POLITECNICO DI TORINO

Master Degree in Computer Engineering
(Automation and Intelligent Cyber-Physical Systems)

Master of Science Thesis

Wind Estimation Using Gaussian Process Regression for Safe Drone Control

Supervisors
Prof. Carlo Masone
Prof. Laura Ferranti
Dr. Dennis Benders

Candidate
Simone Carena

2023-2024

Abstract

Drones are becoming increasingly important in our society. They are used for different purposes ranging from search and rescue to package delivery and entertainment. Reliability is an important aspect to consider for these real-world applications. One of the most significant challenges to drone stability and control is wind disturbances, which can affect flight accuracy, safety, and efficiency. Conventional control systems are often incapable of adapting to such external forces, leading to the risk of collisions and even system failures. This research aims to enhance the stability and robustness of the control system by integrating the traditional Model Predictive Control (MPC) algorithm with Gaussian Processes for real-time online wind predictions.

In this thesis, I present a modified version of the conventional Gaussian Process Regression (GPR), which uses past wind information in a sliding window fashion to estimate future disturbances. This probabilistic approach enables the construction of a data-driven model for the wind dynamics, capable of providing, apart from the simple predictions, also a confidence interval around such estimates. The non-parametric nature of the GP framework allows to easily include new pieces of information and remove less significant ones during the online phase. This also enables the model to capture complex nonlinear dynamics without the need to explicitly consider an underlying model for the wind.

The predictions are included inside the MPC of the drone, which is then used to adjust the drone trajectory and control action based on the estimates given by the model. By incorporating the model's predictions, the drone dynamics are improved to account for external forces, thereby enhancing control effectiveness. Furthermore, the use of the uncertainty resulting from these estimates becomes a key component when considering the obstacle avoidance constraints: not only the system is able to properly avoid the obstacles, but the resulting uncertainty renders the controller more conservative when the model prediction is more uncertain, leveraging the use of chance-constrained optimization.

This study validates using Gaussian Processes to model and predict wind disturbances for quadrotor navigation. This new control formulation is compared with the traditional MPC algorithm and an improved version that considers a constant baseline for wind disturbances. The algorithm's effectiveness is evaluated in terms of real-time feasibility and tested against wind fields consisting of real-world collected data. The proposed algorithm is shown to perform well in windy and cluttered environments, being able to handle complex wind fields while avoiding excessive conservativeness.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Drones are becoming more and more widespread in our society. With applications ranging from search and rescue to more mundane task, like delivery and transportation, the need to develop reliable control algorithms to operate these vehicles becomes imperative.

1.1 Motivation and Scope

As the use of drones for various types of task increases, the need to develop robust and safe control algorithms becomes a necessity. The ability to performs different types of task in disparate environments poses several challenges to traditional control methods. One of the main disturbances that can hinder the success of a flight, are winds destabilizing the drone. The effect that these disturbances have can be detrimental to the undertaken mission, ranging from a simple increase in the control effort, in order to balance the displacement caused by these external forces, to more catastrophic scenarios, where the drone collides with obstacles leading to severe damage to the vehicle.

The focus of this work is then that of finding a control method capable of safely driving a quadrotor along a desired path, even in the presence of obstacle in its way. The proposed approach aims at complementing a more traditional control method with a machine learning model, in order to estimate future wind disturbances affecting the drone. Rather than only improving the control performance, the proposed framework aims at increasing the safety of the flight by accounting for the uncertainty of the machine learning model in its predictions. By considering a model capable of providing not only a simple prediction, but an uncertainty around its estimate, the controller becomes more conservative in areas where the predictions are more uncertain, avoiding excessive zest in the maneuvers. The proposed implementation is designed to adapt to space-varying wind fields, updating its predictions when new data is available, making it able to adapt to changes in the environment.

1.2 State of the Art

Guaranteeing a safe and reliable flight for a drone in the presence of unforeseen external disturbances is a key challenge in today's control community. Due to the increased use of this technologies for an ever-growing number of activities, the need to include safety and reliability guarantees in the presence of external wind forces has become of primary importance.

Some of the existing methods, used to estimate and reject external wind disturbances, rely on the use of filtering techniques like Kalman Filters or Particle Filters (Xing et al. [2023], Hentzen et al. [2019]). The use of these kind of estimators is proven to be useful and effective, but does not provide safety guarantees on the generated actions. Some other traditional control methods, that allow for a more robust approach to the control problem, rely on the use of Sliding Mode Controllers, and modified versions of such framework, to achieve robust control under the presence of external wind disturbances (Perozzi et al. [2018], Fethalla et al. [2018], Mofid et al. [2022]). Alongside this more traditional methods, the field of robust control for quadrotors has also developed toward the use of Robust Model Predictive Control techniques, which combines the advantages of the MPC with means to improve the reliability of the system (Alexis et al. [2016]).

Alongside these traditional methods, progresses in the field of Machine Learning, and its increasingly widespread use, allowed the development of data-driven control methods, that complement classical frameworks with the use of ML model. This addition allows for the inclusion of a component inside the controller that directly stems from observed data, that does not require the use of an underlying model for the disturbances, and able, at the same time, to model the residual dynamics based on the observed system behavior. Proposed approaches rely on the use o LSTMs and ANNs (Zimmerman et al. [2022], Crowe et al. [2020]), which prove to be quite effective, but at the same time do not provide safety guarantees about their predictions. In particular, these type of models suffer from a lack of reliability when they are presented with out-of-distribution data: they might become unreliable, yielding wrong or inaccurate predictions and hindering the safety of the flight.

Toward the concept of achieving safe control using ML approaches, the control community started using Gaussian Processes as a way of achieving both effectiveness in the control action, and reliability in the predictions of the model. While the use of GP has been studied to model the residual dynamics or the complete dynamical model of a drone (Mehndiratta and Kayacan [2020], Torrente et al. [2021], Cao et al. [2017]), few discussions have taken place in regards of using the GP framework to estimate external wind disturbances. Research in the direction of estimating wind disturbances using GP models has been performed by Yang et al. [2017] in a task directed to guaranteeing safe delivery of a payload. The research, though, does include the wind estimates inside an MPC to improve its performance, and focuses on other factors to secure the payload delivery even in windy environment. These researches show how it is possible to estimate exogenous disturbances using GPR and how this ML model can be included inside the MPC to improve control performance.

1.3 Research Objectives and Contribution

The objective of this study is to improve the safety of quadrotor flight, in the presence of an external wind field interfering with the drone flight. This improvement is to be intended both in terms of disturbance rejection, and thus is terms of improved flight performance, but also regarding the safety of the flight itself, in particular when moving close to obstacles.

To achieve this higher performance and reliability, this works complement the classic MPC control framework with the use of GPR to estimate external wind disturbances. The use of this ML model has been deemed well-suited for this approach as it can provide not only an estimate of the disturbance, but also an epistemic uncertainty around it. This uncertainty, expressing the confidence of the model about its prediction, is then propagated within the MPC control horizon and used to perform safe control via chance-constrained optimization. The GP model presented in this work is a modification of the classical framework, adapted to fit the need of this research in better handling unforeseen disturbances and rapidly adapting to them.

The research objectives presented in this work can then be expressed as the following topics:

- How well does the herein implemented GP model predict future wind disturbances, based on past information, in terms of improved flight performance?
- How does the inclusion of the model via chance-constrained optimization help in improving the robustness and reliability of the control?

This Thesis presents the following main contributions:

1. Implementing a GP model capable of estimating future wind disturbances based on a handful of past collected data. This is achieved by modifying the classic GPR framework: the model includes new more relevant data to make predictions, while discarding less significant past information.
2. Complementing the MPC with the predictions made from the GP model, in order to improve the flight performance in presence of unforeseen disturbances.
3. Making use of the predictions' uncertainty to perform chance-constrained optimization, making the control more conservative in areas where the wind estimates are less reliable.

1.4 Thesis Organization

This Thesis work is structured as follows:

Chapter 2. In this chapter it is presented the use of Gaussian Processes for wind estimation. The chapter stars with a mathematical description of this ML framework and how the model works. After that introduction, the applications of GPR for estimating wind

disturbances, and their use in this work is presented, along with the herein developed implementation of the framework.

Chapter 3. This part of the Thesis starts with a description of the drone model and the formulation of the Model Predictive Control algorithm. The subject is then expanded by describing how the standard MPC formulation is complemented by the use of GP. The chapter ends with implementation details about the presented model and considerations on the frameworks used to construct the GP-MPC.

Chapter 4. Here the experiments performed to validate the model are presented. An introduction of the setup is followed by several tests and considerations on the adopted frameworks, controller implementation and GP model use.

Chapter 5. A final chapter containing conclusions and suggestions for future work wraps-up the Thesis and concludes the presentation.

Chapter 2

Wind Disturbances Estimation

2.1 Gaussian Process Regression

The goal of Gaussian Process Regression (Rasmussen and Williams [2006]) is to obtain a probabilistic model approximating a nonlinear function describing the wind disturbances affecting the drone, given a set of noisy observations. The following section provides a mathematical description of this machine learning model and its use to estimate the wind disturbances.

2.1.1 Mathematical Background

A Gaussian Process (GP) is a collection of random variables, any finite number of which is jointly Gaussian, as shown in 2.1. Similarly to a Gaussian distribution, a GP is completely specified by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$. For a real process $f(\mathbf{x})$, these are defined as

$$m(\mathbf{x}) = \mathbb{E}\left[f(\mathbf{x})\right] \tag{2.1}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}\left[\left(f(\mathbf{x}) - m(\mathbf{x})\right)\left(f(\mathbf{x}') - m(\mathbf{x}')\right)\right] \tag{2.2}$$

The Gaussian Process can be then denoted as

$$f(\mathbf{x}) \sim GP\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right) \tag{2.3}$$

To reconstruct the underlying nonlinear function and make predictions over unseen data, the GP framework uses a Bayesian approach, incorporating prior knowledge with observed noisy data. Such GP prior is specified by the mean and covariance functions.

In order to make predictions with the model, we need to define a likelihood function that captures the probability of observing the data given the function values at specific points. Assuming the likelihood to be Gaussian, the dataset $\mathcal{D}$ of n observations, $\mathcal{D} = \{(\mathbf{x}_i, y_i) \,|\, i = 1, \ldots, n\}$, used to train the model is generated according to

$$y_i = f(\mathbf{x}_i) + \epsilon \tag{2.4}$$

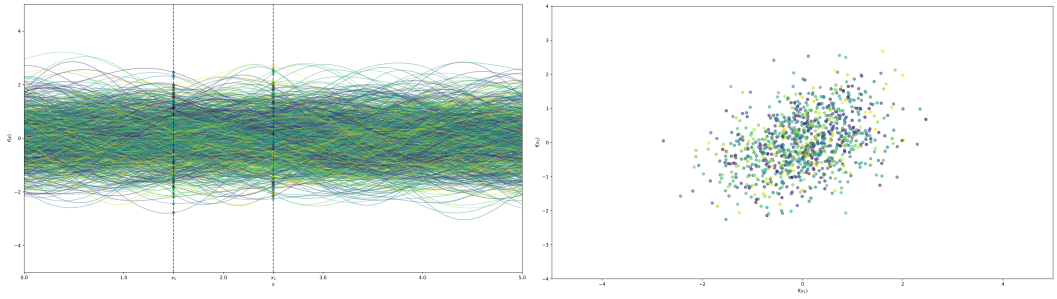

Figure 2.1: On the left, samples from a GP prior distribution, using an RBF kernel. On the right, how the values of the functions are correlated when sampling from 2 input points

where $f : \mathbb{R}^n \to \mathbb{R}$ is the nonlinear function we want to learn, $\epsilon \sim \mathcal{N}\left(0, \sigma_n^2\right)$ is the likelihood noise affecting the observations, and $\mathbf{x}_i \in \mathbb{R}^n$ are the collected input points. The likelihood can then be expressed as

$$p(\mathbf{y}|\mathbf{f}, X) \sim \mathcal{N}\left(f(X), \sigma_n^2 \mathbf{I}\right) \tag{2.5}$$

where X is the collection of input points, $X = \{\mathbf{x}_i \,|\, i = 1, \ldots, n\}$ and $f(X)$ are the values the function assumes at such input points.

Combining the prior and the likelihood, we can obtain the posterior predictive distribution over unseen data-points using the Bayesian framework. In particular, we can express the joint prior distribution of the noisy observations $\mathbf{y}$ and the function value $\mathbf{f}_*$ we want to predict as

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) + \sigma_n^2 \mathbf{I} & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right) \tag{2.6}$$

To get the posterior distribution over functions we need to restrict this joint prior to contain only those functions which agree with the observed data-points (2.2). This operation corresponds to conditioning the joint Gaussian prior distribution on the observations, resulting in

$$p\left(\mathbf{f}_*|\mathbf{y}, X, X_*\right) \sim \mathcal{N}\left(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*\right) \tag{2.7}$$

where

$$\boldsymbol{\mu}_* = K(X_*, X)\left[K(X,X) + \sigma_n^2 \mathbf{I}\right]^{-1} \mathbf{y} \tag{2.8}$$

$$\boldsymbol{\Sigma}_* = K(X_*, X_*) - K(X_*, X)\left[K(X,X) + \sigma_n^2 \mathbf{I}\right]^{-1} K(X, X_*) \tag{2.9}$$

The quantity $K(\cdot, \cdot)$ is the kernel function evaluated at specific input points, in particular the training points X and the prediction points X_*. This discrete formulation comes from the observation used in the model, being them in finite number. This matrix K is then defined as $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and being it a covariance matrix has the constraints of being

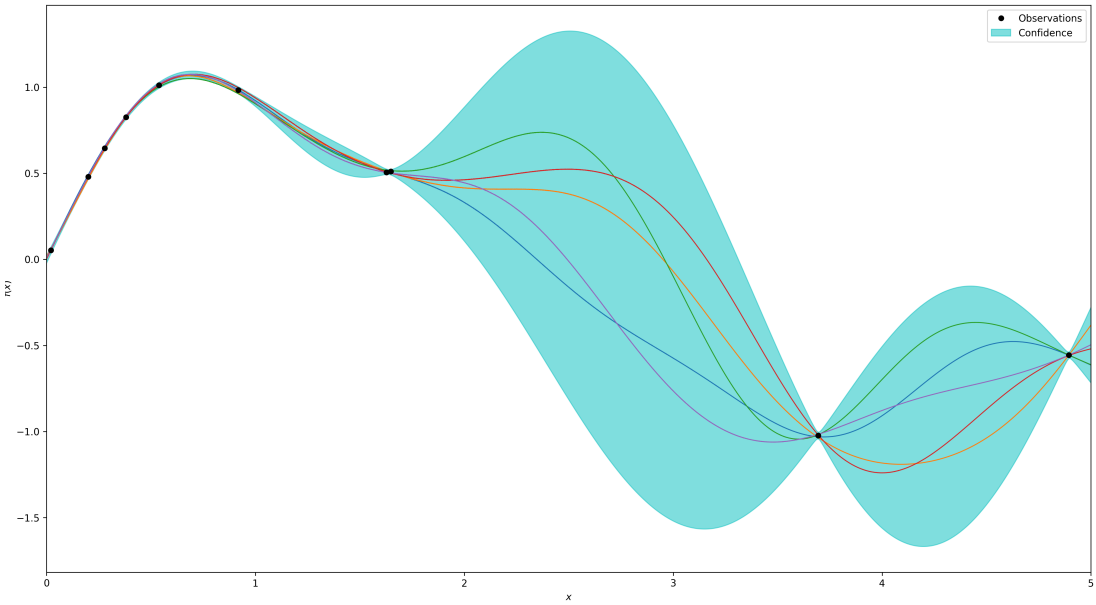

Figure 2.2: Samples drawn from the posterior distribution of a GP, conditioned on 10 observations

symmetric, i.e. $K_{ij} = K_{ji}$, and positive semi-definite (PSD): $K \succeq 0 \Leftrightarrow \mathbf{x}^{\mathrm{T}} K \mathbf{x} \geq 0 \; \forall \mathbf{x} \neq \mathbf{0} \in \mathbb{R}^n$.

In the more general case, i.e. if a non-zero prior mean is considered, the predictive distribution assumes a slightly different formulation

$$\boldsymbol{\mu_*} = m(X_*) - K(X_*, X)\left[K(X, X) + \sigma_n^2 \mathbf{I}\right]^{-1}(\mathbf{y} - m(X)) \tag{2.10}$$

where $m(X)$ is the prior mean evaluated at the training inputs X. Given that no prior assumptions are made, in this work a zero-mean is considered for the prior (2.8), and thus the computations will be done considering that case.

2.1.2 Hyper-parameters and Model Selection

The prior knowledge is encoded in the model by the specification of the mean and covariance function. The prior mean function is usually assumed to be zero if no prior information about the function is available. The kernel function determines the similarity between pairs of points in the domain of the random function, encoding how function values are correlated with one another. Each kernel has a set of *hyper-parameters*, that specify the properties of the resulting GP.

Radial Basis Function (RBF). This kernel, also known as the **Squared Exponential**, is the most commonly used and has the form

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||_2^2}{2\ell^2}\right) \tag{2.11}$$

The *characteristic length-scale* parameter ℓ specifies how correlated two function values are, based on how close the input points $\mathbf{x}$ and $\mathbf{x}'$ are.
Matérn Class of Covariance Functions. This family of kernels is defined as

$$k(\mathbf{x}, \mathbf{x}')_\nu = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{||\mathbf{x} - \mathbf{x}'||_2 \sqrt{2\nu}}{l} \right)^\nu K_\nu \left(\frac{||\mathbf{x} - \mathbf{x}'||_2 \sqrt{2\nu}}{l} \right) \tag{2.12}$$

where $K_\nu(\cdot)$ is a modified Bessel function, and $\Gamma(\cdot)$ is the Gamma function. The ℓ parameter has the same function as in the SE kernel, but, in addition, this kernel family is also parameterized by the ν, which is used to control the smoothness of the resulting GP. The Matérn covariance functions become especially simple when ν is half-integer: $\nu = p + 1/2$, where $p \in \mathbb{N} \setminus \{0\}$, and converges to the SE kernel for $\nu \to \infty$. Th most commonly-used values for ν are $3/2$ and $5/2$.

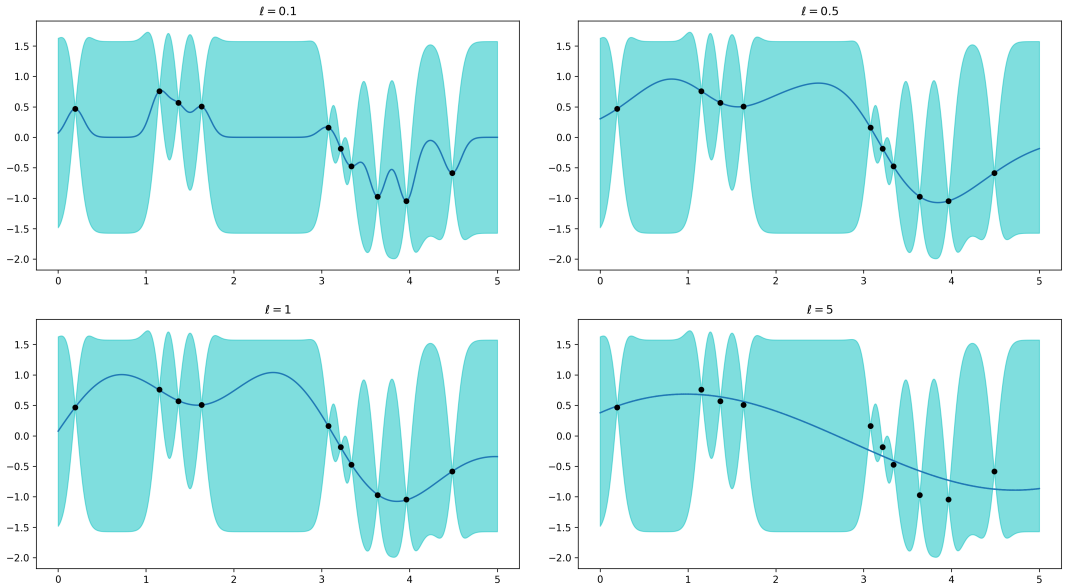

Figure 2.3: Effect of varying the length-scale parameter of an RBF kernel on the predictive distribution

The selection of the model parameters is performed by maximizing the log of the *marginal likelihood*, also known as the *evidence* of the data given the model, with respect to the set of hyper-parameters $\boldsymbol{\theta}$. This quantity measures how likely it is to sample my data from the pior distribution, and when dealing with Gaussian Prior and Gaussian Likelihood, the marginal likelihood can be computed analytically as

$$\log p\left(\mathbf{y} \,|X, \boldsymbol{\theta}\right) = -\frac{1}{2}\mathbf{y}^{\mathrm{T}} K_y^{-1} \mathbf{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log(2\pi) \tag{2.13}$$

where:

- $K_y = K(X, X) + \sigma_n^2 \mathbf{I}$
- The term $-\frac{1}{2}\mathbf{y}^{\mathrm{T}} K_y^{-1}\mathbf{y}$ is the data-fit
- The term $\frac{1}{2}\log|K_y|$ is the complexity penalty depending only on the covariance function
- The term $\frac{n}{2}\log(2\pi)$ is a normalization constant

The best set of hyper-parameters $\boldsymbol{\theta}^*$ for the kernel is obtained by the maximization of 2.13

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log p\left(\mathbf{y}\,|X, \boldsymbol{\theta}\right) \tag{2.14}$$

The marginal likelihood will tend to favour the least complex models able to explain the data. Several tools are available to train GP models, such as `GPyTorch` (Gardner et al. [2018]) and `scikit-learn` (Pedregosa et al. [2011]).

2.1.3 Scalability Issues and Sparse Approximation of GPs

Considering 2.9, it is noticeable how the computational complexity of the problem is dominated by the need to invert $\left[K(X, X) + \sigma_n^2 \mathbf{I}\right]$, which scales with $\mathcal{O}\left(n^3\right)$, being n the number of points used to train the model. This issue may limit the usability of GP, as adding too many data-points to the model may result in a computationally unfeasible problem. To deal with this issue approximations of the posterior distribution have been proposed, to reduce the computational complexity without losing accuracy.

The main approaches to solve this problem come from Titsias [2009] and have been extended by Hensman et al. [2013], and rely on the concept of *inducing points*, a smaller set of training points Z able to represent sufficiently well the nonlinear function we are trying to approximate. Using a variational approach, Hensman et al. [2013], makes the inducing points part of the optimization process in the course of the training. The posterior distribution is approximated as a multivariate Gaussian, whose mean and covariance have been optimized along with the inducing points location. The advantage of this approach is to reduce the computational cost of the GP inference from $O(n^3)$ to $O(nm^2)$, where m is the numer of inducing points.

This approach takes the name of *Stochastic Variational Gaussian Process* (SVGP), and can help reduce the computational complexity of GP. But since this is not the approach used in the thesis the topic is no further expanded.

2.2 GP-Based Wind Estimation

After introducing the theoretical background on GP, this section presents their practical use in the wind estimation process. In particular it is shown how the model is used to make predictions inside the control framework and how the basic GP formulation has been modified to better fit the task.

2.2.1 Disturbance Model

In this framework, the GP are used to estimate the wind forces affecting the drone along the x and y axis. In particular, to make the predictions, the model uses the xy-position of the drone $\mathbf{p} = [\, p_x,\, p_y \,]$ as its input and returns the estimated wind force $\mathbf{F_w} = [\, F_w^x,\, F_w^y \,]$ at such position, as in 2.4. The classic GP formulation, as described in the previous

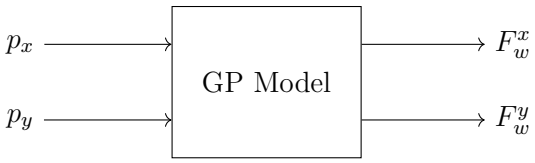

Figure 2.4: GP model for wind prediction

section, can only model a scalar output. As the wind data used in this work is a bi-variate quantity, it becomes imperative to extend the basic formulation to a one able to generate a multi-variate output. Two main approaches have been considered to deal with this issue, which affect how the predicted variables are correlated inside the model. The first approach, presented in Liu et al. [2018], consists in assuming that the variable of interest are correlated, and explicitly modelling such inter-task correlation. The other approach assumes that the two outputs of the model are independent, and thus can be modelled separately. This allows for the possibility of using two distinct GP to estimate the wind disturbances acting of the x and y axes, as in

$$\begin{bmatrix} F_x^w(\mathbf{p}) \\ F_y^w(\mathbf{p}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x(\mathbf{p}) \\ \mu_y(\mathbf{p}) \end{bmatrix}, \begin{bmatrix} \Sigma_x(\mathbf{p}) & 0 \\ 0 & \Sigma_y(\mathbf{p}) \end{bmatrix} \right) \tag{2.15}$$

Even though it does not model the correlation between the outputs, this approach allows to have different hyper-parameters and kernels for each GP. The assumption used in this work is that it is possible to model the wind disturbances using the same kernel, and kernel hyper-parameters, for both the outputs.

$$\begin{bmatrix} F_x^w(\mathbf{p}) \\ F_y^w(\mathbf{p}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_x(\mathbf{p}) \\ \mu_y(\mathbf{p}) \end{bmatrix}, \begin{bmatrix} \Sigma(\mathbf{p}) & 0 \\ 0 & \Sigma(\mathbf{p}) \end{bmatrix} \right) \tag{2.16}$$

$$\mu_x(\mathbf{p}) = K(\mathbf{p}, P) \left[K(\mathbf{p}, P) + \sigma_n^2 \mathbf{I} \right]^{-1} \mathbf{F}_x^w \tag{2.17}$$

$$\mu_y(\mathbf{p}) = K(\mathbf{p}, P) \left[K(\mathbf{p}, P) + \sigma_n^2 \mathbf{I} \right]^{-1} \mathbf{F}_y^w \tag{2.18}$$

$$\Sigma(\mathbf{p}) = K(\mathbf{p}, P) \left[K(\mathbf{p}, P) + \sigma_n^2 \mathbf{I} \right]^{-1} K(P, \mathbf{p}) \tag{2.19}$$

where:

- $P = [\, \mathbf{p}_0,\, \ldots,\, \mathbf{p}_M \,]^{\mathrm{T}}$ is the list of all the input positions used inside the model
- $\mathbf{F}_x^w$ and $\mathbf{F}_y^w$ are, respectively, the measured x and y wind forces used in the model

This stems from the observation that both GP models have the same inputs, and from the adaptation to the model described in the upcoming section.

2.2.2 Online Implementation of Gaussian Processes

Given the task of estimating external wind disturbances over a certain time horizon, the basic approach to GPR becomes unsuitable. As shown in 2.8, the standard formulation expects to have previous training data available to compute the predictive mean. Since the adopted approach uses the position of the system to compute the wind disturbances, training the model on a generic wind field, and using the collected data to preform predictions on another scenario turns out to be meaningless. The approach for this setting is then to either choose the hyper-parameters by hand or training the model on a reasonable scenario and only keeping the hyper-parameters, discarding the input data and labels.

During the online phase, following an active learning strategy, the system collects new data points at each time step, in the form of a (input, label) pair $(\mathbf{p}_k, \mathbf{F}_k^w)$, where the wind force $\mathbf{F}_k^w$ is measured as the mismatch between the nominal quadrotor state at time k, and the actual measured state

$$\mathbf{F}_k^w = \underbrace{\mathbf{B_d}}_{\substack{\text{Selection}\\ \text{Matrix}}} \left(\underbrace{\hat{\mathbf{x}}_{k+1}}_{\substack{\text{Nominal}\\ \text{state}}} - \underbrace{\mathbf{x}_{k+1}}_{\substack{\text{Measured}\\ \text{State}}} \right) \tag{2.20}$$

The selection matrix $\mathbf{B_d}$ chooses which states are used to measure the model mismatch.

Once the system has collected enough (input, label) pairs, the model starts making predictions about the wind disturbances. Every time a new data point is collected, from now on, the least recent one is discarded in favour of that, following a sliding window approach, as in

$$\underbrace{\begin{bmatrix} \mathbf{p}_{k-M} \\ \vdots \\ \mathbf{p}_{k-1} \end{bmatrix}}_{\substack{\text{Inputs at time}\\ k}}, \underbrace{\begin{bmatrix} \mathbf{F}_{k-M+1} \\ \vdots \\ \mathbf{F}_{k} \end{bmatrix}}_{\substack{\text{Labels at time}\\ k}} \Longrightarrow \underbrace{\begin{bmatrix} \mathbf{p}_{k-M+1} \\ \vdots \\ \mathbf{p}_{k} \end{bmatrix}}_{\substack{\text{Inputs at time}\\ k+1}}, \underbrace{\begin{bmatrix} \mathbf{F}_{k-M} \\ \vdots \\ \mathbf{F}_{k+1} \end{bmatrix}}_{\substack{\text{Labels at time}\\ k+1}} \tag{2.21}$$

The number of data-points to collect is specified by the size M of this *sliding window* of past data, and is chosen a priori. The complete algorithm is presented in 1.

Algorithm 1 Sliding Window Gaussian Process Regression

Input: $k(\cdot,\cdot)$, M ▷ Kernel hyper-parameters and sliding window size
  P, $F^w \leftarrow [\,]$ ▷ Empty list of inputs and labels
  loop
    $\hat{\mathbf{x}}_{k+1} \leftarrow$ Compute next state with the nominal model
    $\mathbf{x}_{k+1} \leftarrow$ Measure next state
    if len$(P) \geq M$ **then**
      Remove least recent element from P
      Remove least recent element from F^w
    end if
    Append $\mathbf{p}_k$ to P
    Append $\mathbf{B_d}\left(\hat{\mathbf{x}}_{k+1} - \mathbf{x}_{k+1}\right)$ to F^w
  end loop

Chapter 3

GP-Based Model Predictive Control

A popular choice for the development of a control system is the use of *Model Predictive Control* (MPC), a method that optimizes control actions by predicting future system behavior based on the dynamic model of the system. By complementing the dynamics of the drone with the wind prediction of the GP model, we can increase the effectiveness of the controller in the presence of external disturbances, and thus improve its performance. In the following chapter it is presented a description of the drone model used for the experiments, along with the mathematical formulation of the GP-complemented MPC.

3.1 Drone Control

Achieving effective control of a drone requires a strategy capable of managing its rapidly changing and complex dynamics. Model Predictive Control (MPC) is particularly suited for this purpose, as it determines optimal control actions by predicting the drone's future states over a defined horizon. By continuously optimizing control inputs, MPC ensures that the drone can respond accurately to both planned maneuvers and any immediate changes in desired position or orientation.

This approach relies on a dynamical model of the system, which allows it to anticipate and adjust for the drone's natural behavior. This capability makes the MPC ideal for drones, as it can handle the sudden changes in the control action required for stable flight. In the following sections, the mathematical model of the drone and the core principles of MPC are presented.

3.1.1 Drone Model

The quadrotor's dynamical model considered in this work is the one from Benders et al. [2024], and is defined by the following dynamics

$$\begin{bmatrix} \dot{p}^x \\ \dot{p}^y \\ \dot{p}^z \end{bmatrix} = \begin{bmatrix} v^x \\ v^y \\ v^z \end{bmatrix} \tag{3.1}$$

$$\begin{bmatrix} \dot{v}^x \\ \dot{v}^y \\ \dot{v}^z \end{bmatrix} = \begin{bmatrix} \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi) \\ -\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) \\ \cos(\phi)\cos(\theta) \end{bmatrix} a - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{3.2}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{a} \end{bmatrix} = \operatorname{diag}\left(-\frac{1}{\tau^\phi}, -\frac{1}{\tau^\theta}, -\frac{1}{\tau^\psi}, -\frac{1}{\tau^a}\right) \begin{bmatrix} \phi \\ \theta \\ \psi \\ a \end{bmatrix} + \operatorname{diag}\left(\frac{k^\phi}{\tau^\phi}, \frac{k^\theta}{\tau^\theta}, \frac{k^\psi}{\tau^\psi}, \frac{k^a}{\tau^a}\right) \begin{bmatrix} \phi^c \\ \theta^c \\ \psi^c \\ a^c \end{bmatrix} \tag{3.3}$$

with state $\mathbf{x} = [p^x, p^y, p^z, v^x, v^y, v^z, \phi, \theta, \psi, a]$ containing the position and velocities along the x, y, z axis, the attitude roll, pitch and yaw angles, and the collective mass-normalized thrust. The control inputs of the model are given by $\mathbf{u} = [\phi^c, \theta^c, \psi^c, a^c]$, containing the attitude commands, and the collective mass-normalized thrust. The τ and k are constant values, in particular $\boldsymbol{\tau} = [0.18, 0.18, 0.56, 0.05]$ and $k = 1$.

3.1.2 Model Predictive Control Fundamentals

The core principle of MPC relies on the minimization of a cost function J, that penalizes the mismatch between a desired state and the state predicted over an horizon N. An addition to this formulation also consider a penalization term for the deviation of the control action from a reference input. The resulting cost function can be expressed as

$$\begin{aligned} J = \sum_{k=0}^{N-1} \left(\mathbf{x}_k - \mathbf{x}_k^{ref}\right)^{\mathrm{T}} \mathbf{Q} \left(\mathbf{x}_k - \mathbf{x}_k^{ref}\right) + \left(\mathbf{u}_k - \mathbf{u}_k^{ref}\right)^{\mathrm{T}} \mathbf{R} \left(\mathbf{u}_k - \mathbf{u}_k^{ref}\right) \\ + \left(\mathbf{x}_N - \mathbf{x}_N^{ref}\right)^{\mathrm{T}} \mathbf{P} \left(\mathbf{x}_N - \mathbf{x}_N^{ref}\right) \end{aligned} \tag{3.4}$$

$\mathbf{Q}$ and $\mathbf{R}$ are diagonal matrices used to express the weight each, respectively, state and input deviation from the reference should have on the cost at the intermediate steps. The $\mathbf{P}$ matrix is the cost matrix for the last state, at time N.

The strength of MPC lies in its capability to handle constraints effectively. Unlike many traditional control methods, MPC optimizes the control actions while explicitly considering constraints within its predictive model. Constraints can be expressed as bounds on the states and inputs, but also using more complex formulations involving non-linear functions of $\mathbf{x}$ and $\mathbf{u}$.

$$\mathbf{x}_{\min} \leq \mathbf{x}_k \leq \mathbf{x}_{\max} \quad \forall k \in \{0, \ldots, N\} \tag{3.5}$$

$$\mathbf{u}_{\min} \leq \mathbf{u}_k \leq \mathbf{u}_{\max} \quad \forall k \in \{0, \ldots, N-1\} \tag{3.6}$$

$$g_i(\mathbf{x}_k, \mathbf{u}_k) \leq 0 \quad \forall k \in \{0, \ldots, N-1\} \tag{3.7}$$

$$h_j(\mathbf{x}_k, \mathbf{u}_k) = 0 \quad \forall k \in \{0, \ldots, N-1\} \tag{3.8}$$

The model of the system is in particular used as a dynamic constraint for the optimization problem, expressing how the drone state evolves over time, as in

$$\mathbf{x}_{k+1} = \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) \quad \forall k \in \{0, \dots, N-1\} \tag{3.9}$$

3.2 GP-Based Control and Controller Formulation

The use of Gaussian Processes inside the MPC allows to better optimize the control action and predicted future states, in accordance to the wind estimates of the model. The nominal quadrotor dynamics are extended using this ML algorithm, in order to improve the performance of the system. Not only this, but the uncertainty of the prediction is used to implement chance-constraints for obstacle avoidance, increasing the safety of the flight without being too conservative in the decision-making process.

3.2.1 Extended Drone Dynamics

Considering the wind prediction of GP as described previously, we can extend the dynamics of the drone to take into account this external disturbances. The velocity dynamics of the system, as expressed in 3.2, can be complemented with the model's estimates as

$$\begin{bmatrix} \dot{v}^x \\ \dot{v}^y \\ \dot{v}^z \end{bmatrix} = \begin{bmatrix} \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi) \\ -\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) \\ \cos(\phi)\cos(\theta) \end{bmatrix} a - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \mathbf{d}\left(\mathbf{p}\right) \tag{3.10}$$

where $\mathbf{d}(\mathbf{p}_k)$ represents the wind disturbance estimated by the model in position $\mathbf{p}$. The continuous system dynamics are discretized via the Runge-Kutta 4 method (RK-4), resulting in the following

$$\mathbf{x}_{k+1} = \mathbf{f}\left(\mathbf{x}_k, \mathbf{u}_k\right) + \mathbf{B_d}\, \mathbf{d}\left(\mathbf{p}_k\right) \tag{3.11}$$

The $\mathbf{B_d} \in \mathbb{R}^{9,2}$ matrix is a selection matrix, used to map the disturbance vector $\mathbf{d} \in \mathbb{R}^2$ to the correct state, as the wind only only blows along the x and y axis, and affects only the velocity dynamics directly.

3.2.2 State and Uncertainty Propagation

Since the non-linear wind dynamics are expressed by a GP, and thus follow a Gaussian distribution, the state of the system also become a stochastic variable. Evaluating the posterior of the GP at the next uncertain state input, however, is computationally intractable. Instead, the posterior distribution can be approximated using a Gaussian (Girard et al. [2002]), making the state at time $k+1$ follow a normal distribution too

$$\mathbf{x}_{k+1} \sim \mathcal{N}\left(\boldsymbol{\mu}^x_{k+1}, \boldsymbol{\Sigma}^x_{k+1}\right) \tag{3.12}$$

The mean is propagated over the prediction horizon using the law of iterated expectations (Girard et al. [2002])

$$\boldsymbol{\mu}^x_{k+1} = \mathbf{f}\left(\boldsymbol{\mu}^x_k, \mathbf{u}_k\right) + \mathbf{B_d}\, \boldsymbol{\mu}^{\boldsymbol{d}}\left(\boldsymbol{\mu}^p_k\right) \tag{3.13}$$

where $\boldsymbol{\mu}_k^p$ is the mean of the position at time k, and $\boldsymbol{\mu}^{\boldsymbol{d}}$ is the mean of the GP.

The variance is propagated by approximating the next state $\mathbf{x}_{k+1}$ via a first order Taylor series around the $\boldsymbol{\mu}_k^x$, similarly to what is proposed by Hewing et al. [2019]

$$\mathbf{x}_{k+1} \approx \underbrace{\nabla_{\mathbf{x}} \left(\mathbf{f}\left(\mathbf{x}, \mathbf{u}_k\right) + \mathbf{B_d}\, \boldsymbol{\mu}^{\boldsymbol{d}}\left(\mathbf{x}\right) \right) \Big|_{\mathbf{x}=\boldsymbol{\mu}_k^x}}_{\tilde{\mathbf{A}}_k} \tag{3.14}$$

Where $\nabla_{\mathbf{x}} \boldsymbol{\mu}^{\boldsymbol{d}}\left(\mathbf{x}\right)$ is the derivative of the predictive mean of the GP. The uncertainty is then propagated using the law of conditional variances using this linearized model

$$\boldsymbol{\Sigma}_{k+1}^x = \tilde{\mathbf{A}}_k \boldsymbol{\Sigma}_k^x \tilde{\mathbf{A}}_k^{\mathrm{T}} + \boldsymbol{B_d} \boldsymbol{\Sigma}_k^d \boldsymbol{B_d}^{\mathrm{T}} \tag{3.15}$$

Where $\boldsymbol{\Sigma}_k^x$ is the covariance of the state at time k, and $\boldsymbol{\Sigma}_k^d$ is the covariance of the GP prediction at time k.

3.2.3 Obstacle Avoidance and Chance-Constraint Formulation

Given that the main objective of this controller formulation is to be robust to wind disturbances in particular near obstacles, obstacles avoidance constraints are added to the MPC formulation. Constraints have to be formulated such that the space occupied by the drone at any give time k, does not intersect with the space occupied by the obstacles. In particular, by denoting with $\mathcal{B}(\mathbf{p}_k)$ the region of space occupied by the quadrotor a time k, and by $\mathcal{C}$ the region occupied by the obstacles, it must be ensured that

$$\mathcal{B}(\mathbf{p}_k) \cap \mathcal{C} = \emptyset \quad \forall k \tag{3.16}$$

Given that the state of the drone is affected by the uncertainty derived from the GP, as in 3.12, the obstacle avoidance constraint assumes a probabilistic formulation, where the collision region has to be extended to consider the uncertainty on the position. This constraint can be reformulated by requiring that the probability of the drone colliding with the obstacles is under a certain value δ

$$\Pr\left(\mathcal{B}(\mathbf{p}_k, \boldsymbol{\Sigma}_k^p) \cap \mathcal{C} \neq \emptyset\right) \leq \delta \quad \forall k \tag{3.17}$$

In this work, the obstacles considered are cylinders of radius r_i, each at position $\boldsymbol{p}_i^o = [\, p_i^{o_x}, p_i^{o_y} \,]^{\mathrm{T}}$. The volume occupied by the drone is the one of a sphere of radius r_0. The occupancy volume of the drone is extended in order to account for the uncertainty on the position, by including the axis of the uncertainty ellipsoid of the position at time k. The axis are derived from the covariance matrix of the position $\boldsymbol{\Sigma}_k^p$ which is diagonal due to the observation made in 2.2.1

$$\boldsymbol{\Sigma}_k^p = \begin{bmatrix} \sigma_{x_k}^2 & 0 \\ 0 & \sigma_{y_k}^2 \end{bmatrix} \tag{3.18}$$

The resulting axis ℓ_x and ℓ_y are then derived as

$$\ell_{x_k} = \sqrt{\sigma_{x_k} \cdot \chi_\delta^2} \tag{3.19}$$

$$\ell_{y_k} = \sqrt{\sigma_{y_k} \cdot \chi_\delta^2} \tag{3.20}$$

where χ^2_δ is the chi-squared value associated to a probability δ in the case of 2 degree of freedom. In this work, the probability δ has been considered to be $\delta = 0.9973$.

Since the kernel is the same for both the x and y axis for the wind estimation, the associated position uncertainty will be the same: $\ell_{x_k} = \ell_{y_k} = \ell_k$. The resulting clearance region where the drone avoids the n obstacle can then be represented as the following set of constraints

$$\left(\mathbf{p}_k^0 - \mathbf{p}^i\right)^{\mathrm{T}} \left(\mathbf{p}_k^0 - \mathbf{p}^i\right) \geq \left(r^0 + r^i + \ell_k\right)^2 \quad \forall i \in \{1, \ldots, n\} \text{ and } \forall k \tag{3.21}$$

3.2.4 Complete MPC Formulation

Given the previous consideration, the complete MPC formulation used for this work is the following, where the trajectory the drone has to follow has been expressed in terms of a set of position and velocity references

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u}} \sum_{k=0}^{N-1} & \begin{bmatrix} \mathbf{p}_k - \mathbf{p}_k^{ref} \\ \mathbf{v}_k - \mathbf{v}_k^{ref} \end{bmatrix}^{\mathrm{T}} \mathbf{Q} \begin{bmatrix} \mathbf{p}_k - \mathbf{p}_k^{ref} \\ \mathbf{v}_k - \mathbf{v}_k^{ref} \end{bmatrix} + \left(\mathbf{u}_k - \mathbf{u}_k^{ref}\right)^{\mathrm{T}} \mathbf{R} \left(\mathbf{u}_k - \mathbf{u}_k^{ref}\right) \\ + & \begin{bmatrix} \mathbf{p}_N - \mathbf{p}_N^{ref} \\ \mathbf{v}_N - \mathbf{v}_N^{ref} \end{bmatrix}^{\mathrm{T}} \mathbf{P} \begin{bmatrix} \mathbf{p}_N - \mathbf{p}_N^{ref} \\ \mathbf{v}_N - \mathbf{v}_N^{ref} \end{bmatrix} \end{aligned} \tag{3.22}$$

subject to

$$\boldsymbol{\mu}_0^x = \mathbf{x}_0 \tag{3.23}$$

$$\boldsymbol{\mu}_{k+1}^x = \mathbf{f}\left(\boldsymbol{\mu}_k^x, \mathbf{u}_k\right) + \mathbf{B_d}\, \boldsymbol{\mu}^{\boldsymbol{d}}\left(\boldsymbol{\mu}_k^p\right) \tag{3.24}$$

$$\boldsymbol{\Sigma}_{k+1}^x = \tilde{\mathbf{A}}_k \boldsymbol{\Sigma}_k^x \tilde{\mathbf{A}}_k^{\mathrm{T}} + \boldsymbol{B_d} \boldsymbol{\Sigma}_k^d \boldsymbol{B_d}^{\mathrm{T}} \tag{3.25}$$

$$\mathbf{x}_{\min} \leq \mathbf{x}_k \leq \mathbf{x}_{\max} \tag{3.26}$$

$$\mathbf{u}_{\min} \leq \mathbf{u}_k \leq \mathbf{u}_{\max} \tag{3.27}$$

$$\Pr\left(\mathcal{B}(\mathbf{p}_k, \boldsymbol{\Sigma}_k^p) \cap \mathcal{C} \neq \emptyset\right) \leq \delta \tag{3.28}$$

$$k = 0, \ldots, N-1$$

3.3 Controller Implementation

The MPC implementation has been carried out in `Python` using two different frameworks. The first implementation uses `CasADi` (Andersson et al. [2019]), which is an open-source tool for nonlinear optimization and algorithmic differentiation, that also offers an integration with the `Ipopt` solver (Wächter and Biegler [2006]). The second version adopts the `acados` (Verschueren et al. [2021]) framework, which provides fast and embedded solvers for nonlinear optimal control problems.

3.3.1 General Control Scheme

Both MPC formulations follow a similar scheme, but differ in the realization of GP-MPC block. Given that the GP model needs to collect enough wind data before making the predictions, as in 1, the controller starts without the use of the predictors, and takes advantage of them once enough information are available. As the drone traverses the wind field, the mismatch between the drone model and the measured state is used as an estimate of the external wind force for the GP model, which, in case of a simulator, are directly available. The MPC not only provides the optimal control input, but also the predicted optimal state over the horizon $\mathbf{x}^*_{k:k+N} = \{\mathbf{x}^*_k, \dots, \mathbf{x}^*_{k+N}\}$, which is used as initial guess for the interior point solver. The complete control scheme is presented in 3.1.

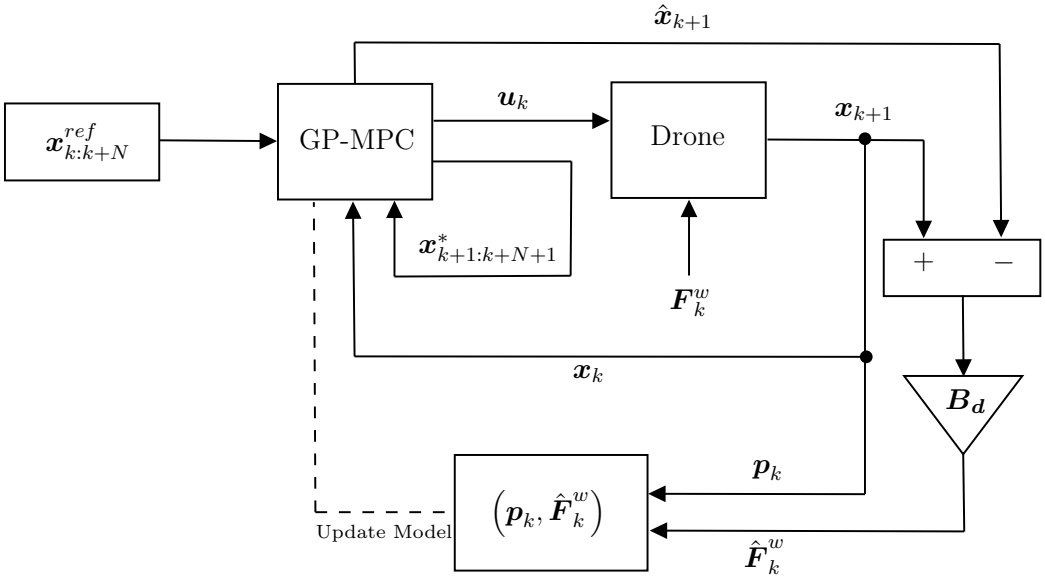

Figure 3.1: GP-MPC Block Diagram

3.3.2 CasAdi Implementation

In the `CasAdi` version of the GP-MPC, the wind prediction resulting from the GP model is carried out completely symbolically inside the optimization problem. This means that, instead of using a specific value to compute the wind disturbances, the position of the drone used for the model is kept symbolical, and the GP posterior distribution is computed symbolically and optimized inside the solver. The scheme of this implementation is shown in 3.2.

The optimizer yields the decision variables for the state $\mathbf{x}^o_i$ (including the position $\mathbf{p}^o_i$) and the control input $\mathbf{u}^o_i$, $i \in \{k, \dots, k+N\}$. These variables are used, first, by the GP model, which computes the wind estimate for the model and the covariance for the uncertainty propagation. The model in turn receives the prediction resulted from the GP and the state and input decision variables from the optimizer; it then computes the next

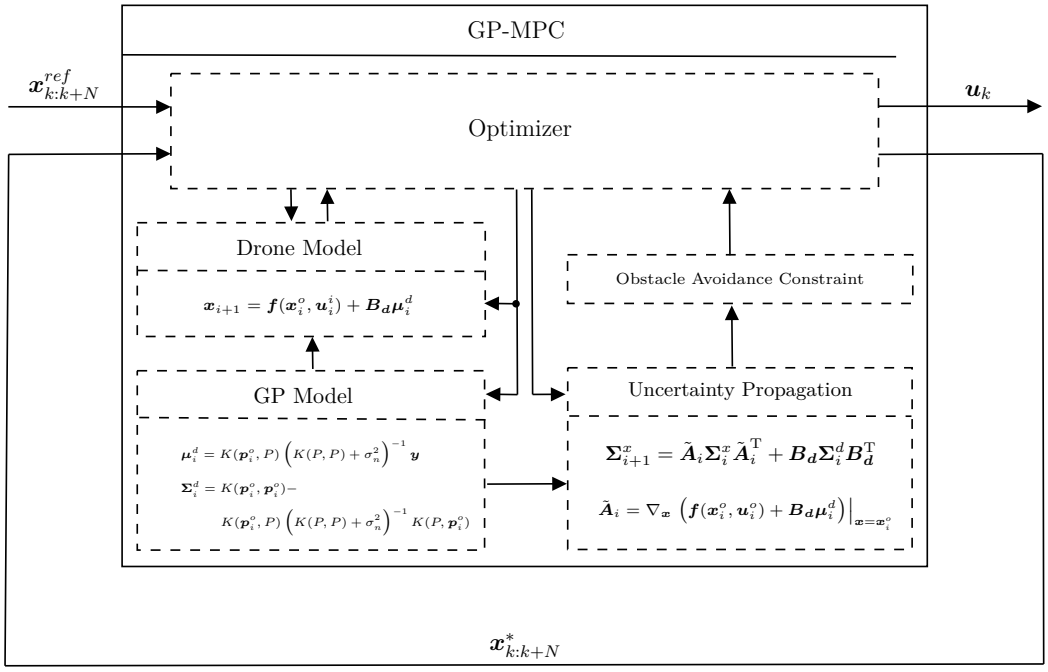

Figure 3.2: Block Diagram of the `CasADi` Implementation of the GP-MPC

state using the drone model and returns it to the optimizer for the dynamical constraints. At the same time the GP outputs the posterior covariance to propagate the uncertainty on the position of the quadrotor and enforce the obstacle-avoidance constraints. The optimizer receives the reference trajectory and the previously computed optimal state to use as initial guess for the solver, and following the before-mentioned scheme outputs the optimal control action $\mathbf{u}_k$ and the predicted optimal state $\mathbf{x}^*_{k:k+N}$.

3.3.3 Acados Implementation

In the `acados` version of the GP-MPC, the wind prediction of the GP model is computed using as position the optimal state resulting from the previous MPC computation. The functioning of this version of the controller is shown in 3.3.

As for the previous case the optimizer yields the decision variables for the state $\mathbf{x}^o_i$ and the control input $\mathbf{u}^o_i$, $i \in \{k, \dots, k+N\}$. The GP model instead of using those to compute the predictive mean and variance, utilizes the previously computed optimal state $\mathbf{x}^*_i$ (which includes the position $\mathbf{p}^*_i$), $i \in \{k, \dots, k+N-1\}$. The predictive mean is used inside the drone model, along with the decision variables, to compute the next state, used for the dynamical constraint in the solver. At the same time, the predictive variance is used to propagate the uncertainty on the quadrotor position, necessary for the obstacle avoidance constraints. As for the previous version, the optimizer returns the optimal control action $\mathbf{u}_k$ and the predicted optimal future state $\mathbf{x}^*_{k:K+N}$.

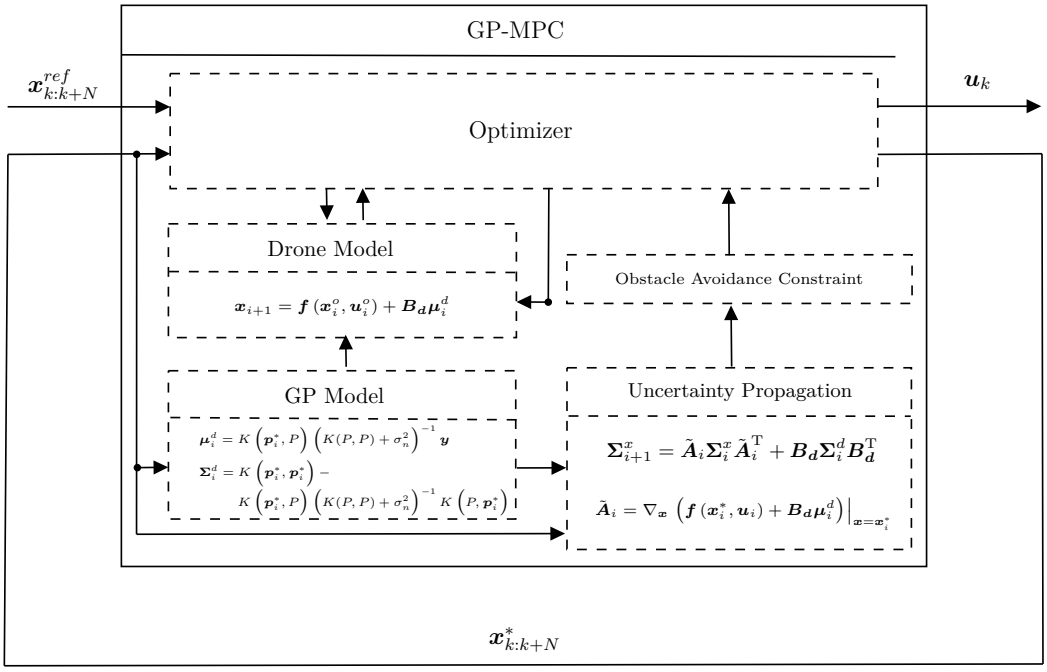

Figure 3.3: Block Diagram of the `acados` implementation of the GP-MPC

3.3.4 Considerations on the Architectural Choices

The choice to test two different MPC frameworks arose from the inclusion of the GP model inside the controller. The inversion of the covariance matrix, necessary to compute the posterior distribution (2.8, 2.9), caused some issues in this work implementation of the GPR. In particular, given that a set of points close to each others is used to compute the predictive distribution, the associated kernel matrix results to be close to singular, having many repeated eigenvalues similar and close to zero, due to the high correlation among the input points.

The inversion of the before-mentioned matrix symbolically results in a badly-scaled optimization problem, which might hinder the functioning of the solver if not properly taken into consideration. This issue is automatically addressed by the `Ipopt` solver, which perform a pre-scaling of the optimization problem before solving it. This action is able to fix the numerical instability deriving from such problem formulation, and allows, as shown in 3.2, to directly compute the GP predictive distribution inside the solver. The negative side of this framework is its slowness in computing the optimal solution, which could lead to the solver not generating the control action within the time threshold.

The implementation of the GP-MPC using `acados` allows it to solve the optimization problem, and thus generate the optimal control action, in a fast and efficient manner, making it suitable for real-time applications. The downside of this version is that the solver inside `acados` does not perform a pre-scaling of the problem, resulting in numerical instability if computing the predictive distribution symbolically, which cause the solver

to fail. This issue led to the implementation of the controller shown in 3.3, which computes the wind estimates and propagates the uncertainty outside the solver, solving the numerical instability caused by the symbolical inversion of the kernel matrix.

Chapter 4

Experiments and Results

To test the effectiveness of the GP-MPC, different scenarios consisting of various trajectories and obstacle placements have been tested. The wind fields used in these experiments consist both of artificially generated forces and real-world collected data. A comparison is performed between the proposed implementation of the MPC, a version that does not account for the disturbances and one that uses a simple baseline consisting of the last wind measurement to account for the external forces affecting the drone. This chapter presents the experimental setup used in this work and the related results obtained.

4.1 Experimental Setup

This section details the simulated environment, including the design of wind fields and the placement of obstacles, as well as the specific trajectories used for the tests. Parameters used in the model and in the MPC are described and discussed throughout this section, as well as the methods used to generate the wind data. The experiments have been performed in a simulation environment, allowing direct access to the state of the drone and the wind data, without the need to extrapolate it from the model mismatch.

4.1.1 Wind Field Generation

Two types of wind fields have been considered in this work. The first type consists of artificially generated wind data, based on simple equations to try mimicking the effect of fans placed throughout the map. This approach allows the maximum flexibility in terms of data-generation, but lacks the effects of a real world scenario. The second setting considered consists of various data collected using an anemometer, having a fan placed in the laboratory as wind source. The advantage of this approach is the more realistic data, but comes at the cost of a less flexible experimental scenario setup.

All wind fields considered in this work are assumed to be time-invariant, but spatially varying, and the area on which the experiments are tested consists of a $4.6 \times 4.6\,m$ grid. The wind is generated only for the x and y axes, it is assumed to be constant along the z axis and not generate an effect on it: $\mathbf{v}^w = [v_x(x, y),\, v_y(x, y),\, 0]^{\mathrm{T}}\ \forall z$. The wind force

deriving from the wind speed is computed using

$$\boldsymbol{F}^w = \frac{1}{2}\,\rho\, C_d\, S\, \boldsymbol{v}^{w^2}\, \mathrm{sgn}\,(\boldsymbol{v}^w) \tag{4.1}$$

This is a simple equation describing the force the wind exert on a surface S, when the air density is ρ. The considered parameters are the following:

1. Drag coefficient $C_d = 0.47$
2. Air density $\rho = 1.225\, kg/m^3$
3. The surface S hit by the wind is assumed, for simplicity to be the cross-sectional area of a sphere of radius r: $S = \pi\, r^2$. The radius of the drone has been assume to be $0.15\, m$

Artificially Generated Wind Fields

The first type of map has been generated by making some consideration on how the wind is generated from a real fan. In particular, it has been assumed to have a linear decrease on the wind speed along the direction perpendicular to the fan $d_\perp$, and a conic spread along the direction parallel to the fan $d_\parallel$. This led to the use of the following equation to artificially generate the wind

$$v(d_\perp, d_\parallel) = \frac{v_0(t)}{2} \frac{\tanh\left(\frac{d_\perp + \frac{L}{2}}{w\cdot(d_\parallel+1)}\right) - \tanh\left(\frac{d_\perp - \frac{L}{2}}{w\cdot(d_\parallel+1)}\right)}{d_\parallel + 1} \tag{4.2}$$

where:

- $v_0(t)$ is the velocity in the center of the fan at time t, which is specific to each fan and each simulation
- L is the width of the fan
- $w = 0.002$ determines the spread of the wind cone
- $d_\perp$ and $d_\parallel$ are the distances between the center of the fan and the point $\mathbf{p} = [x, y]^\mathrm{T}$ along, respectively, the axis perpendicular to the wind direction versor $\mathbf{u}_0$ and the axis parallel to it. Such distances are computed as

$$d_\parallel = |\mathbf{p} \cdot \mathbf{u}_0|$$
$$d_\perp = \left|\mathbf{p} \cdot \mathbf{u}_0^\perp\right|$$

The behavior of this wind generation method is shown in 4.1 The advantage of this approach is the possibility to generate any desired behavior for the wind, including time-varying fields. This comes at the disadvantage of not being able to properly include the obstacles inside the generated map, and the lack of resemblance with the real world. Due to these two issues, this approach won't be taken into account for the experiments, but can be of use as a starting base for future research if time-varying wind fields were to be considered.

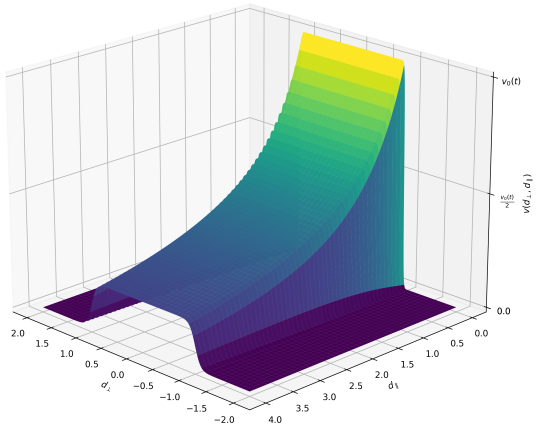

Figure 4.1: How the wind of an artificially generated fan varies along the perpendicular and parallel direction of it, with respect to its center

Real-World Measurements

The measurements collected in the laboratory, using a real fan and an anemometer, consist of a $N \times N$ grid of measurements of the $4.6 \times 4.6\,m$ area, which are then interpolated to generate a more dense $M \times M$ grid. In particular two settings have been measured, and some combinations of the these have been considered in the experiments. The data collected, other than being combined to form more complex settings, has been rescaled to increase the wind speed considered.

The measurements taken consist of two setting, one without obstacles, and one with an obstacle in the middle of the field. The first setting, the one without obstacles, consists of a 11×11 grid of points, while the second scenario, which contains an obstacle in the middle, consists of a grid of 8×8 points. Both of these two maps are interpolated into a 1000×1000 grid of points, using `Scipy`. The first setting, since no obstacle is present has been measured only along the parallel direction of the fan, while for the other, the measurements have been taken along both directions, to properly see the wind curling around the obstacle. The original data and the interpolated version are shown in 4.2 and 4.3.

During the experiments, the wind speed has been rescaled, to better show the effectiveness of the proposed model in rejecting the external disturbances. The wind map generated from the measurements, even though more realistic than the simulated version, does not come without flaws. Two factors mainly contributed to the inaccuracy of the data: first, the measurements where taken by hand using an anemometer, which by itself is not an accurate data-collecting procedure for this kind of phenomenon, as twisting the tool or placing in slightly different positions may result in different measures, not considering the inertia of the device; secondly, having taken the measurements in a closed

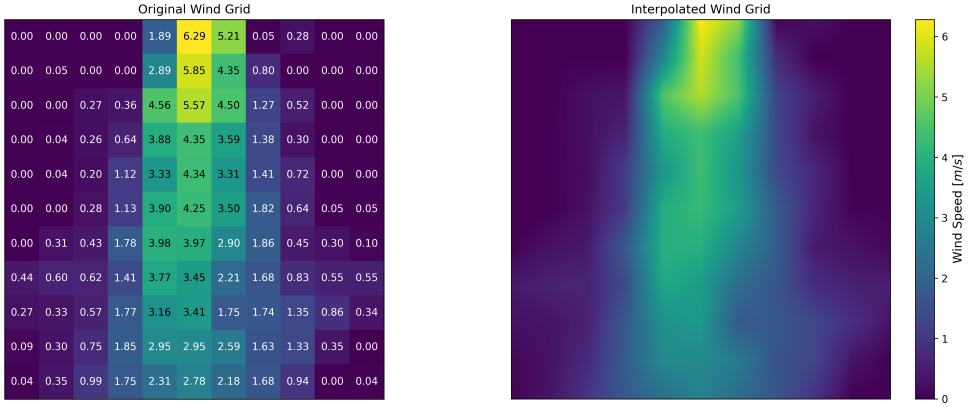

Figure 4.2: Measurements taken for the scenario without obstacles. The data has been collected only along the y axis. On the left, the grid of original measurements, on the right, its interpolated version.

environment, the interaction of wind with walls and other surfaces can create turbulence, reflections, and interference patterns, hindering the measures. Nevertheless, the resulting maps are satisfactory in terms of wind behavior, and as such are used to test the presented framework.

4.1.2 GP-MPC and Simulation Parameters

The experiments have been run considering a set of fixed values for the model and the MPC weight matrices. The models have been trained on the 4.2 wind field, to retrieve their parameters, emptied of their data, and used as described in 1 with a window of M points. The kernel considered in this work is the RBF kernel (2.11), with the addition of an output scale multiplicative factor A

$$k(\mathbf{x}, \mathbf{x}') = A \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||_2^2}{2\ell^2}\right) \tag{4.3}$$

The model parameters, MPC weight matrices and simulation parameters are reported in 4.1.

4.2 Comparison and Results

Both versions of the GP-MPC have been tested against two implementation of the MPC: the first one does not take into account the wind, while the second one uses the last wind measure $\mathbf{F}_{k-1}^{\hat{w}}$ as wind disturbance, and keeps it constant throughout the whole prediction horizon. Two tests have been performed to verify the validity of the approach: the first one consists of a wind field without any obstacles, consisting of a combination of the map shown in 4.2, while the second one represents a scenario containing one obstacle in the

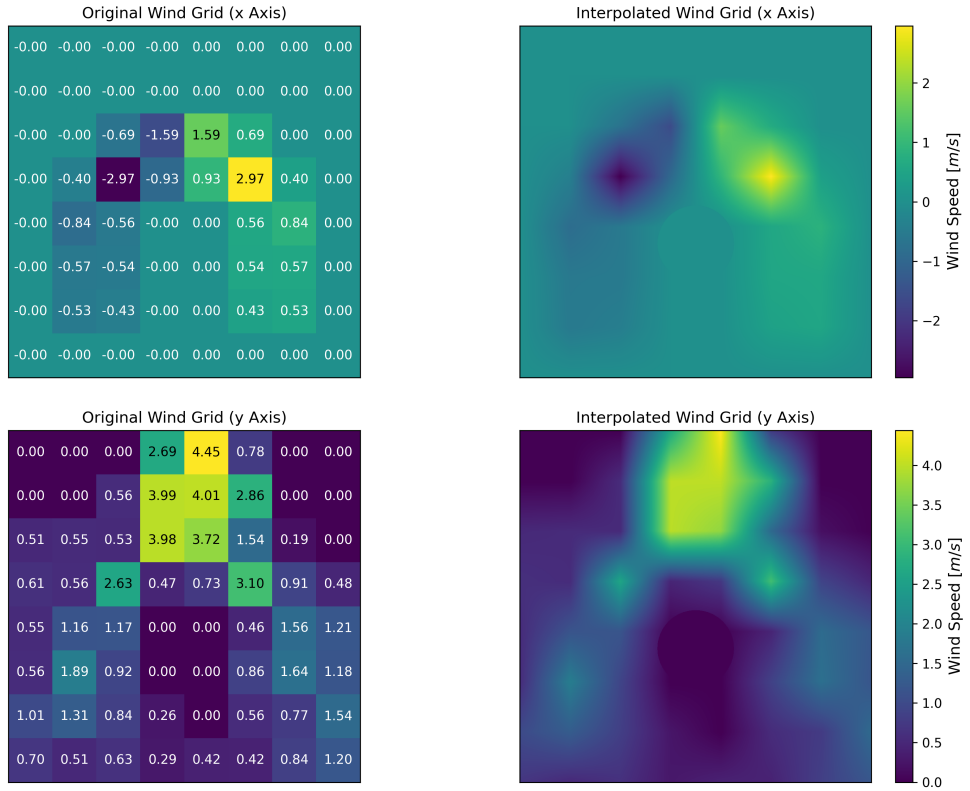

Figure 4.3: Measurements taken for the scenario containing an obstacle. On the left, the two original grid of wind measurements for the x axis (above), and the y axis (below). On the right, their interpolated version.

center, which stems from the data collected in 4.3. Both wind maps have been rescaled by a factor of 3, to better show the effects of the disturbances and the effectiveness of the adopted model. The two wind fields are shown in 4.4 and tested against various trajectories (4.5).

4.2.1 Tracking Performance Comparison

To test the tracking performance of the model in the presence of disturbances, this set of experiments has been run on the wind map without any obstacles, considering 6 different trajectories. To measure the tracking performance, the Root Mean Square Error (RMSE) on the position has been considered, with the exclusion of the first N_0 points, corresponding to 3 s, to properly account for the time when the GP-MPC does not use the predictor yet, and until it has brought back the drone on the proper trajectory

$$\text{RMSE} = \sqrt{\frac{1}{N_0} \sum_{k=N_0} \left|\left| \mathbf{x}_k - \mathbf{x}_k^{ref} \right|\right|_2^2} \tag{4.4}$$

GP Kernel Lengthscale ℓ	0.547
GP Kernel Output scale A	0.013
GP Likelihood noise σ_n^2	$6.892 \cdot 10^{-4}$
GP Window Size M	20
MPC State Weight Matrix $\mathbf{Q}$	diag (10,10,10,2,2,2)
MPC Final State Weight Matrix $\mathbf{P}$	diag (10,10,10,2,2,2)
MPC Control Weight Matrix $\mathbf{R}$	diag (1,1,1,1)
MPC Control Horizon N	10
Control Input Lower Bound $\mathbf{u}_{\min}$	$[-\frac{\pi}{6}$ rad, $-\frac{\pi}{6}$ rad, $-\frac{\pi}{6}$ rad, 5 m/s$^2]$
Control Input Upper Bound $\mathbf{u}_{\max}$	$[\frac{\pi}{6}$ rad, $\frac{\pi}{6}$ rad, $\frac{\pi}{6}$ rad, 15 m/s$^2]$
Discretization Time Δt	50 ms
Total Simulation Time	15 s
Simulation Frequency	1000 Hz
Control Frequency	20 Hz (50 ms)

Table 4.1: GP-MPC and Simulation Parameters Value

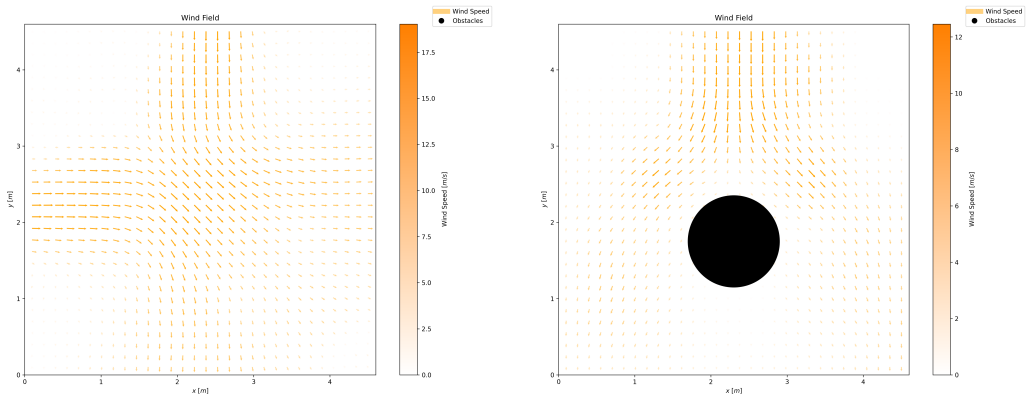

Figure 4.4: On the left, the wind field without any obstacles. On the right, the scenario with one obstacle in the center. The wind field is represented by arrows, whose hue and length describe their intensity.

The results of these simulations are presented in 4.2. As expected, if we do not take into account the wind, the performance degrades considerably, leading to a poor tracking of the desired trajectory. Of more interesting study is the comparison among the two

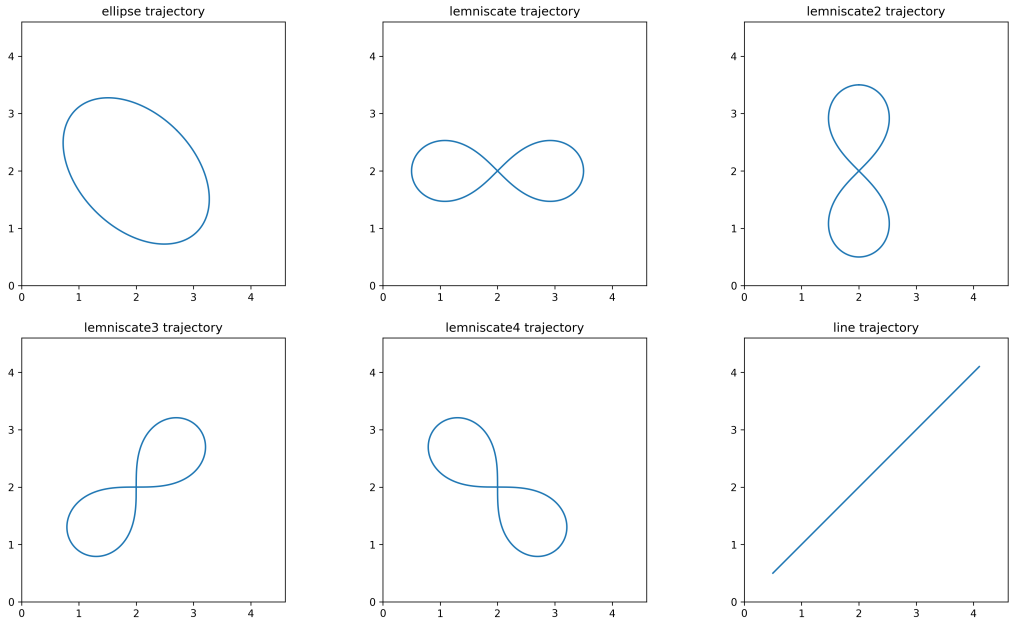

Figure 4.5: Trajectories used in the experiments

GP-MPC implementations and the version of the controller with the baseline. The three implementations all achieve good results in terms of trajectory tracking, with the `acados` version leading in terms of low RMSE.

	Ell.	**Lem. 1**	**Lem. 2**	**Lem. 3**	**Lem. 4**	**Line**
No wind x	0.180 m	0.203 m	0.139 m	0.151 m	0.138 m	0.104 m
No wind y	0.131 m	0.153 m	0.211 m	0.188 m	0.176 m	0.128 m
Baseline x	0.056 m	0.036 m	0.014 m	0.013 m	0.020 m	0.009 m
Baseline y	0.015 m	0.015 m	0.019 m	0.013 m	0.014 m	0.011 m
GP `acados` x	0.025 m	0.016 m	0.002 m	0.002 m	0.003 m	0.001 m
GP `acados` y	0.008 m	0.006 m	0.010 m	0.009 m	0.008 m	0.006 m
GP `CasADi` x	0.051 m	0.041 m	0.022 m	0.012 m	0.009 m	0.010 m
GP `CasADi` y	0.046 m	0.040 m	0.073 m	0.064 m	0.056 m	0.043 m

Table 4.2: RMSE measuring the tracking of each tested MPC implementation against the proposed trajectories. For each model measures for the x and y axis tracking performance have been reported.

4.2.2 Obstacle Avoidance Performance

Considering that the main focus of this Thesis is to work toward safe control in the presence of wind disturbances, of greater interest is the study of the scenario containing an obstacles. This setting highlights the efficacy of the GP model in combination with the MPC controller, which is able to safely fly around the obstacles even in the presence of external unknown forces. For this setting, since the trajectories have not been changed to account for the presence of an obstacle, the test has been performed by only considering which MPC formulation can successfully complete the flight without colliding with the obstacle. The results of such tests have been reported in 4.3.

	Ell.	**Lem. 1**	**Lem. 2**	**Lem. 3**	**Lem. 4**	**Line**
No Wind	✓	✗	✗	✗	✗	✗
Baseline	✓	✗	✗	✗	✗	✗
GP `acados`	✓	✓	✓	✓	✓	✓
GP `CasADi`	✓	✓	✓	✓	✓	✓

Table 4.3: Results of the simulation in the presence of an obstacle obstructing part of the trajectory. The successful flights have been marked with a ✓, while the failed ones with an ✗.

As it is possible to see, when the drone the needs to avoid an obstacle in the presence of a wind field, the standard MPC formulation and the one with the baseline fail to safely drive the quadrotor around it and avoiding collisions. The GP-MPC formulation, both with the `acados` and `CasADi` frameworks, is instead able to safely stay away from it, successfully accounting for the wind disturbances. This stems, apart from the predictive mean itself, also from the uncertainty associated to such estimates, as they provide a broader collision bound around the obstacle, making the MPC more conservative when the GP model is more uncertain about its predictions. This behavior becomes evident when we look at 4.6, the control action is more conservative in regions where the model is more uncertain, while avoids excessive care when the estimates are more reliable. In the figure it is possible to see a notable difference in terms of uncertainty in the two GP-MPC implementations, leading to a more or less conservative flight depending around the obstacles.

4.2.3 Solving Time

An important factor in the regards of a good controller, is the time it takes to generate the optimal control action. If the solver is too slow the generate input, the control can be imprecise, as it was computed for an earlier state, or, in extreme cases, can render the system unstable. For these reasons, one concern when developing this framework has been to guarantee the real-time feasibility of the MPC. This has been tested in simulation by retrieving the solver time after each call, and verifying that it would not exceed the

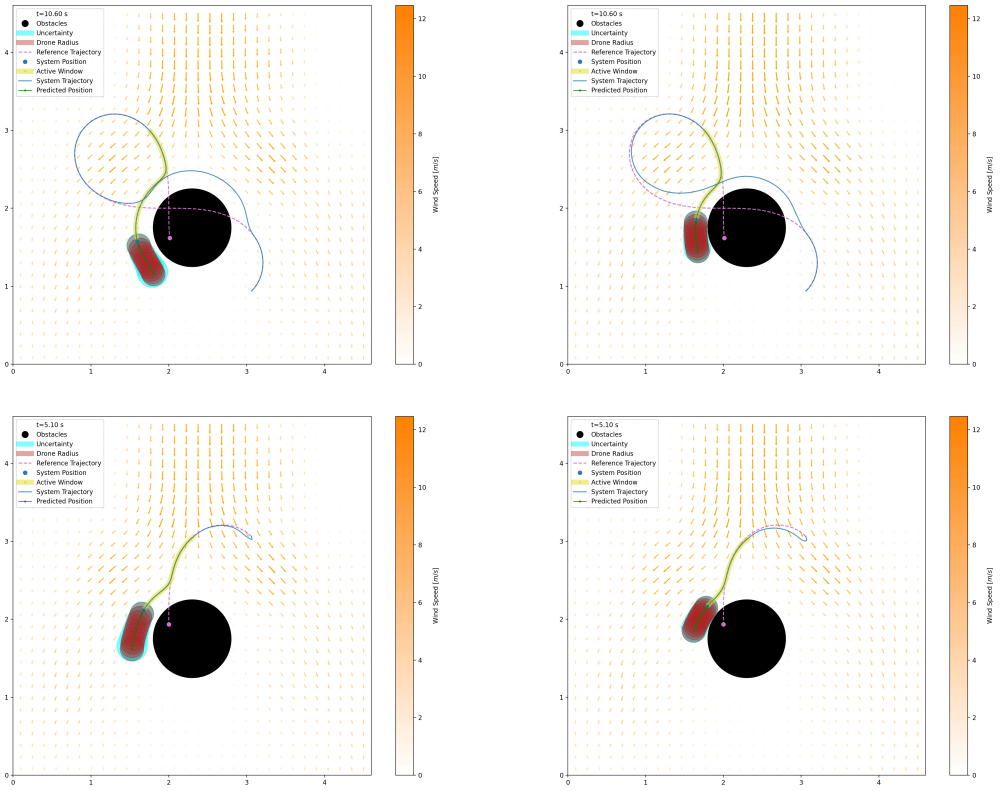

Figure 4.6: Visualization of the GP-MPC flight in the presence of an obstacle for tow different trajectories. The red circle represent the occupancy area of the drone, while the cyan one represent the derived uncertainty from the predictions. In the image is also reported the set of points used to make the predictions (yellow-highlighted line) and the future predicted positions (green dots). On the left the `acados` implementation, on the right the `CasADi` one.

imposed control frequency of 50 ms. The resulting average solving time for each controller formulation has been reported in 4.4.

The table shows how the computation time required by the GP-MPC is considerably higher than for the base versions. Furthermore, comparing the `acados` and `CasADi` implementations, it becomes evident how the former if by far faster than the latter. This slowness in generating the optimal solution likely stems from two main factors: first, the inversion of the covariance matrix which, scaling with $O(n^3)$ is an highly expensive operation; second, the propagation of the uncertainty throughout the horizon, which requires various matrix multiplications to be performed.

As it is possible to see, the `CasADi` implementation is far slower even than the `acados` implementation. This behavior comes from the heavy computations performed inside the solver, as in this version of the controller, the whole optimization process is performed

	Ell.	Lem. 1	Lem. 2	Lem. 3	Lem. 4	Line
No Wind	0.281 ms	0.290 ms	0.450 ms	0.309 ms	0.308 ms	0.198 ms
Baseline	0.339 ms	0.337 ms	0.525 ms	0.352 ms	0.303 ms	0.229 ms
GP `acados`	1.637 ms	1.656 ms	1.837 ms	1.686 ms	1.669 ms	1.563 ms
GP `CasADi`	15.178 ms	15.764 ms	15.571 ms	15.459 ms	15.446 ms	14.744 ms
No Wind	0.165 ms	0.443 ms	0.466 ms	0.310 ms	0.342 ms	0.234 ms
Baseline	0.163 ms	0.272 ms	0.436 ms	0.311 ms	0.353 ms	0.258 ms
GP `acados`	1.477 ms	1.785 ms	1.777 ms	1.620 ms	1.661 ms	1.533 ms
GP `CasADi`	26.791 ms	40.527 ms	41.159 ms	32.972 ms	35.402 ms	31.958 ms

Table 4.4: Average time required by the solver to compute the optimal control action for each considered trajectory without obstacles (above) and with the obstacle (below). The solver time has been computed also for the cases when the drone collided.

inside the solver, especially the computation of the predictive distribution and the uncertainty propagation. Furthermore, the `acados` framework compiles the solver in `C/C++`, making it even faster that its `Python` counterpart.

4.2.4 Comparison between acados and CasADi Implementations

As shown in the previous sections, the `acados` and the `CasADi` implementations of the GP-MPC present significant differences, both in terms of solving time (4.4) and, in particular, in the way the uncertainty affects the flight in the regions where the predictions are less reliable.

Referencing 4.7 it is possible to see how, with respect to the same instant during the simulation, the two implementation behavior is considerably different. The `casadi` version is shown trying getting closer to the reference trajectory as fast as possible, making more sharp turns and speeding the system more in order to reach the target position. This has the the solver generate optimal future points that are further from the current position, which, in turns, means making estimates with the GP model using points that are farther away from the set currently included in the window of data used for the predictions. Using points too far from the current set, makes the predictions less accurate and growing the uncertainty associated with them. This in turns make the model more conservative, making it stay further away from the obstacle.

In contrast, the `CasADi` version, which uses the `Iopot` solver, is less concerned with getting close to the reference trajectory as quickly, making the optimal future states closer to the current position and rendering the prediction of the model less uncertain, which, in turns, allows the drone flight to be less conservative around the obstacle. This behavior likely stems from the fact that the `CasADi` solver internally handles and optimizes the wind

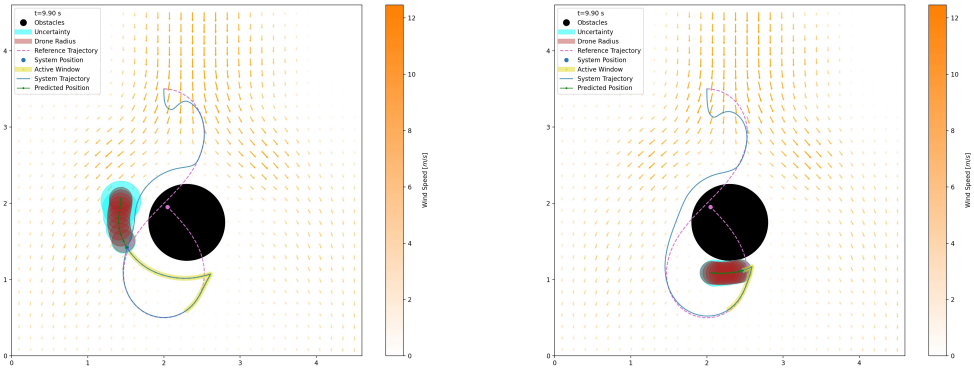

Figure 4.7: Frames extrapolated from the flight simulation performed in the presence of an obstacle. On the left, the `acados` results, on the right the `CasADi` performance. Both frames correspond to the same time instant during the simulation.

prediction and uncertainty propagation. This could lead to a more optimized trajectory that enables this solution to be less conservative than its counter part, allowing it to fly closer to the obstacle without colliding.

Even though the solution with the `Ipopt` solver is less conservative, this aspect is not the only one to take into account. In particular if we look at 4.4, it's evident how this solver is far slower than its `acados` counterpart, being approximately 10 times slower. This aspect becomes crucial when dealing with real-time feasibility and strict solving-time constraint: other than not being able to increase the control frequency, the `CasADi` solution is not always able to generate the control action in time, which could lead to dire consequences if the phenomenon becomes too frequent. In the performed experiments, for each trajectory, the number of violations of the solver-time constraint has been between 10 and 30, while the `acados` version never violated such limit. Increasing the control frequency would most likely lead to the infeasibility of this approach.

With these consideration in mind, it is possible to see how the `acados` framework could be better suited for real-time applications, where the control frequency is high, and reliable control actions need to be delivered quickly. Even though the model tends to be more conservative, due to the uncertainty of the predictor, the capability of the solver to generate the optimal control faster allows for a more frequent control, allowing the GP model to make prediction in a shorter range of positions, yielding less uncertainty and thus making the control less conservative in turn. This effect is shown in 4.8, where the control frequency has been increased to 50 Hz (20 ms as maximum computation time).

4.2.5 Tests Performed on More Complex Scenarios

To stress the capabilities of the implemented approach, other tests have been performed in more complex scenarios, which include more obstacle. The wind field in which these experiments have been performed, even though stems from real-world data, is not representative of a realistic scenario, as the obstacles have been placed only in simulation,

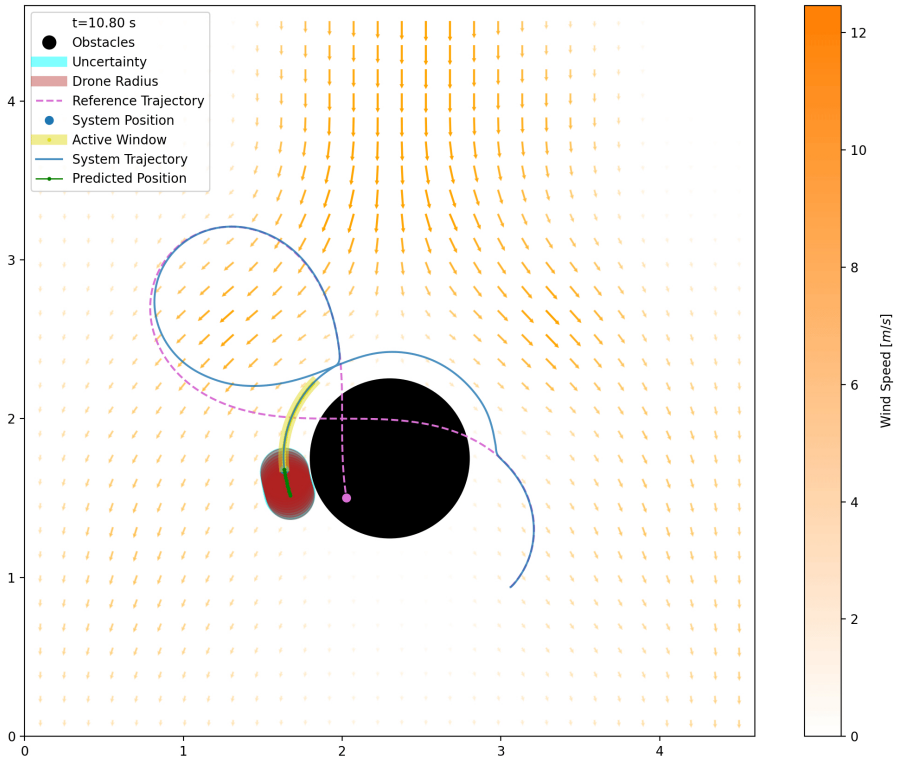

Figure 4.8: Simulation performed with an higher control frequency using the `acados` solver. The model is more confident about its predictions, yielding less uncertainty and making the control action less conservative.

collecting the data in the laboratory without putting them in the setting. The wind field used for testing is the one presented in 4.9, which shows the wind field originated from the collected data, and the version used in simulation, that includes the presence of obstacles. Even though the setting is not representative of a real-world scenario, the wind disturbances alongside the presence of multiple obstacles are an interesting benchmark to validate the performance of the GP-MPC model.

The results of the test, shown in 4.10, highlight how this implementation is able to easily handle more complex scenarios, without violating the solving time constraint nor degrading the performance with respect to the previous tests. In fact, the average time required by the solver to compute the optimal control action is 1.436 s, which is in line with the previous setting. In contrast, the `CasADi` version of the controller required an average of 71.843 s to deliver the input to the controller. Not only this timing is considerably greater than its previous performances, but it is also well above the maximum solving time allowed, rendering the performed computations virtually useless, as they cannot guarantee real-time feasibility.

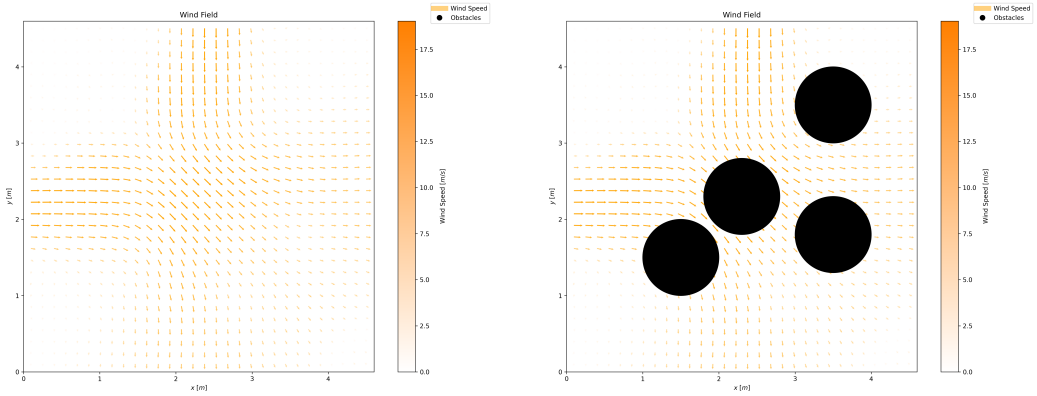

Figure 4.9: More complex scenario on which the GP-MPC framework has been tested. On the left, the original wind field, based on real-world measurements. On the right, the wind field used in the simulation, with the addition of obstacles.

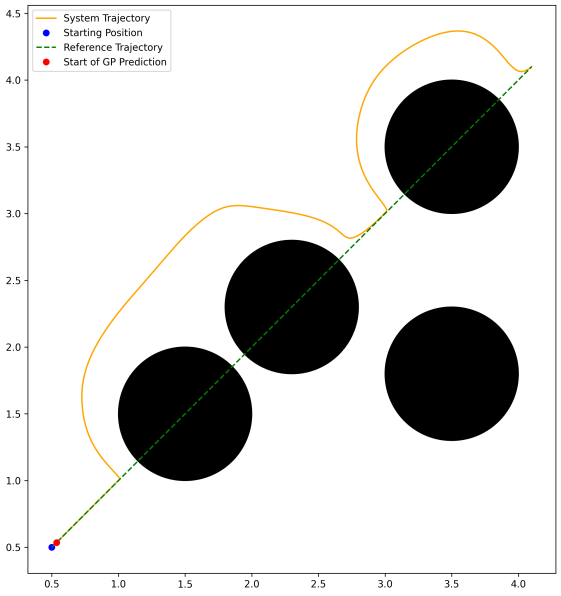

Figure 4.10: Simulation results for a more complex scenario using the `acados` framework to implement the MPC.

4.2.6 Effects of the Window Size on the Control Performance

Considering the implementation of the GP model described in 1, it is possible to see how the window size M of points used to make the predictions is a key components in the parameters supplied to the algorithm. To study the effects this parameter has on the complete GP-MPC, a test on the lemniscate trajectory using the wind field without obstacle has been performed. The window size has been varied between 5 and 50 points,

and the simulation results are shown in 4.11.

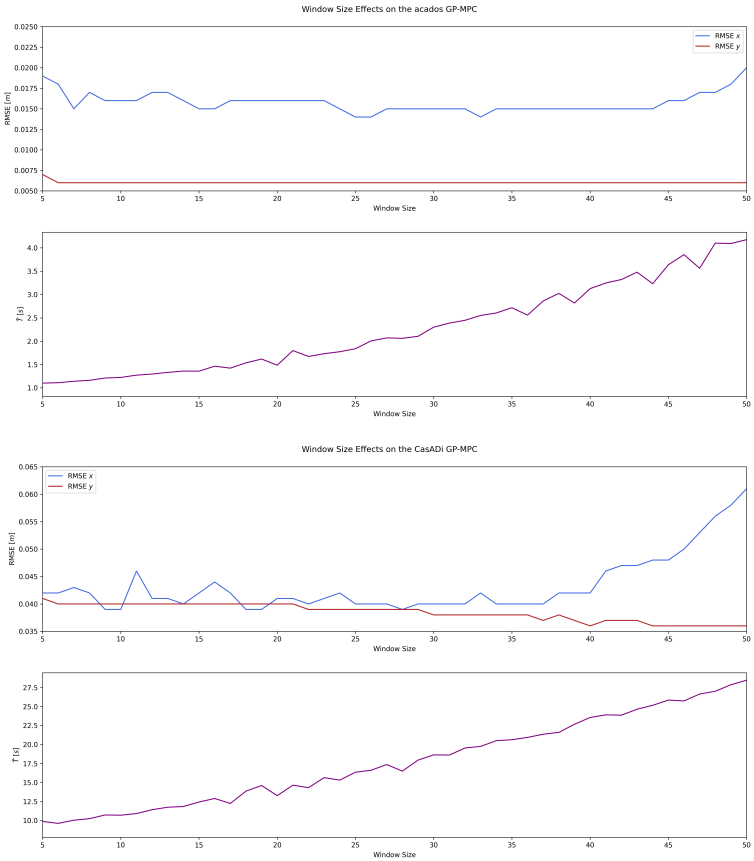

Figure 4.11: Effects on the control action when varying the window size, when tested on the lemniscate trajectory with no obstacles. Above, the results for the `acados` framework, below the results for the `CasADi` implementation.

Looking at the graphs, it is possible to see how there is a trend in the evolution of both frameworks RMSE. In fact, the accuracy of the trajectory tracking tends to decrease when increasing the windows size, until a certain value, after which the model starts performing worse. This happens because the model uses points that are far away from the current position, as inputs to make the predictions, thus yielding a less precise wind estimate, as it accounts for data not necessarily correlated with the current drone location.

At the same time, a more evident trend is visible for the solving time evolution. This correlation is more straightforward, as including more points in the model requires an higher time to both invert the covariance matrix and perform the required operations inside the solver.

Chapter 5

Conclusions and Future Work

This Thesis is concluded by summarizing the key findings and contributions of this work and providing a summary of the research outcomes and conclusions. Additionally, this chapter identifies limitations and areas for improvement, serving as a foundation for future research.

5.1 Conclusions

This research shows how Gaussian Processes can effectively model wind disturbances in unknown environments. The online implementation of the framework, discussed in this worked, proved to be able to handle complex scenarios, taking into account twisting wind fields and complicated obstacle patterns, while outperforming classical MPC implementations.

The inclusion of the GP model inside the MPC algorithm proved to be effective both in improving the control performance, in terms of trajectory tracking, and in guaranteeing a safe flight in the presence of obstacles, via uncertainty propagation and chance-constrained optimization. These features, along with the online adaptability of the model, given by the herein presented GP implementation, allow for the controller to easily adapt to space-varying wind fields in unknown environments.

Nevertheless, some considerations are in order to qualify this work as a starting base for future research. The first limitations is related to the simplifying assumptions made in this Thesis, which could affect the the performance when tested in the real world. The simple time-invariant wind fields, and the use of a wind model that only affects the dynamics of the drone on the x and y axis, are not well representative of a real-world setting. Regarding the implementation itself, the use of GP, as presented in this work, might hinder the effectiveness of the control action, due to the slowness in the computations when the number of data increases significantly.

In conclusion, the work here presented serves as a solid starting base for future researches on GP for wind estimation and their implementation in an MPC framework. The potential of this approach to improve safety during drone flight is significant, allowing for the possibility of online use and exploration of unknown environments while

guaranteeing real-time feasibility. Presenting both advantages and disadvantages of this approach, along with experiments and implementative considerations, this Thesis lays the ground for future works on the topic, suggesting, in the next section, improvements and future research objectives.

5.2 Future Work

This section provides a list of recommendations for future works stemming from this Thesis.

Test in a Real World Scenario

The first recommendation for future research is to implement the GP-MPC on a real drone, and test its performance on a real-world setting. This topic comes with several challenges and requires accounting for various factors that might hinder the effectiveness of the model. The main challenge is related to the quadrotor itself; using a physical drone means having to take into account various factors related to the real world system. The quadrotor is not necessarily behaving as expected, due to model mismatch and intrinsic stochastic factors. The task of testing on a real-time environment, thus, mainly consists in developing a model, and a related MPC, capable of providing a good performance even in the presence of imprecise model dynamics.

Consider Time-Varying Wind Fields

Being able to provide a sufficiently accurate drone model, or developing a control action able to compensate for the mismatch, is a complex challenge per se, but in a real-world environment, the wind disturbances affecting the flight are more complex than what is representable in a simple simulation, and characterized by time-varying effects. To this end, it becomes necessary to test the system considering time-varying wind field, and taking into account the complexity that such settings can generate. On a simpler level, the artificially generated wind fields presented in 4.1.1 could be a good starting point to develop more complex and time-varying scenarios, though they lack turbulent effects and behaviors associated with real phenomenons. To this end, two approaches can be taken into considerations: the first one is to use CFD to generate a reasonably functional wind field, with the possibility to include obstacles and setting up more complex environments; the second viable approach is to fly the drone directly in the real word, where the wind can be assumed to be non-stationary. Both these approaches have their advantages and disadvantages, and careful work and considerations have to be made when deciding the approach.

Improving the GP-MPC Framework

A final note should be made on the improvement of the GP-MPC model. Even though it guarantees real-time feasibility, the current implementation bottleneck is the inversion of the covariance matrix, which, since the data inside changes every MPC call, is performed

every time. An iterative method should be taken into consideration, to avoid computing the inverse every iteration, easing the computational cost of the controller. Other considerations could also be made to speed up the `CasADi` implementation, but, with regards to this work, they are of secondary importance.

Bibliography

Kostas Alexis, Christos Papachristos, Roland Siegwart, and Anthony Tzes. Robust model predictive flight control of unmanned rotorcrafts. *Journal of Intelligent & Robotic Systems*, 81:443–469, 2016.

Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019. doi: 10.1007/s12532-018-0139-4.

Dennis Benders, Johannes Köhler, Thijs Niesten, Robert Babuška, Javier Alonso-Mora, and Laura Ferranti. Embedded hierarchical mpc for autonomous navigation. *arXiv preprint arXiv:2406.11506*, 2024.

Gang Cao, Edmund M-K Lai, and Fakhrul Alam. Gaussian process model predictive control of an unmanned quadrotor. *Journal of Intelligent & Robotic Systems*, 88:147–162, 2017.

David Crowe, Raghava Pamula, Hing Yuet Cheung, and Stephan FJ De Wekker. Two supervised machine learning approaches for wind velocity estimation using multi-rotor copter attitude measurements. *Sensors*, 20(19):5638, 2020.

Nuradeen Fethalla, Maarouf Saad, Hannah Michalska, and Jawhar Ghommam. Robust observer-based dynamic sliding mode controller for a quadrotor uav. *IEEE access*, 6: 45846–45859, 2018.

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 *Advances in Neural Information Processing Systems*, 2018.

Agathe Girard, Carl Rasmussen, Joaquin Q Candela, and Roderick Murray-Smith. Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting. *Advances in neural information processing systems*, 15, 2002.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.

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.

Lukas Hewing, Juraj Kabzan, and Melanie N Zeilinger. Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28(6):2736–2743, 2019.

Haitao Liu, Jianfei Cai, and Yew-Soon Ong. Remarks on multi-output gaussian process regression. *Knowledge-Based Systems*, 144:102–121, 2018.

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.

Omid Mofid, Saleh Mobayen, Chunwei Zhang, and Balasubramanian Esakki. Desired tracking of delayed quadrotor uav under model uncertainty and wind disturbance using adaptive super-twisting terminal sliding mode control. *ISA transactions*, 123:455–471, 2022.

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.

Gabriele Perozzi, Denis Efimov, Jean-Marc Biannic, and Laurent Planckaert. Trajectory tracking for a quadrotor under wind perturbations: sliding mode control with state-dependent gains. *Journal of the Franklin Institute*, 355(12):4809–4838, 2018.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

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

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

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.

Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006.

Zhewen Xing, Youmin Zhang, and Chun-Yi Su. Active wind rejection control for a quadrotor uav against unknown winds. *IEEE Transactions on Aerospace and Electronic Systems*, 59(6):8956–8968, 2023. doi: 10.1109/TAES.2023.3315254.

Shiyi Yang, Nan Wei, Soo Jeon, Ricardo Bencatel, and Anouck Girard. Real-time optimal path planning and wind estimation using gaussian process regression for precision airdrop. In *2017 American control conference (ACC)*, pages 2582–2587. IEEE, 2017.

Steven Zimmerman, Miayan Yeremi, Ryozo Nagamune, and Steven Rogak. Wind estimation by multirotor dynamic state measurement and machine learning models. *Measurement*, 198:111331, 2022.

Glossary

GP Gaussian Process. 2, 10–13, 15–18, 21, 23, 24, 26–28, 38, 40, 41, 43, 45

GP-MPC Gaussian Process-complemented Model Predictive Control. 12, 26–28, 31, 34, 35, 37–40, 42, 43, 46

GPR Gaussian Process Regression. 2, 10, 11, 19, 28

ML Machine Learning. 10, 11, 23

MPC Model Predictive Control. 2, 10–12, 21–28, 31, 34, 38, 45, 46

PSD Positive Semi-Definite. 15

RBF Radial Basis Function. 15, 34

RK-4 Runge-Kutta 4. 23

RMSE Root Mean Square Error. 35, 37, 44

SE Squared Exponential. 16

SVGP Stochastic Variational Gaussian Process. 17