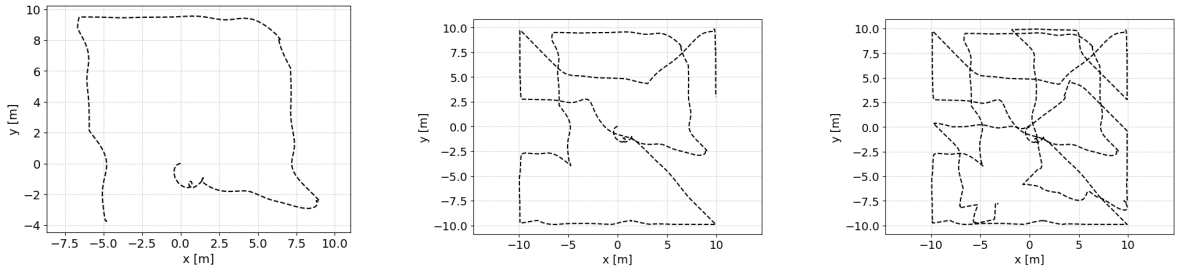


Figure 4-7: Generated path after 1, 3, and 5 active learning iterations with the SE kernel.

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 5 iterations, almost the entire space is covered with similar exploration times.

Accuracy and uncertainty Comparing the MSE shown in Figure 4-9 in the left plot and the uncertainties shown on the right, the training results are similar. The SE kernel is slightly more accurate for more training iterations, but also has a slighly higher uncertainty. In general, also here no big difference for the two types of kernel functions can be identified.

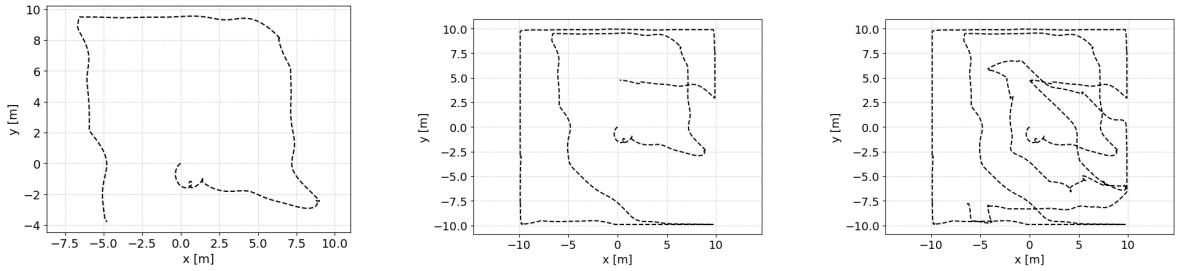


Figure 4-8: Generated path after 1, 3, and 5 active learning iterations with the Matern kernel.

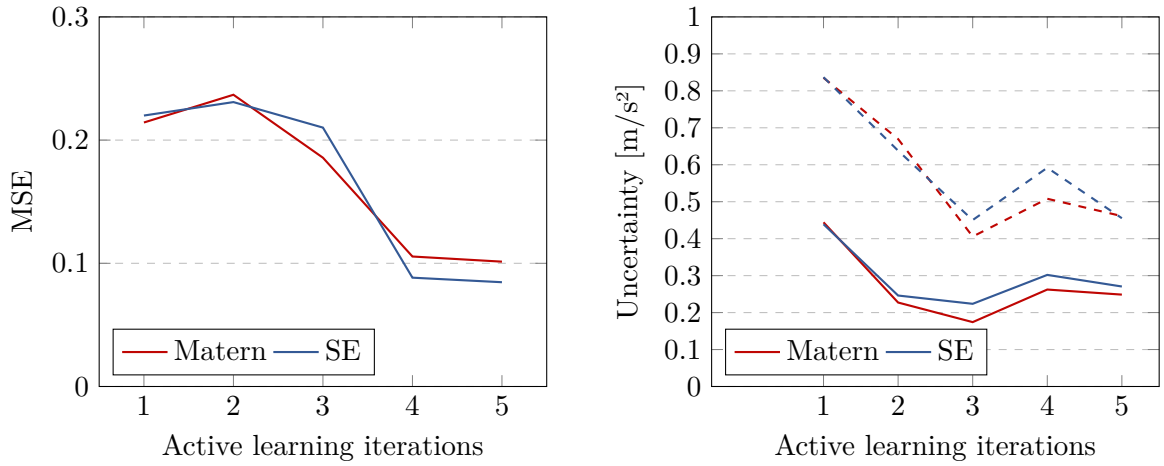


Figure 4-9: MSE and mean(solid) and max(dashed) uncertainty of the trained GP model for the SE and Matern kernel.

Effect of smoothness on the learning result In general, the SE kernel is smoother than 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. The less smooth Matern kernel could perform better.

Chosen kernel type As for the performed experiments the resilts are similar, the SE kernel is chosen to continue with the hyperparameter training in simulation. It has the simpler formulation of the two types of kernel functions, making subsequent computations, particularly derivative computations, easier. The Matern kernel should kept in mind, however, in case the SE kernel does not perform well under real-world conditions, where the wind fields can be expected to be 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, superiority of collecting he data using AL is shown. 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

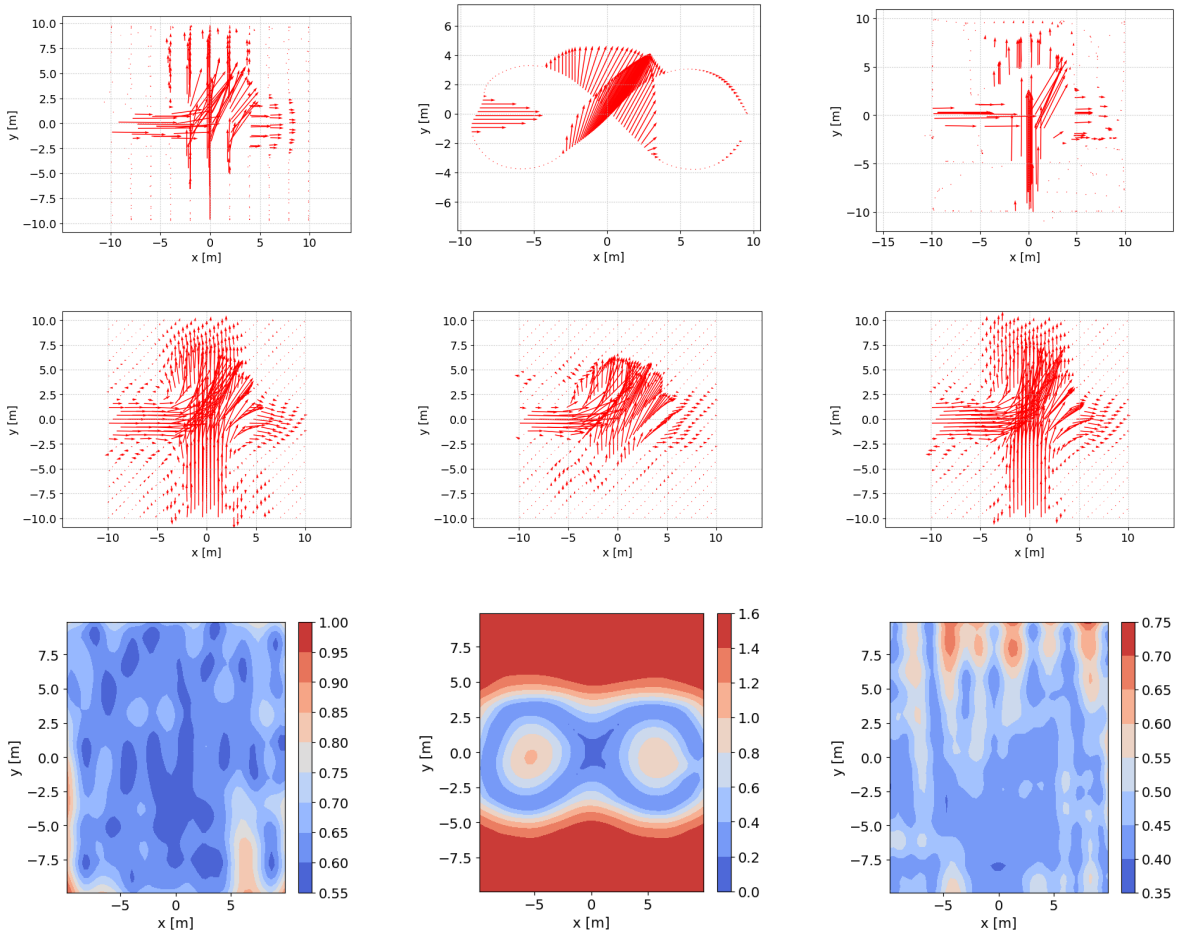
To show the superiority of collecting the data using AL, the data collection method is compared to two other exploration paths. In one scenario, the data is collected by exploring the environment in a structured way, to cover as much area as possible. The resulting quadrotor path is shown in Figure 4-10a. In the second scenario, the environment is explored by flying a lemniscate trajectory shown in the left column of Figure 4-10b. The generated path 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.

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

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

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 twice the amount of time, also nearly doubling the amount of training data. Moreover, comparing the maximum uncertainty, it is lower for the AL approach as the environment was 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 was not covered by the lemniscate reference path and more evenly distributed for the structured and AL path.



(a) Structured path **(b)** Lemniscate path **(c)** AL path

Figure 4-10: Results of collecting data using a strcutured path (left column), a lemniscate path(middle column) and AL path (right) column. The first row shows the collected data, the second row shows the predicted windfields after training and the third row shows the uncertainty of the prediction, for all three cases.

4-4-2 Sparse versus full Gaussian process model

To demonstrate the necessity of using sparse GP for this project, sparse GP training and prediction is compared to full GP training and prediction. This comparison should furthermore show, that reducing the number of training points, does not significantly deteriorate the training results. The reference data 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 5 times higher and the uncertainty is doubled. This is to be expected, as the

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

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

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. Also, the information on the higher uncertainty can be accounted for in the controller design. When comparing the training and prediction times the major advantage of using sparse GP for this application becomes visible: The prediction is around 150 times faster. The training times of the sparse approach is larger due to the stochastic training and the fact that the GP has to learn the predictive distribution in addition to the hyperparameters of the kernel. However, as the GP is not trained online, this is secondary.

4-4-3 Trained disturbance models

After finding the right training conditions, 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). 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. The wind fields are then trained using sparse GP, with 30 inducing points and a SE kernel and the batch size and number of epochs determined in the previous section.

Wind field	Wind-x (4-4a)	Wind-cross (4-4c)
Training iterations	4	4
MSE	0.019	0.060
Mean uncertainty [m/s^2]	0.248	0.498
Maximum uncertainty [m/s^2]	0.3575	0.639
Training time [s]	15.07	14.81
Prediction time [s]	0.080	0.051
Length scales d_{v_x}	(7.875, 1.009)	(2.27, 2.66)
Length scaled d_{v_y}	(7.93, 11.21)	(1.675, 2.597)
Output covariance d_{v_x}	0.096	0.050
Output covariance d_{v_y}	2.8e-6	0.076
Noise covariance d_{v_x}	0.032	0.037
Noise covariance d_{v_y}	2.4e-4	0.043

Table 4-3: Final parameters and performance metrics of the two trained wind disturbance maps.

Trained parameters and performance metrics The final training results are summarized

in Table 4-3, together with the trained hyperparameters. The length scales of the wind field in x are generally larger, indicating that the windfield 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. The GP model is able to train the wind field with an MSE of 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.

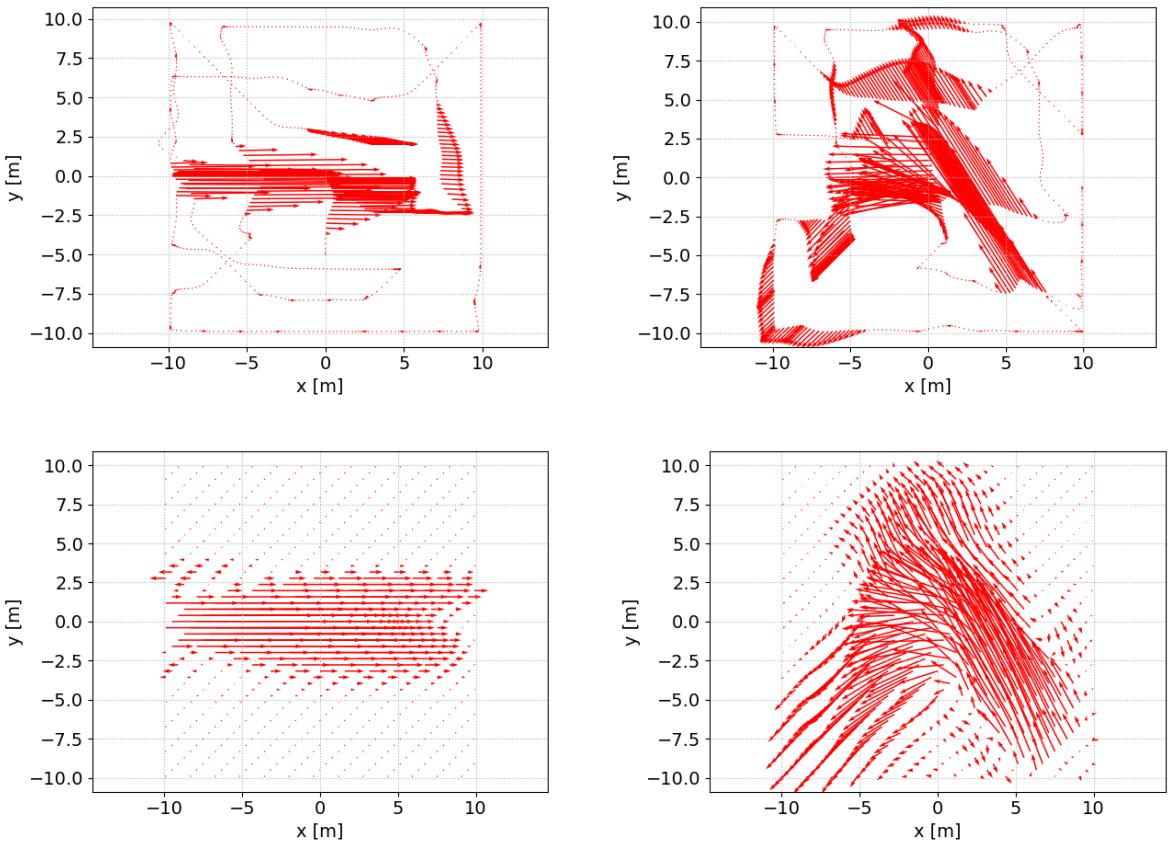
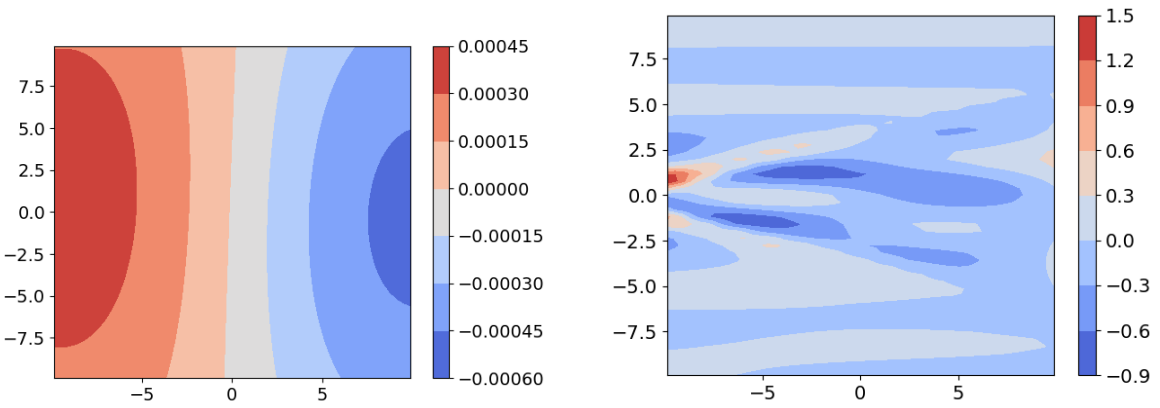


Figure 4-11: Collected training data and trained disturbance maps for the wind field in x-direction (left column) and the randomly crossing fans (right column).

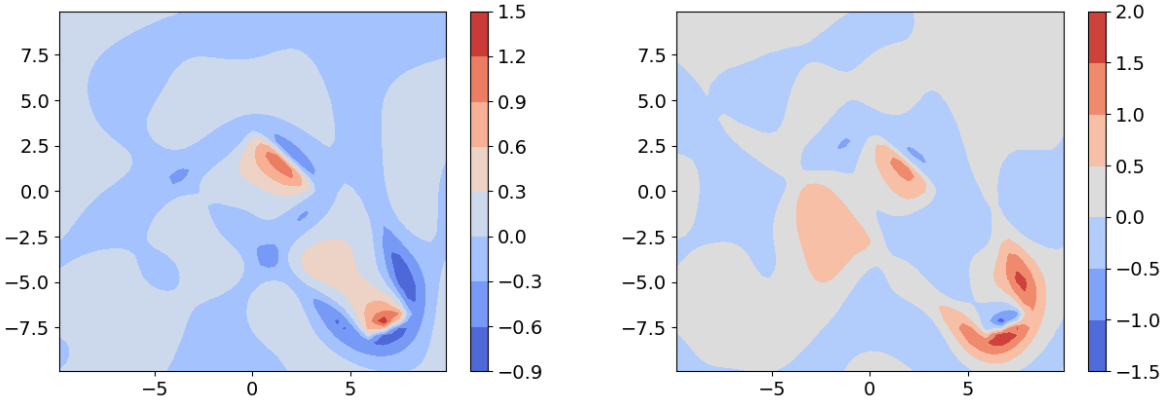
Comparison to ground truth wind field From the results in Figure 4-11, it can already be seen that the mean of the trained wind disturbance map is not accurate everywhere. Figure 4-12 therefore shows the difference between the trained mean $\hat{d}_{v.}$ and the actual mean $d_{v.}$ for the x- and y-direction independently, again for both trained wind fields. Specifically near the fans, the prediction becomes less accurate. This is caused most likely by the fact, that near the fans, the wind drops rapidly, which cannot be captured by the smooth GP model, which moreover has reduced expressiveness due to the sparse GP approach.

Uncertainties of the trained wind fields Finally, the ground truth wind fields are compared to the trained models in terms of the uncertainties to make sure that the uncertainty region covers the actual disturbance despite some model mismatch. Figure 4-13 shows the actual mean disturbance in red and the uncertain regions within 2.807 standard deviations



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

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

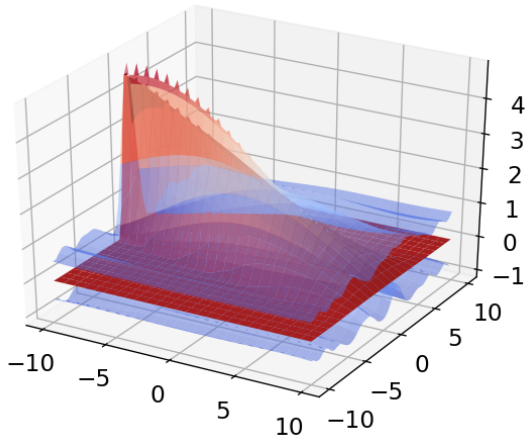


(c) $\hat{d}_{v_x} - d_{v_x}$ for crossing fans.

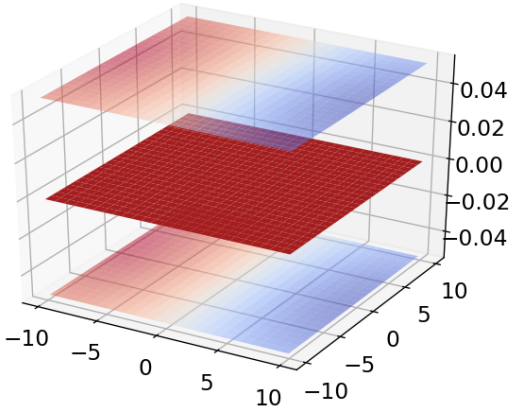
(d) $\hat{d}_{v_y} - d_{v_y}$ for crossing fans.

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

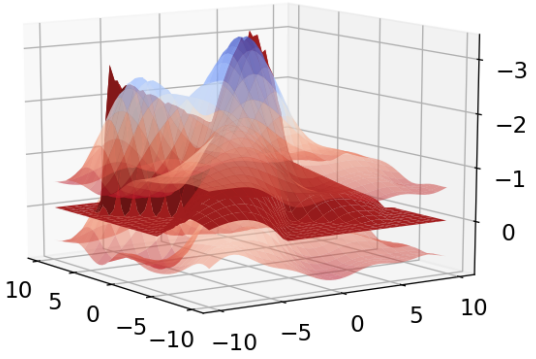
learned by the GP model, corresponding to the 99.5% confidence interval. For the largest part, the actual disturbance lies within the trained wind fields. Despite some mismatch in the mean prediction, the designed motion planner and controller should therefore be robust enough, taking in account the 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 was 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 would 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,



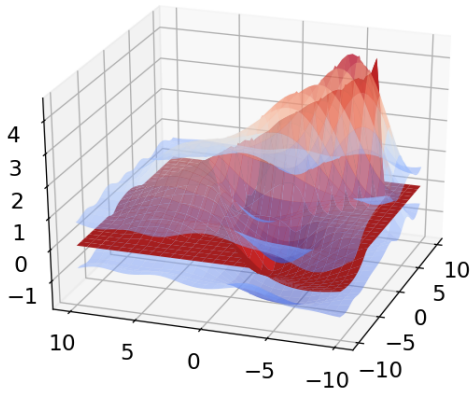
(a) Disturbance bound of the trained wind wind field pointing in x-direction in x.



(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 confidence interval of 99.5% is kept so that the controller is less conservative.

4-5 Summary and Discussion

GP 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 have to be considered.

Despite the hyperparameters 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 was found such that the SE kernel was chosen due to its simpler form.

Additionally, a method of optimal exploration of the environment is presented through the use AL. By collecting the data in using AL, it can be ensured that the data is collected in an optimal way and adapts to the type of disturbance map. This approach allows for reduced exploration times while also decreasing the uncertainty of the trained GP model when compared to manually designed exploration paths.

To be able to use the GP model in the Local Model Predictive Contouring Control (LMPCC), sparse GP methods are implemented. While this approach decreases the accuracy of the GP model and increases uncertainty in predictions, it also increases the feasibility of using the GP model by reducing prediction times by a factor of 100. This decrease in model performance is acknowledged and taken into consideration during controller design. Furthermore, using a specification method to train the model further complicates the process and requires the optimization of the epoch and batch size through grid search.

Two types of wind maps are trained for use in the subsequent LMPCC 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%. However, when the number of training points is reduced during the sparsification process, the trained models may not accurately capture high spikes in the training data. Therefore, it is recommended to explore online learning methods that do not require capturing the entire environment, especially in real-world scenarios where wind conditions may change rapidly, for future research.

Chapter 5

Data-driven local MPCC motion planner

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

5-1 Data-driven controller formulation

The resulting learning-based 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 drone are given as a combination of the nominal model and the GP model. Due to the discrete nature of the derived GP model, the system dynamics are

represented in discrete form, according to:

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

The continuous, nominal dynamics $f(\mathbf{x}_k, \mathbf{u}_k)$ follow the model derived in Eq. (3-6) and are discretized using an RK4 integration scheme. 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 $\mathbf{B}_d$. Additional uncertainty in the system is accounted for by the process noise $\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{\Sigma}_k^{\mathbf{w}})$.

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.

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

with approximate inference using linearization.

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:

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

for the update of the state mean. Here, $\boldsymbol{\mu}_k^{\mathbf{d}}(\boldsymbol{\mu}_k^{\mathbf{p}})$, indicates the evaluation of the GP disturbance map 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:

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

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

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

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

with $\mathbf{B}_d \left(\mathbf{\Sigma}_k^{\mathbf{d}} + \mathbf{\Sigma}_k^{\mathbf{w}} \right) \mathbf{B}_d^T$ being the propagation of the GP uncertainty $\mathbf{\Sigma}_k^{\mathbf{d}}$ 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:

$$\mathbf{\Sigma}_{k+1}^{\mathbf{x}} = \left[\nabla_{\mathbf{x}} f\left(\boldsymbol{\mu}_k^{\mathbf{x}}, \mathbf{u}_k \right) \, \mathbf{B}_d \right] \begin{bmatrix} \mathbf{\Sigma}_k^{\mathbf{x}} & \mathbf{\Sigma}_k^{\mathbf{xd}} \\ \mathbf{\Sigma}_k^{\mathbf{dx}} & \mathbf{\Sigma}_k^{\mathbf{d}} + \mathbf{\Sigma}_k^{\mathbf{w}} \end{bmatrix} \left[\nabla_{\mathbf{x}} f\left(\boldsymbol{\mu}_k^{\mathbf{x}}, \mathbf{u}_k \right) \, \mathbf{B}_d \right]^T \tag{5-7}$$

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

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:

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

$$\mathbf{\Sigma}_{k+1}^{\mathbf{x}} = \left[\mathbf{A} \, \mathbf{B}_d \right] \begin{bmatrix} \mathbf{\Sigma}_k^{\mathbf{x}} & \mathbf{\Sigma}_k^{\mathbf{xd}} \\ \mathbf{\Sigma}_k^{\mathbf{dx}} & \mathbf{\Sigma}_k^{\mathbf{d}} + \mathbf{\Sigma}_k^{\mathbf{w}} \end{bmatrix} \left[\mathbf{A} \, \mathbf{B}_d \right]^T . \tag{5-8b}$$

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

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

The LMPCC 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 LMPCC controller. The formulation discussed hereafter is taken from the research in [3] and slightly modified for the application of this thesis.

Reference path The reference path of the LMPCC 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:

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

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 drone 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 and lag error are defined and combined in an error vector:

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

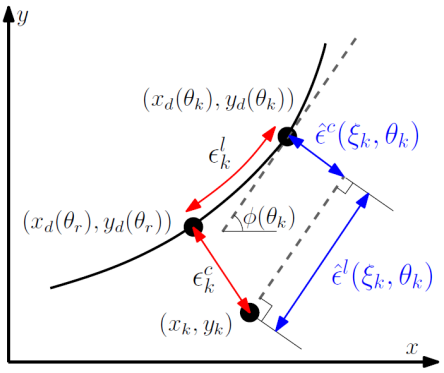


Figure 5-1: Contour and lag error and it's approximations in 2D [26].

The contour and lag error define the approximate 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 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 LMPCC is

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

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

5-1-3 Obstcale avoidance constraints

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

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

where $\mathcal{B}(\mathbf{x}_k, \mathbf{\Sigma}_k^{\mathbf{x}})$ represents the occupancy volume of the quadrotor and $\mathcal{C}$ denotes the collision region. 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:

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

The uncertainty of the position $\mathbf{\Sigma}_k^{\mathbf{p}}$ 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 $\mathbf{\Sigma}_k^{\mathbf{p}}$ [11]. To guarantee a certain collision probability, the ellipsoid is scaled given the confidence interval χ. The obstacle avoidance constraint at each state is [11]:

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

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 was derived by the authors in [56]. The chance constraints are defined as a collision probability:

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

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

$$\tilde{\mathcal{C}}_o := \left\{ \mathbf{x} \mid \mathbf{a}_o^T \left(\mathbf{p} - \mathbf{p}_o\right) \leq b \right\}, \tag{5-20}$$

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

$$\mathbf{p} \sim \mathcal{N}\left(\hat{\mathbf{p}}, \mathbf{\Sigma}_k^{\mathbf{p}}\right), \tag{5-21}$$

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 [2]:

$$\Pr\left(\mathbf{a}^T\mathbf{x} \leq b\right) = \frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{b - \mathbf{a}^T\hat{\mathbf{x}}}{\sqrt{2\mathbf{a}^T\sum\mathbf{a}}}\right). \tag{5-22}$$

where erf denotes the error function:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt. \tag{5-23}$$

This deterministic chance constraints take the form:

$$\mathbf{a}_o^T\left(\hat{\mathbf{p}} - \mathbf{p}_o\right) - b \geq \operatorname{erf}^{-1}\left(1 - 2\delta_o\right)\sqrt{2\mathbf{a}_o^T\left(\mathbf{\Sigma}_k^{\mathbf{p}}\right)\mathbf{a}_o}, \forall o \in \mathcal{I}_o. \tag{5-24}$$

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 $\mathcal{J}_\xi = \|\xi\|^2_{q_\xi} + c_\xi \xi$ for the slack variable is defined. For sufficiently large c_ξ the soft constraint formulation is exact, if feasible [23].

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:

$$\mathcal{J}^*_{\text{LMPCC}} = \min_{\mathbf{x},\mathbf{u},s} \mathcal{J}_{\text{LMPCC}}(\boldsymbol{\mu}^{\mathbf{x}}, \mathbf{u}, s) \tag{5-25a}$$

$$\text{s.t. } \boldsymbol{\mu}^{\mathbf{x}}_0 = \boldsymbol{\mu}^{\mathbf{x}}(0), \tag{5-25b}$$

$$s_0 = s(0), \tag{5-25c}$$

$$\boldsymbol{\mu}^{\mathbf{x}}_{k+1} = f(\boldsymbol{\mu}^{\mathbf{x}}_k, \mathbf{u}_k) + \mathbf{B}_d \left(\boldsymbol{\mu}^{\mathbf{d}}_k(\boldsymbol{\mu}^{\mathbf{p}}_k) + \mathbf{w}_k \right) \tag{5-25d}$$

$$\boldsymbol{\Sigma}^{\mathbf{x}}_{k+1} = [\mathbf{A}\, \mathbf{B}_d] \begin{bmatrix} \boldsymbol{\Sigma}^{\mathbf{x}}_k & \boldsymbol{\Sigma}^{\mathbf{xd}}_k, \\ \boldsymbol{\Sigma}^{\mathbf{dx}}_k & \boldsymbol{\Sigma}^{\mathbf{d}}_k + \boldsymbol{\Sigma}^{\mathbf{w}}_k \end{bmatrix} [\mathbf{A}\, \mathbf{B}_d]^T, \tag{5-25e}$$

$$s_{k+1} = s_k + v_k T_s, \tag{5-25f}$$

$$\mathcal{B}(\mathbf{x}_k, \boldsymbol{\Sigma}^{\mathbf{x}}_k) \cap \mathcal{C} = \emptyset, \tag{5-25g}$$

$$\boldsymbol{\mu}^{\mathbf{x}}_k \in \mathcal{X}, \tag{5-25h}$$

$$\mathbf{u}_k \in \mathcal{U}, \tag{5-25i}$$

$$0 \leq s_k \leq L \tag{5-25j}$$

$$k = 0, \ldots, N_p - 1 \tag{5-25k}$$

with cost function

$$\mathcal{J}_{\text{LMPCC}}(\mathbf{x}, \mathbf{u}, s) = \sum_{k=0}^{N_p-1} \mathcal{J}_{\text{LMPCC}}(\boldsymbol{\mu}^{\mathbf{x}}_k, \mathbf{u}_k, s_k) + \mathcal{J}_{\text{LMPCC}}(\boldsymbol{\mu}^{\mathbf{x}}_{N_p}, \mathbf{u}_{N_p}, s_{N_p}). \tag{5-26}$$

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 LMPCC formulation, an optimization problem is solved to find the control inputs to the drone. This section discusses the implementation of the optimal control problem, the tuned parameters and chosen state constraints and the system interface to control the drone in simulation.

5-2-1 Controller implementation

The data-driven LMPCC problem is implemented with a prediction horizon of $N = 20$ at a sampling rate of $T_\mathrm{s} = 50$ms, resulting in a 1s look-ahead. The underlying optimization problem is solved with the interior-point-based Nonlinear Programming (NLP) solver FORCES PRO [9]. 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 $\boldsymbol{\Sigma}_\mathbf{u}$ 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 LMPCC iteration, and set as a fixed parameter. This choice was made to reduce the optimization variables of the solver and therefore computation times. Furthermore, numerical issues appear when having the state uncertainty as an optimization variable inside the constraints.

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

$$\mathcal{U} = \left\{ \mathbf{u} \in \mathbb{R}^m \;\middle|\; \begin{array}{l} (\phi_c, \theta_c) \in [-40, 40]\text{deg} \\ \dot{\psi}_c \in [-10, 10]\text{deg/s} \\ T_c \in [5, 15]\text{m/s}^2 \end{array} \right\}, \tag{5-27}$$

$$\mathcal{X} = \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{l} (x, y) \in [-\infty, \infty]\text{m} \\ z \in [-1, 1]\text{m} \\ (v_x, v_y, v_z) \in [-10, 10]\text{m/s} \\ (\phi, \theta) \in [-50, 50]\text{deg} \\ \dot{\psi} \in [-15, 15]\text{deg/s} \end{array} \right\}. \tag{5-28}$$

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

$$\mathcal{U} = \{s \in \mathbb{R} \mid s \in [0, 1e4]\text{m}\} \tag{5-29}$$

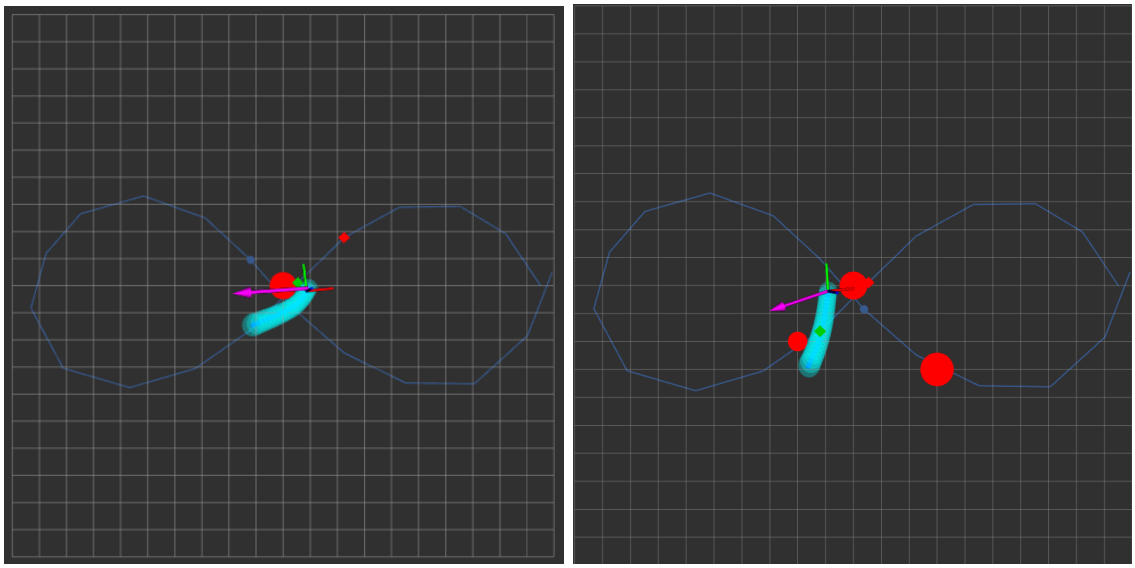
Furthermore, the following parameter values listed in Table 5-1 are used.

Name	Value
Roll and pitch input weight ϵ_ϕ&ϵ_θ	5.0
Yaw rate input weight $\epsilon_{\dot{\psi}}$	1.0
Thrust input weight ϵ_T	1.0
Lag error weight ϵ_l	3.0
Contour error weight ϵ_c	1.0
Reference velocity weight ϵ_v	10.0
v_ref	2.0
Slack weight q_ξ	1000
Slack weight c_ξ	100

Table 5-1: LMPCC parameters.

5-2-2 Simulation interface

The output of the optimal control problem is interfaced with the simple simulation environment described in section 4-2-1, simulated in RViz. The control commands generated by the solver are translated into Robot Operating System (ROS) commands. The drone is subject to the same wind disturbances that were 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 $r_{\text{quad}} = 0.325\text{m}$. 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 $r_{\text{obst}} = 0.5\text{m}$. This scenario is shown in Figure 5-2a.



(a) Simple obstacle scenario. **(b)** Complex obstacle scenario.

Figure 5-2: Quadrotor following a lemniscate reference path while avoiding one obstacle (left) and three obstacles(right). The uncertainty of the quadrotor position is propagated for the training horizon.

5-3 Data-driven controller validation

To show the validity of the data driven LMPCC motion planner, several scenarios are tested for the described simulation environment. First, the GP mean is added to the nominal model to show and improved 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, improved robustness of the controller formulation is shown.

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:

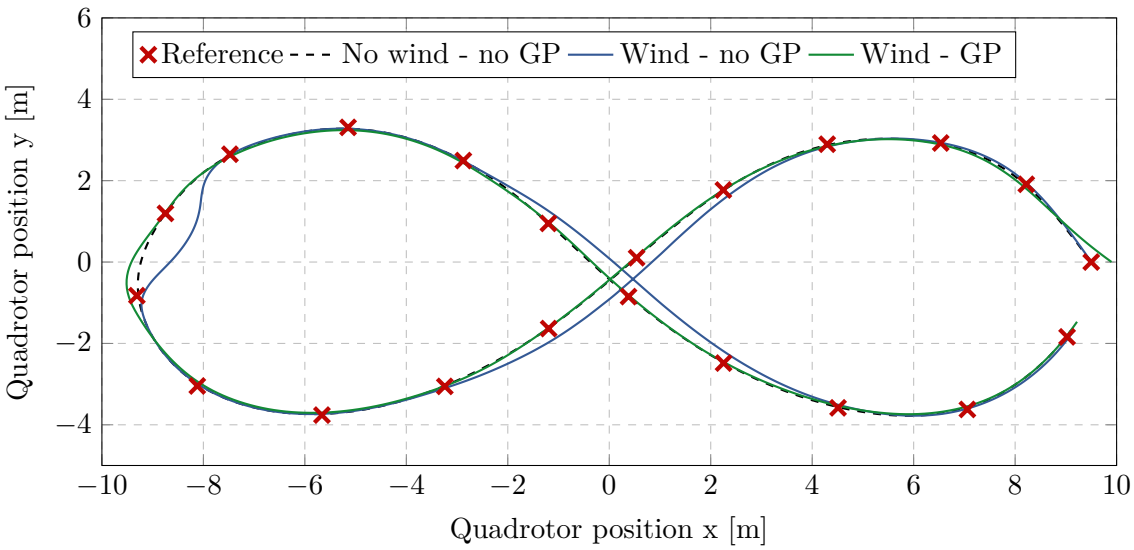


Figure 5-3: Tracking of the lemniscate reference path with and without the GP model for the wind field pointing in x direction.

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. When the wind is acting on the quadrotor and 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 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.

	Wind in x-direction	**Crossing fans**
Wind - no GP	0.127	0.160
Wind - GP	0.0695	0.0673

Table 5-2: Average minimum distance between the reference path points and the path flown by the quadrotor.

Numerical comparison Improved tracking is also validated numerically by computing the average minimum distance between the reference path points and the path flown by the quadrotor, summarized in Table 5-2. By adding the mean of the GP model, tracking becomes around twice as accurate, showing superiority of the trained model.

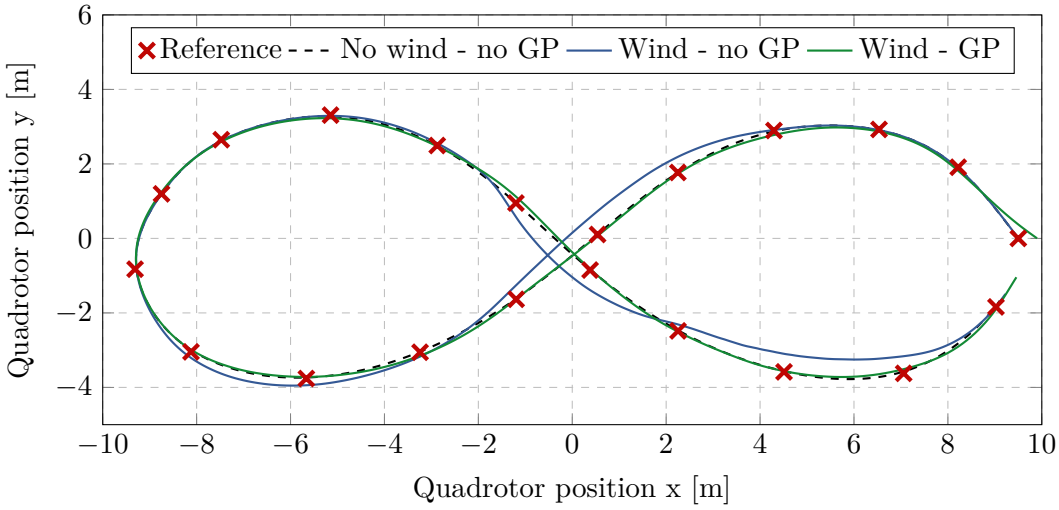
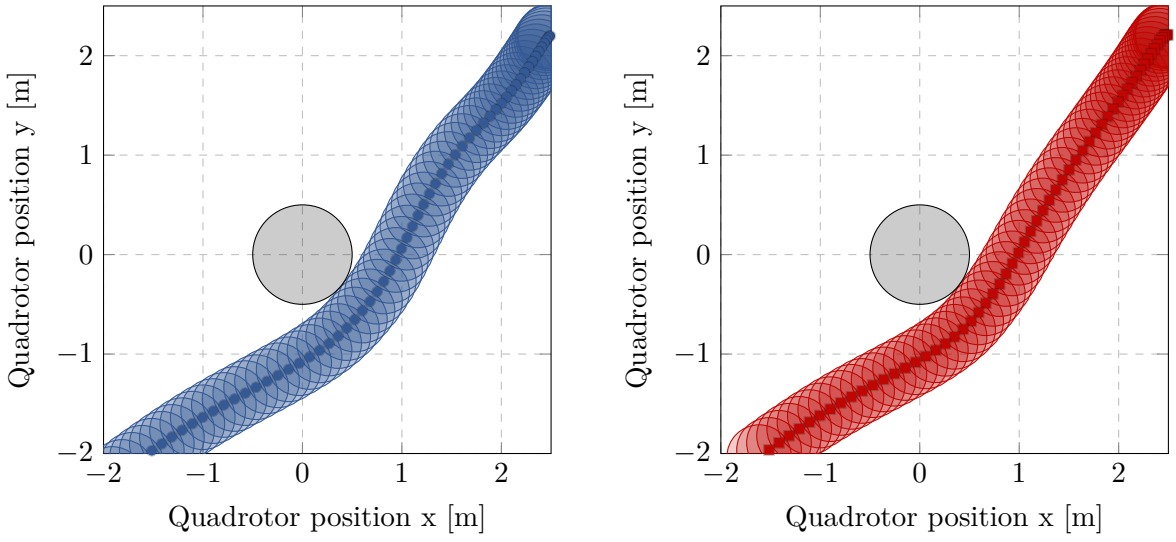


Figure 5-4: Tracking of the lemniscate reference path with and without the GP model for the windfield with two crossing fans.

5-3-2 Ellipsoidal versus chance constraints

Apart from using 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 5-1-3 are compared. In both cases, the collision probability is set to 0.5%. 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

(b) Chance constraints

Figure 5-5: Quadrotor navigating around the obstacle with an outer radius of 0.325m

Conservatism of the constraints Taking a closer look at the propagated uncertainty for

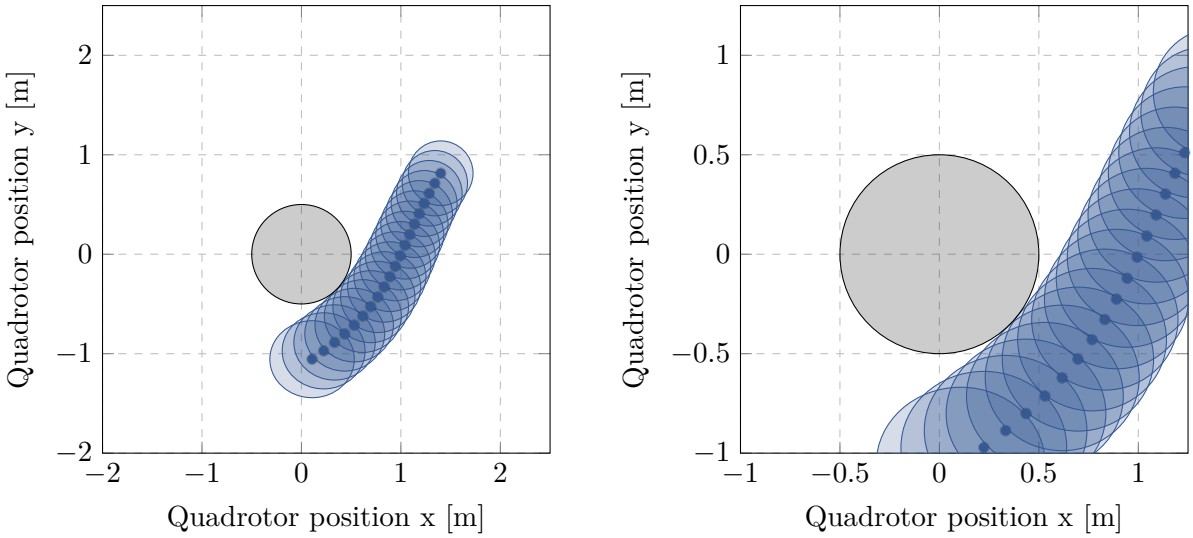


Figure 5-6: Propagation of the quadrotor path and uncertain ellipsoidal bound around the quadrotor for one solver iteration over a horizon of $N = 20$ with ellipsoidal constraints.

one solver iteration over the entire horizon of $N = 20$, 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. 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 were violated over the horizon, guaranteeing the same theoretic collision probability. It can therefore be concluded that the chance constraints are less conservative.

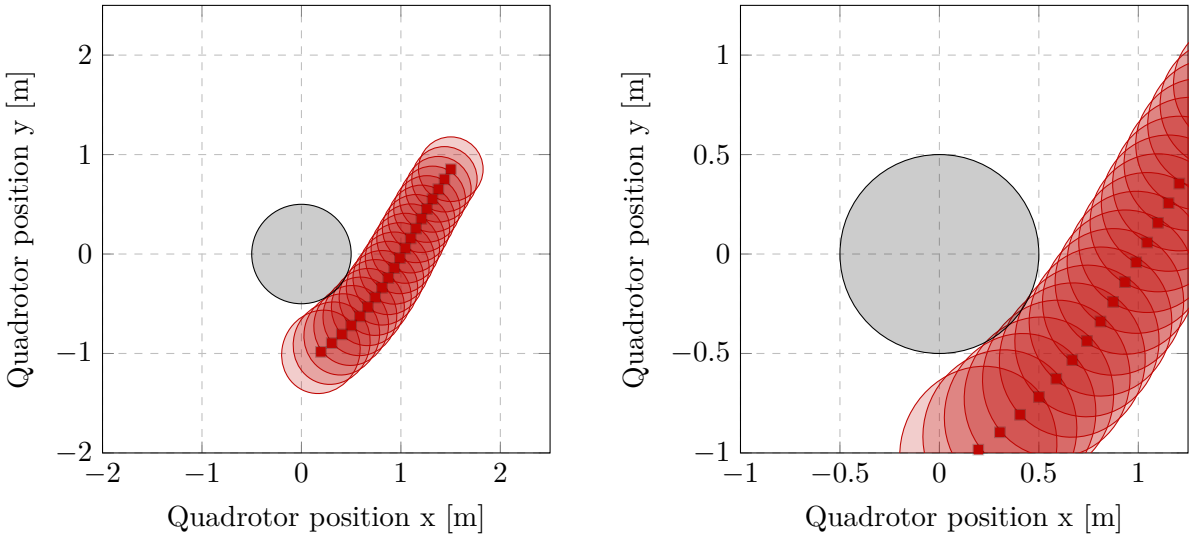


Figure 5-7: Propagation of the quadrotor path and uncertain ellipsoidal bound around the quadrotor for one solver iteration over a horizon of $N = 20$ with chance constraints.

Chosen constraint 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 controller is evaluated by comparing the nominal and GP-based LMPCC controller in a more complex scenario, where the quadrotor is still following the lemniscate path now with more obstacles in the environment as shown in Figure 5-2b. The robustness of both approaches is evaluated in terms of the slack ξ, corresponding to the obstacle-constraint violations.

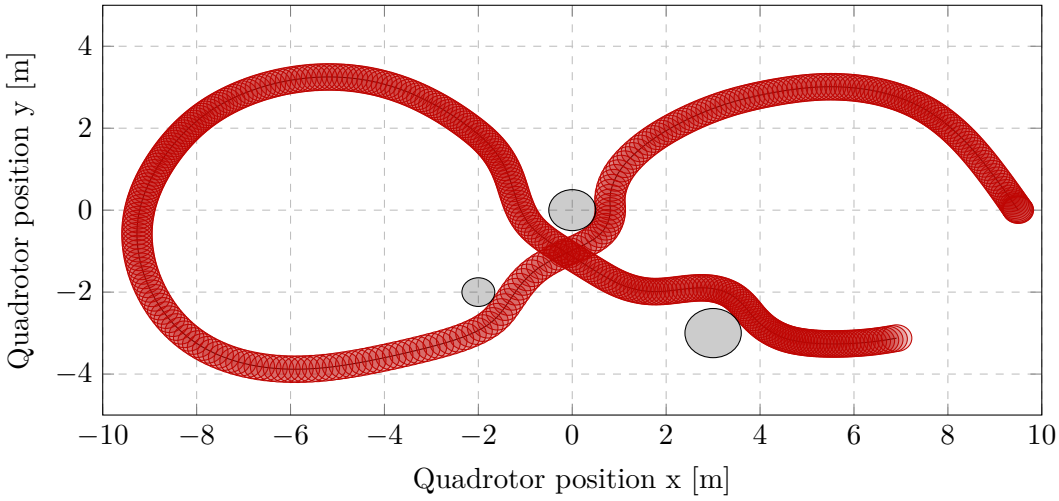


Figure 5-8: Quadrotor navigating around multiple obstacles with the nominal LMPCC controller.

Nominal LMPCC The path of the drone with the nominal LMPCC 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 a crash of the drone with the environment.

GP-based LMPCC The path of the drone with the GP-based LMPCC controller is shown in Figure 5-8. It can be noted, that the quadrotor chose to take a different path taking into account the mean and uncertainty of the GP model. 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 were actually violated. With the GP model and chance constraints in place the drone 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

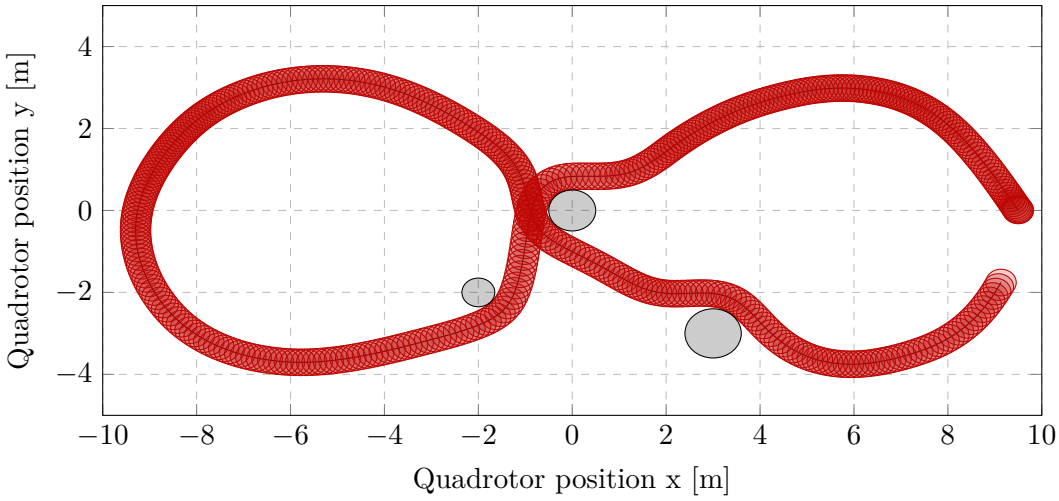


Figure 5-9: Quadrotor navigating around multiple obstacles with the GP-based LMPCC controller.

section with the LMPCC formulation including the GP with uncertainty and the LMPCC 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 were 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.

Nominal LMPCC For the nominal LMPCC controller, the control and optimization loop run at similar sampling times. The optimzation 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 50Hz. However, as the design of the nominal LMPCC 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 LMPCC 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 LMPCC implementation in C++. The uncertainty is then set as a parameter of the solver before the optimization loop.

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 LMPCC 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 computations times need to be sped up. As the real drone 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 an idea to implement the propagation of uncertainty inside the LMPCC implementation in C++. Moreover, the researchers in [25] suggest an optimal way to solve the problem of propagating the uncertainty inside the solver. However, they make use of the acados [52] solver, which also supports a Python interface, unlike FORCES PRO.

5-4 Summary and Discussion

LMPCC as presented in [3], offers a way to follow a reference path in an optimal way, solving the trajectory generation and tracking problem simultaneously. However, the motion planner and controller requires a system model. If the model is not accurate, such as when the drone is subject to wind, the controller performance will be poor and it may even 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 into the LMPCC controller, it was demonstrated that the drone not only exhibited improved tracking but also increased robustness when avoiding obstacles in the environment.

Extending the LMPCC 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 LMPCC controller using a probabilistic model and incorporating uncertainty for obstacle avoidance. Solutions to these challenges were provided in this chapter. Additionally, it was demonstrated that a chance constraint formulation is less conservative for obstacle avoidance and effectively utilizes the uncertainty information provided by the model.

However, the real-time capability of the approach has not yet been fully validated. It was found that incorporating the mean propagation of the GP model does not slow down the solver, however, even the solving times of the nominal LMPCC 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 were discussed in the relevant section.

Chapter 6

Performance comparison

After validating the data-driven Local Model Predictive Contouring Control (LMPCC) algorithm, it is compared to the external forces resilient safe motion planning algorithm by the authors in [55] to benchmark the developed algorithm in existing literature

6-1 Comparative method

The data-driven LMPCC 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 was chosen that is solving the same two issues: The external forces resilient safe motion planning algorithm by the authors in [55].

Motion planning and control The comparative method [55] 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 [55] construct an uncertain ellipsoid around the quadrotor using forward reachable sets.

Force estimation and re-planning The external force is obtained by VID-Fusion [8], 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. Details on the formulation can be found in [55]. A system overview diagram of the re-planning framework is shown in Figure 6-1.

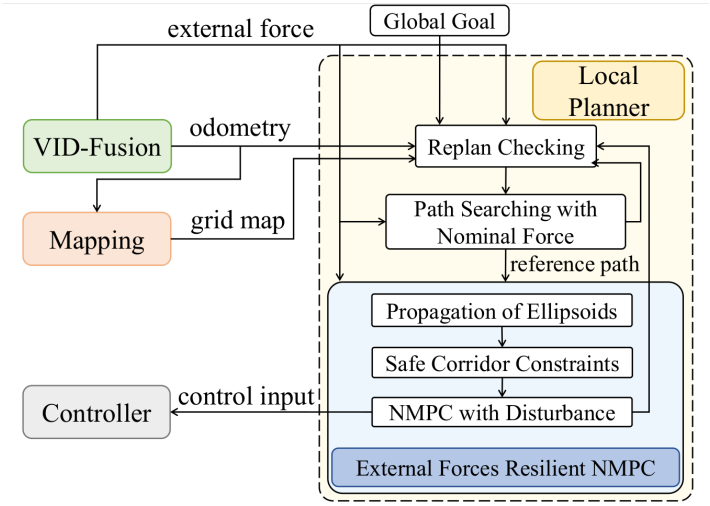


Figure 6-1: System overview diagram of the re-planning framework of the external forces resilient motion planning algorithm [55].

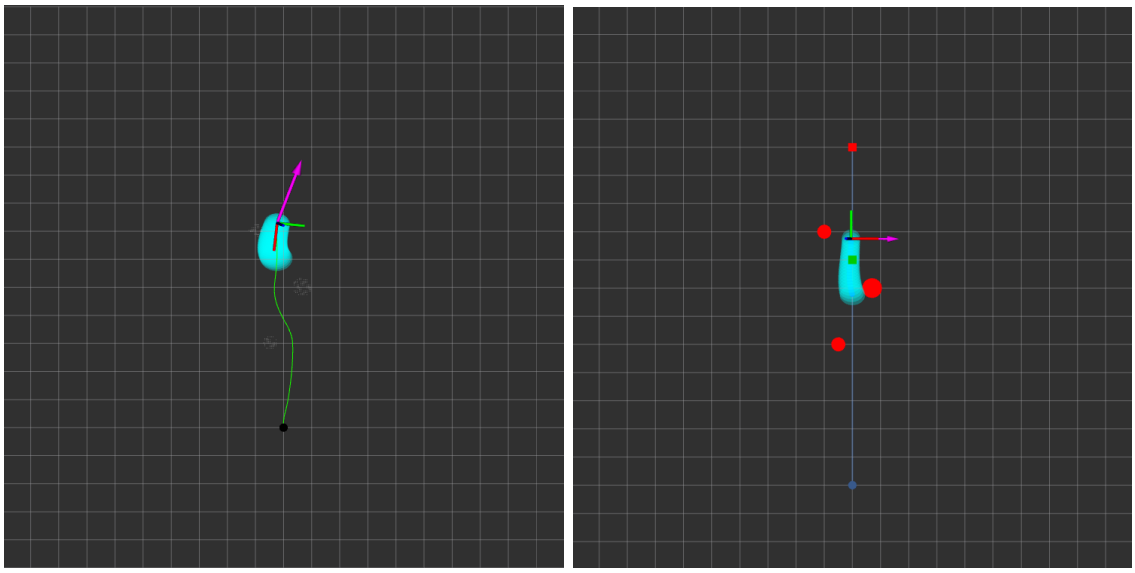
6-2 Scenario

The two methods are compared for the same scenario in the simple simulation environment, displayed in RViz (see 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 $(-0.5, 2)$, $(0.7, 0)$ and $(-1, 2)$ 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 LMPCC are tested with wind pointing in x-direction and the two crossing fans.

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 simple 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 2D. Moreover, the force estimation step with VID-fusion is skipped and the 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 uncertain bounds used the controller and reliability.



(a) External forces resilient planner. **(b)** GP-based LMPCC.

Figure 6-2: Quadrotor traveling between the two goal points with wind acting on it and obstacles populating the environment. The uncertain bounds are shown in cyan.

6-3-1 External forces resilient safe motion planner

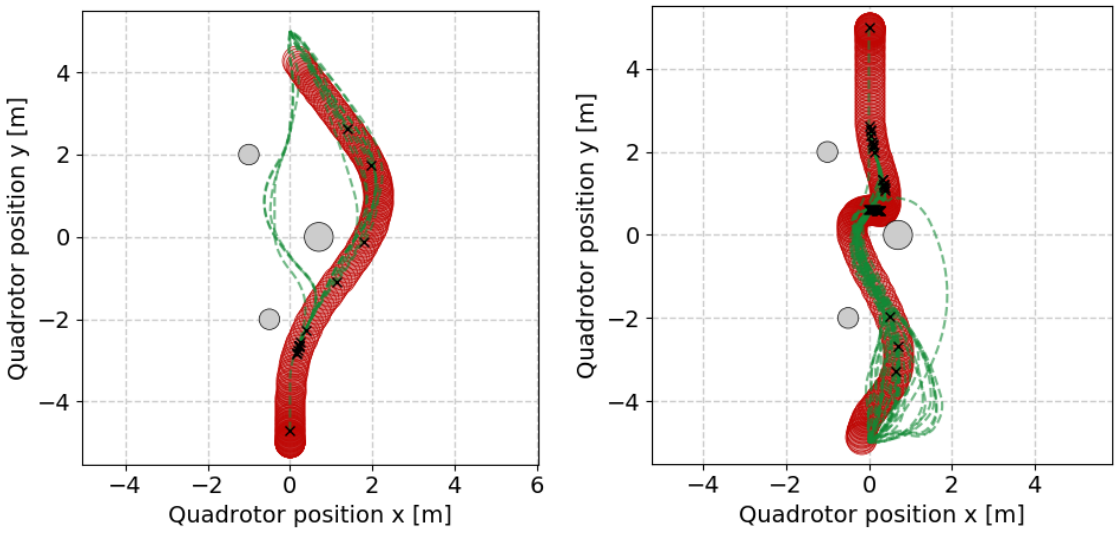
First, the external forces resilient safe motion planner is tested with the wind pointing in x-direction. From the described algorithm, it is expected that the path is re-planned several times, when the magnitude of the wind disturbance exceeds the predefined bound of $0.5\mathrm{m\,s^{-1}}$. Moreover,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

Quadrotor path and re-planning In Figure 6-3, the resulting quadrotor trajectory including the radius of the quadrotor is shown for the quadrotor travelling in both positive (left) and negative (right) y-direction. Moreover, 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 was re-planned and also marks the start of the newly planned path.

Initially, the front-end path planner is planning a path through the obstacles, with 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.

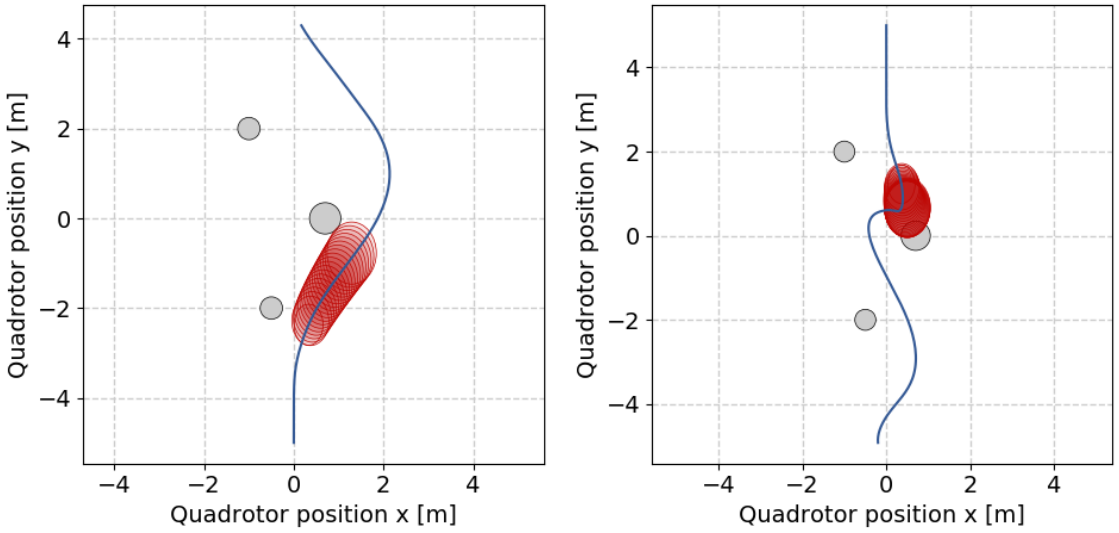
Traveling in the other direction, the algorithm chooses the shorter way through the obstacles. However, when the wind disturbances start acting on the quadrotor this is nearly resulting in a crash. A lot of re-planning iterations are needed to find a feasible path to the goal point, significantly slowing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

The reason for the quadrotor choosing the more conservative path to the goal and multiple



(a) Travelling in positive y-direction. **(b)** 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.



(a) Travelling in positive y-direction. **(b)** Travelling in negative y-direction.

Figure 6-4: Propagated uncertain ellipsoid over one planning horizon by the external forces resilient safe motion planner.

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.

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

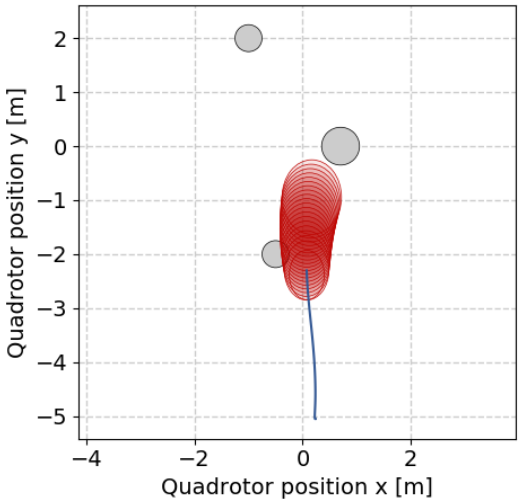
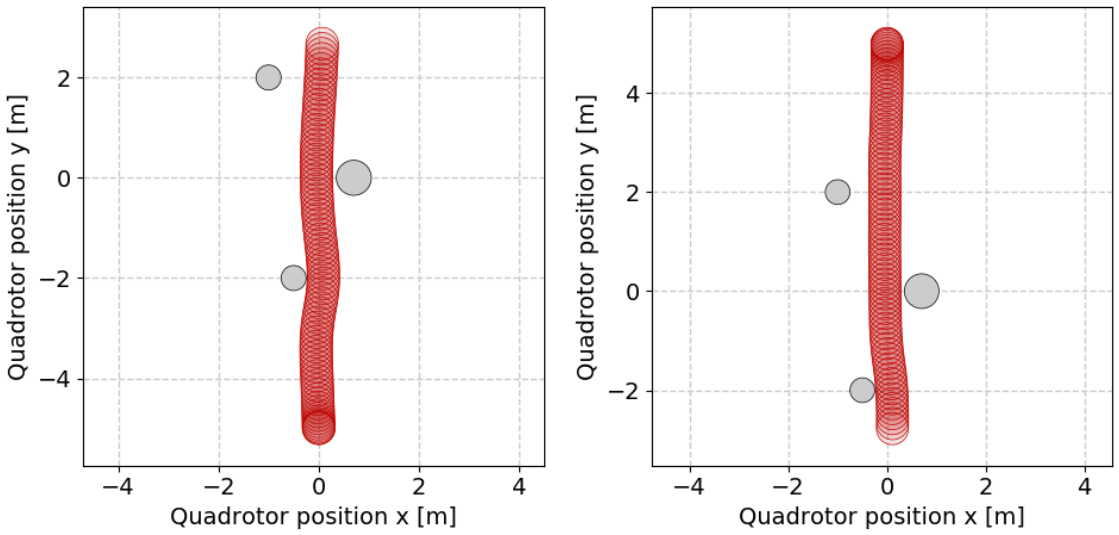


Figure 6-5: Quadrotor getting stuck in the re-planning framework with the wind field generated by two crossing fans.

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 LMPCC

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



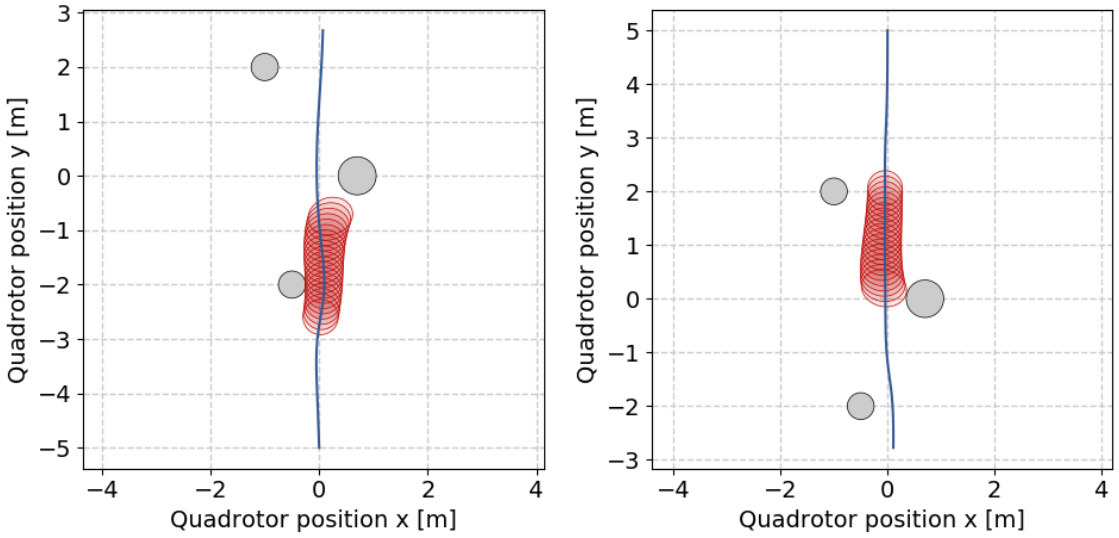
(a) Simple obstacle scenario.

(b) Complex obstacle scenario.

Figure 6-6: Trajectory generated by the Gaussian Process (GP)-based LMPCC controller for the quadrotor travelling from the start point to the end goal.

Quadrotor path As the data-driven LMPCC controller solves the problem of trajectory

planning and control simultaneously, and both the obstacle location and wind field are known beforehand, no re-planning is needed. Instead, onlty 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.



(a) Simple obstacle scenario. **(b)** Complex obstacle scenario.

Figure 6-7: Propagated uncertain ellipsoids over one planning horizon by the GP-based LMPCC controller.

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 formalism to generate an optimal trajectory through the obstacles, results in a less conservative trajectory through the obstacles without having to avoid the obstacles by much.

Other wind fields Repeating the experiment for the wind field with crossing fans results in very similar results. As also here, the force of the wind and its uncertainty are known beforehand, the quadrotor is able to find a feasible trajectory without any crashes.

6-4 Comparison

Based on the theoretical description of the external forces resilient safe motion planner and the experimental validation, the data driven LMPCC motion planner is compared to the state-of-the-art method.

Theoretic comparison A theoretic comparison of the data-driven LMPCC method of this thesis and the method by the authors in [55] 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 forces resilient motion planner obtains information on the force online, and uses it instantly, the data-driven LMPCC 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 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.

External Forces	Data-driven LMPCC
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

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

Experimental comparison The theoretical differences can be validated with the experiments. 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. 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 motion planner 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 motion planner to an online learning setup. It would be expected that here too, the online GP-based motion planner will outperform the forces resilient planner as it provides a model of the force, allowing for predictions into the future.

Moreover, it was validated experimentally that the GP-based motion planner is less conservative than the forces resilient planner despite the missing feedback in the control law that could help contain the uncertainty even more. This 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 was 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.

Finally, it was also shown that combining the motion planner and controller into one optimization problem avoids the re-planning framework which can cause the drone 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.

6-5 Summary and Discussion

Comparing the proposed GP-based motion planner to the current state of the art external forces resilient safe motion planner by [55] showed the potential but also areas of improvement of the proposed 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. In that case 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

Closing this report, the results are summarized and the main findings and conclusions are derived from the results discussed. In addition, the drawbacks of the current implementation are identified, and suggestions for improvement are made for future work.

7-1 Summary

One of the main challenges in developing autonomous drones 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 aimed 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) [3]. 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 drone. A first-order approximation of the attitude dynamics model is identified by sending step signals to the drone 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 drone into an acceleration. The attitude dynamics model had 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 also may 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 Local Model Predictive Contouring Control (LMPCC) to consider both the mean and uncertainty of the resulting probabilistic model. The uncertainty isapproximately propagated using successive linearization and included into the LMPCC controller through the use of ellipsoidal and chance constraints. Chance constraints are chosen to continue with for this research as they have been shown to be less conservative. This implementation of the GP-based LMPCC 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 application 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 LMPCC implementation not being real-time feasible, and also because propagating the uncertainty adds significant computation times to the controller.

The proposed GP-based LMPCC implementation is compared compared to the external forces resilient safe motion planner proposed in [55] 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. Testing both navigation algorithms in the same scenario, it is found that the external forces resilient safe planner struggles in rapidly changing wind conditions, while the GP-based LMPCC controller utilizing a trained disturbance map does not. However, for a fair comparison, the GP-based LMPCC controller should also be implemented online. If the GP-based LMPCC 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 LMPCC 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 was 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 LMPCC 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 theGP-based LMPCC 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 may 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 Model Predictive Control with uncertainty (LMPCC) 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

TODO

Another advantage of using GPs in MPC is that they can be updated in real-time as new data becomes available. This allows the model to adapt to changing conditions and improve the

accuracy of the predictions. This is important for controlling drones in windy environments, where the wind conditions can change rapidly.

- In case different applications of this method taken into consideration, e.g. swarm flight - wind parks where wind conditions are steady TODO: find papers to cite

Bibliography

[1] PX4 Autopilot. https://px4.io/.

[2] Lars Blackmore, Masahiro Ono, and Brian C Williams. Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics*, 27(6):1080–1094, 2011.

[3] Bruno Brito, Boaz Floor, Laura Ferranti, and Javier Alonso-Mora. Model predictive contouring control for collision avoidance in unstructured dynamic environments. *IEEE Robotics and Automation Letters*, 4(4):4459–4466, 2019.

[4] Andrea Carron, Elena Arcari, Martin Wermelinger, Lukas Hewing, Marco Hutter, and Melanie N Zeilinger. Data-driven model predictive control for trajectory tracking with a robotic arm. *IEEE Robotics and Automation Letters*, 4(4):3758–3765, 2019.

[5] Marc Peter Deisenroth. *Efficient reinforcement learning using Gaussian processes*, volume 9. KIT Scientific Publishing, 2010.

[6] Alexandre Didier, Anilkumar Parsi, Jeremy Coulson, and Roy S Smith. Robust adaptive model predictive control of quadrotors. In *2021 European Control Conference (ECC)*, pages 657–662. IEEE, 2021.

[7] Alexandre Didier, Kim P Wabersich, and Melanie N Zeilinger. Adaptive model predictive safety certification for learning-based control. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 809–815. IEEE, 2021.

[8] Ziming Ding, Tiankai Yang, Kunyi Zhang, Chao Xu, and Fei Gao. Vid-fusion: Robust visual-inertial-dynamics odometry for accurate external force estimation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14469–14475. IEEE, 2021.

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

[10] Bara J Emran and Homayoun Najjaran. A review of quadrotor: An underactuated mechanical system. *Annual Reviews in Control*, 46:165–180, 2018.

[11] Laura Ferranti, Bruno Brito, Ewoud Pool, Yanggu Zheng, Ronald M Ensing, Riender Happee, Barys Shyrokau, Julian FP Kooij, Javier Alonso-Mora, and Dariu M Gavrila. Safevru: A research platform for the interaction of self-driving vehicles with vulnerable road users. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1660–1666. IEEE, 2019.

[12] Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7587–7597, 2018.

[13] Gowtham Garimella, Matthew Sheckells, and Marin Kobilarov. Robust obstacle avoidance for aerial platforms using adaptive model predictive control. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5876–5882. IEEE, 2017.

[14] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *Journal of Intelligent and Robotic Systems*, 57(1):65–100, 2010.

[15] Daniel Hentzen, Thomas Stastny, Roland Siegwart, and Roland Brockers. Disturbance estimation and rejection for high-precision multirotor position control. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2797–2804. IEEE, 2019.

[16] Sylvia L Herbert, Mo Chen, SooJean Han, Somil Bansal, Jaime F Fisac, and Claire J Tomlin. FaSTrack: A modular framework for fast and guaranteed safe motion planning. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1517–1522. IEEE, 2017.

[17] HoverGames. https://www.hovergames.com/.

[18] Kuan-Hao Huang. DeepAL: Deep active learning in Python. *arXiv preprint arXiv:2111.15258*, 2021.

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

[20] Mina Kamel, Michael Burri, and Roland Siegwart. Linear vs nonlinear MPC for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine*, 50(1):3463–3469, 2017.

[21] Mina Kamel, Thomas Stastny, Kostas Alexis, and Roland Siegwart. Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system. In *Robot operating system (ROS)*, pages 3–39. Springer, 2017.

[22] Farid Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 29(2):315–378, 2012.

[23] Eric C Kerrigan and Jan M Maciejowski. Soft constraints and exact penalty functions in model predictive control. 2000.

[24] Shreyas Kousik, Sean Vaskov, Fan Bu, Matthew Johnson-Roberson, and Ram Vasudevan. Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots. *The International Journal of Robotics Research*, 39(12):1419–1469, 2020.

[25] Amon Lahr, Andrea Zanelli, Andrea Carron, and Melanie N Zeilinger. Zero-order optimization for Gaussian process-based model predictive control. *arXiv preprint arXiv:2211.15522*, 2022.

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

[27] Lebao Li, Lingling Sun, and Jie Jin. Survey of advances in control algorithms of quadrotor unmanned aerial vehicle. In *2015 IEEE 16th international conference on communication technology (ICCT)*, pages 107–111. IEEE, 2015.

[28] Miao Liu, Girish Chowdhary, Bruno Castra Da Silva, Shih-Yuan Liu, and Jonathan P How. Gaussian processes for learning and control: A tutorial with examples. *IEEE Control Systems Magazine*, 38(5):53–86, 2018.

[29] Giuseppe Loianno and Davide Scaramuzza. Special issue on future challenges and opportunities in vision-based drone navigation. *Journal of Field Robotics*, 37(4):495–496, 2020.

[30] Matthias Lorenzen, Mark Cannon, and Frank Allgöwer. Robust MPC with recursive model update. *Automatica*, 103:461–471, 2019.

[31] Robert Mahony, Vijay Kumar, and Peter Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics and Automation magazine*, 19(3):20–32, 2012.

[32] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.

[33] Mohit Mehndiratta and Erdal Kayacan. Gaussian process-based learning control of aerial robots for precise visualization of geological outcrops. In *2020 European Control Conference (ECC)*, pages 10–16. IEEE, 2020.

[34] Tiago P Nascimento and Martin Saska. Position and attitude control of multi-rotor aerial vehicles: A survey. *Annual Reviews in Control*, 48:129–146, 2019.

[35] Huan Nguyen, Mina Kamel, Kostas Alexis, and Roland Siegwart. Model predictive control for micro aerial vehicles: A survey. In *2021 European Control Conference (ECC)*, pages 1556–1563. IEEE, 2021.

[36] Michael O'Connell, Guanya Shi, Xichen Shi, and Soon-Jo Chung. Meta-learning-based robust adaptive flight control under uncertain wind conditions. *arXiv preprint arXiv:2103.01932*, 2021.

[37] Chris J Ostafew, Angela P Schoellig, and Timothy D Barfoot. Robust constrained learning-based NMPC enabling reliable mobile robot path tracking. *The International Journal of Robotics Research*, 35(13):1547–1563, 2016.

[38] 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*, 12:2825–2830, 2011.

[39] Robert Penicka and Davide Scaramuzza. Minimum-time quadrotor waypoint flight in cluttered environments. *IEEE Robotics and Automation Letters*, 7(2):5719–5726, 2022.

[40] Lun Quan, Luxin Han, Boyu Zhou, Shaojie Shen, and Fei Gao. Survey of UAV motion planning. *IET Cyber-systems and Robotics*, 2(1):14–21, 2020.

[41] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, 2017.

[42] Angel Romero, Sihao Sun, Philipp Foehn, and Davide Scaramuzza. Model predictive contouring control for near-time-optimal quadrotor flight. *arXiv preprint arXiv:2108.13205*, 2021.

[43] Wilko Schwarting, Javier Alonso-Mora, Liam Paull, Sertac Karaman, and Daniela Rus. Safe nonlinear trajectory generation for parallel autonomy with a dynamic vehicle model. *IEEE Transactions on Intelligent Transportation Systems*, 19(9):2994–3008, 2017.

[44] Sumeet Singh, Benoit Landry, Anirudha Majumdar, Jean-Jacques Slotine, and Marco Pavone. Robust feedback motion planning via contraction theory. *The International Journal of Robotics Research*, 2019.

[45] Ewoud JJ Smeur, Guido CHE de Croon, and Qiping Chu. Cascaded incremental nonlinear dynamic inversion for MAV disturbance rejection. *Control Engineering Practice*, 73:79–90, 2018.

[46] Nitin Sydney, Brendan Smyth, and Derek A Paley. Dynamic control of autonomous quadrotor flight in an estimated wind field. In *52nd IEEE Conference on Decision and Control*, pages 3609–3616. IEEE, 2013.

[47] Andrea Tagliabue and Jonathan P How. Airflow-inertial odometry for resilient state estimation on multirotors. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5736–5743. IEEE, 2021.

[48] Michalis Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR, 2009.

[49] Teodor Tomić and Sami 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*, pages 4197–4204. IEEE, 2014.

[50] Chau T Ton and William 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)*, pages 937–942. IEEE, 2012.

[51] Guillem Torrente, Elia Kaufmann, Philipp Föhn, and Davide Scaramuzza. Data-driven MPC for quadrotors. *IEEE Robotics and Automation Letters*, 6(2):3769–3776, 2021.

[52] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. acados – a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 2021.

[53] Steven Waslander and Carlos Wang. Wind disturbance estimation and rejection for quadrotor position control. In *AIAA Infotech@ Aerospace conference and AIAA unmanned... Unlimited conference*, page 1983, 2009.

[54] Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[55] Yuwei Wu, Ziming Ding, Chao Xu, and Fei Gao. External forces resilient safe motion planning for quadrotor. *IEEE Robotics and Automation Letters*, 6(4):8506–8513, 2021.

[56] Hai Zhu and Javier Alonso-Mora. Chance-constrained collision avoidance for MAVs in dynamic environments. *IEEE Robotics and Automation Letters*, 4(2):776–783, 2019.

[57] Andrew Zulu and Samuel John. A review of control algorithms for autonomous quadrotors. *arXiv preprint arXiv:1602.02622*, 2016.

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
NRMSE	Normalized Root Mean Squared Error
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
LMPCC	Local Model Predictive Contouring 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

List of Symbols

β	Time delay
$\boldsymbol{\theta}$	Hyperparameter vector
$\dot{\phi}$	Roll rate
$\dot{\psi}$	Yaw rate
$\dot{\theta}$	Pitch rate
$\mathbf{B}_d$	Mapping of the uncertainty
$\mathbf{d}$	Disturbance vector
$\mathbf{d}$	Disturbance vector
$\mathbf{d}(\mathbf{p}_k)$	Disturbance vector
$\mathbf{p}$	Position
$\mathbf{u}$	Inducing points
$\mathbf{u}$	Input space
$\mathbf{w}_k$	Process noise
$\mathbf{x}$	State space
$\mathcal{B}$	Occupancy volume
$\mathcal{C}$	Collision region
$\mathcal{D}$	Dataset of observations
$\mathcal{I}_o$	Obstacle space
$\mathcal{U}$	Input constraint set
$\mathcal{X}$	State constraint set
ν	Smoothness parameter of the Matern kernel
ϕ	Roll
ψ	Yaw
θ	Pitch
$\{A\}$	World frame
$\{B\}$	Body frame
a_T	Linear scaling between the desired thrust and thrust command
b_T	Linear scaling between the battery voltage and thrust command
g	Gravitation constant
I	Inertia
k	Steady-state gain
$K(X, X)$	Kernel matrix
k_D*	Normalized drag coefficient
k_D	Drag coefficient
l_d	Horizontal length scale for output dimension d
m	Mass
R	Rotation matrix
s	Path variable

T_{c}	Thrust command
T_{h}	Hover thrust
T_s	Sampling time
U	Battery voltage
x	Position in x
y	Position in y
z	Position in z
σ_ε	Noise covariance
σ_f	Output variance
τ	Time constant