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

Johanna Probst

Master of Science Thesis

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 3, 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

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 3, 2023

Johanna Probst

Chapter 1

Nominal quadrotor model

This chapter will cover the derivation of the nominal quadrotor equations of motion (EOM) based on first-principle models, as well as discussing how the inner-loop attitude 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.

1-1 First principle quadrotor model

The nominal quadrotor model can be found using first principle modeling techniques based on simplifying assumptions of real-world dynamics. We make use of Newton-Euler equations of motion (EOM) to describe the quadrotor dynamics.

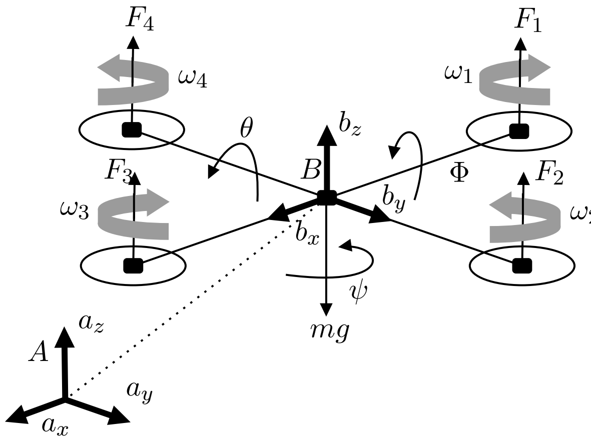


Figure 1-1: Quadrotor configuration with world frame A and body frame B.

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 and z, and the three Euler angles roll, pitch, and yaw. 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 1-1 to describe the quadrotor motion: The position of the quadrotor $\mathbf{p} = [x, y, z]^T \in \mathbb{R}^3$, and velocity $\mathbf{v} = [v_x, v_y, v_z] \in \mathbb{R}^3$ are expressed in the world frame $\{A\}$. The roll, pitch, and yaw angles of the quadrotor are denoted by ϕ, θ, and ψ, respectively. They specify the orientation of the quadrotor body frame $\{B\}$.

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 is described as:

$$R = R_{B\to A} = R_Z(\psi)R_X(\phi)R_Y(\theta) \tag{1-1a}$$
$$= \begin{bmatrix} c\psi c\theta & c\psi s\phi s\theta - c\phi s\psi & s\psi s\phi + c\psi c\phi s\theta \\ c\theta s\psi & c\phi c\psi + s\psi s\phi s\theta & c\phi s\theta s\psi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\phi c\theta \end{bmatrix}. \tag{1-1b}$$

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

$$\dot{\mathbf{p}} = \mathbf{v} \tag{1-2a}$$
$$m\dot{\mathbf{v}} = -mg\mathbf{a}_3 + R\mathbf{T} \tag{1-2b}$$

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

$$m\dot{v}_x = T(\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi) \tag{1-3a}$$
$$m\dot{v}_y = T(-\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi) \tag{1-3b}$$
$$m\dot{v}_z = T(\cos\phi\cos\theta) - mg. \tag{1-3c}$$

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:

$$\dot{\phi} = \frac{1}{\tau_\phi}(k_\phi\phi_d - \phi) \tag{1-4a}$$
$$\dot{\theta} = \frac{1}{\tau_\theta}(k_\theta\theta_d - \theta) \tag{1-4b}$$
$$\dot{\psi} = \dot{\psi}_d \tag{1-4c}$$
$$T = T_d \tag{1-4d}$$

with system inputs $\mathbf{u} = [\phi_d, \theta_d, \psi_d, T_d]$. The coefficients of the first-order model are found using system identification.

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 [7]. One can account for these effects by adding an external drag force proportional to the system velocity:

$$m\dot{\mathbf{v}} = -mg\mathbf{a}_3 + R\mathbf{T} - D\mathbf{v} \tag{1-5}$$

where $D = \mathrm{diag}\left(k_D, k_D, k_D\right)$ and k_D is the drag coefficient.

Derived quadrotor model The resulting quadrotor model is summarized below.

$$\dot{\mathbf{p}} = \mathbf{v} \tag{1-6a}$$

$$m\dot{v}_x = T_d\left(\sin(\psi)\sin(\phi) + \cos(\phi)\sin(\theta)\cos(\psi)\right) - k_D v_x \tag{1-6b}$$

$$m\dot{v}_y = T_d\left(-\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi)\right) - k_D v_y \tag{1-6c}$$

$$m\dot{v}_z = T_d\left(\cos(\phi)\cos(\theta)\right) - mg - k_D v_z \tag{1-6d}$$

$$\dot{\phi} = \frac{1}{\tau_\phi}(k_\phi \phi_d - \phi) \tag{1-6e}$$

$$\dot{\theta} = \frac{1}{\tau_\theta}(k_\theta \theta_d - \theta) \tag{1-6f}$$

$$\dot{\psi} = \dot{\psi}_d \tag{1-6g}$$

with state vector $\mathbf{x} = \left[\mathbf{p}^T, \mathbf{v}^T, \phi, \theta, \psi\right]^T$ and the input vector $\mathbf{u} = \left[\phi_d, \theta_d, \dot{\psi}_d, T_d\right]^T$.

Linear quadrotor model For propagating the uncertainties, we will make use of the linear system model. Assuming small attitude angles, the quadrotor dynamics model can be linearized around its hovering condition with $\dot{v}_z = 0$. Furthermore, the vehicle heading is aligned with the inertial frame x-axis, such that the yaw angle $\psi = 0$. The linearized system dynamics are:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -k_D* & 0 & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 0 & -k_D* & 0 & -g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -k_D* & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\phi} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\theta} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ \phi \\ \theta \\ \psi \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k_\phi}{\tau_\phi} & 0 & 0 & 0 \\ 0 & \frac{k_\theta}{\tau_\theta} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \phi_d \\ \theta_d \\ \dot{\psi}_d \\ T_d \end{bmatrix} \tag{1-7}$$

with the mass-normalized drag coefficient $k_D* = \frac{k_D}{m}$ [4].

1-2 System identification of the attitude dynamics

This section describes the system identification experiments that are performed to identify the coefficients of the attitude dynamics model described in section TODO. Additionally, a thrust model is identified that translates the thrust control inputs to the desired acceleration of the drone.

1-2-1 Attitude dynamics and thrust model

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

$$\dot{\phi} = \frac{1}{\tau_\phi} \left(k_\phi \phi_d - \phi\right) \tag{1-8a}$$

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

Here k_ϕ and k_θ are the steady-state gains and τ_ϕ and τ_θ are the time constants. 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:

$$\dot{\phi} = \frac{1}{\tau_\phi} \left(k_\phi (t-\beta) \phi_d - \phi\right) \tag{1-9a}$$

$$\dot{\theta} = \frac{1}{\tau_\theta} \left(k_\theta (t-\beta) \theta_d - \theta\right). \tag{1-9b}$$

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 $T_c = 0 \ldots 1$ that is translated into an PWM (TODO) signal to the motors, resulting in a desired rotor speed and thrust of the drone. However, for the model in TODO, the desired thrust command T_d 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:

$$T_c = a_T \cdot T_d + T_h. \tag{1-10}$$

The linear scaling a_T and an offset, corresponding to the hovering thrust T_h, are to be identified. 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 at $14.5V$, resulting the thrust dynamics model:

$$T_c = a_T \cdot T_d + b_T \cdot (U - 14.5V) + T_h. \tag{1-11}$$

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

1-2-2 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.

Experimental setup

Physical experiments are performed on the HoverGames (TODO: cite) 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 (TODO: citee). 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 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 ROS software library. PX4 supports MAVROS messages as a bridge between the flight controller 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.

TODO: put picture of the physical drone in the lab.

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

TODO: put picture of the drone in gazebo environment.

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 and yaw rate 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 0.5 Hz 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 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 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:

$$T_d = \frac{(m_{\mathbf{tot}} - m_{\mathbf{nom}})}{m_{\mathbf{tot}}} g + g \tag{1-12}$$

with the nominal mass of the drone $m_{\mathbf{nom}}$ and $m_{\mathbf{tot}}$ the total mass, given the added or removed weight.

1-2-3 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 tfest function to cover a wide range of scenarios.

Roll and pitch model parameters with time delay To estimate the time delay, the MATLAB function deleyest 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 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 $(1 - NRMSE)100\%$, where $NRMSE$ is the normalized root mean square error between the predicted and measured output of the model.

1-2-4 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 TODO and for the real-world experiment in Table TODO. 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.

Parameter	$k_\phi = k_\theta$	$\tau_\phi = \tau_\theta$	β	a_T	T_h
Value	0.963	0.104	0.08	23.258	0.675

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

Parameter	k_ϕ	k_θ	τ_ϕ	τ_θ	β	a_T	b_T	T_h
Value	0.954	0.950	0.065	0.074	0.1	0.0329	-0.0456	0.2911

Table 1-2: Identified parameter values for the attitude and thrust dynamics on the physical drone.

The response to the input command for the roll dynamics and the behavior of the fitted model are shown in Figure TODO. The left picture shows the behavior of the system for one of the datasets it was trained with while the right picture 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.

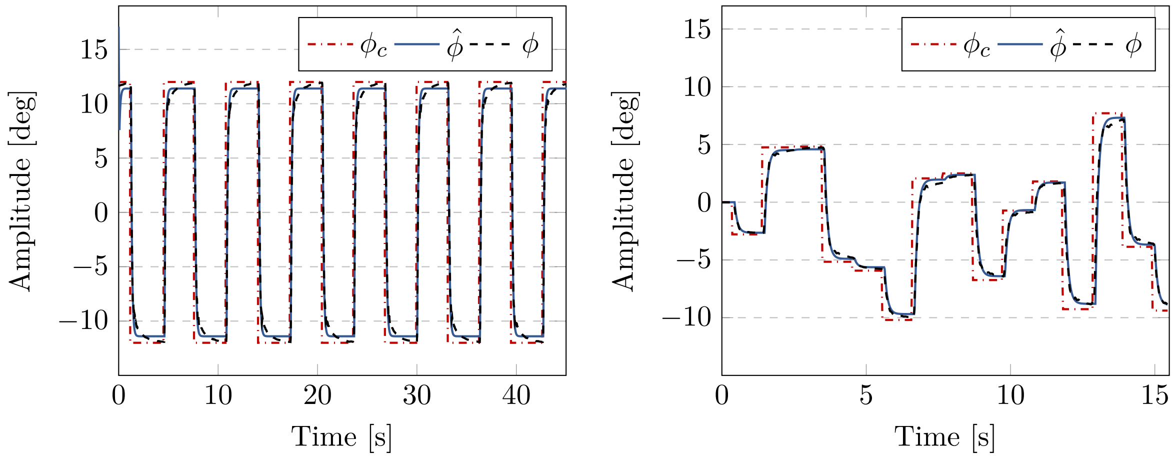


Figure 1-2: test1

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 TODO. 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. 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.

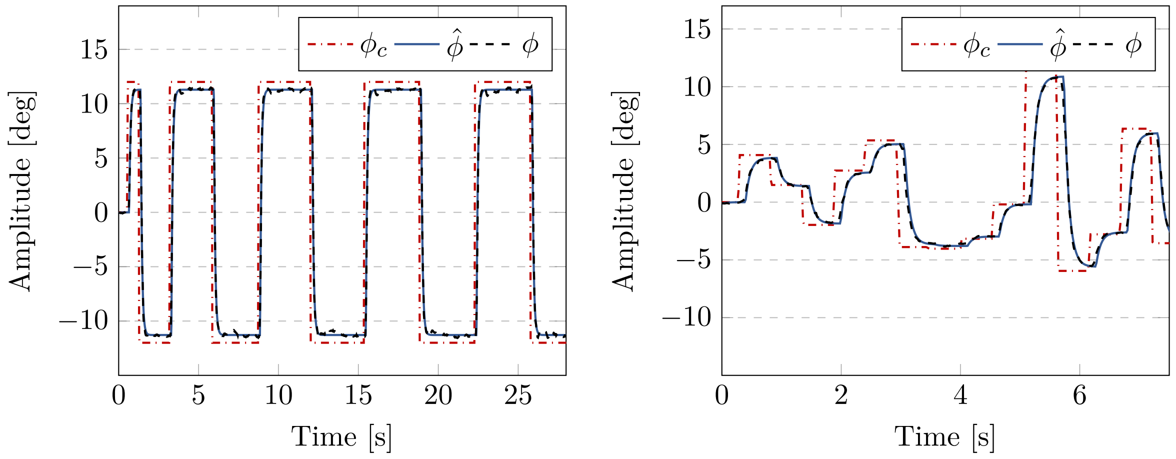


Figure 1-3: test2

The input responses in roll and pitch direction on the real drone are shown in Figure TODO and TODO, 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.

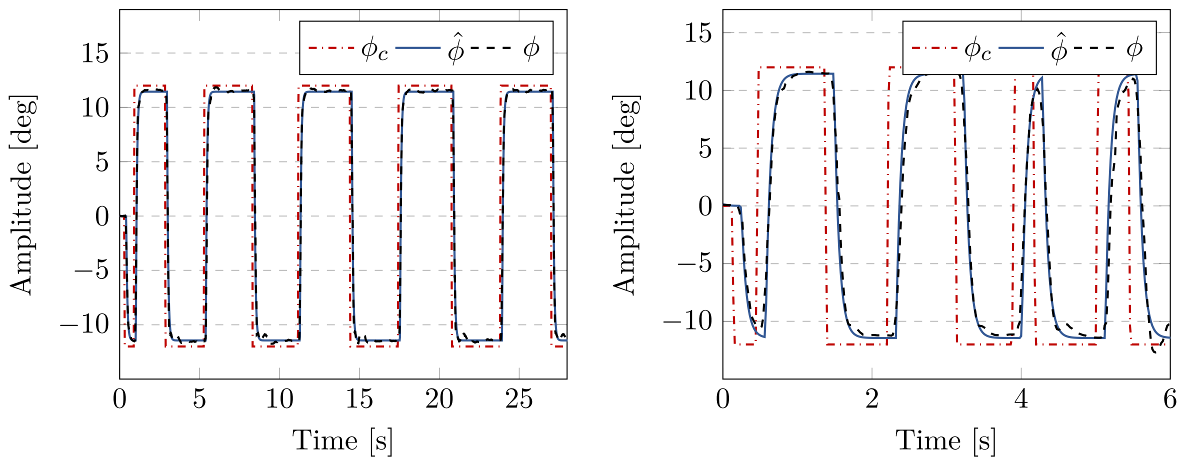


Figure 1-4: test3

For the thrust dynamics model in simulation, the input command T_c between 0 and 1 is scaled according to the identified hovering thrust and linear scaling factor to find the desired thrust T_d and compared to the measured acceleration of the system a. This is shown in Figure TODO. Again, the training data on the left is compared, and a validation dataset 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 MPC like controller that does correct for model mismatch by design.

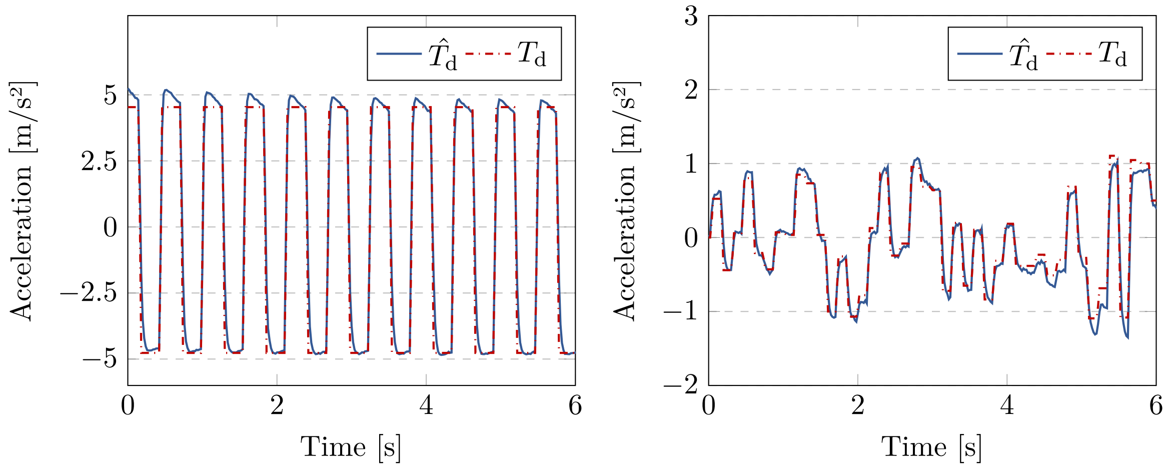


Figure 1-5: test4

As the experiments for identifying the thrust dynamics are performed slightly different, the fitted curves are also shown in a different way. Figure TODO 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 $T_{d,1}$ indicates the thrust curve without any added weight. For $T_{d,2}...T_{d,6}$, weight is added in steps of $200g$, resulting in a maximum added weight of $1kg$. To this behavior, the linear model in TODO is fitted, resulting in the solid curves displayed in Figure TODO. The goodness of fit of this model is 65.85%.

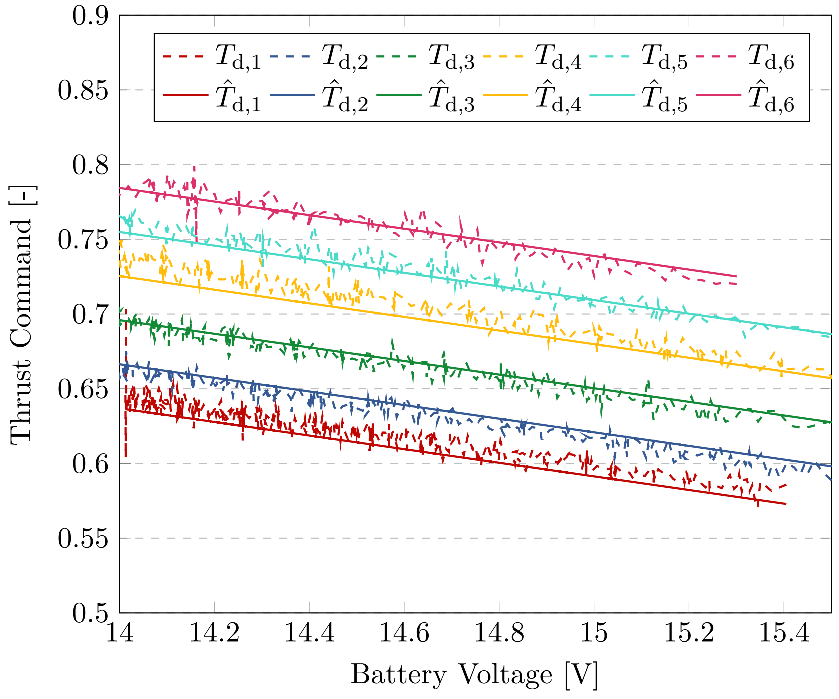


Figure 1-6: test

However, when comparing the model to validation 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 TODO. This is also the case for validation datasets collected on another day. 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 TODO.

1-3 Summary

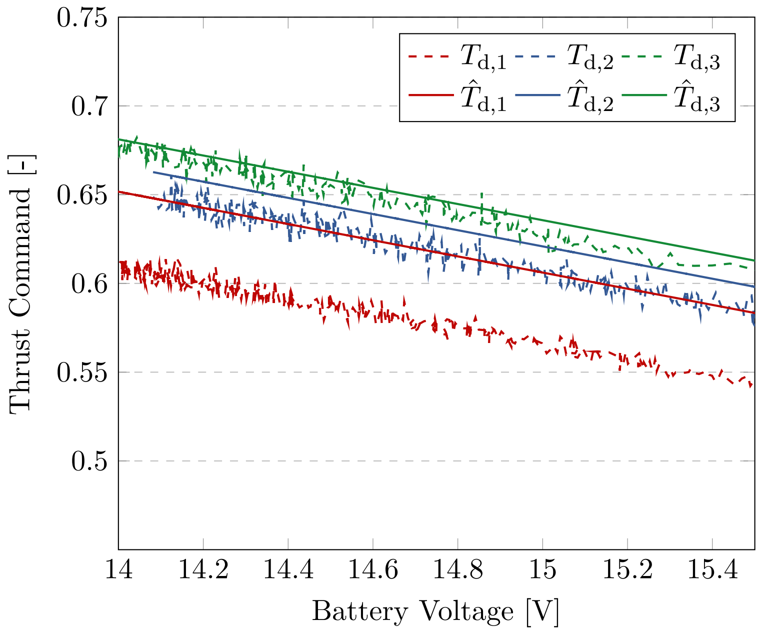


Figure 1-7: test

	Roll		Pitch		Thrust	
	Sim	Exp	Sim	Exp	Sim	Exp
Training	95.12%	95.89%	-	95.76%	84.59%	65.85%
Validation	91.16%	92.37%	-	87.08%	57.86%	10.82%

Table 1-3: Goodness of fit for the conducted experiments to identify the parameters of the roll, pitch and thrust dynamics model.

Chapter 2

Gaussian process disturbance map

TODO: write introduction here connecting the chapters

2-1 Gaussian Process Regression

In the following section, the mathematical background of the Gaussian process regression [9] (GPR) is explained. The goal of GPR is to obtain an approximation of the nonlinear function that describes the behavior of wind disturbances with uncertainty. An introduction to the mathematical foundation that Gaussian processes are built on is given and how this foundation can be used to train and predict the nonlinear model. An explanation of sparse Gaussian processes to reduce computational costs and the idea of active learning with Gaussian processes to optimally capture the nonlinear perturbation map is also given.

2-1-1 Mathematical Background

The basic building block of the Gaussian process 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 $\mathbf{x}_*$ [9].

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

$$y_i = f(\mathbf{x}_i) + \varepsilon_i \tag{2-1}$$

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

denotes the $n \times n$ matrix of the 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:

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

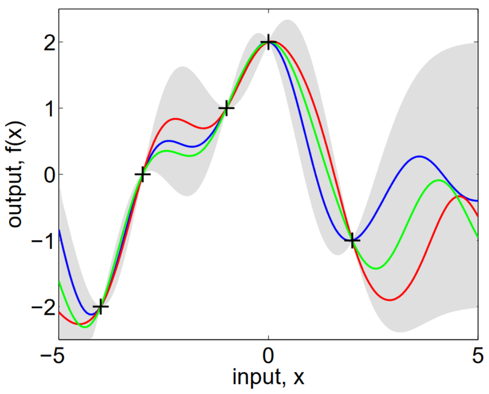


Figure 2-1: Gaussian process posterior distribution conditioned on five, noise free observations. The grey area represents the mean ± two times the standard deviation. In colour are three random samples drawn from the posterior distribution [9].

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

$$p(\mathbf{f}_* \mid X, \mathbf{y}, X_*) \sim \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \text{ where} \tag{2-3a}$$

$$\boldsymbol{\mu}_* = K(X_*,X)\left[K(X,X)+\sigma_\varepsilon^2 I\right]^{-1}\mathbf{y}, \tag{2-3b}$$

$$\boldsymbol{\Sigma}_* = K(X_*,X_*) - K(X_*,X)\left[K(X,X)+\sigma_\varepsilon^2 I\right]^{-1}K(X,X_*). \tag{2-3c}$$

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

$$\mu_* = \mathbf{k}_*^\top\left(K+\sigma_\varepsilon^2 I\right)^{-1}\mathbf{y} \tag{2-3d}$$

$$\Sigma_* = k(x_*,x_*) - \mathbf{k}_*^\top\left(K+\sigma_\varepsilon^2 I\right)^{-1}\mathbf{k}_* \tag{2-3e}$$

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 3-3.

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

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

Another relevant kernel is the Matern kernel

$$k_{\mathrm{MAT}}\left(\mathbf{x}, \mathbf{x}'\right) = \frac{1}{\Gamma(\nu) 2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{l} d\left(\mathbf{x}, \mathbf{x}'\right) \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu}}{l} d\left(\mathbf{x}, \mathbf{x}'\right) \right). \tag{2-5}$$

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. Hyperparameters of the SE and Matern kernel are the horizontal length scale for each input dimension l_d and the output variance ${\sigma_f}^2$. 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 ${\sigma_f}^2$ determines the average distance of the function away from its mean. In addition, the noise covariance ${\sigma_\varepsilon}^2$ 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 $K_\theta = K(\mathbf{x}, \mathbf{x})(\boldsymbol{\theta})$:

$$\mathcal{L} = \log p(\mathbf{y} \mid X, \boldsymbol{\theta}) = -\frac{1}{2} \mathbf{y}^\top \left(K_\theta + \sigma_\varepsilon^2 I\right)^{-1} \mathbf{y} - \frac{1}{2} \log \left|K_\theta + \sigma_\varepsilon^2 I\right| - \frac{n}{2} \log(2\pi). \tag{2-6}$$

The hyperparameter vector:

$$\hat{\boldsymbol{\theta}} \in \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{y} \mid X, \boldsymbol{\theta}) \tag{2-7}$$

is the maximum likelihood estimate of the hyperparameters, which trades off model complexity versus the data fit [2]. Maximizing the likelihood is a non-convex, nonlinear optimization problem which can be solved by hand or using avialable libraries such as GPyTorch (TODO:cite) or sklearn.

Computational complexity Training the GP with a training set of size n requires $\mathcal{O}\left(n^3\right)$ per gradient step due to the inversion of the Kernel matrix $K(X, X)$ which makes training very expensive [9]. To make predictions, the inversion of the kernel matrix demands most operations and results in a computational complexity of $\mathcal{O}\left(n^3\right)$ per prediction. Cholesky factorization can help reduce the computational complexity of the matrix inversion to $\mathcal{O}\left(n^3/6\right)$. Sparse GPs can further reduce the complexity, which motivates the use of it.

2-1-2 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 or active test set is used to form the covariance matrix $K(X, X)$. The complexity of inverting the covariance matrix is thereby reduced to $\mathcal{O}\left(nm^2\right)$ - a significant speed up.

Choosing inducing inputs Using fewer inputs to train and predict GPs also reduces the accuracy and expressiveness of the GP. Selecting the inducing inputs can be difficult. If it is selected greedily from the available data, it becomes difficult to include all the necessary information. A variational approach [8] 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 u, that are optimized for in the course of training. The posterior distribution in TODO is approximate as a multivariate Gaussian $q(u)$ over the inducing variables u with variance μ_u and covariance Σ_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 TODO 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 u, one can define the inducing variable $w := \mathit{chol}(K_u u)^{-1} v$, where $v = u - \mu_u$ and chol refers to Cholesky decomposition. The variational parameters are now the mean vector m_w and covariance matrix $S_w w$. From this, the posterior sparse GP formulation can be computed as follows:

$$f(\cdot) \sim \mathcal{GP}\left(\mu(\cdot) + \mathbf{k_u} \mathbf{L_{uu}^{-\top}} \mathbf{m_w}, k\left(\cdot, ,'\right) - \mathbf{k_{\cdot u}} \mathbf{L_{uu}^{-\top}} \left(\mathbf{I} - \mathbf{S_{ww}}\right) \mathbf{L_{uu}^{-1}} \mathbf{k_{u'}}\right)$$

denoting chol $(\mathbf{K_{uu}})$ as $\mathbf{L_{uu}}$ and making use of the identity $\mathbf{K_{uu}^{-1}} = \mathbf{L_{uu}^{-\top}} \mathbf{L_{uu}^{-1}}$. This parametrization of the inducing variables can facilitate the optimization and speed up the prediction. The method has been implemented in GPyTorch [3] which is a GP library using PyTorch for standard and sparse GP inference.

2-1-3 Active Learning

Active Leaning 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 active learning are the model, the uncertainty measure and the querying strategy.

The advantage of using active learning with Gaussian processes directly provides the uncertainty as part of it's model. The query strategy is chosen such that the most uncertain points are sampled at each iteration. Afterwards, the Gaussian process 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 active learning is shown in Figure TODO.

2-2 Experiment Setup

(TODO: find better title)

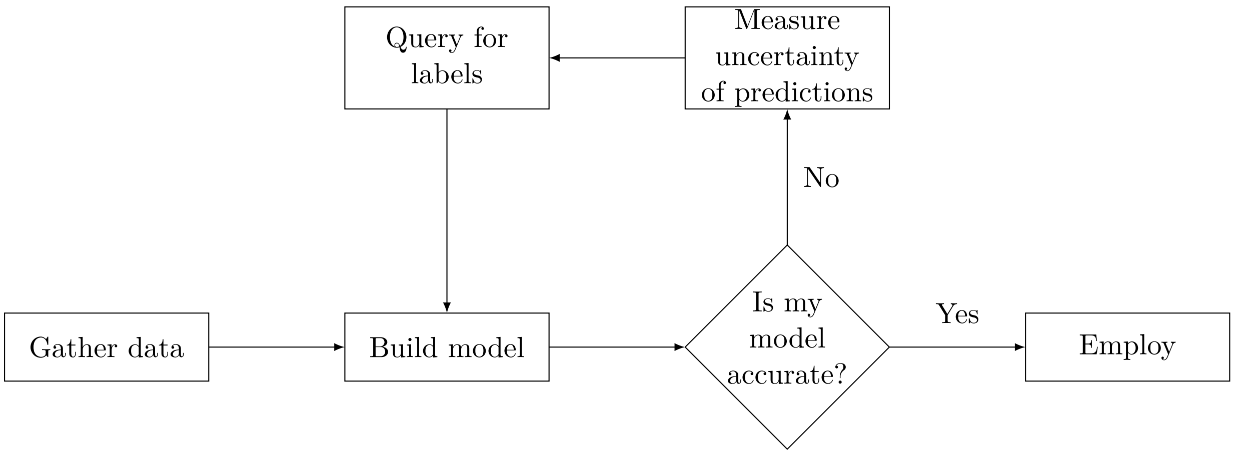


Figure 2-2: Caption

2-2-1 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 nearly match the derived quadrotor model in section TODO, including the identified model parameters. To introduce some model mismatch which is to be expected in real-world, a higher drag is added to the simulated model and the identified parameters are varied slightly. The environment is programmed using a Python client library for ROS, to keep the quadrotor control in the ROS environment.

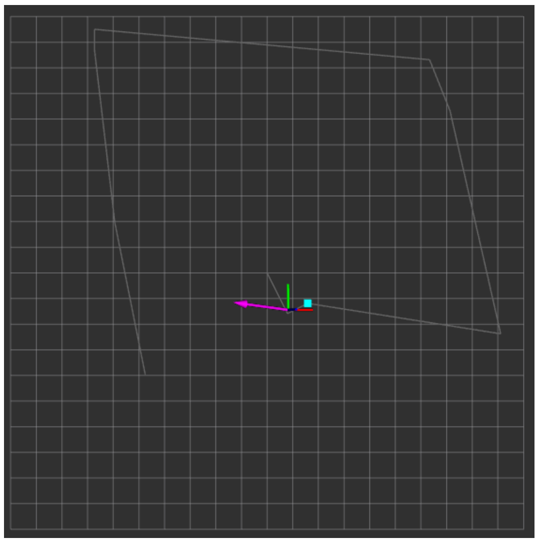
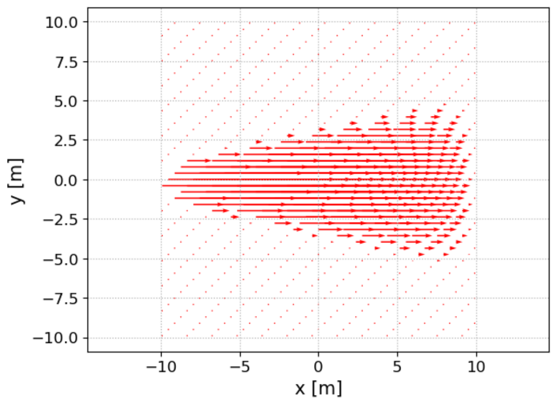


Figure 2-3: TODO

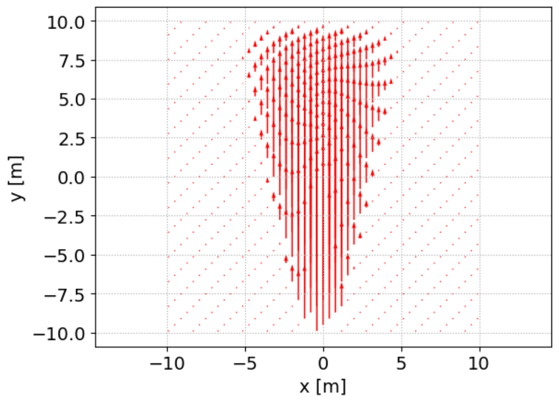
The quadrotor is controlled by sending attitude commands, i.e. $\mathbf{u_c} = \left[\phi_c, \theta_c, \dot{\psi}_c, T_c\right]^T$. The simple simulation environment uses an RK4 method to integrate the quadrotor dynamics and returns the odometry data of the quadrotor. This happens at a rate of 20Hz. The quadrotor is assumed to move in a grid of 20 m × 20 m, in which the wind acts on the quadrotor. 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 TODO, where 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.

Generated wind fields

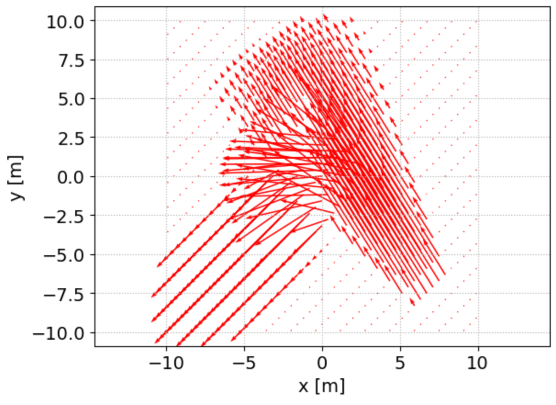
The wind field is simulated by adding a force term to the quadrotor dynamics in v_x and v_y, changing the speed of the quadrotor. The simple simulation supports adding a constant wind in both directions, which is equal across the entire environment. That force can moreover be altered with Gaussian noise, simulating the randomness of the wind.



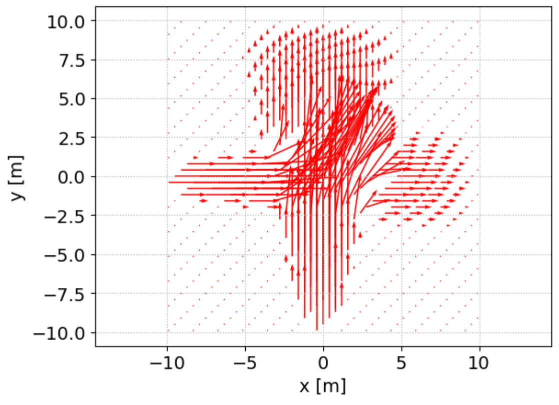
(a) Network 1



(b) Network 2



(c) Network 3



(d) Network 4

Figure 2-4: Generated wind fields

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.

The wind plugin is used to generate wind fields that can easily be recreated 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 spead 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 it is also experimented with more complex cases where two fans cross and add noise to the scenarios. A visualization of the different wind fields is shown in Figures TODO to TODO.

2-2-2 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 current quadrotor position $\mathbf{p}_k = [x_k, y_k]^T$ can be calculated as follows:

$$\begin{aligned} d_{v_x}(\mathbf{p}_k) &= \frac{\hat{v}_{x,k} - v_{x,k}}{\Delta T} \\ d_{v_y}(\mathbf{p}_k) &= \frac{\hat{v}_{y,k} - v_{y,k}}{\Delta T}. \end{aligned} \tag{2-8}$$

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

Multivariate GP model Classic GPs, as described in section TODO, are only capable of modelling nonlinear dynamics with one output. As the disturbance output of the Gaussian process model is a multivariate target, i.e. $\mathbf{d} \in \mathbb{R}^2$, each output is trained independently. Two GP's are trained for the two disturbance directions using the same quadrotor position as input, but only one of the disturbances as training target. This assumes that the two wind directions are computationally independent. The resulting disturbance vector is:

$$\mathbf{d}(\mathbf{p}) = \begin{bmatrix} d_{v_x}(\mathbf{p}) \sim \mathcal{N}\left(\mu^{d_{v_x}}(\mathbf{p}), \Sigma^{d_{v_x}}(\mathbf{p})\right) \\ d_{v_y}(\mathbf{p}) \sim \mathcal{N}\left(\mu^{d_{v_y}}(\mathbf{p}), \Sigma^{d_{v_y}}(\mathbf{p})\right) \end{bmatrix}. \tag{2-9}$$

The predictive distribution is given by:

$$\boldsymbol{\mu}_d = [\mu_{v_x}, \mu_{v_y}]^\top \tag{2-10a}$$

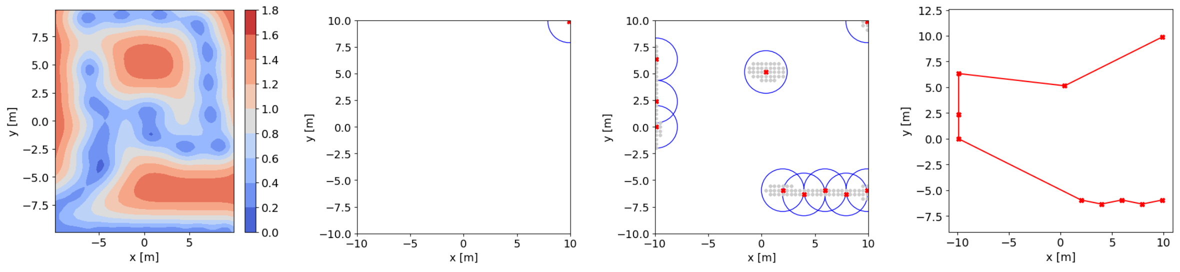
$$\boldsymbol{\Sigma}_d = \operatorname{diag}\left(\begin{bmatrix} \Sigma^2_{v_x} & \Sigma^2_{v_y} \end{bmatrix}\right) \tag{2-10b}$$

2-2-3 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 to cover as much area as possible. 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 active learning, explained in section TODO.

Sampling N 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 $\bar{\sigma}$.



(a) Network 1 **(b)** Network 2 **(c)** Network 3 **(d)** Network 3

Figure 2-5: Active Learning illustration

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.

The procedure of the active learning experiment design is shown in Figure TODO. The left Figure shows the 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 measured are certain, while points of the map that have not been visited yet are less certain. From the information on the uncertainty, the most uncertain point is sampled as shown in Figure TODO (b). 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 TODO (c). The optimal sequence of points computed with the TSP is shown in Figure TODO (d), which is sent as a path reference to the drone.

The quadrotor is navigated in the environment using a PID position controller/ the LMPCC controller without specifying any obstacles. The data is collected along the entire quadrotor path at a frequency of 20Hz. However, for GP training, the data is down-sampled to 4Hz to avoid too large data sets and training times.

2-3 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.

2-3-1 Gaussian process training

The GP is trained using stochastic variational GP Regression in GPyTorch, which is the sparse GP regression method explained in section TODO 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.

The sparse Gaussian process is initialized with 30 inducing points. This number was selected based on related literature (see. e.g. TODO) and is a balance between the speed of the prediction and accuracy of the GP.

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. Two types of kernel functions are explored: The squared exponential 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.

2-3-2 Hyperparameter tuning

The model parameters that need to be explored to find the parameters of the GP model training are the epoch and patch size and type of kernel function.

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 TODO for a total of four iterations. This is done for the SE and Matern kernel with the wind generated from a fan pointing in x-direction is shown in Figure TODO. As it was found that the influence of epoch and batch size is independent of the choice of kernel function, the results here are discussed for the SE kernel.

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 TODO.

The most accurate results with a MSE of 0.02 can be achieved for a large number of epochs or small batch sizes. However, this results in very large training times of over two minutes.

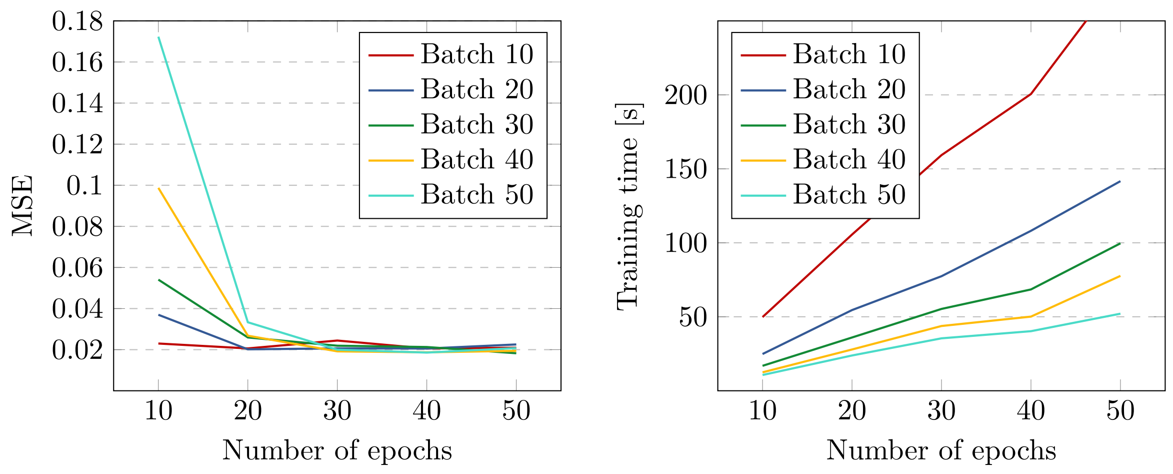


Figure 2-6: Epochs and batch size

On the other hand, if the number of epochs is too low, or the batch size is too large, training is fast but the GP model is inaccurate. While for this combination of epochs and batch size, overfitting is not yet an issue, it is desired to lower the number of epochs and batch sizes such that training is accurate while the training time remains small. 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.

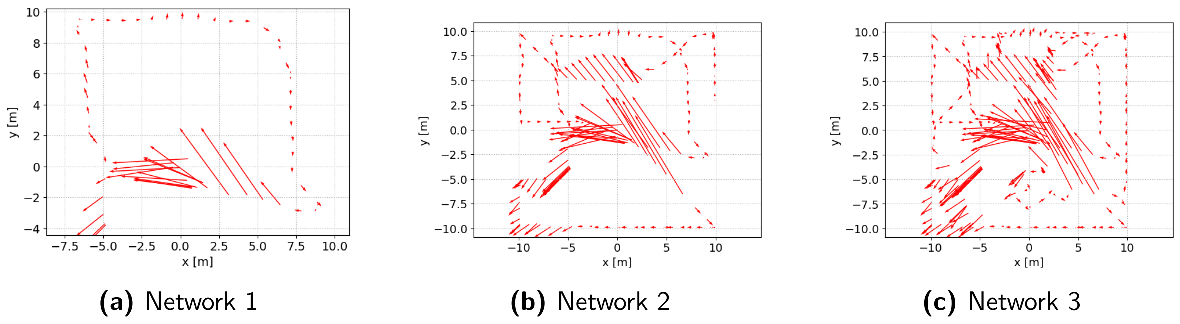
Performing this step for less experiment iterations, i.e. with less data, it was furthermore found that the optimal batch size depends on the size of the training data. If the training dataset is smaller, the batch size has to be lowered as well in order to achieve similar results for the accuracy. The 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.

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 TODO, 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 GP. In total, 5 active learning iterations are performed. The uncertainty is calculated from the confidence region, corresponding to 2 standard deviations.

Figure TODO shows the generated path and collected wind data for $1, 3, \text{and } 5$ training iterations. 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.

Comparing the MSE shown in Figure TODO in the left plot and the uncertainties shown on the right, the training results are similar. However, the GP model with SE kernel learns the windfield better with less training data, while the GP model with Matern kernel becomes more



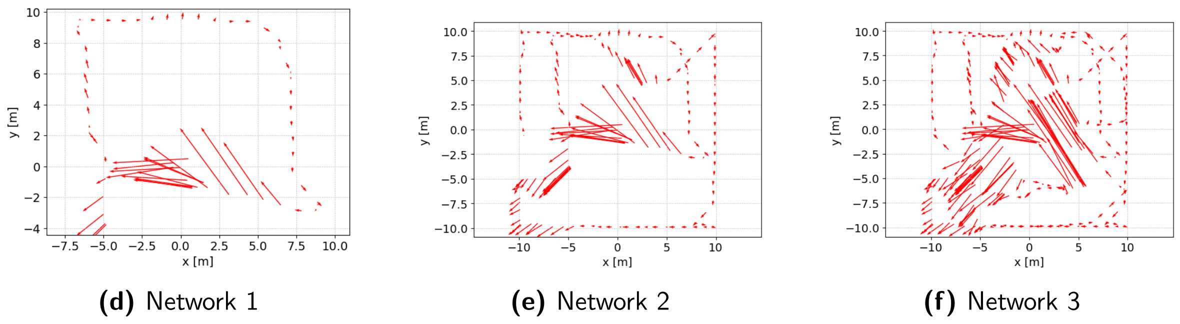


Figure 2-7: Generated wind fields

accurate with more training iterations. While the uncertainties are similar for less available data, the SE kernel gives less uncertainty for more training iterations.

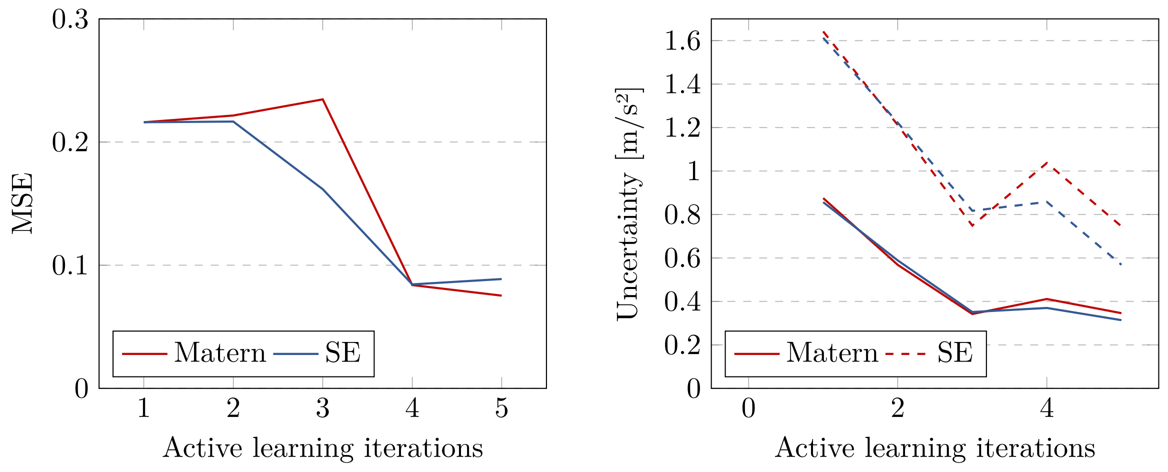


Figure 2-8: Epochs and batch size

This behavior can be explained knowing that 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 winfields in simulation are also smooth, this combination works well for the training data resulting in accurate models with less training data and less uncetainty. 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. In that case the less smooth Matern kernel could perform better.

Based on these results, the SE kernel is chosen to continue with the hyperparameter training in simulation, while keeping in mind the Matern kernel in case the SE kernel does not perform well under real-world conditions.

2-4 Results

After deciding on the disturbance model, experiment setup and the hyperparameters of the model, the GP model can be trained and evaluated in terms of its performance. This section is divided in three parts: First, we show the superiority of the data collection using active learning. Second, we compare the sparse GP approach to training a full GP. Third, we trained the disturbance maps for two types of windfields that are going to be used in the following chapters.

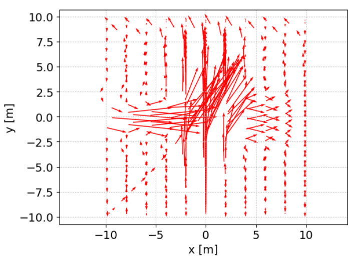
2-4-1 Data collection with active learning

To show the superiority of collecting the data using active learning, we compare this data collection method 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 TODO (a). In the second scenario, the environment is explored by flying a lemniscate trajectory shown in Figure TODO (b). The generated path using the AL approach with four iterations is shown in Figure TODO (c). All paths are tracked under the same conditions, with the wind acting in both x- and y- directions as shown in Figure TODO (d) 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 of the mean and uncertainty. The results are summarized in Table TODO.

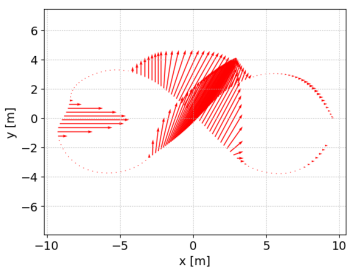
Path	Structured	Lemniscate	Active Learning
Exploration time [s]	471	TODO	274
Data samples	1864	767	1196
Training time [s]	46	17	18.03
MSE	0.034	0.282	0.051
mean uncertainty [m/s^2]	0.589	2.147	0.529
max uncertainty [m/s^2]	1.34	3.19	1.12

Table 2-1: Active learning comparison

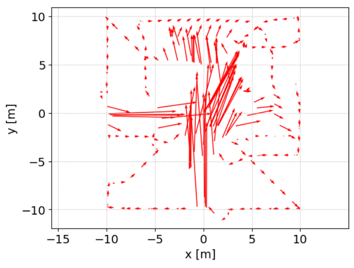
Both the structured and AL(TODO: short) path learn the distribution of the windfield well.with similar MSE and mean uncertainty. However, tracking the structured path takes almost twice the amount of time, also nearly doubling the amount of training data which increases the training time. Moreover, comparing 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 trajectory 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.



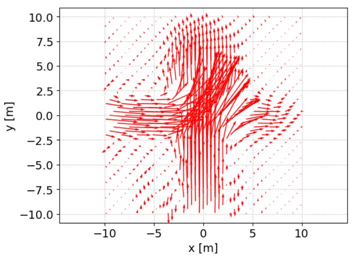
(a) Network 1



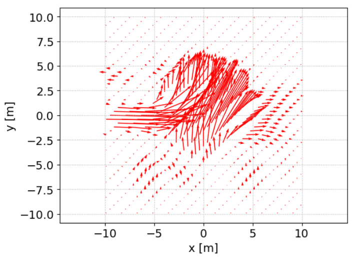
(b) Network 2



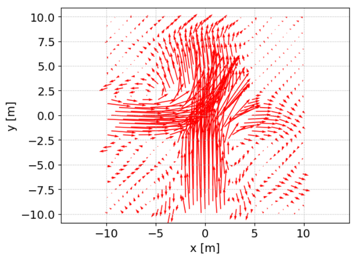
(c) Network 3



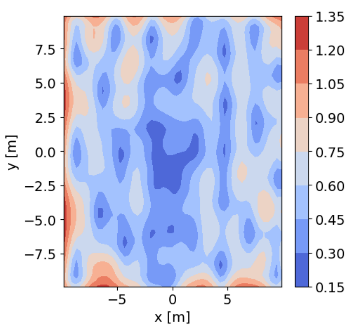
(d) Network 1



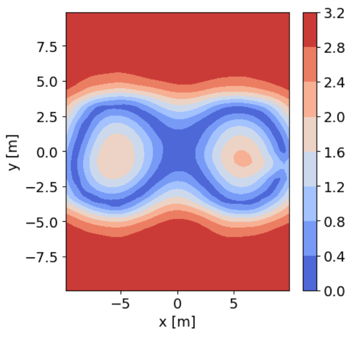
(e) Network 2



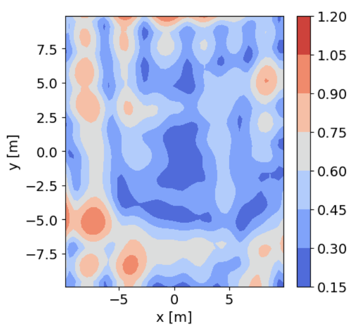
(f) Network 3



(g) Network 1



(h) Network 2



(i) Network 3

Figure 2-9: Generated wind fields

This is also reflected in Figure TODO where we show the predicted wind fields and uncertainties. The structured path and AL path can reconstruct the wind field over the entire grid, while the prediction for the lemniscate trajectory is only accurate near areas covered by the drone. This is also visible in the uncertainty plots shown in Figure TODO (g) to (i), where the uncertainty is large where the environment was not covered by the lemniscate reference path and about evenly distributed for the structured and AL path.

2-4-2 Sparse versus full GP model

To demonstrate the necessity of using sparse Gaussian processes 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 deteriorate the training results. The reference data for this comparison is generated with a simple wind

	Full GP	Sparse GP
MSE	0.025	0.029
mean_unc [m/s^2]	0.438	0.441
max_unc [m/s^2]	1.071	1.018
training time [s]	3.262	16.18
prediction time time [s]	9.224	0.063

Table 2-2: Full versus sparse GP

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.

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 TODO. From the results on the accuracy and uncertainty, it can be concluded that the performance of the GP prediction is not affected by much when using only 30 inducing points for making predictions. 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 we do not train the GP online, this is secondary.

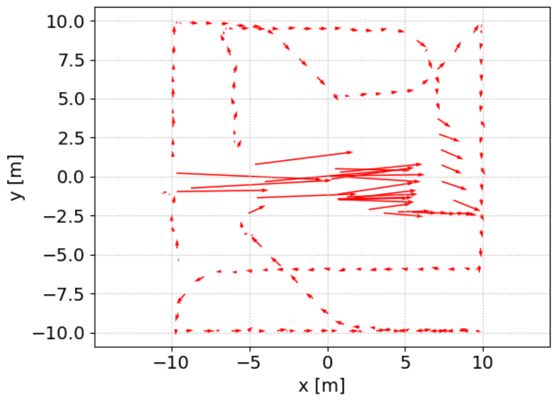
2-4-3 Trained disturbance models

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 TODO (a)) and the crossing wind fields (see Figure TODO (c)). Additionally, noise with a variance of 0.01 is added to the data to simulate more real conditions. For the first windfield, the environment is explored until and uncertainty threshold of 1 m/s^2 is reached for the 2 sigma confidence region. This is the case after 3 iterations. As the other windfiels is more complex, we set an uncertainty threshold of 1.5 m/s^2 which is reached after 4 iterations. The generated paths and collected data are shown in Figure TODO (a) and (b). The windfields are then trained using sparse GP, with 30 inducing points and a squared exponential kernel.

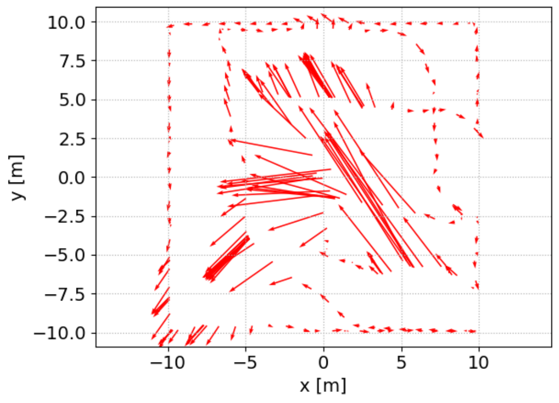
	Windfield (a)	Windfield (c)
Training iterations	3	4
MSE	0.065	0.097
mean_unc [m/s^2]	0.435	0.697
max_unc [m/s^2]	0.828	1.44
training time [s]	14.76	13.30
prediction time [s]	0.053	0.053
length scales d_x	(4.65, 2.41)	(3.32, 3.31)
length scaled d_y	(4.66, 4.64)	(2.53, 3.72)
output variance d_x	0.384	0.372
output variance d_y	0.063	0.431

Table 2-3: Final params

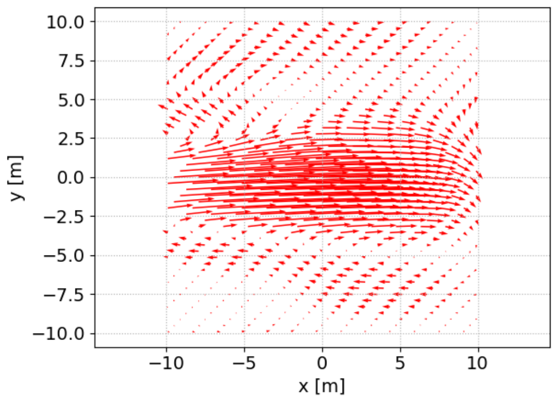
The final training results are summarized in Table TODO, together with the trained hyperparameters. The length scales for both models are in a similar range, indicating a similar smoothness of the trained disturbance map. Also, a successful training of the hyperparameters can also be concluded from the fact, that the output variance in y-direction is very small for the wind field pointing mostly in x-direction, while all other output variances are also similar. Despite the added noise, the GP model is able to train the wind field with an MSE of 0.065 and 0.097, respectively. With similar training times, the accuracy is lower for the more complex wind field, which is to be expected. Also, the uncertainties are higher. However, both trained models can capture most of the underlying wind dynamics which is also visible in Figure TODO.



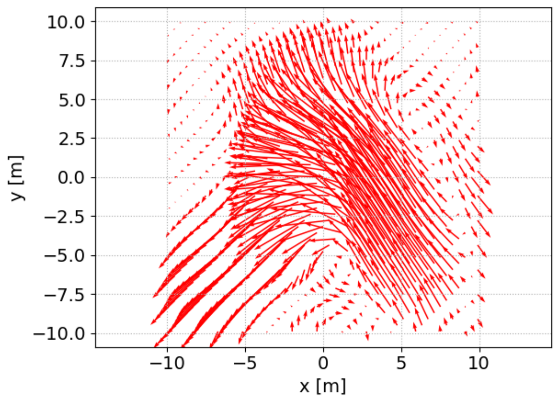
(a) Network 1



(b) Network 2



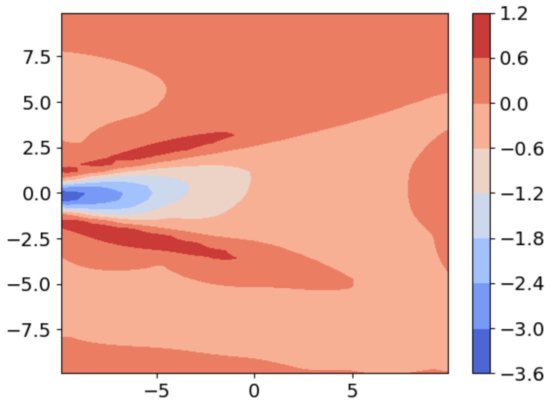
(c) Network 1

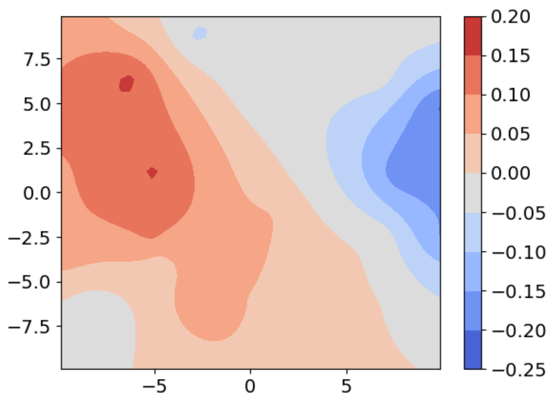


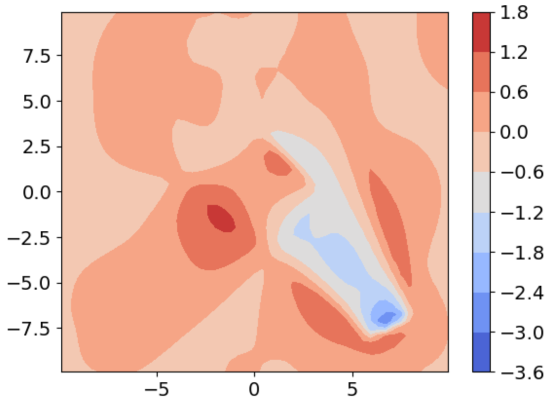
(d) Network 2

Figure 2-10: Final wind fields

Finally, we also compare the ground truth wind fields 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 TODO shows the actual mean disturbance in red and the uncertain regions withing two standard deviations learned by the GP model. For the largest part, the actual disturbance lies within the trained windfields, however for large spikes of the disturbance, the GP model fails to capture the disturbances within the uncertain bounds. Repeating the experiment for a less smooth kernel, does not change this behavior such that


(a) Network 1


(b) Network 2


(c) Network 1

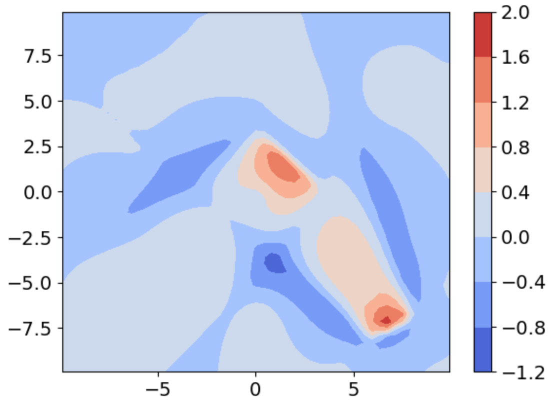
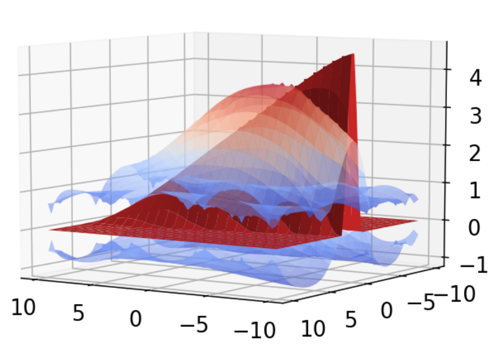

(d) Network 2

Figure 2-11: Final wind fields

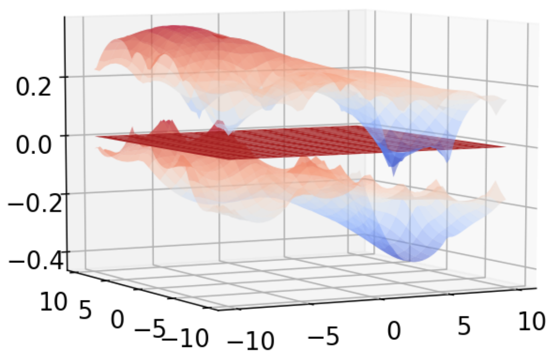
the smoothness of the SE kernel is likely not the cause. Another guess is that this might be caused by the interpolating scheme of the wind fields and the PID position controller, such that the high spikes in the data are never recorded during flight. Another cause might be the fact that the spikes occur around the edges where the wind field quickly drops to zero outside the defined region, which cannot be captured by the GP properly. Two ways to work around the problem are to avoid the marginal areas in subsequent flights or to increase the confidence bounds to make sure the true wind field lies within the uncertain bounds.

2-5 Summary and Discussion

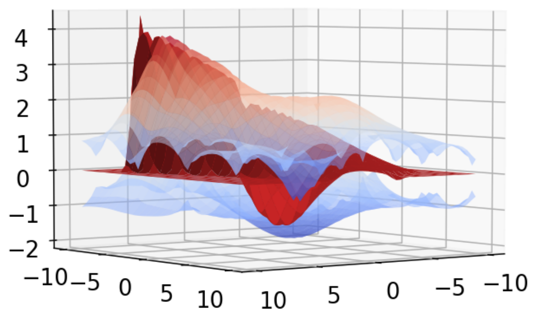
TODO



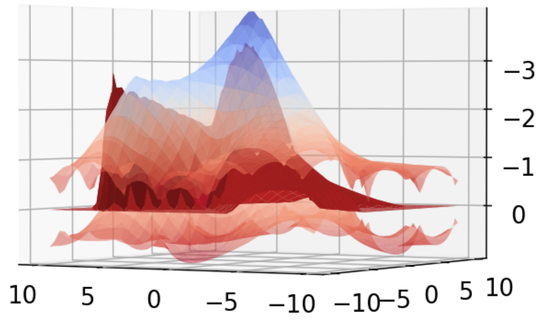
(a) Network 1



(b) Network 2



(c) Network 1



(d) Network 2

Figure 2-12: Final wind fields

Chapter 3

Data-driven local MPCC motion planner

This chapter covers the derivation of the controller formulation of the LMPCC controller that navigates the drone through the windy environment using a simple reference path. The resulting learning-based controller formulation consists of several building blocks that are familiar from the classical 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.

The second part of this chapter will go into the details of the implementation of the LMPCC controller.

3-1 Data-driven controller formulation

3-1-1 System dynamics

The resulting system dynamics are a combination of the nominal quadrotor model and the disturbance map. As the disturbance map is represented by a GP, the resulting system dynamics become probabilistic. This section covers how the model can be propagated across multiple states given the probabilistic formulation.

Data-driven quadrotor model

The system equations of the drone are given as a combination of the nominal model and Gaussian process (GP) model:

$$f(\mathbf{x}, \mathbf{u}) = f_n(\mathbf{x}, \mathbf{u}) + f_{\mathrm{GP}}(\mathbf{z}) \tag{3-1}$$

where $f_n(\mathbf{x},\mathbf{u})$ is the nominal quadrotor model derived in chapter TODO and $f_{\text{GP}}(\mathbf{z}) \sim \mathcal{N}(\boldsymbol{\mu}(\mathbf{z}), \boldsymbol{\Sigma}(\mathbf{z}))$ is the GP model capturing the wind dynamics. As the trained GP disturbance map

$$\mathbf{d}(\mathbf{p}) = \begin{bmatrix} d_{v_x}(\mathbf{p}) \sim \mathcal{N}\left(\mu^{d_{v_x}}(\mathbf{p}), \Sigma^{d_{v_x}}(\mathbf{p})\right) \\ d_{v_y}(\mathbf{p}) \sim \mathcal{N}\left(\mu^{d_{v_y}}(\mathbf{p}), \Sigma^{d_{v_y}}(\mathbf{p})\right) \end{bmatrix}. \tag{3-2}$$

is only defined for v_x and v_y, we need to map the disturbance vector $\mathbf{d}(\mathbf{p})$ to the correct states. With the mapping:

$$B_d = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3-3}$$

the resulting system equations are:

$$f(\mathbf{x},\mathbf{u}) = f_n(\mathbf{x},\mathbf{u}) + B_d\mathbf{d}(\mathbf{z}). \tag{3-4}$$

TODO: Get mathematical formulation straight and define dimensions of the model.

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.

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

with approximate inference using linearization. With the next state estimate given by:

$$x_{k+1} = f(x_k, u_k) + B_d\mathbf{d}(z_k) \tag{3-6}$$

the mean of the next state estimate can be computed by passing the mean values through the system model. This gives:

$$\mu^x_{k+1} = f(\mu^x_k, u_k) + B_d\mu^d_k(\mu^z_k) \tag{3-7}$$

for the update of the state mean.

For the propagation of the uncertainty, the first-order taylor approximation of the next state estimate is used, which is:

$$\begin{aligned} x_{k+1} &\approx x^0_k + \left(\frac{df}{dx} f(x_k, u_k)\Big|_{x=\mu_x} + \frac{df}{dx} B_d\mathbf{d}(z_k)\Big|_{x=\mu_x} \right) x_k \\ x_{k+1} &\approx x^0_k + \left(\nabla_x f\left(\mu^x_k, u_k\right) + B_d \nabla_x \mu^d_k\left(\mu^z_k\right)\right) x_k \end{aligned} \tag{3-8}$$

where $\nabla_x \mu^d_k\left(\mu^z_k\right)$ is the derivative of the GP with respect to it's state vector evaluated at the mean input state vector. Given the function linearization, one can propagate the uncertainty as for the linear case:

$$\Sigma_{k+1}^{x}=[\nabla_x f\left(\mu_k^x, u_k\right)+B_d \nabla_x \mu_k^d\left(\mu_k^z\right)]\Sigma_k[\nabla_x f\left(\mu_k^x, u_k\right)+B_d \nabla_x \mu_k^d\left(\mu_k^z\right)]^T+B_d \Sigma_k^d B_d^T \tag{3-9}$$

with $B_d \Sigma_k^d B_d^T$ being the propagation of the GP uncertainty Σ_k^d at the k-th state.

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

$$\Sigma_{k+1}^{x}=\left[\nabla_x f\left(\mu_k^x, u_k\right) B_d\right]\left[\begin{array}{cc}\Sigma_k^x & \Sigma_k^{xd} \\ \Sigma_k^{dx} & \Sigma_k^d\end{array}\right]\left[\nabla_x f\left(\mu_k^x, u_k\right) B_d\right]^T \tag{3-10}$$

with $\Sigma_k^{dx}=\nabla_x \mu_k^d\left(\mu_k^z\right) \Sigma_k$ and $\Sigma_k^{xd}=\Sigma_k^{dx T}$. Replacing the nonlinear function dynamics with it's linear model in the uncertainty propagation results in the mean and variance propagation given by:

$$\begin{aligned} \mu_{k+1}^x & =f(\mu_k^x, u_k)+B_d \mu_k^d(\mu_k^z) \\ \Sigma_{k+1}^x & =[A\, B_d]\left[\begin{array}{cc}\Sigma_k^x & \Sigma_k^{xd} \\ \Sigma_k^{dx} & \Sigma_k^d\end{array}\right][A\, B_d]^T . \end{aligned} \tag{3-11}$$

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

3-1-2 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 [1] 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 θ. It is assumed that the desired path is parameterised by $\mathbf{p}(\theta)$. For a given longitudinal vehicle speed $\mathbf{v}_k$ at time step k, the approximate progress along the reference path can be described by:

$$\theta_{k+1}=\theta_k+\mathbf{v}_k \Delta t . \tag{3-12}$$

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.,

$$J_{\text{speed }}\left(\boldsymbol{x}_k\right)=Q_v\left(v_{\text{ref }}-v_k\right)^2 \tag{3-13}$$

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:

$$\mathbf{e}_k=\left[\begin{array}{c}\hat{e}^l\left(\mathbf{x}_k, \theta_k\right) \\ \hat{e}^c\left(\mathbf{x}_k, \theta_k\right)\end{array}\right] . \tag{3-14}$$

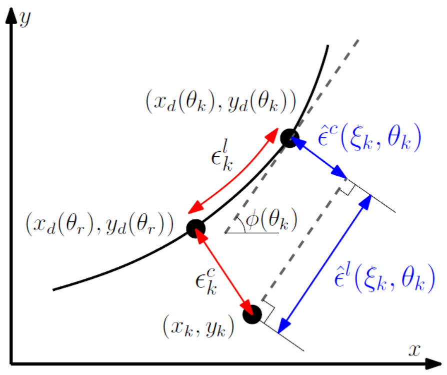

y
x
$(x_d(\theta_k), y_d(\theta_k))$
ϵ^l_k
$\hat{\epsilon}^c(\xi_k, \theta_k)$
$(x_d(\theta_r), y_d(\theta_r))$
$\phi(\theta_k)$
ϵ^c_k
$\hat{\epsilon}^l(\xi_k, \theta_k)$
(x_k, y_k)

Figure 3-1: 2D [5].

Figure 3-2: Contour and lag error and it's approximations in 2D.

The contour and lag error define the approximate distance to the reference path, given the progress along the path θ_k as shown in Figure TODO. The tracking cost is designed to minimize the distance to the path:

$$J_{\text{tracking}}(\boldsymbol{x}_k, \theta_k) = \boldsymbol{e}_k^T Q_\epsilon \boldsymbol{e}_k, \tag{3-15}$$

where Q_ϵ is a design weight.

Penalizing the inputs Additionally, the inputs are penalized with

$$J_{\text{input}}\ (\boldsymbol{u}_k) = \boldsymbol{u}_k^T Q_u \boldsymbol{u}_k, \tag{3-16}$$

where Q_u is a design weight.

Total cost The total stage cost of the LMPCC is

$$J\left(\mathbf{z}_k, \mathbf{u}_k, \theta_k\right) := J_{\text{tracking}}\ (\boldsymbol{x}_k, \theta_k) + J_{\text{speed}}\ (\boldsymbol{x}_k) + J_{\text{input}}\ (\boldsymbol{u}_k)\,. \tag{3-17}$$

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

3-1-3 Obstcale avoidance constraints

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

$$\mathcal{B}\left(\mathbf{x}_k\right) \cap \mathcal{O} = \emptyset \tag{3-18}$$

where $\mathcal{B}\left(\mathbf{x}_k\right)$ represents the occupancy volume of the quadrotor and $\mathcal{O}$ is the obstacle space. 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. We therefore extract the uncertainty matrix:

$$\Sigma_{k,\text{pos}} = \begin{bmatrix} \sigma_x & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix} \tag{3-19}$$

from the propagated uncertainty at each state k.

Two types of constraints are considered: Ellipsoidal constraints and Gaussian constraints. Both types of constraints assume static, circular obstacles.

Ellipsoidal constraints

TODO: cite Laura paper

For the ellipsoidal constraints, an ellipsoidal uncertainty bound is constructed around the quadrotor, given the uncertainty matrix and a confidence interval χ, which represents the collision probability. The obstacle avoidance constraint at each state is:

$$c_k^{\text{obst},j}(\boldsymbol{z}_k) = \begin{bmatrix} \Delta x_k^j \\ \Delta y_k^j \end{bmatrix}^{\mathrm{T}} R(\psi)^{\mathrm{T}} \begin{bmatrix} \frac{1}{\alpha^2} & 0 \\ 0 & \frac{1}{\beta^2} \end{bmatrix} R(\psi) \begin{bmatrix} \Delta x_k^j \\ \Delta y_k^j \end{bmatrix} > 1 \tag{3-20}$$

,

where Δx_k^jandΔy_k^j is the distance between the quadrotor and the obstacle. $R(\psi)$ is the rotation matrix, describing the rotation of the uncertain ellipsoid and $\alpha = a + r_{\text{obst}} + r_{\text{quad}}$, $\beta = a + r_{\text{obst}} + r_{\text{quad}}$, where a and b the major and minor axes of the ellipsoid. The rotation and major and minor axes of the ellipsoid can be found using a singular value decomposition of the uncertainty of the quadrotor position $\Sigma_{k,\text{pos}}$. The axes are enlarged according to the collision probability χ.

TODO: describe ellipsoid

Chance constraints

TODO: cite Hai paper

The chance constraints are defined as a collision probability:

$$\Pr\left(\mathbf{x}^k \notin \mathcal{C}_o^k\right) \geq 1 - \delta_o, \forall o \in \mathcal{I}_o. \tag{3-21}$$

The collision region is given as an enlarged half space of the actual collision region:

$$\tilde{\mathcal{C}}_o := \left\{\mathbf{x} \mid \mathbf{a}_o^T \left(\mathbf{p} - \mathbf{p}_o\right) \leq b\right\}, \tag{3-22}$$

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

$$\mathbf{p} \sim \mathcal{N}\left(\hat{\mathbf{p}}, \Sigma_p\right), \tag{3-23}$$

the deterministic obstacle position $\mathbf{p}_o$ and nominal distance between the obstacle and quadrotor $b = r_{\text{quad}} + r_{\text{obst}}$.The chance constraints can be reformulated into deterministic ones using the following relation (TODO cite):

$$\Pr\left(\mathbf{a}^T\mathbf{x} \leq b\right) = \frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{b - \mathbf{a}^T\hat{\mathbf{x}}}{\sqrt{2\mathbf{a}^T \sum \mathbf{a}}}\right). \tag{3-24}$$

where erf denotes the error function:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt. \tag{3-25}$$

This deterministic chance constraints take the form:

$$\mathbf{a}_o^T \left(\hat{\mathbf{p}} - \mathbf{p}_o\right) - b \geq \operatorname{erf}^{-1}\left(1 - 2\delta_o\right) \sqrt{2\mathbf{a}_o^T \left(\Sigma_{\mathrm{p}}\right) \mathbf{a}_o}. \tag{3-26}$$

3-1-4 Resulting controller formulation

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

$$
\begin{aligned}
\mathcal{J}^*_{\text{MP}} = \min_{\boldsymbol{x}(\cdot|t),\boldsymbol{u}(\cdot|t),s(\cdot|t)} & \mathcal{J}_{\text{MP}}(\boldsymbol{x}(\cdot \mid t), \boldsymbol{u}(\cdot \mid t), s(\cdot \mid t)) && \text{(3-27a)}\\
\text{s.t. } & \boldsymbol{x}(0 \mid t) = \hat{\boldsymbol{x}}(t), && \text{(3-27b)}\\
& s(0 \mid t) = \hat{s}(t), && \text{(3-27c)}\\
& \boldsymbol{x}(k+1 \mid t) = f(\boldsymbol{x}(k \mid t), \boldsymbol{u}(k \mid t)), && \text{(3-27d)}\\
& s(k+1 \mid t) = s(k \mid t) + v(k \mid t)T_s, && \text{(3-27e)}\\
& \mathcal{B}(\boldsymbol{x}(k \mid t)) \cap \mathcal{O}_{\text{static}} = \emptyset, && \text{(3-27f)}\\
& \boldsymbol{x}(k \mid t) \in \mathcal{X}, && \text{(3-27g)}\\
& \boldsymbol{u}(k \mid t) \in \mathcal{U}, && \text{(3-27h)}\\
& 0 \leq s(k) \leq L && \text{(3-27i)}\\
& k = 0, \ldots, N_p - 1 && \text{(3-27j)}
\end{aligned}
$$

TODO: get this formulation straight.

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.

3-2 Data-driven controller implementation

With the resulting data-driven local MPCC 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.

3-2-1 Controller implementation

The data-driven LMPCC problem is implemented with a prediction horizon of $N = 20$ at a sampling rate of $T_\text{s} = 50\text{ms}$, resulting in a 1s look-ahead. The underlying optimization problem is solved with the interior-point-based NLP solver FORCES PRO (TODO: cite, see robot arm). The maximum number of iterations is set to 60 to ensure consistent maximum solve times.

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 u, μ_u, and Σ_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 and to avoid numerical issues that appear when having the state uncertainty as an optimization variable inside the constraints. The continuous, nonlinear dynamics are propagated into the future using the RK4 method, such that at each optimization step the GP is evaluated four times.

The input and state constraints are given by:

$$\mathcal{U} = \left\{ \boldsymbol{u} \in \mathbb{R}^m \;\middle|\; \begin{array}{l} (\phi_d, \theta_d) \in [-40, 40]\text{deg} \\ \dot{\psi}_d \in [-10, 10]\text{deg/s} \\ T_d \in [5, 15]\text{m/s}^2 \end{array} \right\}, \tag{3-28}$$

$$\mathcal{X} = \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{l} (x, y) \in [-\infty, \infty]\text{m} \\ z \in [-1, 1]\text{m} \\ (v_x, v_y, v_z) \in [-10, 10]\text{m/s} \\ (\phi, \theta) \in [-50, 50]\text{deg} \\ \dot{\psi} \in [-15, 15]\text{deg/s} \end{array} \right\}. \tag{3-29}$$

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

$$\mathcal{U} = \{s \in \mathbb{R} \mid s \in [0, 1e4]\text{m}\} \tag{3-30}$$

Furthermore, the following parameter values are used:

Name	**Value**
Cost weight	5.0
Cost weight	1.0
Cost weight	1.0
Cost weight	0.02
Cost weight	1.0
Cost weight	10.0
v_ref	2.0

Table 3-1: LMPCC parameters.

3-2-2 Simulation interface

The output of the optimal control problem is interfaced with the simple simulation environment described in section TODO, simulated in RViz. The control commands generated by the solver are translated into 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 $r_{\text{obst}} = 0.3\text{m}$. This scenario is shown in Figure TODO.

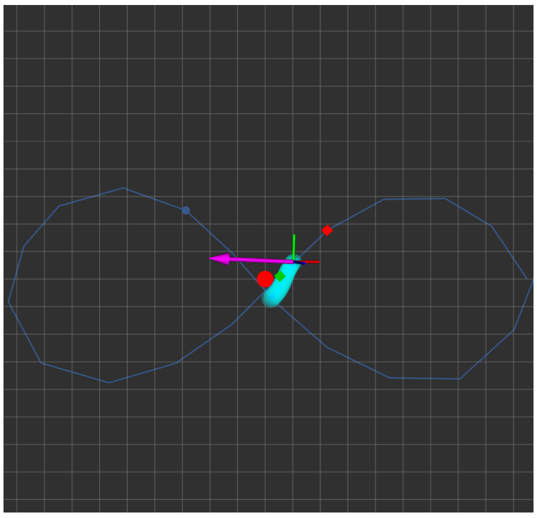


Figure 3-3: LMPCC rviz

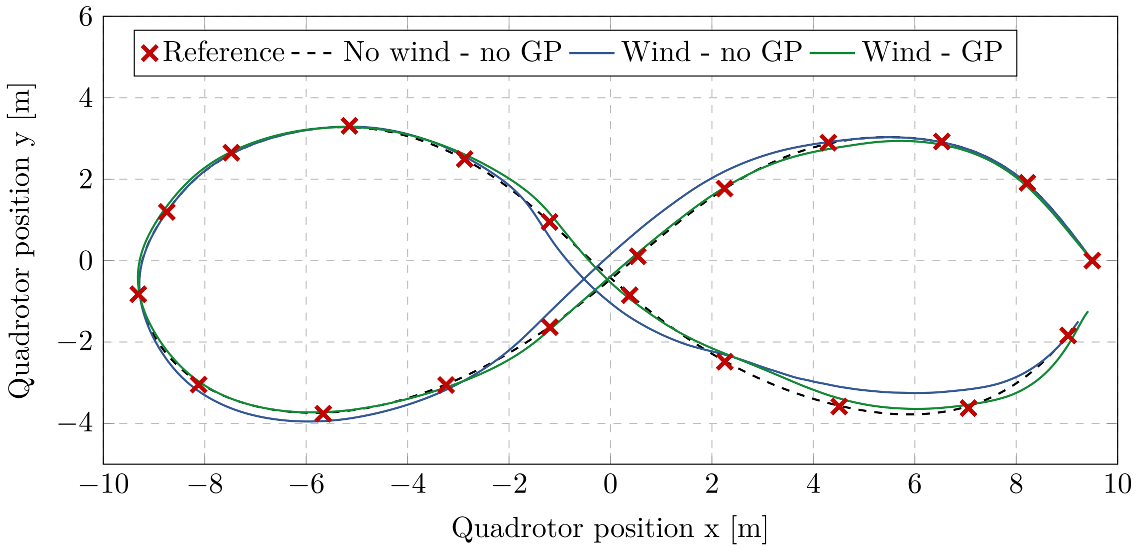


Figure 3-4: test

	fit x	fit xy
No wind - no GP	0.030	
Wind - no GP	0.127	0.160
Wind - GP	0.075	0.076

Table 3-2: Goodness of fit

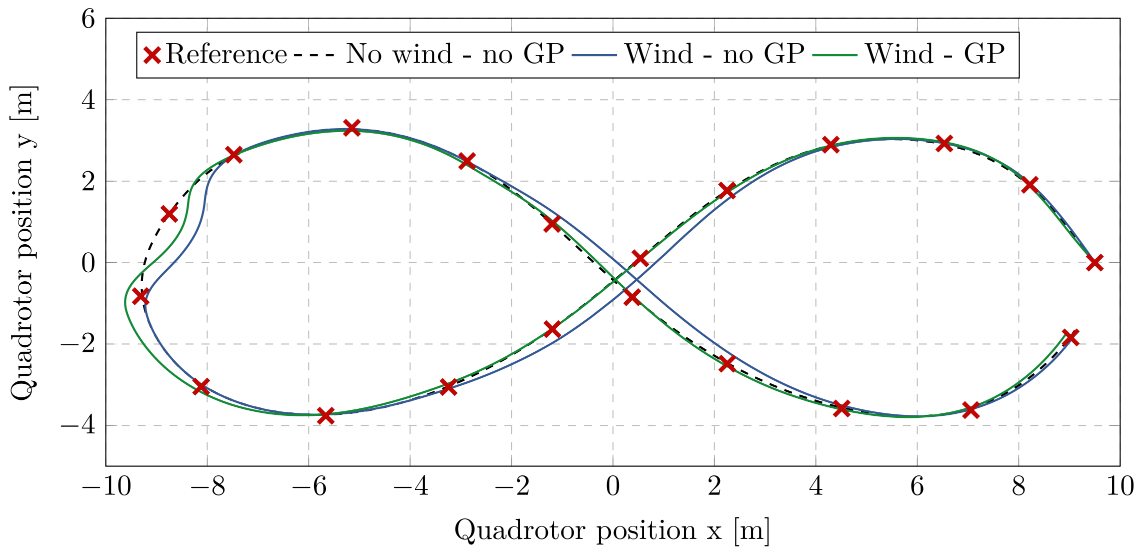


Figure 3-5: test

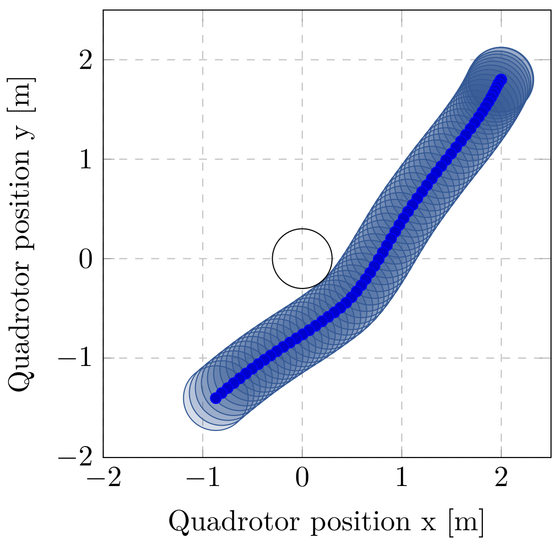


Figure 3-6: test

Bibliography

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

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

Glossary

List of Acronyms