Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Control Systems Engineering

Learning-based Nonlinear MPC for Quadrotor Control

Master Candidate

Filippo Simonetti

Student ID 2055715

Supervisor

Prof. Mattia Bruschetta

University of Padova

Co-supervisor

Prof. Angelo Cenedese

University of Padova

Academic Year
2022/2023

To my parents Letizia and Fabio, and my family,
the roots of my accomplishments;
to Marta, who've supported me in every step of the way;
to Tommaso, a great help for both my humor and my intuition;
to my colleague and friend Alvise;
and to all my dearest ones, for their unceasing support and constant belief.

Abstract

This work aims at investigate the application of different learning based techniques for the enhancement of the Nonlinear Model Predictive Control (NMPC) framework, in the context of trajectory control for a quadrotor unmanned aerial vehicle (UAV). In particular, a gaussian process regression technique and a neural network approach are both taken into account in order to improve the knowledge of the model that constitutes the basis of the effectiveness of the NMPC.

Sommario

Questa tesi si propone di indagare l'applicazione di diverse tecniche learning-based per il miglioramento del NMPC di un quadrirotore. In particolare, una tecnica di regressione con Gaussian Process e un approccio basato su Neural Networks sono utilizzati per migliorare la conoscenza del modello predittivo che costituisce la base della efficacia del NMPC.

Contents

List of Figures

List of Tables

List of Acronyms

UAV	Unmanned Aerial Vehicles
MPC	Model Predictive Control
NMPC	Nonlinear Model Predictive Control
LQR	Linear Quadratic Regulator
RHC	Receding Horizon Control
ARE	Algebraic Riccati Equation
ODE	Ordinary Differential Equations
PDE	Partial Differential Equations
NLP	Nonlinear Programming
QP	Quadratic Programming
SQP	Sequential Quadratic Programming
IPM	Interior Point Method
KKT	Karush-Kuhn-Tucker
RTI	Real-Time Iteration
ML	Machine Learning
GP	Gaussian Process
SE	Squared Exponential

NN	Neural Network
Lb-MPC	Learning-based Model Predictive Control
Lb-NMPC	Learning-based Nonlinear Model Predictive Control
GP-NMPC	Gaussian Process based Nonlinear Model Predictive Control
NN-NMPC	Neural Network based Nonlinear Model Predictive Control
MSE	Mean Squared Error
ReLU	Rectified Linear Unit
LReLU	Leaky ReLU
ELU	Exponential-Linear Unit
API	Application Program Interface
MEX	MATLAB Executable
CPU	Central Processing Unit
RAM	Random Access Memory
CPT	Computational Time
VFE	Variational Free Energy
i.e.	id est
w.r.t.	with respect to
e.g.	exempli gratia

1 Introduction

1.1 Thesis introduction

In recent years, Unmanned Aerial Vehicles (UAV) have achieved great popularity in a number of fields of use, such as logistics, agriculture, security and rescuing to name a few. In particular, quadrotor platforms represent an interesting topic in robotics research and applications, thanks to their low cost and high maneuverability. These mobile platforms, alongside said desirable features, present also significant catches, including underactuation, nonlinearities and bounded control inputs. Such characteristics prevent classical control approaches from exploiting the capabilities offered by these drones, as they typically relay on model linearizations and related small-angle-like assumptions [37].
The NMPC framework lends itself well to the control of nonlinear constrained systems as quadrotors, as it allows to rely on an accurate model of the controlled system to define an high performance control law, while systematically handling the system constraints. Such control technique, which was originally adopted mainly for the control of slow-dynamics systems, has recently benefited from the increase of computational power and from the development of efficient and optimized toolbox. Specifically, the University of Padova developed in the past years an open-source MATLAB toolbox called MATMPC [13], which allows a real-time implementation of the NMPC algorithm. Thanks to these advancements, NMPC offers now a promising approach for the control of fast-dynamics systems as quadrotors. Indeed, NMPC implementations have

been widely adopted on quadrotors to solve position control tasks [10], [37], [48] or to tackle specific critical maneuvers [4], [44], leading to satisfactory results. However, NMPC performance are highly dependent on the quality of the considered model. For this reason, whenever the system presents unpredicted dynamical behaviours, introduced by manufacturing defects, platform customization, damaged mechanical components or simply due to an inaccurate model, the control performance could easily deteriorate. To overcome this limitation, different research paths have been explored in recent years, both in the fields of adaptive control and of learning-based control. Among the various adaptive control methods, $\mathcal{L}_1$ adaptive control has been widely adopted, as it grants fast adaptation and robustness [25], [41], [6]. Nonetheless, despite its desirable properties, this approach is prone to constraint violations when applied in agile control tasks [15], [35]. On the other hand, great efforts have been devoted to the development of the Learning-based Nonlinear Model Predictive Control (Lb-NMPC) framework, namely a research field dedicated to the implementation of learning techniques into the NMPC control scheme [31]. In particular, learning dynamics approaches are extensively used to replace or improve the predictive model of the controller.
In this context, Gaussian Process (GP) regression has resulted remarkably effective in learning the residual errors between true and predicted dynamics. Such data-driven prediction error models are suitable for the definition of a grey-box description of the controlled system. GP-based models have been successfully implemented in NMPC schemes and consequently used for the control of quadrotor platforms in presence of different kind of model uncertainties [50], [36]. These models suffer from scalability problems and present huge computational complexity. However, said limitation can be effectively countered thanks to sparse GP approximations [2].
Recently, a significant research effort has been dedicated to the adoption of Neural Network (NN) regression models in NMPC implementations, as an alternative to the aforementioned GP approach. These learning models have proven effective in learning complex dynamics and have been adopted for the development of different data-driven NMPC approaches for quadrotor platforms [47], [46], [11]. NN-based NMPCs present lower computational burden if compared with the GP-based methods, yet their implementation can be troublesome, as NN learning models are prone to overfittig and suffer from poor interpretability.
In order to deepen the understanding of the tradeoff problem offered by the

choice between GP and NN paths for the Lb-NMPC implementation, this thesis is focused on the development of both approaches for the position control of a quadrotor platform. GP and NN models are here adopted to derive a correction law for the NMPC continuous time prediction model, based on direct measurements of the residual error between true and predicted acceleration of the system. Following the intuition presented in [39] for the development of said GP-based control, a GP-NMPC and a novel NN-NMPC algorithm are implemented and tested in the MATLAB and Simulink environment, thanks to dedicated modifications to the MATMPC framework. In this scenario, GP-based and NN-based Lb-NMPC performances are compared with the ones of a traditional NMPC, in order to test the beneficial impact of the considered data-drive techniques. Moreover, as both learning-based approaches are implemented with reference to the same problem, this work offers a fair comparison between these two innovative tools.

1.2 Thesis structure

This thesis is structured as follows:

- Chapter 2 presents a model of the considered quadrotor platform. The dynamics described by this model will be adopted both for the simulation of the quadrotor platform in the Simulink environment and for the definition of the discussed Nonlinear Model Predictive Control approaches;

- Chapter 3 introduces the MPC framework, starting from a discussion on optimal control and its application in the *receding horizon* control algorithm. Subsequently, specifics on the implementation of a NMPC approach for the control of a quadrotor platform are here presented;

- Chapter 4 is devoted to the introduction of the Lb-NMPC approach. First, the control algorithm and the associated learning problem are presented. Hence, the GP and the NN regression frameworks are discussed, with particular focus on the application of interest;

- Chapter 5 provides a description of the simulation environment adopted to test the considered control approaches, followed by a commented report of the simulation results;

- Chapter 6 is dedicated to a commented summary of the thesis results. A suggestion on future works and development on the considered research themes is finally provided.

2 Agent Modeling

All the control algorithms described in this thesis are discussed and developed with reference to a quadrotor platform. This chapter provides a mathematical model for the description of this aerial vehicle.

2.1 Quadrotor mathematical model

The model of interest of this work is the same described and adopted in [4] by Beniamino Pozzan, Badr Elaamery and Angelo Cenedese. Let $\mathfrak{F}_{\mathfrak{w}}$ denote the common world frame and $\mathfrak{F}_{\mathfrak{b}}$ the quadrotor body frame, whose origin coincides with the center of mass of the UAV. Let $\boldsymbol{p} \in \mathbb{R}^3$ and $\boldsymbol{v} \in \mathbb{R}^3$ denote respectively the position and the linear velocity of the quadrotor in the world frame $\mathfrak{F}_{\mathfrak{w}}$. Moreover, the quadrotor attitude is described by means of a unitary quaternion representation: $\boldsymbol{q} \in \mathbb{S}^3$ represents the pose of $\mathfrak{F}_{\mathfrak{b}}$ with respect to $\mathfrak{F}_{\mathfrak{w}}$, while $\boldsymbol{w} \in \mathbb{R}^3$ represents the angular velocity of the body frame $\mathfrak{F}_{\mathfrak{b}}$ with respect to $\mathfrak{F}_{\mathfrak{w}}$.

Each propeller of the quadrotor platform has rotation axis parallel to $\boldsymbol{e}_z = [0, 0, 1]^T$ w.r.t. $\mathfrak{F}_{\mathfrak{b}}$. As for the most common UAV stuctures, half of the propellers performs a clockwise rotation, while the other half performs a counter-clockwise rotation. The quantities Ω_i with $i = 1, 2, 3, 4$ indicate the propellers spinning rates. A representation of $\mathfrak{F}_{\mathfrak{w}}$ and $\mathfrak{F}_{\mathfrak{b}}$ and of the drone configuration can be found in Figure 2.1. Each propeller, thanks to the blade configuration, applies a thrust force $f_i = c_t\Omega_i^2$ along $\boldsymbol{e}_z$ and a drag torque $\tau_i = c_d\Omega_i^2$ with opposite direction to the angular velocity of the corresponding propeller. The quantities c_t and c_d

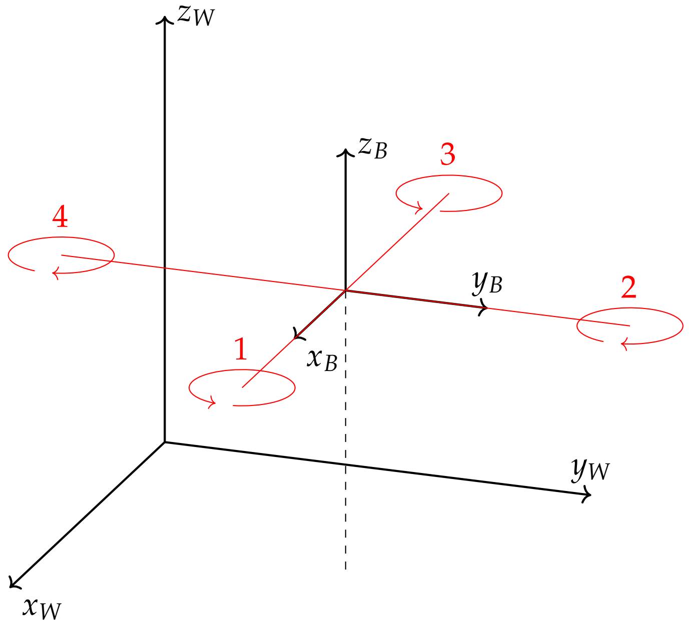


Figure 2.1: Drone schematic representation with reference to the world frame and the body frame.

denote respectively thrust and drag coefficients of the quadrotor. Overall, a total thrust force of $f_z = \sum_{i=1}^{4} c_t \Omega_i^2$ and a total steering moment $\boldsymbol{\tau} = \sum_{i=1}^{4}(c_t \boldsymbol{r}_i \times \boldsymbol{e}_z + c_d \boldsymbol{e}_z)\Omega_i^2$ are applied to the UAV, where $\boldsymbol{r}_i$ is the position of the i-th propeller with respect to $\mathfrak{F}_\mathfrak{b}$. The complete quadrotor model adopted in this work can therefore be described as follows:

$$\begin{aligned} \dot{\boldsymbol{p}} &= \boldsymbol{v} \\ \dot{\boldsymbol{v}} &= \frac{1}{m} f_z \boldsymbol{R}(\boldsymbol{q})\, \boldsymbol{e}_3 - \boldsymbol{g} \\ \dot{\boldsymbol{q}} &= \frac{1}{2} \boldsymbol{q} \circ \boldsymbol{\omega}^+ \\ \dot{\boldsymbol{\omega}} &= \boldsymbol{J}^{-1} (\boldsymbol{\tau} - \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega}) \end{aligned} \tag{2.1}$$

where $m > 0$ is the quadrotor mass, $\boldsymbol{R}(\boldsymbol{q}) \in SO(3)$ is the rotation matrix associated to $\boldsymbol{q} \in \mathbb{S}^3$, $\boldsymbol{g} = [0, 0, g]^T$ is the gravitational acceleration expressed in $\mathfrak{F}_\mathfrak{w}$, $\boldsymbol{J} \in \mathbb{R}^{3\times3}$ is the inertia matrix of the quadrotor in $\mathfrak{F}_\mathfrak{b}$, which is assumed to be diagonal, and lastly $\boldsymbol{\omega}^+ = [0, \boldsymbol{\omega}^T]^T$. In order to control the system described above, it is assumed that the whole state $\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{v}, \boldsymbol{q}, \boldsymbol{\omega}]^T$ can be measured, while $\boldsymbol{u} = [\Omega_1^2, \Omega_2^2, \Omega_3^2, \Omega_4^2]^T$ is considered as control input.
Finally, Table 2.1 displays the collection of constants used for the nominal quadrotor model.

Name	**Symbol [unit]**	**Value**
Mass	$m\ [kg]$	1.5
Moment of inertia	$\boldsymbol{J}\ [kgm^2]$	$\begin{bmatrix} 0.029 & 0 & 0 \\ 0 & 0.029 & 0 \\ 0 & 0 & 0.055 \end{bmatrix}$
Arm length	$r\ [m]$	0.255
Thrust constant	$c_t\ [N/(rad/s)^2]$	0.06
Drag constant	$c_d\ [Nm/(rad/s)^2]$	$7.8 \cdot 10^{-4}$

Table 2.1: Nominal model constants for the characterization of the considered quadrotor platform.

3 Model Predictive Control

Model Predictive Control (MPC) is a form of control that exploits, at each sampling instant, an estimation of the state of a system and its mathematical model, in order to predict its evolution. This prediction is used to solve an optimal control problem over a finite horizon, leading to a finite control sequence; the first control action in this sequence is then applied to the plant [27].
In this chapter a description of the concepts of Linear Quadratic Regulator (LQR) and Receding Horizon Control (RHC) will be given, leading to the formulation of the Model Predictive Control framework and to its application to the control of a quadrotor platform.

3.1 Linear Quadratic Regulator

In order to introduce the formalization of the Receding Horizon Control first, and of Model Predictive Control later, this section will be dedicated to discuss the infinite horizon optimal control and, more specifically, the Linear Quadratic Regulator.
Considering a generic continuous time system, described as

$$\begin{aligned} \dot{\boldsymbol{x}} &= f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0 \end{aligned} \tag{3.1}$$

where $x, x_0 \in \mathbb{R}^n$ are the state and the initial state of the system, while $u \in \mathbb{R}^m$ is the input, one can define the infinite horizon optimal control as

$$\min_{\boldsymbol{u}(\cdot)} \int_0^\infty V(\boldsymbol{x}(t), \boldsymbol{u}(t))\, dt \tag{3.2}$$

i.e. finding the control sequence $u^*(t)$ that minimizes a given cost function V in the interval $t \in [0, +\infty[$.
In the case of LQR, the problem is restricted to the optimal control of a linear time-invariant system of the form

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0 \end{aligned} \tag{3.3}$$

with a quadratic objective function, generally leading to the following optimization problem

$$\min_{\boldsymbol{u}(\cdot)} \int_0^\infty \boldsymbol{x}^\top(t)\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}^\top(t)\boldsymbol{R}\boldsymbol{u}(t)\, dt \tag{3.4}$$

with symmetric matrices $\boldsymbol{Q} \geq \boldsymbol{0} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{R} \geq \boldsymbol{0} \in \mathbb{R}^{m \times m}$. Notice that, thanks to this assumption on $\boldsymbol{Q}$ and $\boldsymbol{R}$, the objective function is non negative and finite for every $\boldsymbol{u}(\cdot) \in L^2$, therefore the problem is well posed.
Consider now the Algebraic Riccati Equation (ARE)

$$\boldsymbol{A}^\top\boldsymbol{P} + \boldsymbol{Q} + \boldsymbol{P}\boldsymbol{A} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top\boldsymbol{P} = \boldsymbol{0} \tag{3.5}$$

Given $\boldsymbol{Q}^{1/2} \in \mathbb{R}^{n \times n} | \boldsymbol{Q} = \boldsymbol{Q}^{\frac{1}{2}\top}\boldsymbol{Q}^{\frac{1}{2}}$, equation (3.5) has unique solution $\boldsymbol{P}_\infty \geq \boldsymbol{0}$ if and only if $(\boldsymbol{A}, \boldsymbol{B})$ is stabilizable and $(\boldsymbol{A}, \boldsymbol{Q}^{\frac{1}{2}})$ is detectable [20]. In case $(\boldsymbol{A}, \boldsymbol{Q}^{\frac{1}{2}})$ is also observable, then $\boldsymbol{P}_\infty \geq \boldsymbol{0}$.
Defining

$$\boldsymbol{K}_\infty = \boldsymbol{R}^{-1}\boldsymbol{B}^\top\boldsymbol{P}_\infty \tag{3.6}$$

it is possible to outline a solution for problem (3.4) as a feedback control law

$$\boldsymbol{u}^*(t) = -\boldsymbol{K}_\infty\boldsymbol{x}(t) \tag{3.7}$$

Finally, this control law makes the system asymptotically stable if and only if the pair $(\boldsymbol{A}, \boldsymbol{Q}^{\frac{1}{2}})$ is detectable.

3.2 Receding Horizon Control

The Linear Quadratic Regulator has very desirable properties: an explicit formula for a stabilizing feedback control law is provided and the existence and uniqueness of a solution are guaranteed under mild and reasonable assumptions. However, the LQR may fail in presence of input saturations or other constraints on either the state or the control variables.
A first effort in order to provide an optimal control law that could include the effects of saturation was carried out by Pontryagin and his colleagues, leading to the Pontryagin's minimum principle, originally presented in [32]. This approach has some drawbacks: computing the solution is difficult and only an open-loop optimal control is provided, missing all the advantages that a closed-loop one could guarantee.
In order to counter the aforementioned computational difficulty, it is possible to discretize the original problem in time and then solve it numerically over a finite time-horizon. This indeed is also quite natural for digital controllers. Moreover, it is important to find a way to implement a closed-loop control law. For this purpose, one can rely on the Receding Horizon Control. At each time instant a discretized version of the optimal control problem is solved over a finite horizon

$$\begin{aligned} \min_{\boldsymbol{u}(\cdot)} \quad & \sum_{k=0}^{N-1} J(\boldsymbol{x}(k), \boldsymbol{u}(k)) + J_N(\boldsymbol{x}(N)) \\ \text{s.t.} \quad & \boldsymbol{x}(t+1) = \Phi(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ & \boldsymbol{x}(0) = \boldsymbol{x}(t_{start}) \\ & \boldsymbol{x}(k) \in \mathcal{X} \\ & \boldsymbol{u}(k) \in \mathcal{U} \end{aligned} \tag{3.8}$$

considering the system's condition at that instant $\boldsymbol{x}(t_{start})$ as initial condition for its evolution prediction, expressed in a dsicretized form as $\Phi(\boldsymbol{x}(t), \boldsymbol{u}(t))$. Here $J(\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot))$ represents the cost function for each time instant, while $J_N(\boldsymbol{x}(N))$ represents the terminal cost. The conditions $\boldsymbol{x}(k) \in \mathcal{X} \subseteq \mathbb{R}^n$ and $\boldsymbol{u}(k) \in \mathcal{U} \subseteq \mathbb{R}^m$ finally represent the adopted constraints over the state and the control input respectively. Once the solution to problem (3.8) has been retrieved, namely the control sequence $\boldsymbol{u}^*(1), \boldsymbol{u}^*(2), ..., \boldsymbol{u}^*(N)$, only the first term is applied to the sys-

tem, then the problem is formulated again starting from $x(0) = x(t_{start} + 1)$. This process is repeated iteratively in order to obtain a control sequence that allows to operate over an arbitrarily long time horizon.
The procedure that characterizes the Receding Horizon Control is summarized by Alg. 1.

Algorithm 1 Receding Horizon Control

```
Require: $N > 0$
  loop
    $x_0 \leftarrow x(t_{start})$
    Compute optimal $u^*(\cdot)$ over a $N$ length horizon
    Apply the first element $u^*(1)$ to the system
    $t_{start} \leftarrow t_{start} + 1$
  end loop
```

3.3 Nonlinear Model Predictive Control

Model Predictive Control is an implementation of RHC, in which the optimal control law is iteratively computed on-line. MPC is basically composed of three key ingredients: a dynamical model of the system, an objective function that has to be minimized and a sets of constraints.
This control technique allows to obtain great performance, while implicitly handling state and control input constraints. However, in order to approach this framework with proper caution, one should also consider some significant drawbacks: since the optimal control problem is formulated over a finite time horizon at every instant, an analysis of stability and optimality of the MPC control law is not straightforward; for the same reason, the optimization problem may also become infeasible at some time step, in case no control sequence is able to satisfy all the constraints. Moreover, each optimal control problem has to be solved in real-time, namely within the sampling interval of the control system, making the applicability of this approach highly dependent on the adopted hardware. Finally, the control performance are deeply influenced by the accuracy of the model [34].
In order to tackle the control of highly nonlinear systems, the MPC approach needs to be extended, introducing the Nonlinear Model Predictive Control framework. A nonlinear dynamical model is adopted in the formulation of the optimal

control problem and to predict the evolution of the considered system. However, this introduces an obvious limitation: while the optimization problems related to a MPC implementation can be solved reliably and quickly, the optimization problems that are needed to be tackled in a NMPC approach are typically hard to solve, especially within real-time computational constraints. Due to the consequent computational complexity of the problem, NMPC was historically adopted for the control of slow varying nonlinear systems. However, thanks to the recent improvements in terms of software and hardware, this technique is now widely adopted for the real-time control of systems with fast dynamics as well.

The remaining part of this section is dedicated to the formulation of the NMPC problem and to a brief discussion of its modern solutions, that are extensively treated and summarized in [12], with focus on the ones adopted for the control of the considered quadrotor platform.

3.3.1 Direct multiple shooting

Given a nonlinear system modeled in continuous time with a set of Ordinary Differential Equations (ODE), generally expressed according to the formulation (3.1), both MPC and NMPC require to solve at each sampling instant a finite horizon optimal control problem

$$\begin{aligned} \min_{\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)} \quad & \int_{t_0}^{t_f} J(\boldsymbol{x}(t),\boldsymbol{u}(t))\,dt + J_N(\boldsymbol{x}(t_f)) \\ \text{s.t.} \quad & \dot{\boldsymbol{x}} = f(\boldsymbol{x}(t),\boldsymbol{u}(t)) \\ & \boldsymbol{x}(t_0) = \hat{\boldsymbol{x}}_0 \\ & r(\boldsymbol{x}(t),\boldsymbol{u}(t)) \leq 0 \end{aligned} \tag{3.9}$$

where the input and state constraints are here expressed in form of inequality constraints $r(\boldsymbol{x}(t),\boldsymbol{u}(t)) \leq 0$. Thus, starting from a state measurement $\hat{\boldsymbol{x}}_0$ the solution of problem (3.9) is given as an optimal trajectory $\{\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)\}$ in the interval $t_f - t_0$.

Potentially, this optimal control problem can be solved by means of three alternative approaches [42]:

- *Dynamic Programming* based on Partial Differential Equations (PDE);

- *Indirect methods* for optimal control, based on Pontryagin's maximum principle;

- *Direct methods* for optimal control, based on the numerical solution of a finite dimensional parametrization of the original continuous-time optimal control problem (3.9). Such parametrization of the problem results in a Nonlinear Programming (NLP) problem.

Among these, direct methods are definitely more popular for NMPC implementations, thanks to their flexibility and to the availability of dedicated numerical optimization solvers for the resulting NLP problem. In particular, *direct multiple shooting* has been found to be particularly effective for NMPC [17].
In direct multiple shooting, the control signal $\boldsymbol{u}(t)$, the state trajectory $\boldsymbol{x}(t)$ and the constraints inequality $r(\boldsymbol{x}(t), \boldsymbol{u}(t)) \leq 0$ are parametrized over N *shooting intervals* that divide the prediction horizon. Specifically, the control signal can be parametrized by means of a piecewise constant representation over the considered shooting intervals $[t_k, t_{k+1}), k = 0, 1, ..., N-1$, namely

$$\boldsymbol{u}(t) = u_k, \quad t \in [t_k, t_{k+1}) \tag{3.10}$$

For the parametrization of state trajectory and constraints inequality, $N+1$ *shooting points* $\{\boldsymbol{s}_0, ...\boldsymbol{s}_N\}$ need to be introduced as additional optimization variables, with each shooting point $\boldsymbol{s}_k$ defined in correspondence of the time grid point t_k. The system dynamics, expressed within each shooting interval with initial condition $\boldsymbol{s}_k$

$$\begin{aligned} \dot{\boldsymbol{x}} &= f(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad t \in [t_k, t_{k+1}) \\ \boldsymbol{x}(t_k) &= \boldsymbol{s}_k \end{aligned} \tag{3.11}$$

can be re-formulated in a discretized form as follows

$$\boldsymbol{s}_{k+1} = \hat{\Phi}(\boldsymbol{s}_k, \boldsymbol{u}_k), \quad k = 0, 1, ..., N-1 \tag{3.12}$$

where $\hat{\Phi}(\cdot)$ is a numerical integrator operator. Similarly, the inequality constraints can also be parametrized on the considered shooting points

$$r(\boldsymbol{s}_k, \boldsymbol{u}_k) \leq 0, \quad k = 0, 1, ..., N-1 \tag{3.13}$$

Thanks to this parametrization, the objective of the optimal control problem (3.9) can be approximated by means of a discrete sum [42]:

$$\sum_{k=0}^{N-1} \int_{t_k}^{t_{k+1}} J(\boldsymbol{s}_k, \boldsymbol{u}_k)\, dt + J_N(\boldsymbol{s}_N) \approx \sum_{k=0}^{N-1} J(\boldsymbol{s}_k, \boldsymbol{u}_k) + J_N(\boldsymbol{s}_N) \tag{3.14}$$

Therefore, thanks to (3.12), (3.10), (3.13) and (3.14), the optimal control problem (3.9) can be formulated in form of a NLP problem as

$$\begin{aligned} \min_{\boldsymbol{s},\boldsymbol{u}} \quad & \sum_{k=0}^{N-1} J(\boldsymbol{s}_k, \boldsymbol{u}_k) + J_N(\boldsymbol{s}_N) \\ \text{s.t.} \quad & \boldsymbol{s}_{k+1} = \hat{\Phi}(\boldsymbol{s}_k, \boldsymbol{u}_k), \quad k = 0, 1, \dots, N-1 \\ & \boldsymbol{s}_0 = \hat{\boldsymbol{x}}_0 \\ & r(\boldsymbol{s}_k, \boldsymbol{u}_k) \le 0, \quad k = 0, 1, \dots, N-1 \end{aligned} \tag{3.15}$$

where the optimal trajectory is expressed in discrete form as $\{\boldsymbol{s} = [\boldsymbol{s}_0^T, \dots, \boldsymbol{s}_N^T]^T, \boldsymbol{u} = [\boldsymbol{u}_0^T, \dots, \boldsymbol{u}_N^T]^T\}$. The first control input of this trajectory is the one that will be actually applied to the system, in a receding horizon fashion.
The popularity of the multiple shooting approach is given by a set of desirable features. First of all, a number of existing softwares are available for the implementation of the numerical integrator used in (3.12) and for the solution of the NLP problem (3.15). For the latter, an initial guess of state and control trajectory over the prediction horizon is needed; however, such guess does not need to be feasible. Moreover, the NLP problem is numerically stable, even in case the considered dynamics is unstable, thus a numerical solution can be achieved anyway. Also, as state and constraints can be decoupled on different shooting intervals, the solution of the NLP problem is well suited for parallel computation implementations.

3.3.2 Sequential quadratic programming

Essentially, the NLP problem (3.15) can be solved by resorting to two classes of optimization algorithms, namely Sequential Quadratic Programming (SQP) and Interior Point Method (IPM) [51]. The NMPC implementations in this thesis rely on the first, as fast and robust solvers are available for this approach. Further

details on the IPM approach and a comparison between the two can be found in [12] and [5].
Consider now a general NLP problem, compactly expressed as

$$\begin{aligned} \min_{z} \quad & a(z) \\ \text{s.t.} \quad & b(z) = 0 \\ & c(z) \leq 0 \end{aligned} \tag{3.16}$$

where $z = [z_0^T, ..., z_{N-1}^T, s_N]^T$, $z_k = [s_k^T, u_k^T]^T$ contains all the optimization variables, while functions $b(\cdot)$ and $c(\cdot)$ express the dynamics and the inequality constraints respectively, since

$$b(z) = \begin{bmatrix} \hat{x}_0 - s_0 \\ \hat{\Phi}(s_0, u_0) - s_1 \\ \vdots \\ \hat{\Phi}(s_{N-1}, u_{N-1}) - s_N \end{bmatrix}, \quad c(z) = \begin{bmatrix} r(s_0, u_0) \\ \vdots \\ r(s_{N-1}, u_{N-1}) \end{bmatrix} \tag{3.17}$$

A necessary condition for the solution of the optimization problem 3.16, given a local minimizer z^*, is the existence of multiplayers λ^*,μ^* that solve the Karush-Kuhn-Tucker (KKT) system [7]

$$\begin{aligned} & \text{Stationarity} \\ & \nabla_z \mathcal{L}(z^*, \lambda^*, \mu^*) := \nabla_z a(z^*) + \nabla_z b(z^*)^T \lambda^* + \nabla_z c(z^*)^T \mu^* = 0 \\ & \text{Primal feasibility} \\ & b(z^*) = 0 \\ & c(z^*) \leq 0 \\ & \text{Dual feasibility} \\ & \mu^* \geq 0 \\ & \text{Complementary slackness} \\ & \mu_k^* c_k(z^*) = 0, \quad k = 0, 1, ..., N-1 \end{aligned} \tag{3.18}$$

where $\mathcal{L}(\cdot)$ is the Lagrangian function. Moreover, strict complementary slackness, linear independence constraint qualification and second order sufficient conditions will be assumed throughout this thesis, since they are reasonable as-

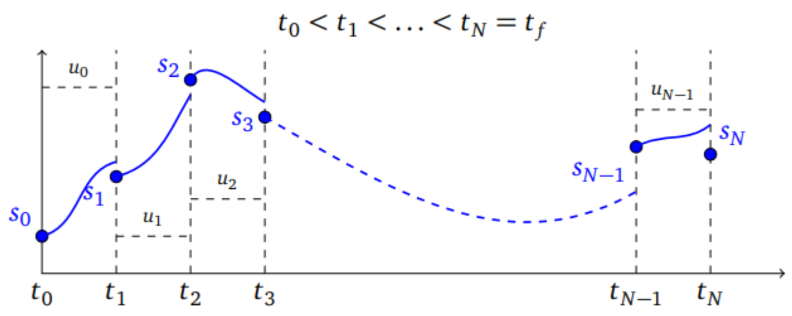
sumptions in most optimization problems [51].
A SQP approach solves the NLP problem of interest iteratively, by adopting at each iteration a local approximation of the objective function and linearized constraints, until the KKT conditions are satisfied with desired accuracy. Specifically, at each iteration a Quadratic Programming (QP) problem is formulated starting from an initial guess value for the multipliers $\boldsymbol{y}^i = [\boldsymbol{z}^{i\,T}, \boldsymbol{\lambda}^{i\,T}, \boldsymbol{\mu}^{i\,T}]^T$, namely

$$\begin{aligned} \min_{\Delta z} \quad & \frac{1}{2}\Delta z^T H(z^i)\Delta z + g(z^i)^T \Delta z \\ \text{s.t.} \quad & b(z^i) + B(z^i)\Delta z = 0 \\ & c(z^i) + C(z^i)\Delta z \leq 0 \end{aligned} \tag{3.19}$$

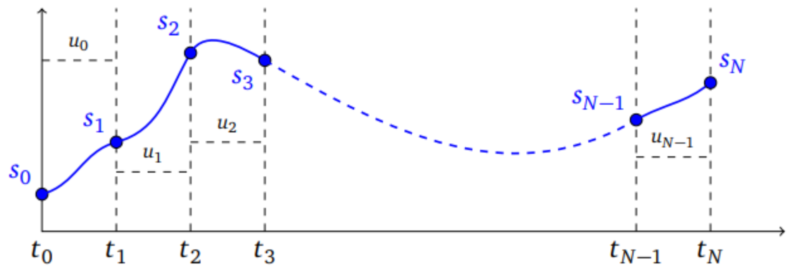
where Δz is called *primal increment*, $H(z^i) := \nabla_z^2 \mathcal{L}(z^i, \boldsymbol{\lambda}^i, \boldsymbol{\mu}^i)$ is the Hessian of the Lagrangian function, while $g(z^i) := \nabla_z a(z^i)$, $B(z^i) := \nabla_z b(z^i)$ and $C(z^i) := \nabla_z c(z^i)$ are the Jacobian matrices of objective function, dynamics constraint and inequality constraint respectively. Notice that this formulation requires the computation of an Hessian matrix at each iteration of the solver algorithm. It is well known that this operation is quite computationally expensive; indeed, it represents one of the principal contributions to the computational burden of this approach.
Problem (3.19) can be solved with popular and reliable QP algorithms, providing an increment value Δz. The multiplier guess is then updated as follows

$$\begin{aligned} \boldsymbol{z}^{i+1} &= \boldsymbol{z}^i + \alpha^i \Delta z \\ \boldsymbol{\lambda}^{i+1} &= (1-\alpha^i)\boldsymbol{\lambda}^i + \alpha^i \boldsymbol{\lambda}^{i+1} \\ \boldsymbol{\mu}^{i+1} &= (1-\alpha^i)\boldsymbol{\mu}^i + \alpha^i \boldsymbol{\mu}^{i+1} \end{aligned} \tag{3.20}$$

where α^i is a *step size* that can be chosen with dedicated globalization strategies [51]. As anticipated, this iterative procedure is repeated until the satisfaction of the KKT system (3.18). In particular, it has been proven that SQP iterations reach the optimum with a quadratic or superlinear convergence rate with a good enough initial guess [23]. It should be noted that with this iterative method the continuity constraint introduced by (3.12) is not satisfied at the beginning of the solving procedure; the trajectory continuity is achieved, with desired accuracy, only when the optimal solution is reached. This behaviour can be observed in Figure 3.1.



(a) Discontinuous trajectory corresponding to the initial guess of the SQP procedure.



(b) Continuous trajectory achieved at the optimum.

Figure 3.1: Figure from [12], showing how the continuity constraint of the state trajectory expressed in a multiple shooting fashion is generally achieved only when the optimum is reached, with iterative NLP solvers.

Delving more into the algorithms that can be used for the solution of the QP subproblems, mainly three classes can be identified [29], [18]:

- *First-order methods* are simple and fast algorithm for QPs with simple constraints. However, the numerical performance of these algorithms could be unreliable, as their convergence rate can highly vary and they can even lead to convergence failure, depending on the characteristics of the problem [18].
- *Interior point methods* do not address the inequality constraints directly. Instead, constraint violations are penalized by means of additional slack variables, that are inserted in the objective function.
- *Active-set methods* solve an easier equality-constrained QP problem in place of the inequality-constrained one, with the assumption that the considered

inequality constraints hold with equality at the optimum. In case the guess is correct, an optimal solution is achieved. Otherwise, in case the guess is not correct, it is updated by adding or removing inequality constraints from the problem, until an optimal solution is reached. In this last category falls a reliable and fast QP solver called qpOASES [19]. This solver has been specifically designed for MPC application and it is the one adopted in this thesis.

Even though the aforementioned method present significant perks for linear MPC, it is crucial to consider that said perks are not always guaranteed for NMPC implementations. In order to tackle the critical issues arising from real-time NMPC applications, further considerations are needed.

3.3.3 Real-Time Iteration

In order to deal with the time restrictions that typically afflict NMPC applications, it is possible to speed up the solution of optimal control problem (3.9) by exploiting some peculiar features of the related NLP formulation. First of all, one should notice that NLP problems given by two consecutive iterations of the NMPC algorithm are very similar. Thus, information that are obtained during the solution of the previous problem could come at hand to tackle the next one. Moreover, for NMPC applications great accuracy in each NLP solution is often unnecessary, hence inexact NLP algorithms are generally sufficient to obtain satisfactory results.
Starting from these intuitions, the Real-Time Iteration (RTI) scheme [17], [16] can be adopted to speed up the NLP solution procedure. The main idea behind this approach is that each NLP problem should be approximately solved with a single SQP iteration, namely by solving a single QP problem, while exploiting information given by previous solutions. Moreover, since a measurement of the initial state $\hat{x}_0$ of optimal control problem (3.9) is not necessary for the formulation of the related QP problem, the real-time optimization process can be divided into a *preparation* phase, where the QP problem is formulated, and a *feedback* phase, where the QP problem is solved right after $\hat{x}_0$ is measured.

Preparation
In this phase, the QP problem needs to be defined without a measurement of the

initial state $\hat{\boldsymbol{x}}_0$. The usual initial state condition is replaced by a linear constraint

$$\Delta \boldsymbol{s}_0 = \hat{\boldsymbol{x}}_0 - \boldsymbol{s}_0 \tag{3.21}$$

In this way, the problem is formulated by accepting a violation of the initial condition constraint. Thanks to the linearity of constraint (3.21), the violation will be immediately corrected, with a single full Newton step that will be performed in the feedback phase. This strategy is called an *initial value embedding*.
In order to speed up the computations, the hessian of the Lagrangian function related to the NLP problem is typically approximated using the Gauss-Newton method, for which only the first derivative of the objective function is required. This approximation grants good numerical performance and excludes the Lagrangian multipliers from the Hessian computation. As a consequence, the QP problem can be initialized with an initial trajectory guess $\boldsymbol{z}^i$. This guess can be obtained thanks to a *warm up* strategy, namely by using the predicted trajectory that can be obtained starting from the past feedback solution as initial guess for the new QP problem.
The QP problem can be written as [12]

$$\begin{aligned}
\min_{\Delta \boldsymbol{s}, \Delta \boldsymbol{u}} \quad & \sum_{k=0}^{N-1} \Big(\frac{1}{2} \begin{bmatrix} \Delta \boldsymbol{s}_k \\ \Delta \boldsymbol{u}_k \end{bmatrix}^T H_k^i \begin{bmatrix} \Delta \boldsymbol{s}_k \\ \Delta \boldsymbol{u}_k \end{bmatrix} + g_k^{i\,T} \begin{bmatrix} \Delta \boldsymbol{s}_k \\ \Delta \boldsymbol{u}_k \end{bmatrix} \Big) + \frac{1}{2} \Delta \boldsymbol{s}_N^T H_N^i \Delta \boldsymbol{s}_N + g_N^{i\,T} \Delta \boldsymbol{s}_N \\
\text{s.t.} \quad & \Delta \boldsymbol{s}_0 = \hat{\boldsymbol{x}}_0 - \boldsymbol{s}_0 \\
& \Delta \boldsymbol{s}_k = A_{k-1}^i \Delta \boldsymbol{s}_{k-1} + B_{k-1}^i \Delta \boldsymbol{u}_{k-1} + d_{k-1}^i, \quad k = 1, \dots, N \\
& C_k^i \begin{bmatrix} \Delta \boldsymbol{s}_k \\ \Delta \boldsymbol{u}_k \end{bmatrix} \le -c_k^i, \quad k = 0, \dots, N-1
\end{aligned} \tag{3.22}$$

where $A_k^i := \frac{\partial \hat{\Phi}}{\partial s}(\boldsymbol{s}_k^i, \boldsymbol{u}_k^i)$, $B_k^i := \frac{\partial \hat{\Phi}}{\partial u}(\boldsymbol{s}_k^i, \boldsymbol{u}_k^i)$ are called sensitivities w.r.t. initial states and controls, respectively. The quantity $d_{k-1}^i := \hat{\Phi}(\boldsymbol{s}_{k-1}^i, \boldsymbol{u}_{k-1}^i) - \boldsymbol{s}_k^i$ denotes the discontinuity gap between a trajectory point at the end of a shooting interval and the point at the beginning of the next one (see Figure 3.1a). Finally, H_k^i and C_k^i are the k-th blocks of Hessian and constraint Jacobian matrices, while g_k^i is the k-th sub-vector of $g(\boldsymbol{z}^i)$.

Feedback
Once the state $\hat{x}_0$ has been measured, QP problem (3.22) can be solved. The resulting increment Δz can be used to perform a full Newton step

$$z^{i+1} = z^i + \Delta z \tag{3.23}$$

The first control input of z^{i+1} is applied to the system, while a new preparation phase begins, adopting the information given by the updated trajectory for the warm start of the next QP problem and for the next initial state guess.

3.4 MATMPC

Given the popularity that the MPC framework has gained over the years, many different open source software and packages were created for implementation and simulation of MPC and NMPC algorithms. Among these, some popular tools are the Model Predictive Control Toolbox [3] for MPC simulations in MATLAB, ACADO [24], that allows the auto-generation of C code for the RTI scheme and CasADi [1], a symbolic differentiation tool that embeds many NLP and QP solvers.
However, the NMPC algorithms used in this thesis are implemented by means of an open source software developed at the University of Padova called MATMPC [13], [12]. This tool provides an easy-to-use and optimized NMPC implementation, with different pre-implemented options in terms of discretization procedures and optimal control problem solvers. Differently from other MPC software, the main advantage offered by MATMPC is that it provides a user-friendly environment both for NMPC implementation and for algorithmic coding and modification. Moreover, while the MATMPC code is mainly written in MATLAB and can be easily embedded in Simulink, its time critical modules are written in the MATLAB API for C and can be compiled into MATLAB Executable (MEX) functions by using MinGW or GCC compilers, on Windows or on Linux and OS X respectively. In this way, MATMPC grants a fast runtime performance, while preserving the readability of the MATLAB environment.
The MATMPC structure can be divided in six modules [12], as shown in Figure 3.2:

1. *Model*: in MATMCP the models are defined in form of continuous-time

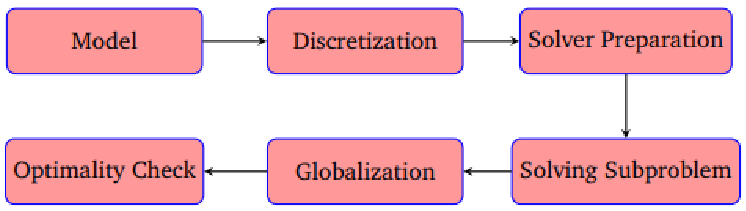


Figure 3.2: Figure from [12] that represents the structure of MATMPC.

dynamical models. The modelling language is based on CasADi. Starting from a model definition, objective function, dynamic equation, and constraints are expressed by means of CasADi functions.

2. *Discretization*: The continuous-time model is discretized with a direct multiple shooting approach. In this phase, a numerical integrator operator is needed to build the NLP problem (3.15). This operator is typically implemented with CasADi symbolic coding.

3. *Solver preparation*: Since MATPMC allows to use different NLP solvers, once the NLP problem is formulated it is necessary to adapt the related information to the chosen solver. Specifically, the solver adopted for this thesis, namely qpOASIS, solves a condensed QP problem. For condensing, full, partial and null-space condensing algorithms are implemented [12].

4. *Solving subproblem*: after the solver preparation phase, the NLP problem can be solved, either by means of existing solvers or with newly implemented algorithms. Both SQP and IPM approaches are supported.

5. *Globalization*: Globalization is used to find local minimum from arbitrary initial points. For this purpose, in MATMPC different efficient line search algorithms are implemented.

6. *Optimality check*: constraints and KKT value, namely the first order derivative of the Lagrangian of the NLP problem, are evaluated, thanks to differentiation toolbox offered by CasADi.

3.5 NMPC FOR QUADROTOR CONTROL

This section is devoted to detail the application of Nonlinear Model Predictive Control to the quadrotor platform of interest. For this purpose, the key components of the NMPC discussed in section 3.3 will be outlined, namely the model, the objective function and the selected constraints.

3.5.1 Model

In order to predict the system evolution and formulate the optimal control problem, the dynamical model expressed by system (2.1) is adopted, together with the nominal model constants reported in Table 2.1.
The adopted model is discretized with a direct multiple shooting approach. In particular, the model dynamics is discretized by means of the Explicit Runge-Kutta algorithm, since it guarantees accurate high-order numerical approximations [45] [8]. More in detail, this operator discretizes the considered model with a fixed shooting time T_s. This parameter, together with the prediction horizon $N = \frac{t_f - t_0}{T_s}$, needs to be carefully selected. Indeed, a small time horizon NT_s doesn't allow to exploit the predictive strategy of the NMPC. On the other hand, a too long time horizon will possibly lead to a control based on incorrect long-term predictions, in case the nominal model isn't sufficiently accurate. Moreover, a small T_s grants greater precision, but a bigger N value will be required to achieve a long enough time horizon, leading to an increased computational burden.

3.5.2 Cost function

Referring to the quantities defined in section 2.1, consider

$$\boldsymbol{h} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{v} \\ \boldsymbol{\omega} \\ \boldsymbol{u} \end{bmatrix} \in \mathbb{R}^{13} \tag{3.24}$$

namely a collection of values of relevance for the formulation of an objective function for the quadrotor control. The time dependency of the considered quantities is here omitted to simplify the notation. The control problem of interest for this thesis will be outlined as a trajectory tracking task, with reference in position $\boldsymbol{p}_{ref}$. The remaining terms will be considered in order to place a cost over the drone linear and angular velocities and the control effort. With this idea, the cost function is chosen as quadratic and can be formulated as follows

$$J(\boldsymbol{h}) = \frac{1}{2}(\boldsymbol{h} - \boldsymbol{h}_{ref})^T \boldsymbol{Q}(\boldsymbol{h} - \boldsymbol{h}_{ref}) \tag{3.25}$$

where $\boldsymbol{h}_{ref} = [\boldsymbol{p}_{ref}, 0, ..., 0]^T \in \mathbb{R}^{13}$ is the reference vector and $\boldsymbol{Q} > 0 \in \mathbb{R}^{13\times13}$ is a diagonal weight matrix. The entries quantities of this latter specify the relative importance of each term of vector $\boldsymbol{h} - \boldsymbol{h}_{ref}$ in the objective function and require extensive tuning to achieve satisfactory performance.
The terminal cost can be defined in a similar fashion, simply ignoring the cost term on the control effort. Thus, given

$$\boldsymbol{h}_N = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} \in \mathbb{R}^9 \tag{3.26}$$

one can write the terminal cost as

$$J_N(\boldsymbol{h}_N) = \frac{1}{2}(\boldsymbol{h}_N - \boldsymbol{h}_{refN})^T \boldsymbol{Q}_N(\boldsymbol{h}_N - \boldsymbol{h}_{refN}) \tag{3.27}$$

with $\boldsymbol{Q}_N \geq 0 \in \mathbb{R}^{9\times9}$ and $\boldsymbol{h}_{refN} = [\boldsymbol{p}_{ref}, 0, ..., 0]^T \in \mathbb{R}^9$.

3.5.3 Constraints

As discussed, one of the main advantages given by the NMPC approach is the systematic handling of state and control input constraint. For the problem of interest, three constraints have been taken into account.
First, a simple constraint on the position has to be considered, since the drone can't fly below the ground level. Denoting the position components of the state as $\boldsymbol{p} = [p_x, p_y, p_z]^T$, one can simply enforce

$$p_z > 0 \tag{3.28}$$

Since the drone pose is represented by means of a quaternion, one has to ensure that the quaternion norm is unitary, namely

$$\|\boldsymbol{q}\| = 1 \tag{3.29}$$

Finally, a control input saturation has to be taken into account, representing a physical limitation on the rotation speed of each rotor. Trivially, one also has to consider

$$0 \leq \boldsymbol{u} \leq \boldsymbol{u}_{max} \tag{3.30}$$

where the entries of vector $\boldsymbol{u}_{max}$ represent the maximum squared rotation speed of each motor of the drone.

3.5.4 Final problem formulation

Combining (3.25), (3.27), (3.28), (3.29) and (3.30) and considering model (2.1), the optimization problem (3.9) that has to be solved at each NMPC iteration can be re-formulated as follows, to tackle a trajectory tracking task for the considered quadrotor platform

$$\begin{aligned}
\min_{\boldsymbol{u}} \quad & \int_{t_0}^{t_f} J(\boldsymbol{h})\, dt + J_N(\boldsymbol{h}_N) \\
\text{s.t.} \quad & \dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \\
& \boldsymbol{x}(t_0) = \hat{\boldsymbol{x}}_0 \\
& p_z > 0 \\
& \|\boldsymbol{q}\| = 1 \\
& 0 \leq \boldsymbol{u} \leq \boldsymbol{u}_{max}
\end{aligned} \tag{3.31}$$

4 Learning-based NMPC

Recent advancements in the field of Machine Learning, as well as the availability of increasing computational capabilities in control systems, have led to a growing interest in data-driven and data-enhanced control techniques. As discussed in chapter 4, the Model Predictive Control highly relies on the availability of a good model of the controlled system. For this reason, this control approach lends itself to exploit the opportunities offered by learning algorithms, in order to improve the knowledge of the system starting from collected data [31].
In this chapter, a learning problem will be outlined in order to exploit experience data for the formulation of an improved model for the MPC prediction. Two suitable regression tools will then be introduced, namely the GP and the NN approaches.

4.1 Learning-based continuous dynamics

In order to exploit the information given by recorded data, the model adopted by the MPC algorithm for the prediction of the system's behaviour and for the optimal problem formulation can be substituted by a law of the form

$$\begin{aligned} \dot{\boldsymbol{x}} &= f(\boldsymbol{x}(t), \boldsymbol{u}(t)) + g(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0 \end{aligned} \tag{4.1}$$

where $f(\boldsymbol{x}(t), \boldsymbol{u}(t))$ represents the nominal dynamics of the system, while $g(\boldsymbol{x}(t), \boldsymbol{u}(t))$ is an additional correction term, learned from data.

Being based both on the physical knowledge of the system and data, formula (4.1) lies in the category of the so called grey-box models. Alternatively, it would be possible to exclusively rely on the learned dynamics, in a black-box fashion. However, the grey-box approach is often preferred to the latter, as it allows to obtain an accurate model from a significantly smaller amount of data [40]. The two methods lead to almost equivalent results with enough data, yet the computational burden of learning algorithms is almost always proportional to the dimension of the considered training data set. In this thesis only the grey-box approach will be considered, even though an extension to the black-box one would be fairly straightforward.
In order to be properly exploited by a MPC algorithm, the data-enhanced model represented in formula (4.1) has then to be detailed for the control problem of interest. In particular, two approaches can be adopted: the learning-based model can be formulated in continuous time and subsequently discretized by a numerical integrator operator, as discussed in paragraph 3.5.1, or it can be considered in a discrete formulation, inferring an expression of the system state at the next discrete time instant thanks to both a discrete nominal model and experience data. The two methods rely on different learning problems. In the continuous formulation a learning-based law is used to describe an acceleration mismatch between the one predicted by the nominal model and the actual acceleration measurements. On the other hand, the discrete approach focuses on learning a similar mismatch in terms of velocities. In practice, the two methods lead to very similar results [39]. Even though the discrete approach is more computationally efficient, the continuous one is way more intuitive and it lends itself to an easier and more interpretable implementation. For this reason, this thesis focuses on the discussion of the continuous learning-based strategy and on its application to the control problem of interest.

Let the state of a dynamical system be formulated as

$$\boldsymbol{x} = [\boldsymbol{\rho}, \dot{\boldsymbol{\rho}}] \tag{4.2}$$

where $\boldsymbol{\rho}$ and $\dot{\boldsymbol{\rho}}$ are respectively the position and the velocity of a physical system. While interacting with the system, state $\boldsymbol{x}_k$ and control input $\boldsymbol{u}_k$ as well as acceleration measurements $\ddot{\boldsymbol{\rho}}_k$ can be acquired for T step times. Using the first two quantities, the nominal acceleration $\widetilde{\ddot{\boldsymbol{\rho}}}_k$ can also be computed, according to the nominal dynamical model of the system of interest, which is given by model

(2.1) for the quadrotor. This data set can then be grouped as follows

$$\begin{aligned} \mathcal{X} &= \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_T\}, \\ \mathcal{U} &= \{\boldsymbol{u}_1, \dots, \boldsymbol{u}_T\}, \\ \ddot{\mathcal{P}} &= \{\ddot{\rho}_1, \dots, \ddot{\rho}_T\}, \\ \widetilde{\ddot{\mathcal{P}}} &= \{\widetilde{\ddot{\rho}}_1, \dots, \widetilde{\ddot{\rho}}_T\} \end{aligned} \tag{4.3}$$

These quantities can be used for the regression of a correction law for the nominal model, adopting $\mathcal{X}, \mathcal{U}$ as learning input and $\ddot{\mathcal{P}}$,$\widetilde{\ddot{\mathcal{P}}}$ as learning target, in order to define a learning-based grey-box model of the form

$$\begin{bmatrix} \widehat{\dot{\rho}}(t) \\ \widehat{\ddot{\rho}}(t) \end{bmatrix} = \begin{bmatrix} \widetilde{\dot{\rho}}(t) \\ \widetilde{\ddot{\rho}}(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \psi_{\ddot{\rho}}(t) \end{bmatrix} \tag{4.4}$$

with $\psi_{\ddot{\rho}}(t)$ being an estimate of the acceleration prediction error. Finally, this model can be used for the definition of each NMPC optimization problem (3.9), in order to be used in a Learning-based Nonlinear Model Predictive Control implementation.

4.2 Gaussian Process

According to the *function-space view* discussed by Rasmussen and Williams [9], a GP can be described as a distribution over functions. More specifically, a Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution. A GP $f(\mathrm{x})$ is completely defined by its mean function $m(\mathrm{x})$ and its covariance or kernel function $k(\mathrm{x}, \mathrm{x}')$, namely

$$\begin{aligned} m(\mathrm{x}) &= \mathbb{E}[f(\mathrm{x})], \\ k(\mathrm{x}, \mathrm{x}') &= \mathbb{E}[(f(\mathrm{x}) - m(\mathrm{x}))(f(\mathrm{x}') - m(\mathrm{x}'))] \end{aligned} \tag{4.5}$$

leading to the following notation

$$f(\mathrm{x}) \sim \mathcal{N}(m(\mathrm{x}), k(\mathrm{x}, \mathrm{x}')) \tag{4.6}$$

Therefore, a crucial aspect of the definition of a GP model for regression is the selection of a mean function, that will be here assumed to be always zero, and of a kernel function, for which the quite popular Squared Exponential kernel will be adopted for any GP model presented in this thesis, since it is generally preferred when no specific knowledge on the modelled function is available

$$k(\mathrm{x}, \mathrm{x}') = s_f^2 \exp -\frac{(\mathrm{x} - \mathrm{x}')\boldsymbol{P}^{-1}(\mathrm{x} - \mathrm{x}')}{2} \tag{4.7}$$

Here, s_f^2 represents the overall GP variance and $\boldsymbol{P}$ is the length-scales diagonal matrix.
In realistic modelling situation it is reasonable to assume that only noisy measurements of the data are available. For this reason, also this model relays on the assumption that the target data are possibly obtained as

$$\boldsymbol{y} = f(\mathrm{x}) + \boldsymbol{e} \tag{4.8}$$

where the $\boldsymbol{e}$ components are sampled from independent Gaussian distributions with variance σ_n^2. Therefore, the hyperparameters of this model can be identified as

- length-scales $[\boldsymbol{P}]_{i,i}$;
- the signal variance σ_f^2;
- the noise variance σ_n^2.

An example of how these hyperparameters influence the model regression properties can be seen in Figure 4.1 for a scalar case. Regression made with matching hyperparameters allows to obtain a good fitting of the training data and good predictions with low variance for domain regions sufficiently close to the training data. Lower length-scales instead are associated with larger flexibility of the regression model, which may lead to a perfect fitting of the training data at the cost of poor generalization. On the contrary, higher length-scales are associated to smoother and slower-varying models, affected by higher noise variance, resulting in a poor fit of the training data.
Starting from the considered training data set, the GP model hyperparameters can be finally tuned by means of empirical methods, such as cross validation, or by maximization of the training data marginal likelihood [9]. In this thesis, this second method will be adopted.

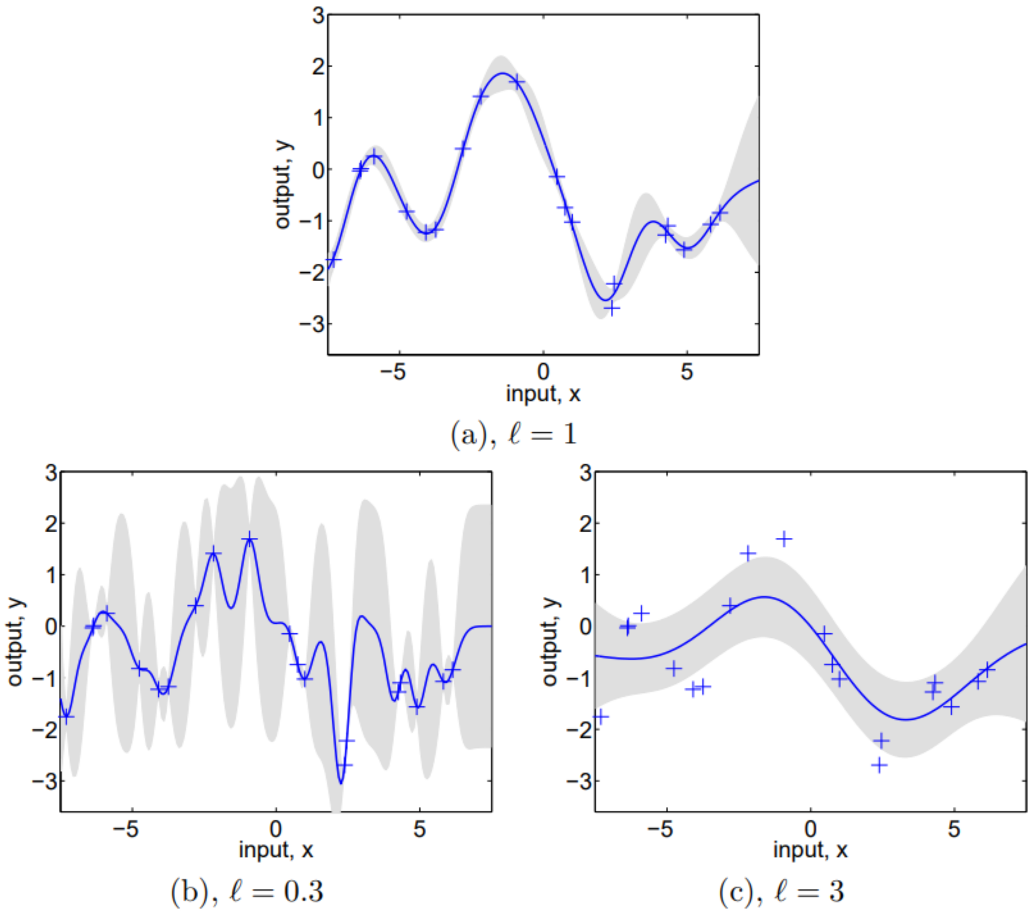


Figure 4.1: Figure from [9], showing the effect of the GP model hyperparameters. The training data, shown as +, have been generated with length-scale $l = 1$, signal variance $\sigma_f^2 = 1$ and noise variance $\sigma_f^2 = 0.1$. Panel (a) shows the GP model prediction ±2 standard deviation with matching hyperparameters, while panels (b) and (c) show the prediction obtained with suboptimal model hyperparameters.

4.2.1 Gaussian Process Regression

Starting from the training data set presented in (4.3), one can model the estimation error for each component of the acceleration vector $\ddot{\boldsymbol{p}}$ with a dedicated independent GP, according to the procedure presented in [39]. The GP's input vector at the k-th time instant, denoted as x_k^{gp}, generally accounts for both the state $\boldsymbol{x}_k$ and the control input value $\boldsymbol{u}_k$, namely $\mathrm{x}_k^{gp} = [\boldsymbol{x}_k^T, \boldsymbol{u}_k^T]^T$. However, the regression procedure could benefit from a dedicated feature selection, that can be performed starting either from some prior knowledge of the system or algorithmically. The corresponding output for the i-th GP is given by $y_k^i = \ddot{p}_k^i - \widetilde{\ddot{p}}_k^i$, where $\ddot{\rho}_k^i$ and $\widetilde{\ddot{\rho}}_k^i$ are the measured and the nominal i-th component of $\ddot{\boldsymbol{\rho}}$ respectively. The following model is thus considered

$$\boldsymbol{y}^i = \begin{bmatrix} y_1^i \\ \vdots \\ y_T^i \end{bmatrix} = \begin{bmatrix} \ddot{\rho}_1^i - \widetilde{\ddot{\rho}}_1^i \\ \vdots \\ \ddot{\rho}_T^i - \widetilde{\ddot{\rho}}_T^i \end{bmatrix} = \begin{bmatrix} \overline{\psi}_{\ddot{\rho}}^i(\mathrm{x}_1^{gp}) \\ \vdots \\ \overline{\psi}_{\ddot{\rho}}^i(\mathrm{x}_T^{gp}) \end{bmatrix} + \begin{bmatrix} e_1^i \\ \vdots \\ e_T^i \end{bmatrix} = \overline{\boldsymbol{\psi}}_{\ddot{\rho}}^i + \boldsymbol{e}^i \tag{4.9}$$

where e_1^i, e_T^i are zero-mean independent Gaussian noises with standard deviation σ_n. According to what previously discussed, $\overline{\boldsymbol{\psi}}_{\ddot{\rho}}^i$ is assumed to be a zero-mean GP, namely $\overline{\boldsymbol{\psi}}_{\ddot{\rho}}^i \sim \mathcal{N}(0, \boldsymbol{K}^i)$, where $\boldsymbol{K}^i$ is a covariance matrix defined by a Squared Exponential kernel function $k^i(\cdot,\cdot)$. Specifically, the entry of $\boldsymbol{K}^i$ at row k and column j is given by $k^i(\mathrm{x}_k^{gp}, \mathrm{x}_j^{gp})$. Moreover, the model hyperparameters are assumed to be tuned, as previously discussed, by maximization of the training data marginal likelihood. Such model will then be used to provide a prediction law for the target of interest, namely a correction acceleration term for the nominal model. Indeed, given a general input x_*^{gp}, the posterior distribution of $\overline{\psi}_{\ddot{\rho}}^i(\mathrm{x}_*^{gp})$ is conveniently Gaussian. Therefore, the maxiumum a posteriori estimator of the target value is given by the posterior mean of $\overline{\psi}_{\ddot{\rho}}^i(\mathrm{x}_*^{gp})$ [9], that can be expressed in closed form by

$$\psi_{\ddot{\rho}}^i(\mathrm{x}_*^{gp}) = \boldsymbol{k}_*^i \boldsymbol{\alpha}^i \tag{4.10}$$

where

$$\begin{aligned} \boldsymbol{k}_*^i &= [k^i(\mathrm{x}_*^{gp}, \mathrm{x}_1^{gp}), \dots, k^i(\mathrm{x}_*^{gp}, \mathrm{x}_T^{gp})] \\ \alpha^i &= (\boldsymbol{K}^i + \sigma_n^2 \mathbb{I})^{-1} \boldsymbol{y}^i \end{aligned}$$

where $\mathbb{I} \in \mathbb{R}^{T \times T}$ is the identity matrix.

4.3 Neural Networks

Feedforward Neural Networks, also called multi-layer perceptrons, are a class of parametric functions that is widely used in order to describe models for the solution of either classification or regression learning problems [26]. These models can be associated to a directed acyclic graph, representing the composition of different functions in a *network* structure, as the name suggests: e.g. given

three functions $g^{(1)}$, $g^{(2)}$, $g^{(3)}$, they can be connected in a chain structure to form $g(\mathrm{x}) = g^{(3)}(g^{(2)}(g^{(1)}(\mathrm{x})))$. Here, $g^{(1)}$ is commonly called *first layer*, $g^{(2)}$ is called *second layer* and so on. In general, the final layer of the network structure is labeled as *output layer*, while the others take the name of *hidden layers*.
These networks are called *neural* as they are loosely inspired by neuroscience. Each hidden layer is typically vector-valued. However, rather than thinking of each layer as representing a single vector-to-vector function, one can also consider it as composed of many parallel vector-to-scalar relations, generally referred to as *neurons*. Representing the output value of the l-th layer as $\boldsymbol{o}^{(l)}$, where $\boldsymbol{o}^{(0)} = \mathrm{x}^{NN}$ is the input vector of the Neural Network, the general function of the j-th neuron composing layer $l+1$ can be expressed as

$$\begin{aligned} a_j^{(l+1)} &= \boldsymbol{w}_j^{(l+1)T}\boldsymbol{o}^{(l)} + b_j^{(l+1)} \\ o_j^{(l+1)} &= \sigma^{(l+1)}(a_j^{(l+1)}) \end{aligned} \tag{4.11}$$

where the weight vector $\boldsymbol{w}_j^{(l+1)}$ and the bias term b_j characterize an affine function, while $\sigma^{(l+1)}(\cdot)$ is a non-linear function that takes the name of activation. Hence, the whole layer can be expressed as

$$\begin{aligned} \boldsymbol{a}^{(l+1)} &= \boldsymbol{W}^{(l+1)}\boldsymbol{o}^{(l)} + \boldsymbol{b}^{(l+1)} \\ \boldsymbol{o}^{(l+1)} &= \sigma^{(l+1)}(\boldsymbol{a}^{(l+1)}) \end{aligned} \tag{4.12}$$

where, with a slight abuse of notation, $\sigma^{(l+1)}$ is here applied to all the elements of its argument vector. Matrix $\boldsymbol{W}^{(l+1)}$ collects all the weights of the layer, having vector $\boldsymbol{w}_j^{(l+1)}$ as its j-th row, while $\boldsymbol{b}^{(l+1)}$ simply contains all the layer biases.
In most cases, a single non-linear function $\sigma(\cdot)$ is applied as activation for all the hidden layers, while the output layer is characterized by a dedicated activation, accordingly to the learning task of interest. For the NN models in this thesis, the output layer will simply apply an identity function, as its the most common choice for regression NN models.

4.3.1 Neural Networks training

Starting from the training data set described in (4.3), it is possible to use Neural Network models in order to derive a correction law for the nominal quadrotor acceleration. In particular, two neural networks can be used, for the correction of the linear acceleration $\dot{\boldsymbol{v}}$ and of the the angular acceleration $\dot{\boldsymbol{\omega}}$ respectively, according to the notation presented in section 2.1.

In practice, the NN weights and biases $\mathcal{W} = \{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}\}_{l=1,\dots,L}$ can be trained by means of gradient descent methods applied over a dedicated loss function $\mathcal{L}(\mathcal{W})$. In order to tackle the regression problems of interest, the loss function choice falls on the Mean Squared Error (MSE) loss, that can be expressed as follows

$$\mathcal{L}(\mathcal{W}, \mathcal{D}) = \frac{1}{2}\sum_{k=1}^{T} \|\boldsymbol{y}_k^{NN} - \boldsymbol{t}_k\|_2^2 \tag{4.13}$$

where $\boldsymbol{y}_k^{NN}$ and $\boldsymbol{t}_k$ are respectively the network prediction and the learning target for the k-th considered data, while $\mathcal{D}$ compactly represent the considered data set. Generally, both state $\boldsymbol{x}_k$ and control input $\boldsymbol{u}_k$ are taken into account for each NN input vector, that is hence defined as $\mathrm{x}_k^{NN} = [\boldsymbol{x}_k^T, \boldsymbol{u}_k^T]^T$. However, also in this case a feature selection procedure could bring significant benefit to the learning results. Moreover, given that the quadrotor acceleration vector can be expressed as $\ddot{\boldsymbol{\rho}} = [\dot{\boldsymbol{v}}^T, \dot{\boldsymbol{\omega}}^T]^T$, the learning targets for the two considered networks can be formulated as $\boldsymbol{t}_k^{lin} = [\ddot{\rho}_k^1 - \widetilde{\ddot{\rho}}_k^1, \dots, \ddot{\rho}_k^3 - \widetilde{\ddot{\rho}}_k^3]^T$ and $\boldsymbol{t}_k^{ang} = [\ddot{\rho}_k^4 - \widetilde{\ddot{\rho}}_k^4, \dots, \ddot{\rho}_k^6 - \widetilde{\ddot{\rho}}_k^6]^T$, where $\ddot{\rho}_k^i$ and $\widetilde{\ddot{\rho}}_k^i$ are respectively the measured and the nominal i-th component of $\ddot{\boldsymbol{\rho}}$ at instant k.
Even though gradient descent methods are widely used for the training of a number of different learning models, Neural Networks presents some important peculiarities. First, the nonlinearity of NN models causes loss functions to be non-convex. For this reason, the adopted iterative gradient descent optimizers only drive the cost function to local minima and results to be significantly sensitive to the initialization of the network parameters. Moreover, given the complexity of a generic Neural Network model, the numerical evaluation of the loss gradient w.r.t. the network parameters can be very computationally expensive. However, the network structure can be exploited in order to achieve an efficient exact gradient evaluation, thanks to the popular *back-propagation* algorithm [14]. In contrast with the *forward propagation* that characterizes the the network prediction, where input x_k^{NN} provides the initial information that propagates up to the hidden layers and finally produces y_k^{NN}, the back-propagation algorithm allows the information given by the cost function to flow backward through the network, in order to compute the overall loss gradient.

The back-propagation procedure relies on the chain rule of calculus to obtain the gradient evaluation: given composite functions $\mathrm{y} = F(\mathrm{x}), \mathrm{x} = G(\mathrm{z})$, namely $\mathrm{y} = F(G(\mathrm{z}))$, it holds that

$$\frac{\partial \mathrm{y}}{\partial \mathrm{z}} = \frac{\partial F}{\partial \mathrm{x}} \frac{\partial G}{\partial \mathrm{z}} \tag{4.14}$$

In order to compute the loss derivative w.r.t. weight $w_{j,i}^{(l)}$ that connects neuron i of layer $l-1$ with neuron j of layer l, one can apply the chain rule to relation (4.11), obtaining

$$\frac{\partial \mathcal{L}}{\partial w_{j,i}^{(l)}} = \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{j,i}^{(l)}} = \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} o_i^{(l-1)} \tag{4.15}$$

While $o_i^{(l-1)}$ can be simply stored during the forward propagation, $\frac{\partial \mathcal{L}}{\partial a_j^{(l)}}$ is generally called error message $\delta_j^{(l)}$ and depends on layers of indices $l' > l$. By applying again the chain rule for the computation of this term, it results that

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} = \sum_h \frac{\partial \mathcal{L}}{\partial a_h^{(l+1)}} \frac{\partial a_h^{(l+1)}}{\partial a_j^{(l)}} = \sum_h \delta_h^{(l+1)} \frac{\partial a_h^{(l+1)}}{\partial a_j^{(l)}} \tag{4.16}$$

hence a weighted sum of the error messages of all the neurons of the following layer is needed in order to compute the error message $\delta_j^{(l)}$. The coefficients $\frac{\partial a_h^{(l+1)}}{\partial a_j^{(l)}}$ can be once again computed by means of the chain rule, recalling formula (4.11):

$$\frac{\partial a_h^{(l+1)}}{\partial a_j^{(l)}} = \frac{\partial a_h^{(l+1)}}{\partial o_j^{(l)}} \frac{\partial o_j^{(l)}}{\partial a_j^{(l)}} \tag{4.17}$$

with

$$\frac{\partial a_h^{(l+1)}}{\partial o_j^{(l)}} = w_{h,j}^{(l+1)} \qquad \frac{\partial o_j^{(l)}}{\partial a_j^{(l)}} = \sigma'(a_j^{(l)})$$

Notice that these terms can be both computed thanks to information that can be stored during the forward propagation as well. Finally, the error message of the output layer can be obtained as

$$\delta_j^{(L)} = \frac{\partial \mathcal{L}}{\partial a_J^{(L)}} = \frac{\partial \mathcal{L}}{\partial o_J^{(L)}} \sigma^{(L)}{}'(a_J^{(L)}) \tag{4.18}$$

Starting from this, the error message, and hence the loss derivative, can be computed backward thanks to the recursive law outlined by equations (4.16) and (4.17):

$$\begin{aligned} \delta_j^{(l)} &= \sigma'(a_j^{(l)}) \sum_h \delta_h^{(l+1)} w_{j,h}^{(l+1)} \\ \frac{\partial \mathcal{L}}{\partial w_{j,i}^{(l)}} &= \delta_j^{(l)} o_i^{(l-1)} \end{aligned} \tag{4.19}$$

Similar considerations can be carried out for the computation of $\frac{\partial \mathcal{L}}{\partial b_j^{(l)}}$, leading to

$$\frac{\partial \mathcal{L}}{\partial b_j^{(l)}} = \delta_j^{(l)} \tag{4.20}$$

4.3.2 L2 Regularization

A central problem in Machine Learning is how to make an algorithm capable of performing well not only on training data, but also on test inputs. The great regression power offered by Neural Network models allows to learn very complex laws, at the cost of making those models prone to overfitting. In order to counter this behaviour, many strategies are explicitly designed to reduce test error, possibly at the expense of an increased training error. These strategies are collectively known as regularization.
Many of these regularization approaches are based on limiting the model regression capacity by adding a parameter norm penalty $\Omega(\boldsymbol{\theta})$ to the loss function

$$\widetilde{\mathcal{L}}(\mathcal{W}; \boldsymbol{\theta}) = \mathcal{L}(\mathcal{W}) + \alpha\Omega(\boldsymbol{\theta}) \tag{4.21}$$

where $\alpha \in [0, \infty[$ is a hyperparameter that weights the relative contribution of the norm penalty $\Omega(\boldsymbol{\theta})$ and of the loss function $\mathcal{L}(\mathcal{W})$. Different kind of parameter norm $\Omega(\boldsymbol{\theta})$ can result in different learning behaviours.
In this thesis, the L^2 norm of the network weights vector $\boldsymbol{w}$ is used as a regularizing term $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$ for the training of the considered NN models. This widely used method, commonly known as weight decay, allows to preserve relatively intact the parameters that contribute significantly to the loss reduction, while irrelevant parameters are decayed away during the learning procedure

[26]. The regularized loss function hence takes the form

$$\widetilde{\mathcal{L}}(\mathcal{W};\boldsymbol{\theta}) = \frac{1}{2}\sum_{k=1}^{T}\|\boldsymbol{y}_k^{NN} - \boldsymbol{t}_k\|_2^2 + \alpha\frac{1}{2}\|\boldsymbol{w}\|_2^2 \tag{4.22}$$

4.3.3 Activation functions

The choice of the hidden layers activation function $\sigma(\cdot)$ plays a crucial role in the design of a Neural Network model. Clearly, a network model composed of linear functions only is able of learning linear functions exclusively. For this reason, activation functions are almost always non linear. Some popular activation functions will be discussed in the following.
First Neural Network used to adopt the Sigmoid function as activation

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{4.23}$$

This function intuitively makes a neuron *active* for $z > 0$ and *inactive* otherwise. It is also differentiable, which is a good property for the application of gradient descend optimization methods. However, its derivative

$$\sigma'(z) = \sigma(z)(1-\sigma(z)) \tag{4.24}$$

is *contractive*, namely its absolute value belongs to $]0,1]$. For this reason, the error message can easily shrink while back-propagating, potentially leading the loss gradient to vanish, hence stopping the training. Moreover, since the function is not zero centered, at each layer a bias is introduced, leading to a phenomenon usually called bias drift that can significantly slow down the training procedure as well.

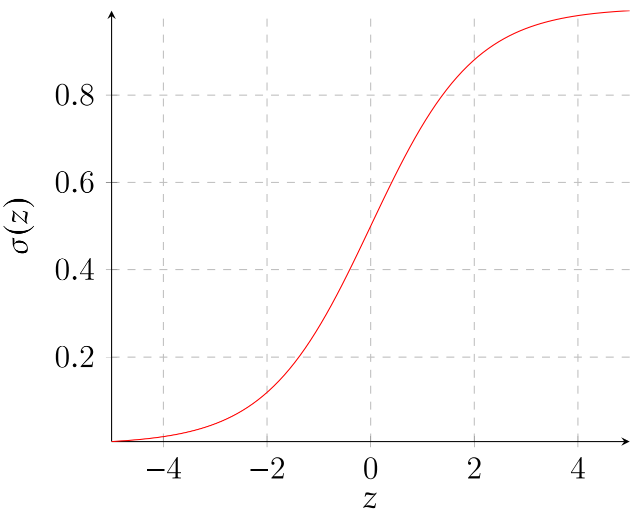


Figure 4.2: Sigmoid function plot.

A common alternative to the Sigmoid function is given by the hyperbolic tangent

$$\sigma(z) = \tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1} \tag{4.25}$$

This activation avoids bias drift, but still presents contractive derivative.

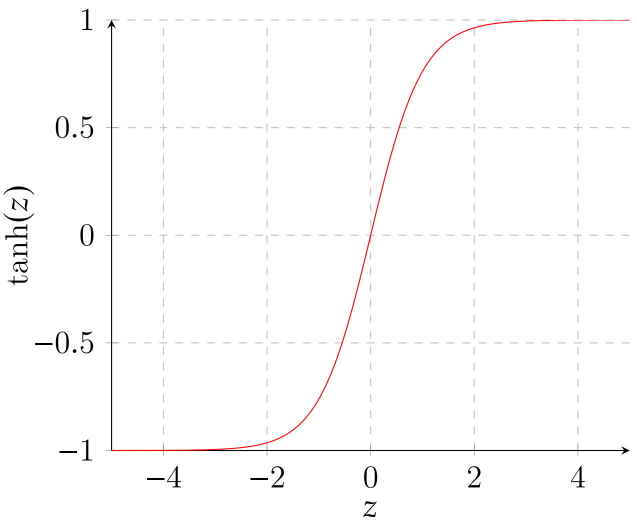


Figure 4.3: Hyperbolic tangent function plot.

Modern Neural Networks often adopt a class of activations commonly called rectified linear units. These activation functions are designed in order to maintain the simplicity of a linear activation while guaranteeing non linearity. A first example of these functions is given by the ReLU activation

$$\sigma(z) = \mathrm{ReLU}(z) = \max(0, z) \tag{4.26}$$

This function significantly speeds up the learning, while partially solving the contractive derivative problem. However, it still presents bias drift. Moreover, negative argument values still present zero gradient, possibly leading some neurons to *die*, namely to stop the gradient based learning procedure for those neurons. Notice also that this function is non differentiable in $z = 0$. It may seems that this invalidates the use of gradient based algorithm. In practice, gradient descent optimizers still perform well for these models.

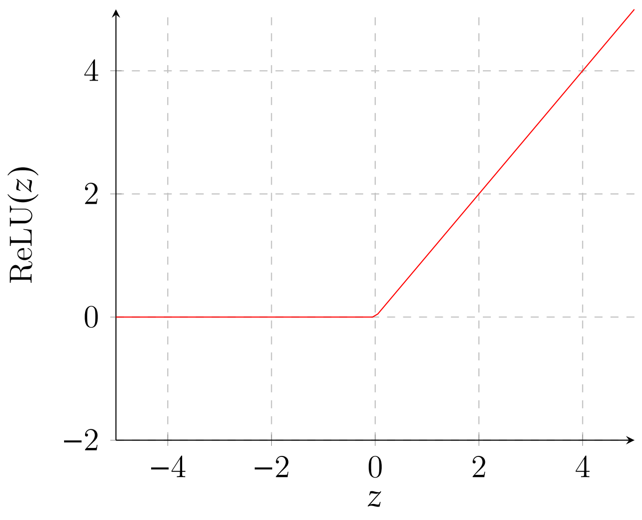


Figure 4.4: ReLU function plot.

In order to solve this issue, the Leaky ReLU (LReLU) function can be introduced

$$\sigma(z) = \mathrm{LReLU}(z) = \begin{cases} z, & \text{if z > 0} \\ \alpha z, & \text{if z < 0} \end{cases} \tag{4.27}$$

with $0 \leq \alpha < 1$. This activation mitigates the dying neuron problem. Notice however that negative values of the argument can however assume arbitrarily negative activation values, since the function is not lower bounded. This is not a desirable property for an activation function. Despite that, Leaky ReLU is still one of the most popular activations.

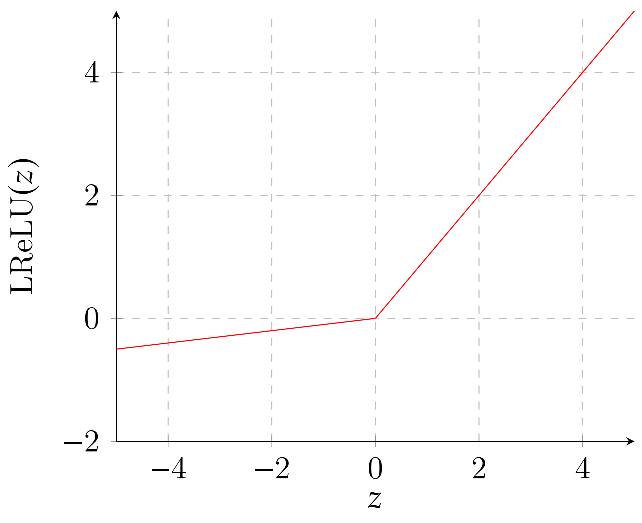


Figure 4.5: LReLU function plot with $\alpha = 0.1$.

Finally, a valid alternative to LReLU is given by the Exponential-Linear Unit (ELU), that asimptotically saturates for large negative arguments, granting more robustness to noise

$$\sigma(z) = \mathrm{LReLU}(z) = \begin{cases} z, & \text{if z > 0} \\ \alpha(e^z - 1), & \text{if z < 0} \end{cases} \tag{4.28}$$

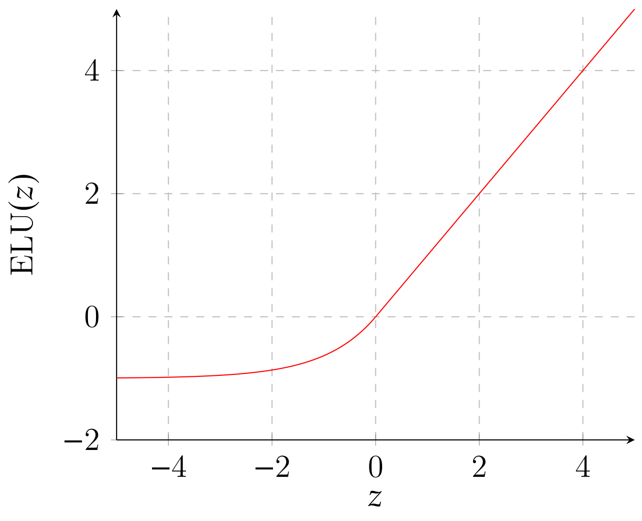


Figure 4.6: ELU function plot with $\alpha = 1$.

However, this last function is more complex to compute and to differentiate than the Leaky ReLU and it can slow down both the learning procedure and the

network prediction on new inputs.
The Leaky ReLU will be adopted as activation function for the NN models in this thesis. However, since it is not possible to predict which activation will give better results for a given learning problem, all the mentioned alternatives shall be generally taken into account also for future new applications.

4.3.4 Weights initialization

As discussed in section 4.3.1, Neural Networks training is possibly influenced by the network parameters initialization. Even though an analytical law for the selection of good initial parameters is not available, it is possible to identify some common sense rules:

- The network weights should be initialized randomly, in order to ensure diversity;
- Large numerical values should be avoided, since they can cause instability and numerical issues in the learning procedure.

One of the most popular weight initialization law is known as Glorot initialization, proposed in [21] by Glorot and Bengio. According to this procedure, the network weights are sampled from an uniform distribution over an interval defined by network input and output size, respectively noted as d and n

$$w_{j,i}^{(l)} \sim \mathcal{U}(-\sqrt{\frac{6}{d+n}}, \sqrt{\frac{6}{d+n}}) \tag{4.29}$$

Alternatively, a common choice is to initialize the network weights by sampling from a zero-mean Gaussian distribution, prioritizing small initialization values

$$w_{j,i}^{(l)} \sim \mathcal{N}(0, \beta) \tag{4.30}$$

Finally, the network bias terms are usually initialized to 0 or, more rarely, to other constant values.

5 Simulations and Results

This chapter provides a description of the simulation environment adopted for the validation of the proposed control methods and a commented report of the simulation results.

5.1 Trajectory tracking problem

The control methods that will be assessed in the following are designed in order to tackle a trajectory tracking task for the quadrotor platform described in chapter 2. The reference trajectory is defined as a timed sequence of L position coordinates $\boldsymbol{p}_{ref}$ and is represented in Figure 5.1. The tracking performance can be reasonably evaluated in terms of position error w.r.t. the reference. Moreover, the average position error

$$\mathrm{Av}_{\mathrm{err}} = \frac{1}{L}\sum_{k=1}^{L} \|\boldsymbol{p}_k - \boldsymbol{p}_{ref\,k}\|_2 \tag{5.1}$$

can be used as a compact indicator of the control performance.

5.2 Simulation setup

All the simulations that will be discussed in the following are carried out in the Simulink environment. Model (2.1) is used for the simulation of the quadrotor dynamics, while the Nonlinear Model Predictive Control approach is adopted

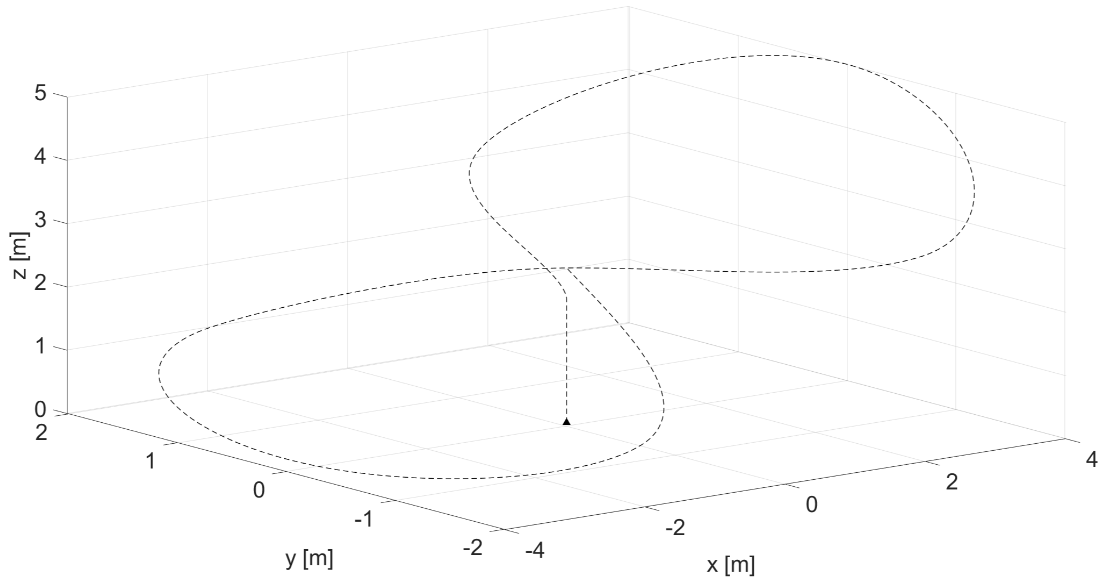


Figure 5.1: Position reference for the trajectory control task of interest. The quadrotor is required to take off and to subsequently perform an eight shaped trajectory, changing its altitude in the process. The whole trajectory requires $40s$ to be completed.

to perform the control task of interest, using a control frequency of $f_c = 20\ Hz$. Specifically, the NMPC is implemented by means of MATMPC, an an open source algorithm developed at the University of Padova that is introduced in section 3.4. The NMPC optimal control problem is discretized with direct multiple shooting, by using a 4-th order Runge-Kutta [30] as numerical integrator. The resulting NLP is addressed with a SQP approach, while qpOASES [19] is used to solve each QP subproblem. The derivatives needed to perform the control optimization are computed with CasADi [1], a popular state of the art differentiation toolbox.
The control scheme adopted in the simulations can be finally observed in Figure 5.2.

5.2.1 NMPC DESIGN AND TUNING

As discussed in section 3.5, the NMPC parameters require a careful tuning in order to grant desirable performance of the controlled system. Specifically, the adopted NMPC algorithm considers cost function (3.25) and terminal cost

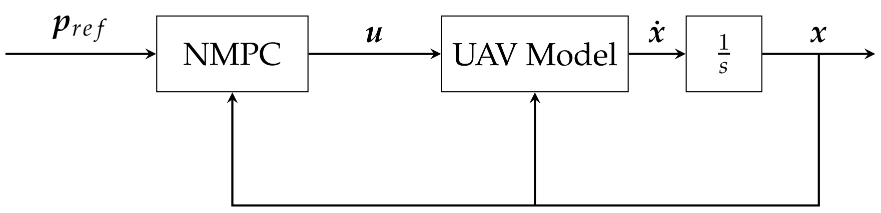


Figure 5.2: Control scheme for the simulation of the adopted control strategies. The NMPC block expoilts the MATMPC code starting from a nominal model of the quadrotor. The $\frac{1}{s}$ block performs a numerical integration, in order to obtain the state value x starting from $\dot{x}$.

(3.27) with weights matrices composed as

$$\begin{aligned} Q &= \text{diag}([\boldsymbol{q}_p, \boldsymbol{q}_v, \boldsymbol{q}_\omega, \boldsymbol{q}_u]^T) \\ Q_N &= \text{diag}([\boldsymbol{q}_p, \boldsymbol{q}_v, \boldsymbol{q}_\omega]^T) \end{aligned} \tag{5.2}$$

where $\boldsymbol{q}_p$ is the position error weights vector, and $\boldsymbol{q}_v$, $\boldsymbol{q}_\omega$ and $\boldsymbol{q}_u$ are respectively the costs vectors for linear velocity, angular velocity and control input. Moreover, the adopted NMPC algorithm performs predictions over an $N = 10$ steps horizon with shooting time of $T_s = 0.1s$, for a total prediction time horizon of $NT_s = 1s$ for each algorithm iteration. Finally, the control input saturation constant is set at $\boldsymbol{u}_{max} = 91\ (rad/s)^2$. All the considered parameters are displayed in Table 5.1.

Name	**Symbol [unit]**	**Value**
Position error weight	$\boldsymbol{q}_p$	$[70, 70, 300]$
Linear velocity weight	$\boldsymbol{q}_v$	$[1, 1, 10]$
Angular velocity weight	$\boldsymbol{q}_\omega$	$[1, 1, 1]$
Control input weight	$\boldsymbol{q}_u$	$[10^{-4}, 10^{-4}, 10^{-4}, 10^{-4}]$
Horizon steps	N	10
Shooting time	$T_s\ [s]$	0.1
Input saturation	$\boldsymbol{u}_{max}\ [(rad/s)^2]$	91

Table 5.1: Parameters of the adopted NMPC algorithm.

5.3 NMPC results

In this section, the results obtained in simulation from the application of the described NMPC control to the considered drone platform are displayed and discussed. Alongside a careful parameter tuning, the NMPC approach performance are deeply influenced by the accuracy of the adopted prediction model. In particular, consider model (2.1) with nominal parameters displayed in Table 2.1 as the NMPC prediction model. Then, in order to assess and subsequently tackle the effects of unpredicted dynamics on the control performance, an ideal case of perfect matching between the NMPC model and the simulated quadrtotor model will be observed and compared with some scenarios of NMPC application in presence of model mismatch.

5.3.1 Matching models

In case the prediction model adopted by the NMPC perfectly matches the quadrotor model used for the simulations, the control should lead to an almost perfect trajectory tracking. Indeed, an average position error of $\mathrm{Av}_{\mathrm{err}} = 7.2\ mm$ is obtained in simulation, while by observing the drone trajectory, showed in Figure 5.3, one can notice a neat overlap with the position reference.
On top of that, working on a machine with an 10-th generation i7 CPU and a

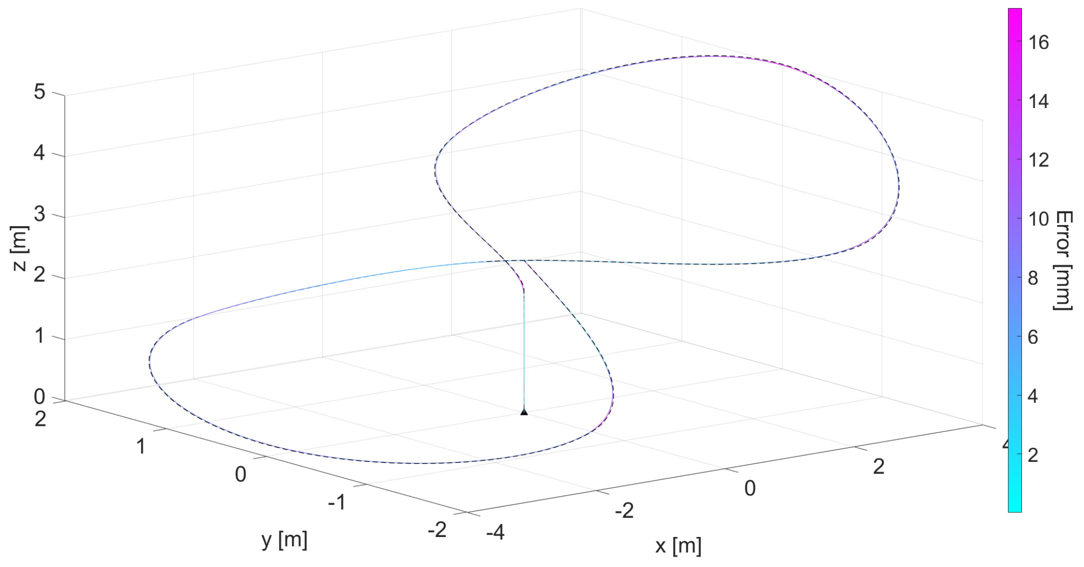


Figure 5.3: Simulation result obtained by applying the NMPC with a perfectly accurate prediction model. As expected, the drone follows with good precision the reference trajectory. The drone trajectory is color coded w.r.t. the tracking error value at each point.

16 GB RAM, the NMPC block performs each iteration of the control algorithm with an average Computational Time (CPT) of $0.3\ ms$, which is compatible with a real-time implementation of the controller.
This scenario, both in terms of average position error and of CPT, will be considered as a benchmark for the following simulations.

5.3.2 Mass mismatch

In real world scenarios, an unknown additional payload could be added to the quadrotor, e.g. for a transport task or in case some new component is added to the platform. As an example, consider an additional payload of 0.25 Kg with respect to the nominal parameter displayed in Table 2.1, corresponding to a 16.7% increase of the drone mass. In such case, the NMPC algorithm can't properly predict the quadrotor behaviour, leading to a position error in the trajectory tracking task, as shown in Figure 5.4. The resulting average position error amounts to $\text{Av}_{\text{err}} = 43.8\ mm$, representing a significant increase in error w.r.t. the ideal case. Even though this performance could still be considered ac-

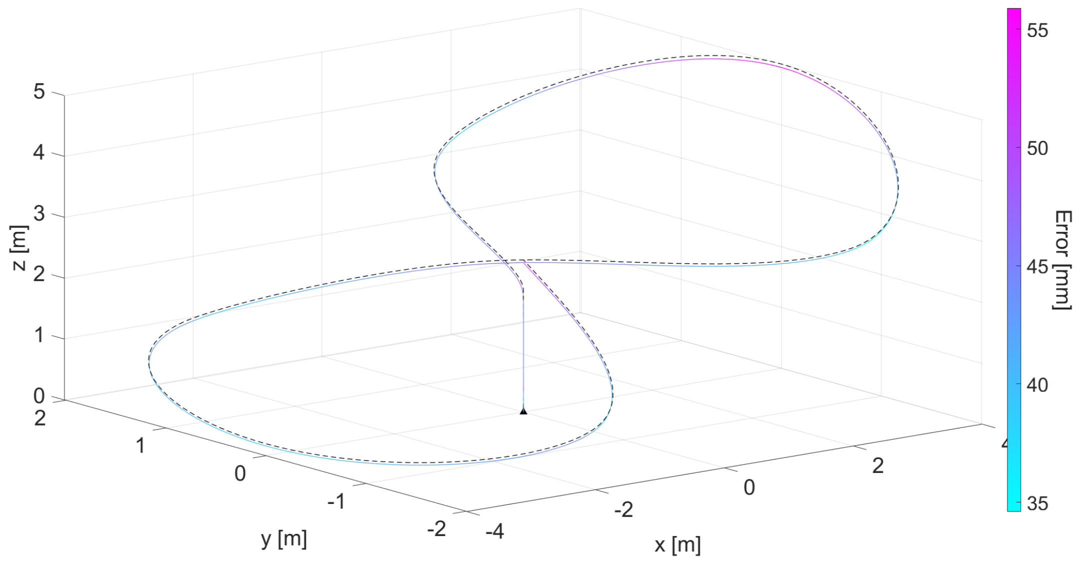


Figure 5.4: Simulation result obtained by applying the NMPC algorithm with nominal model in case of a mass mismatch. The quadrotor performs the tracking task with an altitude offset that becomes more evident during the ascending portion of the trajectory.

ceptable in a real world application, it is clear how even a small linear mismatch between the prediction model and the controlled system dynamics can lead to significant relative deterioration in performance.

5.3.3 Motor rotation speed mismatch

Consider either a mechanical or an electrical malfunction in one of the drone propellers that persistently slows down its rotation of a 10.6% factor, corresponding to a 20% decrease in one of the components of the drone control input $\boldsymbol{u}$. This not only weakens the overall thrust of the platform, but it also brakes the symmetry of the propeller configuration. In this case, the NMPC algorithm leads to an average position error of $\mathrm{Av}_{\mathrm{err}} = 146.9\ mm$. By looking at the resulting simulation trajectory displayed in Figure 5.5, one can indeed notice how the drone is following the reference trajectory with evident position mismatch.

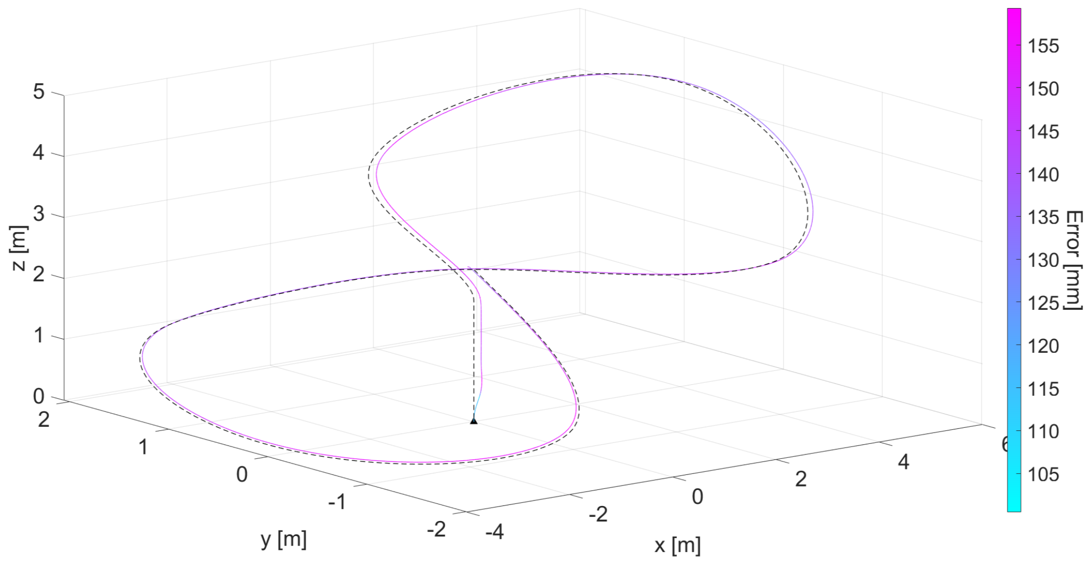


Figure 5.5: Simulation result obtained by applying the NMPC algorithm with nominal model in presence of a rotor speed mismatch. Due to the thrust lack, the quadrotor follows the reference trajectory with an altitude offset, while the unpredicted loss of symmetry of the rotor configuration causes a lateral deviation.

The effect that these unpredicted dynamics have on the position error, when the drone is controlled with an NMPC based on the quadrotor nominal model, can be observed in Figure 5.6. In both the mass mismatch and the rotor speed mismatch cases the position error w.r.t. the reference trajectory is consistently

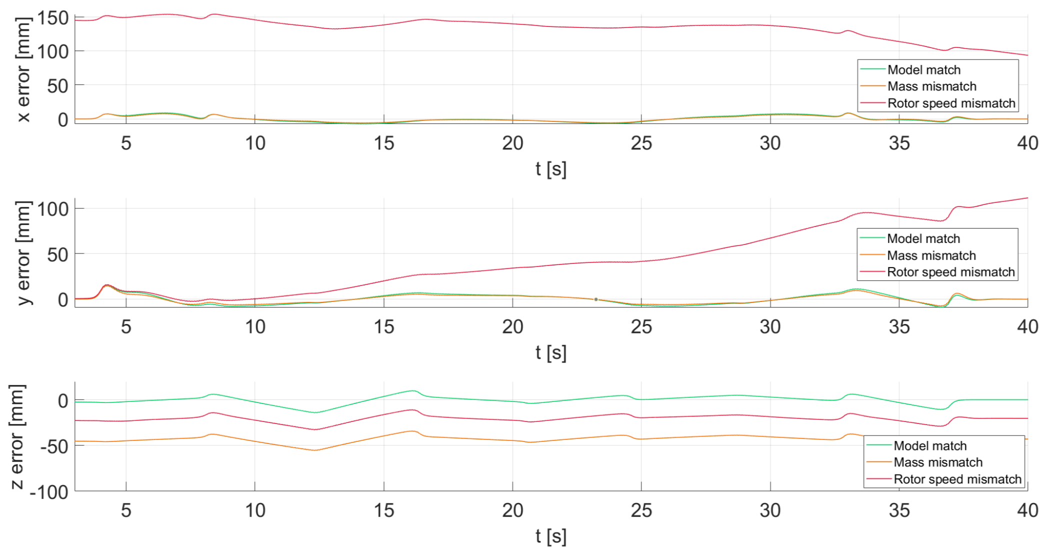


Figure 5.6: Position error of the controlled quadrotor w.r.t. the reference trajectory. The case of perfect match between NMPC model is compared with the mass mismatch and the rotor speed mismatch scenarios. Notice that here the first seconds of simulation data are not considered, since the takeoff is generally a troublesome maneuver and it is often tackled with a dedicated controller.

higher if compared with the one achieved in case the NMPC model perfectly matches the simulation model. Finally, the results in terms of average position error are compared in Table 5.2. These results clearly confirm how the NMPC performance is highly dependent on the accuracy of the adopted prediction model.

Mismatch	Av_{err} $[mm]$
Exact model match	7.2
Mass mismatch	43.8 (+508.3%)
Rotor rotation speed mismatch	146.9 (+1940.3%)

Table 5.2: Average position errors achieved by NMPC with nominal model in presence of model mismatches.

5.4 Learnig the model mismatch

In order to deal with the effect of the unmodeled dynamics listed in the previous section and to reduce the position error that they introduce, it is possible to

replace the nominal NMPC prediction model with a grey-box model that contains a learned correction law, following the intuition discussed in chapter 4. Specifically, a GP or a NN grey-box model can be adopted as prediction model for the NMPC algorithm, while using the same parameters tuning described in section 5.2.1. The resulting controllers are hereafter denoted as GP-NMPC and NN-NMPC respectively, as specific expressions of the Lb-MPC framework.
Both the GP models and the NN models are previously trained by measuring the drone linear and angular accelerations over a dedicated training trajectory. Subsequently, the learned models are tested on the trajectory presented in Figure 5.1.
In order to improve the learning results, by avoiding misleading correlations between input data and training target, a feature selection is manually applied to the input data of the learning models. In fact, instead of considering the whole $[\boldsymbol{x}^T, \boldsymbol{u}^T]^T$ state-control vector as learning input, it is here preferable to reduce the state component to the sole pose of the drone. Specifically, by representing such pose with an Euler angle representation $\boldsymbol{\epsilon} = [\rho, \theta, \psi]^T$, the considered learning input can be expressed as $\mathrm{x} = [\boldsymbol{\epsilon}^T, \boldsymbol{u}^T]^T$.
The remaining part of this section is dedicated to the presentation of the training trajectory, to the discussion of the specifics of the adopted learning methods and to a presentation of the learning results.

5.4.1 Training trajectory

The adopted learning models are trained off-line on a training trajectory for $2000s$, collecting 20000 data points with a sampling time of $0.1s$; $T = 16000$ of them will be used for the actual training procedure, while the last 4000 will be dedicated to the validation. The training trajectory, which is displayed in Figure 5.7, is designed in order to repeatedly perform different types of maneuvers with changing amplitude and velocity:

1. *Helix*: during this maneuver the quadrotor performs a number of horizontal turns, connected by ascending movements at first and descending ones then;
2. *Slalom*: during this maneuver the quadrotor executes a sequence of right and left alternated turns, following in the meantime an ascending movement to each a certain altitude. Once this altitude is reached, same kind of turns are performed in a descending direction;

3. *Up and Down*: during this maneuver the quadrotor performs a quick slope, followed by a dive.

Each of these maneuvers is repeatedly performed a number of times, while increasing the *amplitude* of the maneuver. This simple approach is used to obtain a trajectory of the desired length with just few maneuvers definitions. In fact, by repeating each movement with changed amplitude also the drone acceleration varies, allowing to obtain more significant data. As an alternative, one could use a randomized trajectory for training. However, the adopted method allows to train the drone while performing maneuvers with controlled characteristics, both in terms of learning input values and measured accelerations.
Notice that while NN models require a significant amount of learning data to achieve a satisfactory training, GP regression could be performed with a shorter learning trajectory. However, all the considered learning methods are trained on the same trajectory, in order to deliver a better comparison of the results.

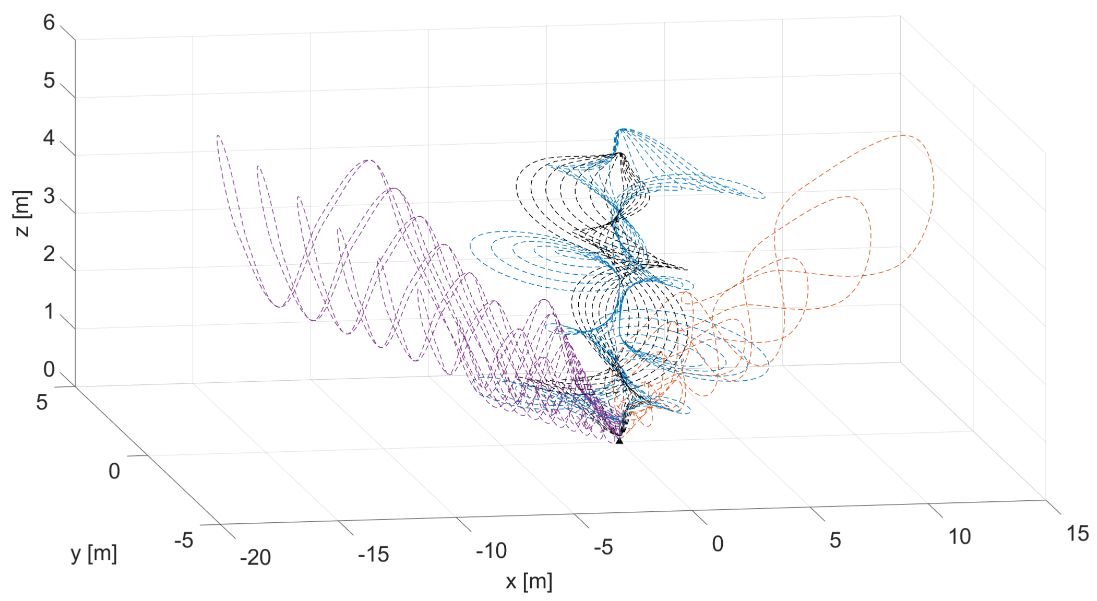


Figure 5.7: Training trajectory for the off-line learning procedure. Three types of maneuvers are performed with different amplitude and velocities in order to train the grey-box models in a number of different significant scenarios. The *helix* maneuvers are highlighted in blue, the *slalom* movements are highlighted in orange and the *up and down* maneuvers are colored in purple. The remaining black section of the trajectory is finally used for validation.

5.4.2 GP approach and results

The mismatch between nominal and real accelerations, introduced either by the mass or by the rotor speed model mismatches, can be learned and therefore predicted thanks to a set of Gaussian Process models. In particular, as anticipated in section 4.2, six GP models are used to derive a correction law, one for each component of linear acceleration $\dot{\boldsymbol{v}}$ and angular acceleration $\dot{\boldsymbol{\omega}}$ of the drone platform. For each GP, a SE kernel is adopted for the modeling of the covariance function, while the model mean is imposed to be zero. The GP models hyperparameters are tuned by maximization of the training data marginal likelihood, thanks to a MATLAB toolbox developed by Rasmussen and colleagues [43].
The learning results for the mass mismatch case, tested on the reference trajectory, are hence displayed in Figure 5.8. As one can notice, the GP model predicts almost perfectly the learning target, leading to a neat superposition between the measured accelerations and the ones predicted by the grey-box model.

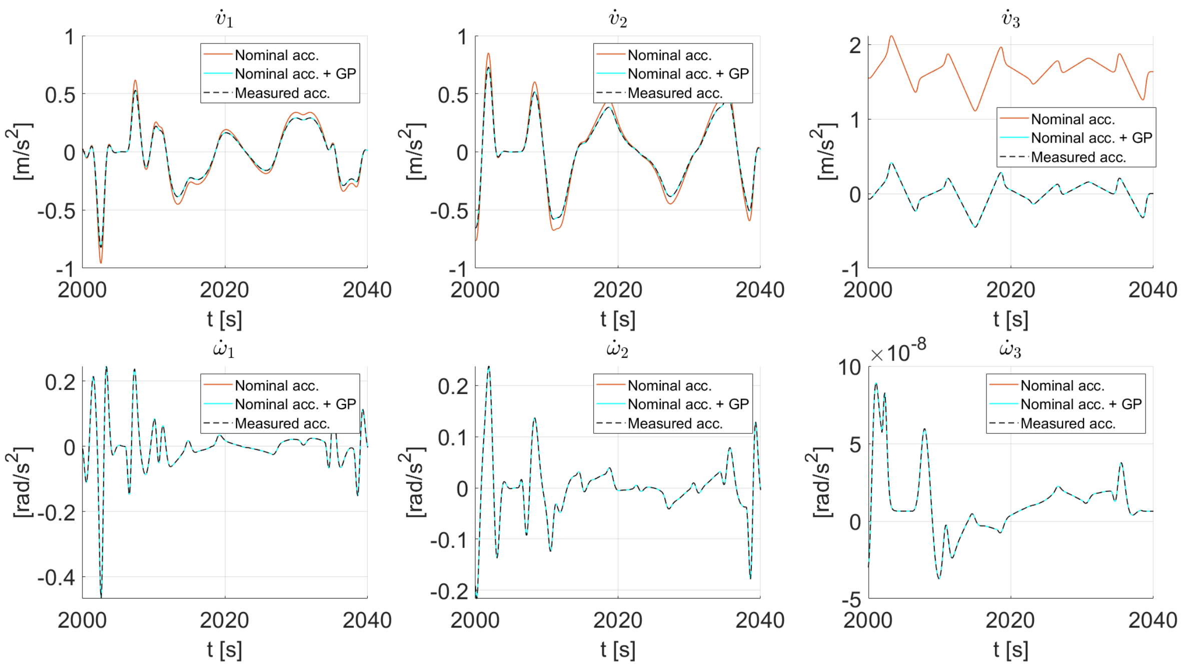


Figure 5.8: Comparison between measured accelerations, nominal model prediction and GP grey-box model prediction, in presence of mass model mismatch. The data are sampled over the test reference trajectory. As one can expect from first principles, the main acceleration mismatch is the one relative to the z component of the linear acceleration.

Even though GPs provide a precise and powerful regression framework, they severely suffer from scalability problems. In fact, the GP prediction procedure has cubic complexity w.r.t. the number of considered support points of the model. In order to cope with this limitation, a lot of research has been devoted towards sparse Gaussian Process model approximations [33]. In particular, a global approximation generally known as Variational Free Energy (VFE) [2] is adopted in order to approximate the GP posterior, by selecting a set of n_u inducing points in place of the T training points adopted to support the model. By reducing the inducing points of the GP model to $n_u = 34$, starting from $T = 16000$ training data, the VFE approximation is capable of maintaining the mismatch prediction nearly unchanged, as displayed in Figure 5.9. Thanks to this approximation, the average CPT on the simulation machine for one GP-NMPC iteration can be reduced to $3.7\ ms$, allowing to adopt the grey-box model in the NMPC formulation under the real-time constraint.

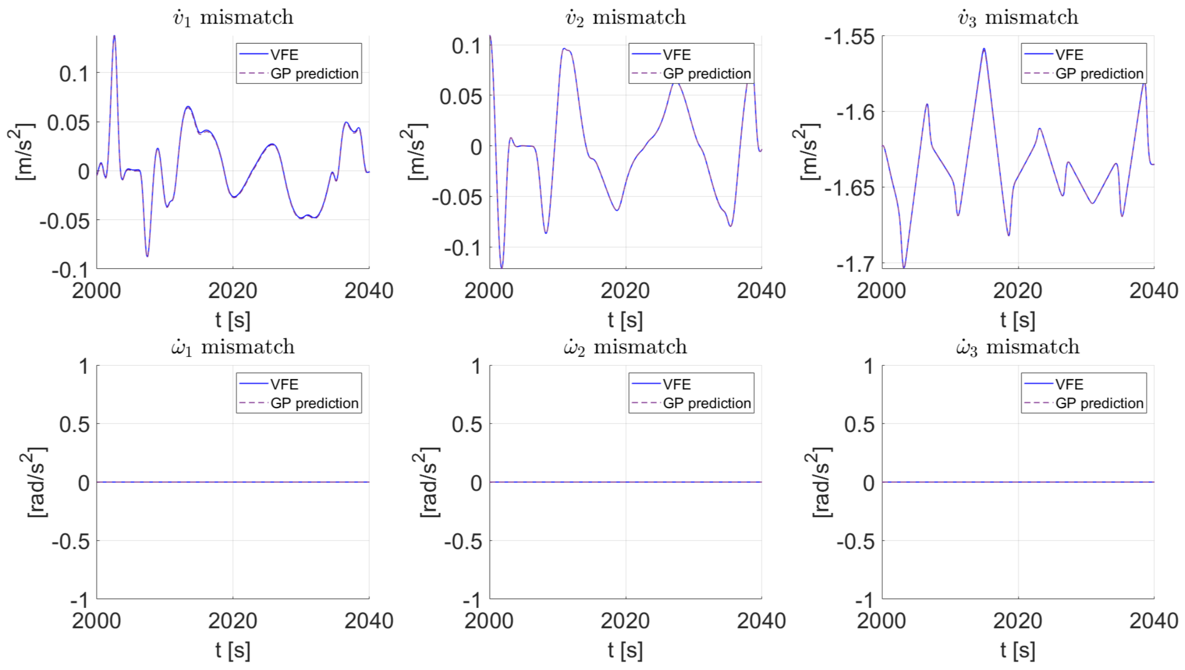


Figure 5.9: Plot of the mismatches between nominal accelerations and measured accelerations along the reference trajectory, in presence of mass mismatch between nominal and simulation models. A full GP model prediction, based on $T = 16000$ training data, is compared to a VFE approximation with $n_u = 34$ inducing points. Both the full GP model and the VFE approximation matches the displayed learning target.

Furthermore, a GP-based grey-box model can be adopted in a similar fashion also in presence of the motor rotation speed mismatch. Even though this second mismatch can be considered more complex than the previous one, the GPs can properly learn an effective correction law, as shown in Figure 5.10.

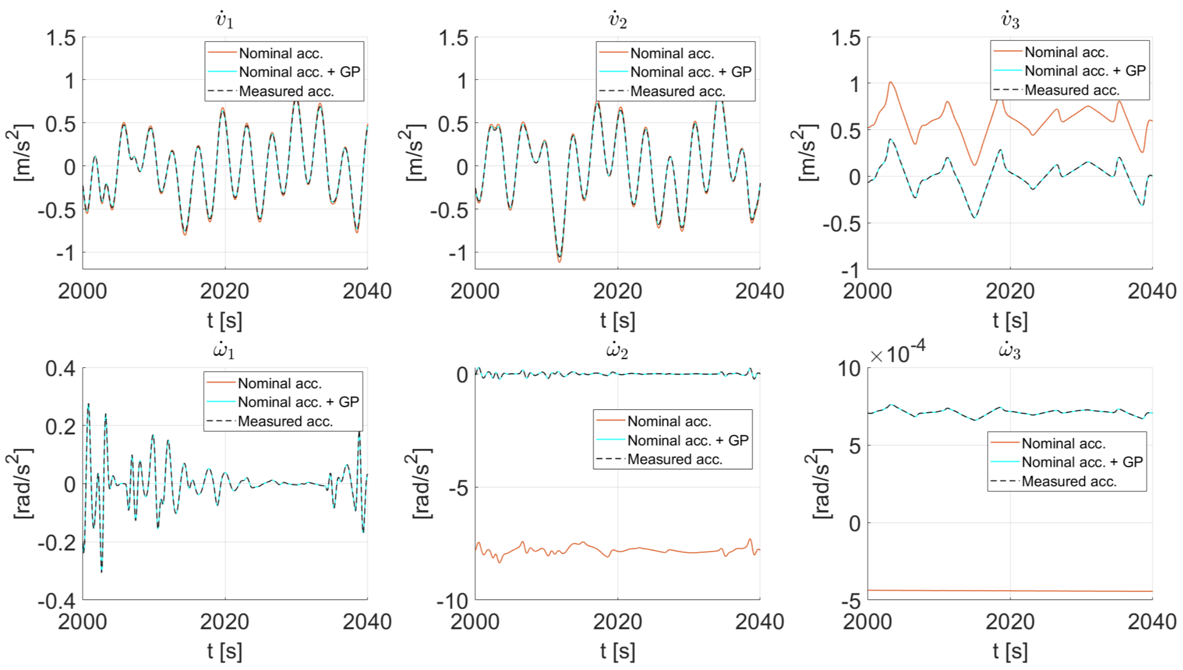


Figure 5.10: Comparison between measured accelerations, nominal model prediction and GP grey-box model prediction, in presence of rotor speed mismatch. The thrust lack causes a significant mismatch on the z component of the linear acceleration. The unpredicted configuration asymmetry causes a mismatch on the y angular acceleration. Both the effects are correctly predicted by the GP grey-box model.

5.4.3 NN approach and results

Alternatively to the Gaussian Process approach, the unmodeled dynamics can be learned by means of Neural Network models. In particular, two NN are adopted to learn the acceleration error made by the nominal model w.r.t. the linear acceleration $\dot{v}$ and the angular acceleration $\dot{\omega}$ respectively. The learning models are both designed as three hidden layers feedforward Neural Networks, with Leaky ReLU activations and 8 neurons for the first layer, 16 for the second one and 36 for the third.

The network models are trained by means of the MATLAB Deep Learning Toolbox, with a popular stochastic gradient descent algorithm commonly known as Adam [28]. During the training procedure, a Mean Squared Error loss is adopted in order to evaluate the prediction performance of the networks, regularized with an L^2 penalty term weighted by a relative coefficient $\alpha = 10^{-4}$. The network weights are initialized by sampling from a Gaussian distribution with standard deviation $\beta = 0.01$.

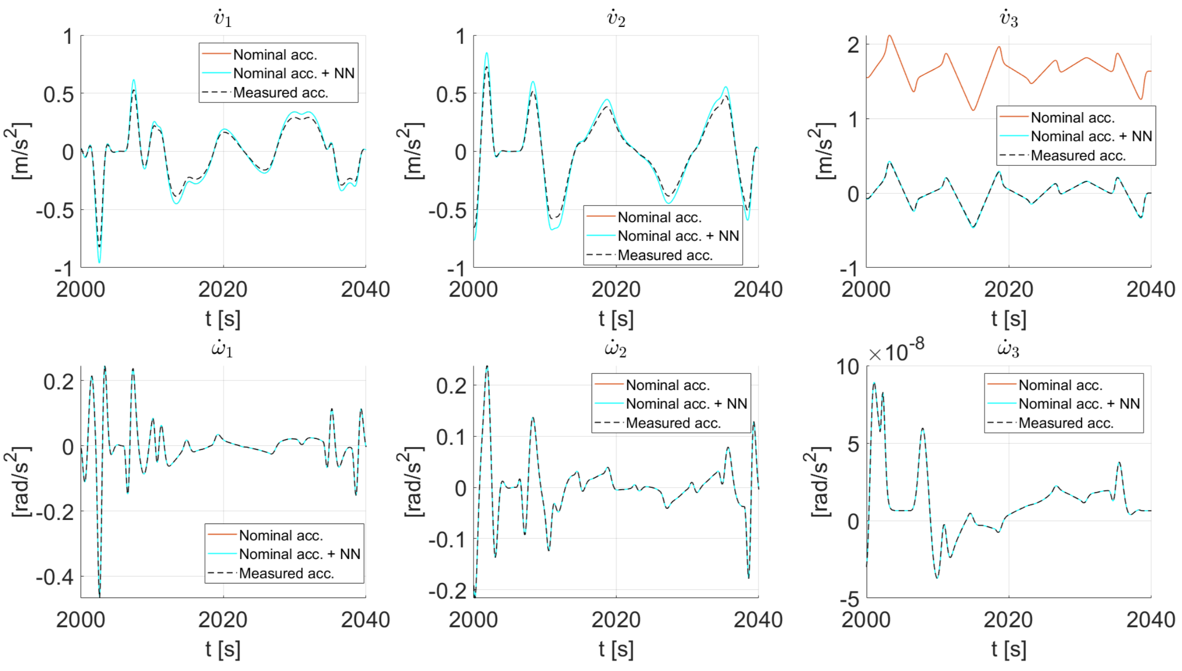


Figure 5.11: Comparison between measured accelerations, nominal model prediction and NN grey-box model prediction, in presence of mass model mismatch. The z linear component is correctly predicted by the NN model, while the other components are nearly unchanged w.r.t. the nominal model prediction.

In presence of mass mismatch, the Neural Network models are able to retrieve a good correction law for the nominal model predicted accelerations, as shown in Figure 5.11. Notice however that the regression is less precise with respect to the one performed by the GP approach. In particular, while the main mismatch on the z component of the linear acceleration seems to be properly corrected, the other linear acceleration components remain completely unchanged even after the introduction of the NN correction term in the prediction model. Nonetheless, it can be noticed how the NN model intuitively prioritize the regression of the most influential components of the learning target. If a wider and hence more complex network is used for the regression task, the target components affected by minor mismatches can also be addressed. Indeed, by using a network with 512 neurons in the first layer, 1024 in the second layer and 2042 in the third one, the mismatches on x and y components of linear acceleration are also tackled by the grey-box model, as shown in Figure 5.12. However, such network would require higher computational and memory resources if used to define a grey-box prediction model for an NMPC implementation, thus the less complex design presented before is highly preferable.

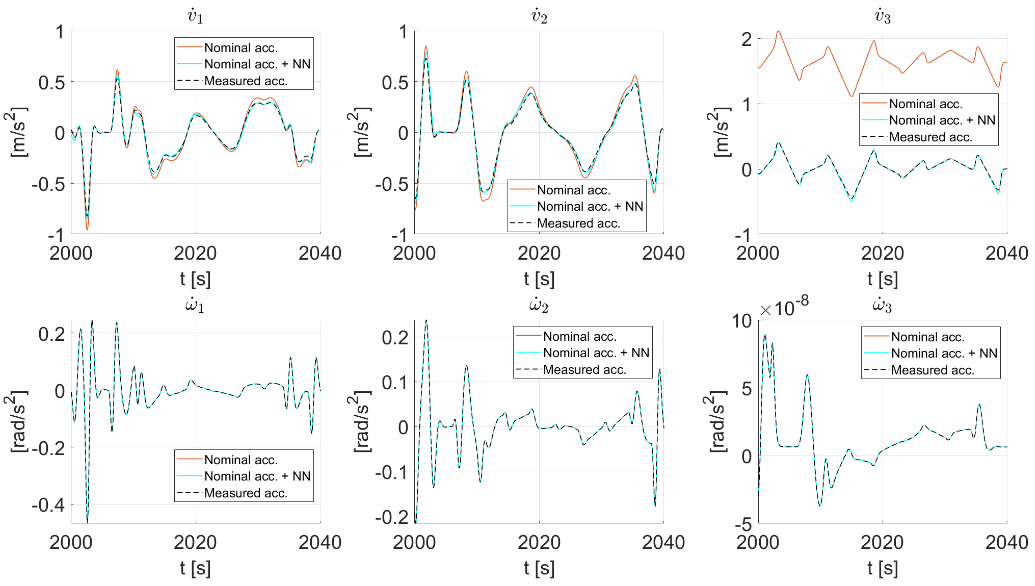


Figure 5.12: Comparison between measured accelerations, nominal model prediction and NN grey-box model prediction, in presence of mass model mismatch, obtained by adopting a Neural Network model with 512-1024-2042 neurons divided in three hidden layers. All the components of the linear acceleration are predicted by the grey-box model with higher precision w.r.t. the nominal model.

Similar behaviour can be observed also in presence of the rotor speed mismatch, as one can see in Figure 5.13. Here the z thrust mismatch is correctly learned by the NN model, while the angular acceleration mismatch is only partially addressed. However, such shallow feedforward Neural Network models allow to perform the prediction procedure simply by applying affine operations and activation functions evaluation, potentially speeding up the NMPC algorithm iterations. Indeed, each NN-NMPC iteration is completed with an average CPT of $1.8\ ms$ on the simulation machine. As it will be shown in the following paragraphs, this computational advantage can be achieved without significant costs in terms of average position error w.r.t. the GP-NMPC approach.

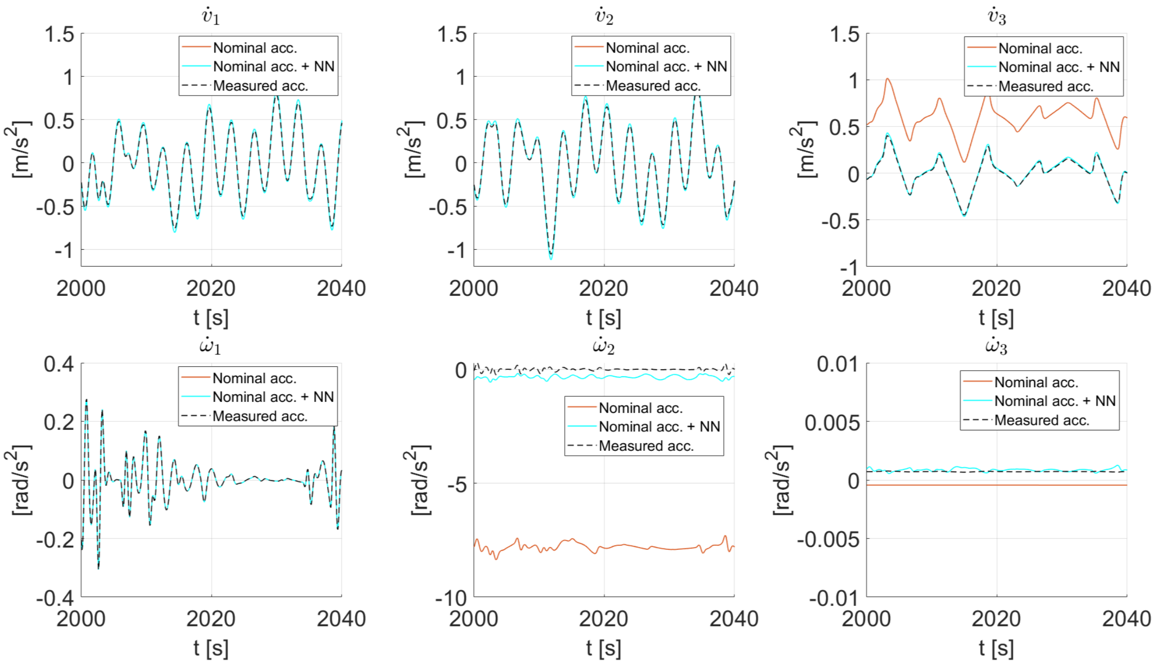


Figure 5.13: Comparison between measured accelerations, nominal model prediction and NN grey-box model prediction, in the case of rotor speed mismatch. The z linear component is correctly learned by the model, while the angular acceleration prediction is only partially improved with respect to the nominal model.

5.5 Lb-NMPC results

This section is dedicated to the discussion of the simulation results obtained by applying the Lb-MPC approach for the control of the quadrotor platform. First, the GP-NMPC and the NN-NMPC approaches are compared in terms of achieved position error during the trajectory tracking task of interest, in presence of mass or rotor speed mismatch. Afterwards, some non-ideality are taken into account, in order to test the learning procedure effectiveness under more realistic conditions.

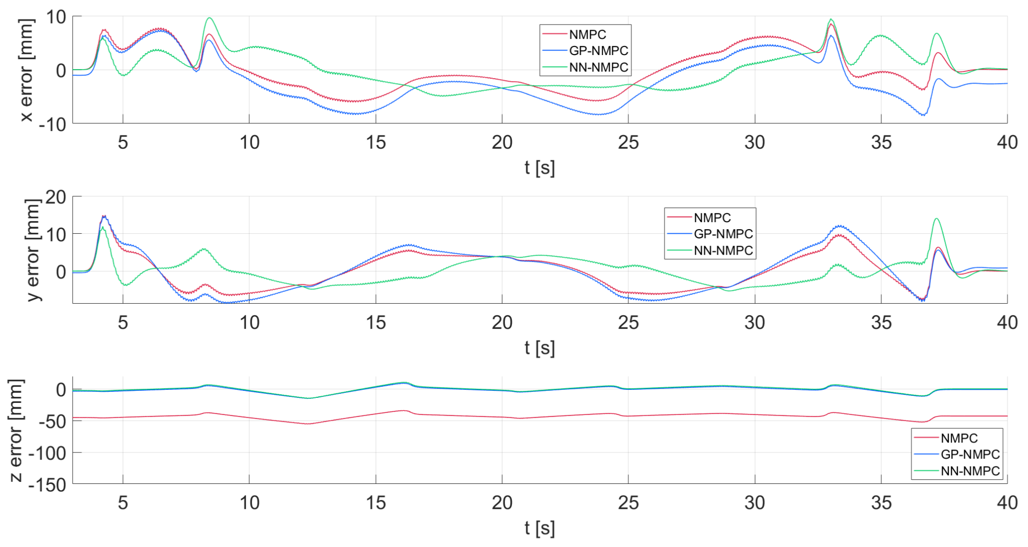
5.5.1 Ideal case

After training the considered learning models in presence of mass mismatch, these can be used in order to perform a more precise predictive control of the quadrotor. Starting from the GP-NMPC approach, it is possible to observe a significant improvement in terms of position error w.r.t. the standard NMPC approach. Indeed, thanks to the learned correction law, the GP-NMPC is able to achieve an average error of $\text{Av}_{\text{err}} = 7.8\ mm$, reducing the one obtained by using the nominal model by 82.2% and reaching a result that is almost identical to the one obtained by the nominal NMPC in case of perfect model match. Similarly, the NN-NMPC approach is able to reduce the average position error to $\text{Av}_{\text{err}} = 5.8\ mm$ in the same scenario, which is equivalent to a 86.8% reduction. In conclusion, both the techniques definitely lead to a significant reduction of the position error over the whole trajectory, as shown in Figure 5.14a. This clearly reflects in satisfactory tracking performance, as one can confirm by looking at Figure 5.15a.
Considering the more complex scenario of a rotor speed mismatch, the two Lb-MPC approaches equivalently achieve satisfactory results. The GP-NMPC completes the trajectory tracking task with an average position error of $\text{Av}_{\text{err}} = 7.6\ mm$, while the NN-NMPC attain an average error of $\text{Av}_{\text{err}} = 7.2\ mm$. These results, if compared to the average error obtained by relying on the nominal model, correspond to a 94.8% reduction with the GP-based model and a 95.1% reduction with the NN-based one. More in detail, also in this case the position error achieved by the Lb-NMPC approaches is significantly reduced w.r.t. the one resulting from a NMPC application over the whole trajectory, as shown in Figure 5.14b. Moreover, these two learning-based control approaches offer a good tracking of the reference trajectory, as confirmed by the simulations in Figure 5.15b.
Observe that the two learning techniques lead to very similar results in terms of position error. Indeed, even though the considered Neural Networks are shown to grant a less precise regression of the acceleration mismatch with respect to the GP models, the registered control performance do not underline any significant difference between the two approaches. However, the Neural Network prediction procedure is less computationally demanding than the GP one, as

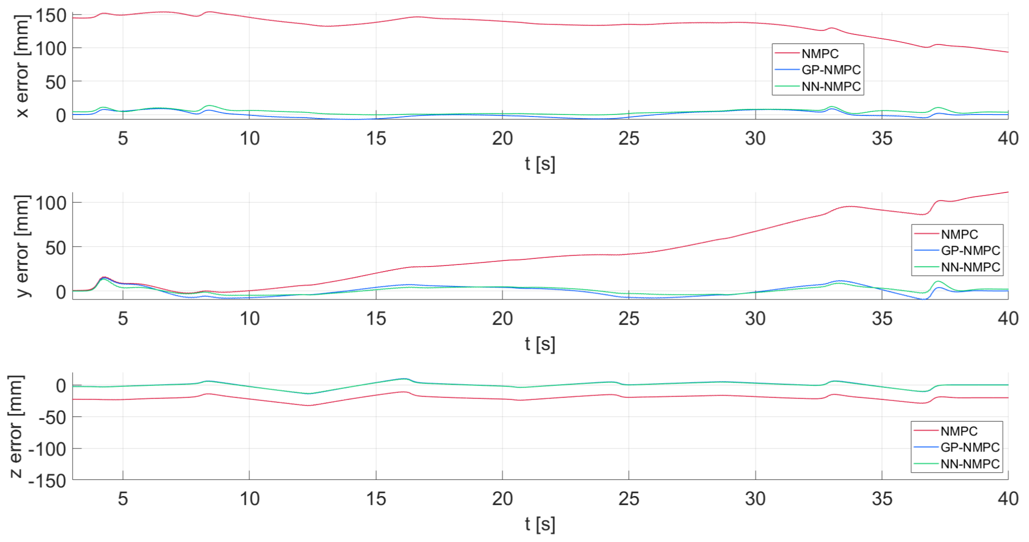
evidenced by the measured CPTs during the simulations.
Table 5.3 offers a summary of the comparison between the considered control techniques, in terms of average position error and computational burden.

Control approach	**Mass mism.** $\mathrm{Av}_{\mathrm{err}}$ $[mm]$	**Rotor speed mism.** $\mathrm{Av}_{\mathrm{err}}$ $[mm]$	**CPT** $[s]$
NMPC	43.8	146.9	0.3
GP-NMPC	7.8 (−82.2%)	7.6 (−94.8%)	3.7
NN-NMPC	5.8 (−86.8%)	7.2 (−95.1%)	1.8

Table 5.3: Performance comparison between the NMPC based on the nominal model, GP-NMPC and NN-NMPC in an ideal learning scenario.

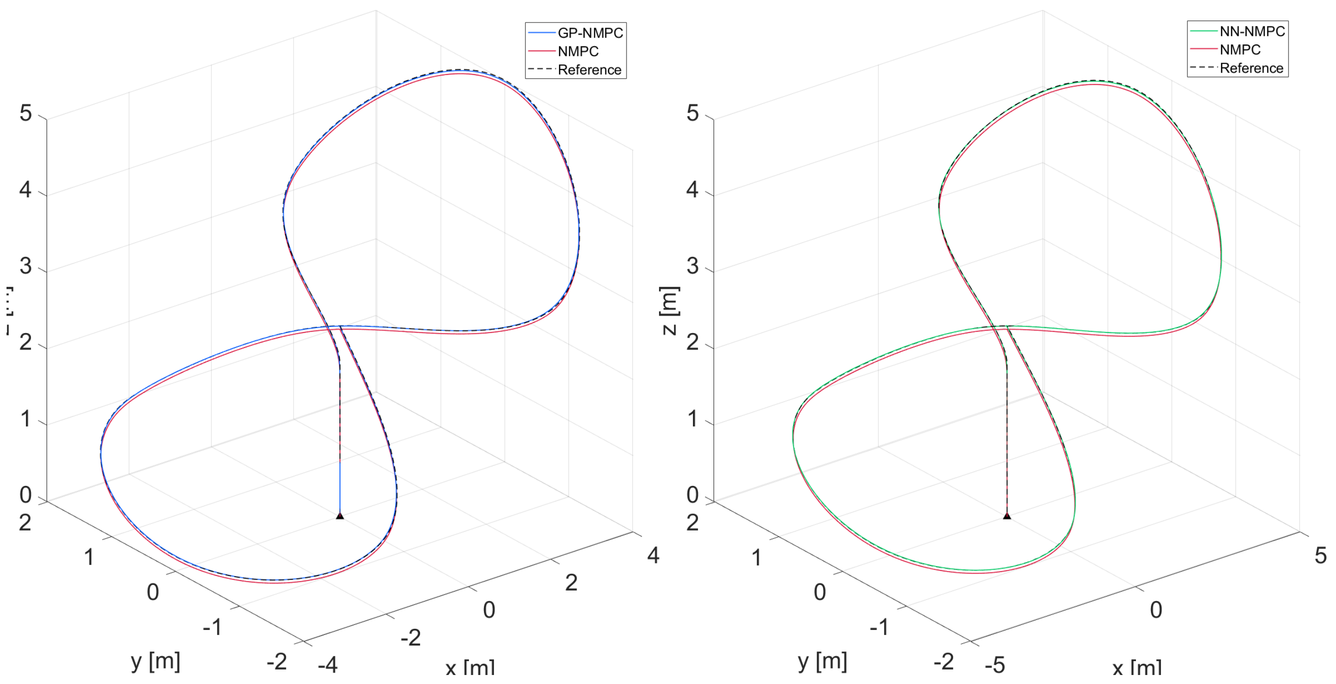


(a) Mass mismatch scenario

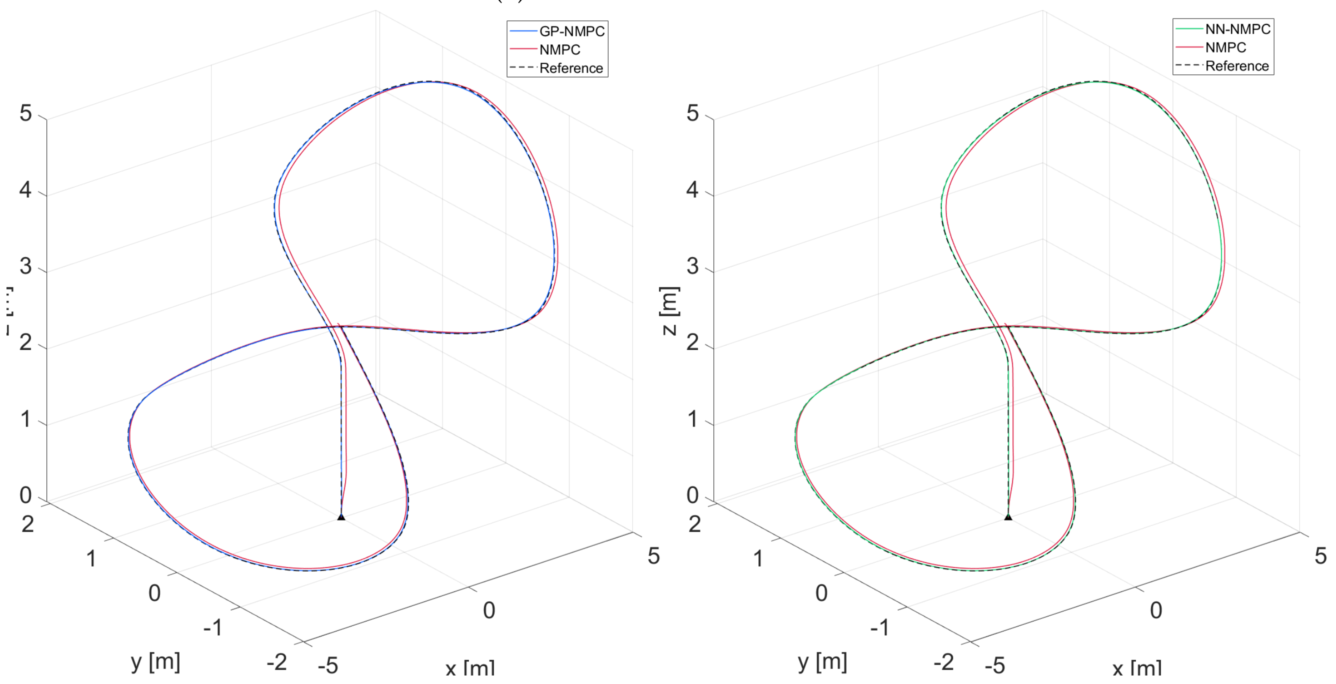


(b) Rotor speed mismatch scenario

Figure 5.14: Position error obtained in simulations, for the ideal learning scenario. The results obtained by GP-NMPC and NN-NMPC are compared with the ones of a NMPC based on the nominal knowledge of the model, either in presence of mass mismatch (a) or rotor speed mismatch (b).



(a) Mass mismatch scenario



(b) Rotor speed mismatch scenario

Figure 5.15: Simulation results obtained from the application of GP-NMPC and NN-NMPC in an ideal learning scenario, compared with the results achieved by the nominal NMPC. Panel (a) shows the results obtained in case of mass mismatch. Panel (b) is dedicated to the application in case of rotor speed mismatch. In both cases, GP-NMPC and NN-NMPC achieve an almost perfect overlap between the drone trajectory and the reference.

5.5.2 Control signal delay

Differently from what happens in the previous simulations, in real life scenarios the control input signal is affected by a certain delay between the moment of its determination by the controller and its actual application to the system. In particular, a dynamic delay of $0.05\ s$ is here taken into account. Such delay, generally denoted in the Laplace domain with a term $e^{-0.05s}$, is expressed in the simulation system as a first order Padè approximation, given by the following transfer function

$$D(s) = \frac{1 - \frac{0.05}{2}s}{1 + \frac{0.05}{2}s} \tag{5.3}$$

Therefore, the resulting simulation system can be schematized as in Figure 5.16.

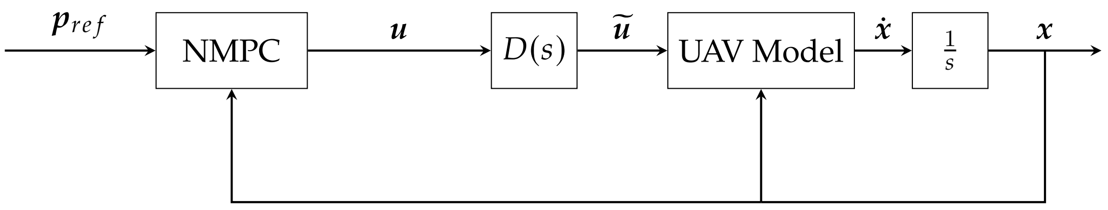


Figure 5.16: Control scheme for the simulation of the system in presence of input delay. The control input $\boldsymbol{u}$ computed by the controller is affected by a dynamic delay, hence $\widetilde{\boldsymbol{u}}(t) \approx \boldsymbol{u}(t - 0.05)$ is applied to the controlled system.

This delay does not appear to affect in any significant way the control performance during the simulations, as the adopted control frequency is high enough to tackle the delay effect. Nonetheless, it is crucial to consider that its presence can alter the learning procedure of the grey-box models. Indeed, as the rotors speed is not typically measured for a commercial quadrotor, it would be reasonable to consider vector $\mathrm{x} = [\boldsymbol{\epsilon}^T, \boldsymbol{u}^T]^T$ as input for the learning problem of interest, with $\boldsymbol{u}$ simply taken as output of the control algorithm. However, this would introduce an additional mismatch between the nominal model acceleration prediction and the measured acceleration, since the first is based on the application of input $\boldsymbol{u}$ to the system, while the latter is consequent to the application of $\widetilde{\boldsymbol{u}}$.

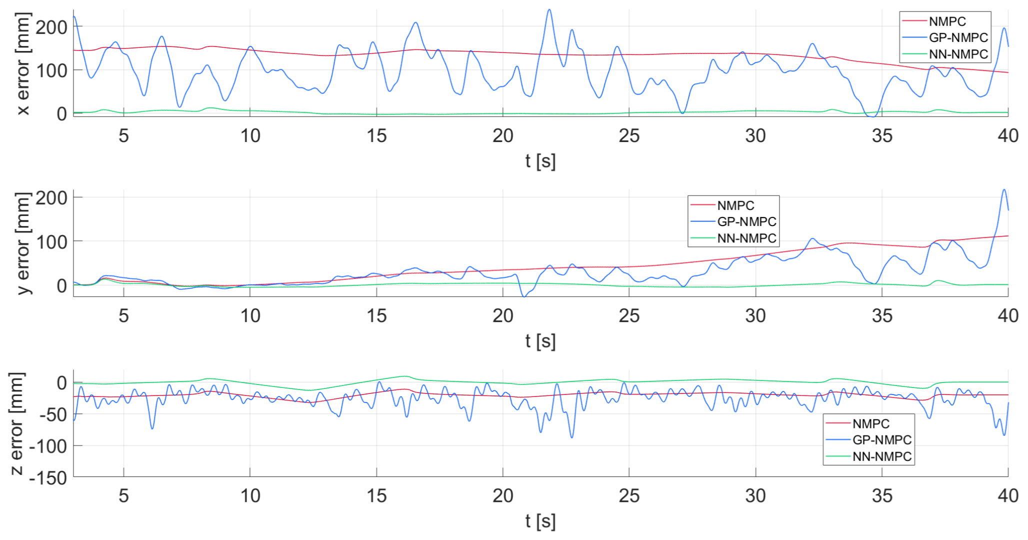


Figure 5.17: Comparison of the position errors obtained by NMPC, GP-NMPC and NN-NMPC in presence of a rotor speed mismatch. Here the learning-based models are trained in presence of control input delay, by adopting the predicted control $\boldsymbol{u}$ as part of the learning input.

In support of this consideration, it is possible to observe the consequences of the adoption of learning input $\mathtt{x} = [\boldsymbol{\epsilon}^T, \boldsymbol{u}^T]^T$ in presence of control delay in the case of rotor speed mismatch. As one can observe in Figure 5.17, only the NN-NMPC is able to present a significant improvement with respect to a NMPC based on the nominal quadrotor model. On the contrary, the considered GP-NMPC design appears to present some difficulties in the regression of the mismatch law. Specifically, the NN-NMPC achieves an average position error of $\mathrm{Av}_{\mathrm{err}} = 7.1\ mm$, corresponding to a 95.2% reduction of the usual $\mathrm{Av}_{\mathrm{err}} = 146.9$ mm error obtained by the nominal NMPC, which remains unchanged by the control delay. Instead, the GP-NMPC only reduces the nominal NMPC error by 24.8%, as it achieves an average position error of $\mathrm{Av}_{\mathrm{err}} = 110.5\ mm$. Also by looking at the simulation trajectories, which are displayed in Figure 5.18, it is possible to notice that NN-NMPC achieves a satisfactory reference tracking, while GP-NMPC presents not only a less precise tracking, but also a less *smooth* trajectory. Intuitively, one could attribute the better performance of NN-NMPC to the fact that the delay induced mismatch is almost ignored during the learning procedure while prioritizing major mismatches. A similar behavior was discussed in section 5.4.3, where the minor mismatches that affected a part of the components of the learning target were naturally ignored by the NN model. On the other hand, the GP-based grey-box model tackles the delay induced mismatch, without properly being able to retrieve a good regression law.

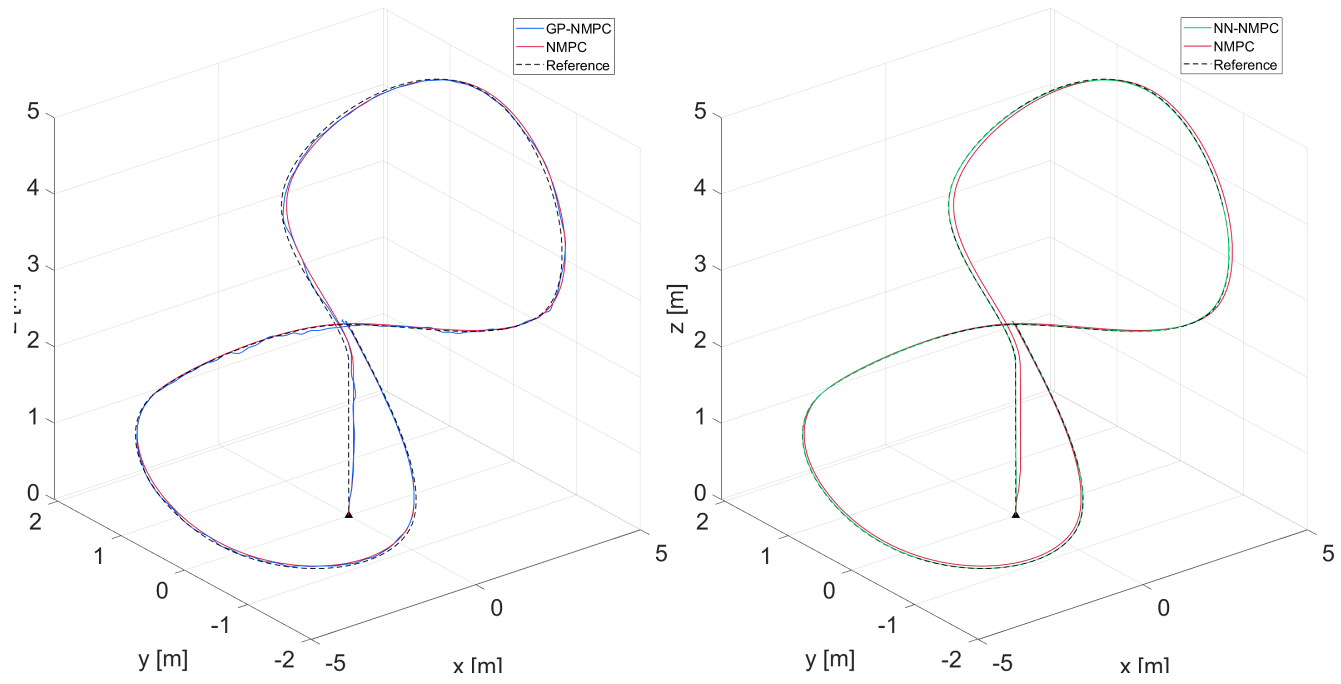


Figure 5.18: GP-NMPC and NN-NMPC simulation results in presence of control input delay and rotor speed mismatch, achieved by adopting the predicted control input $\boldsymbol{u}$ as part of the learning input for the grey-box models. GP-NMPC is not able to provide a good overlap between the drone trajectory and the reference, as the considered GP grey-box model does not deliver a good prediction of the drone dynamics. On the contrary, NN-NMPC achieves an almost perfect overlap between drone trajectory and reference.

In order to completely avoid this issue, one could consider to adopt $\mathrm{x} = [\boldsymbol{\epsilon}^T, \widetilde{\boldsymbol{u}}^T]^T$ as input for the learning problem. This clearly leads to an improvement in the performance of the considered Lb-MPC methods, as one can see in in Figure 5.19 and 5.20. In this scenario, both GP-NMPC and NN-NMPC achieve an average position error of $\mathrm{Av}_{\mathrm{err}} = 7.5\ mm$, corresponding to a 94.9% reduction w.r.t. the nominal NMPC error. However, in order to obtain $\widetilde{\boldsymbol{u}}$ it is necessary to directly measure the drone rotors speed. For this purpose, a set of additional sensors would be needed, making this solution not ideal for a real world application.

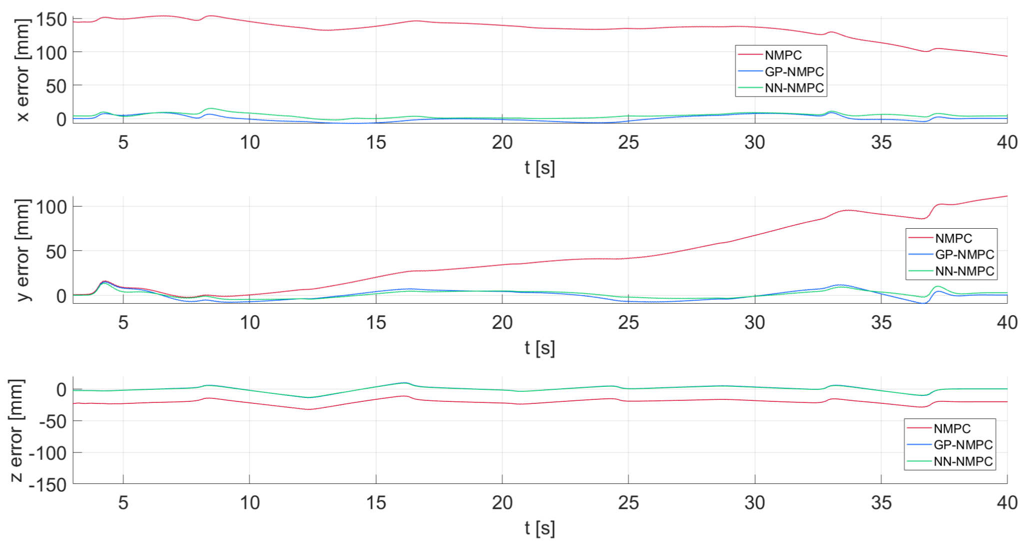


Figure 5.19: Control error comparison in presence of rotor speed mismatch and control input delay, with GP-NMPC and NN-NMPC trained with measured control $\widetilde{u}$ as part of the learning input.

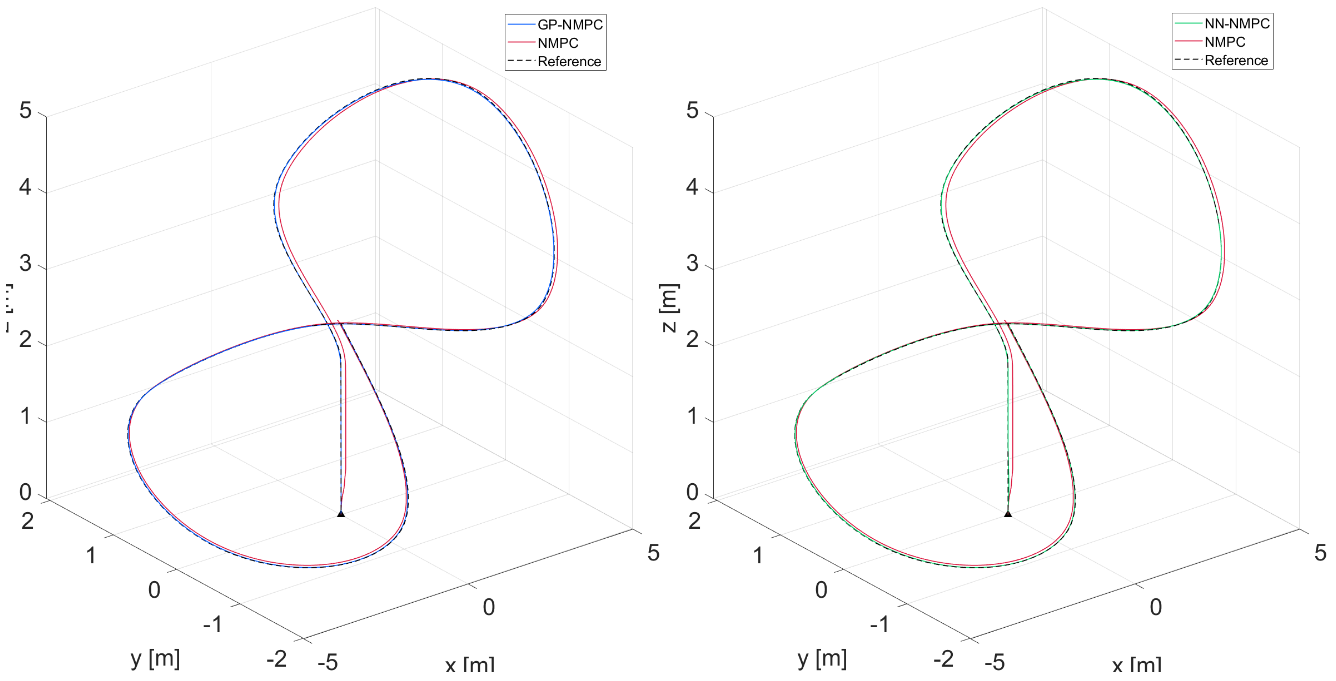


Figure 5.20: GP-NMPC and NN-NMPC simulation results in presence of control input delay and rotor speed mismatch, achieved by using the measured control input $\widetilde{u}$ as part of the control input. In this case, both the GP-NMPC and the NN-NMPC are able to provide a good tracking of the reference trajectory.

5.5.3 Training target noise

So far it has been assumed that the learning target can be measured without any noise. However, a common problem in regression is that only noisy measurements of the target are typically available. Consider then an additive Gaussian measurement noise with zero mean and variance $\sigma_n^{\dot{v}} = 10^{-5} \frac{m}{s^2}$ and $\sigma_n^{\dot{\omega}} = 10^{-9} \frac{rad}{s^2}$ for linear and angular acceleration data respectively. Recall that the presence of such Gaussian noise is part of the assumptions of the Gaussian Process models described in chapter 4. The considered Neural Network models instead do not take target noise into account directly. For this reason, better results are expected from the GP-NMPC approach in this scenario. Indeed, testing the considered controller in presence of the rotor speed mismatch, the GP-NMPC performance is not influenced by the presence of the target noise, as it achieved an average position error of $\mathrm{Av}_{\mathrm{err}} = 7.7\,mm$. On the other hand, the NN model is still able to properly generalize the mismatch law, leading NN-NMPC to obtain an average position error of $\mathrm{Av}_{\mathrm{err}} = 11.5\,mm$. This result is still completely acceptable, as one can also evaluate by looking at the position errors over the whole trajectory in Figure 5.21 and at the resulting trajectories in Figure 5.22.

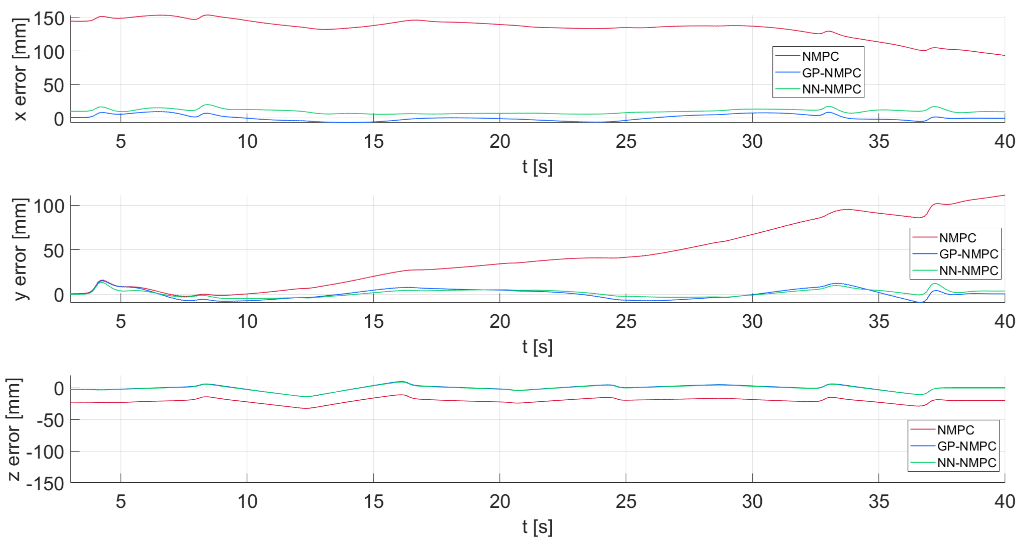


Figure 5.21: Control error comparison in presence of rotor speed mismatch and noisy learning target.

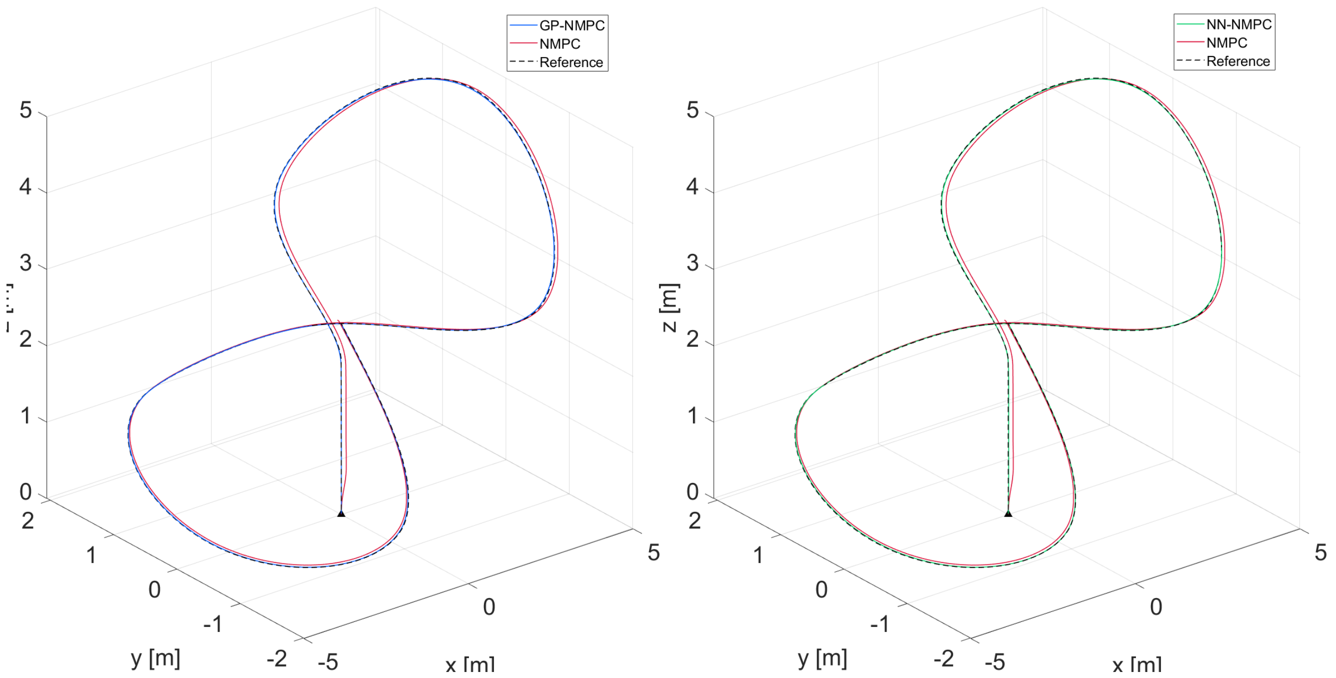


Figure 5.22: GP-NMPC and NN-NMPC simulation results in presence of rotor speed mismatch, with GP and NN grey-box models trained with unfiltered noisy learning target measurements. Both GP-NMPC and NN-NMPC grant a good overlap between drone trajectory and reference, even though the NN-NMPC error is slightly deteriorated w.r.t. the ideal case.

Yet, from both figures one can observe that GP-NMPC performs slightly better than NN-NMPC in this scenario. Moreover, notice that the average position error achieved by NN-NMPC presents a small deterioration with respect to the ideal learning case, where achieved an average error equal to $\text{Av}_{\text{err}} = 7.2\ mm$. The presence of target noise can also be tackled with simple data processing techniques. Indeed, by filtering the measured acceleration data it is possible to retrieve a good representation of the signal before the introduction of the Gaussian noise. However, in order to avoid the delay that would be introduced on the filtered signal by a naive low-pass filter, this procedure should be carried out by adopting a *zero-phase filter*. That means that after filtering the data in the forward direction, the filtered sequence needs to be reversed and run through the filter once again. The result has the following characteristics:

- Zero phase distortion;
- A filter transfer function equal to the squared magnitude of the original filter transfer function;

- A filter order that is double the order of the original filter.

In the simulation setup, this procedure is carried out by a MATLAB function that implements the algorithm proposed by Gustafsson in [22], starting from a third order low-pass filter. The result of such procedure can be observed in Figure 5.23, where the original signal, the noisy signal and the filtered one are compared in an example.

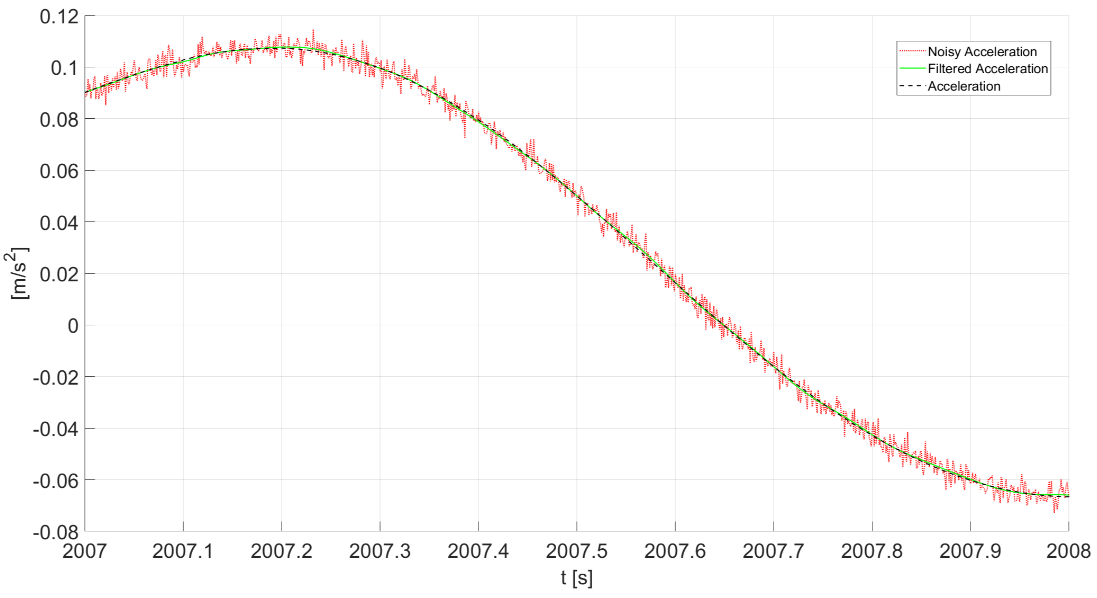


Figure 5.23: Example of comparison between real acceleration, noisy acceleration and a zero-phase filtered version of the latter. The filtered signal provides a good representation of the real acceleration, without presenting any time delay.

By adopting the filtered target during the learning procedure, one can notice a slight improvement in the NN-NMPC results, with an average position error of $\mathrm{Av}_{\mathrm{err}} = 8.3\ mm$. In spite of that, the GP-NMPC performance are actually deteriorated by the adoption of the filter. Indeed, by looking at Figures 5.24 and 5.25 one can see how the position error achieved by GP-NMPC in this case is significantly higher than the one obtained with NN-NMPC. In particular, the average position error achieved by GP-NMPC is $\mathrm{Av}_{\mathrm{err}} = 44.0\ mm$. If compared to the error achieved by the nominal NMPC, this last result is still quite positive, since the GP-based grey-box model introduces a 70.0% error reduction. Yet, it is clear that the GP models not only can be used without any target filtering, but it is even preferable to do so.

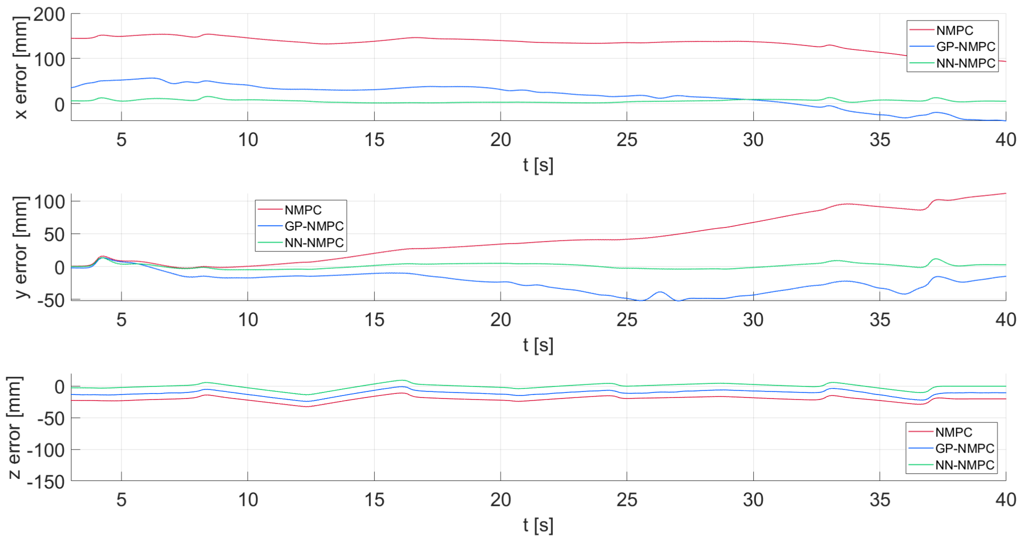


Figure 5.24: Position error comparison between NMPC, GP-NMPC and NN-NMPC in presence of rotor speed mismatch and noisy learning target. The learning-based models are trained with zero-phase filtered targets.

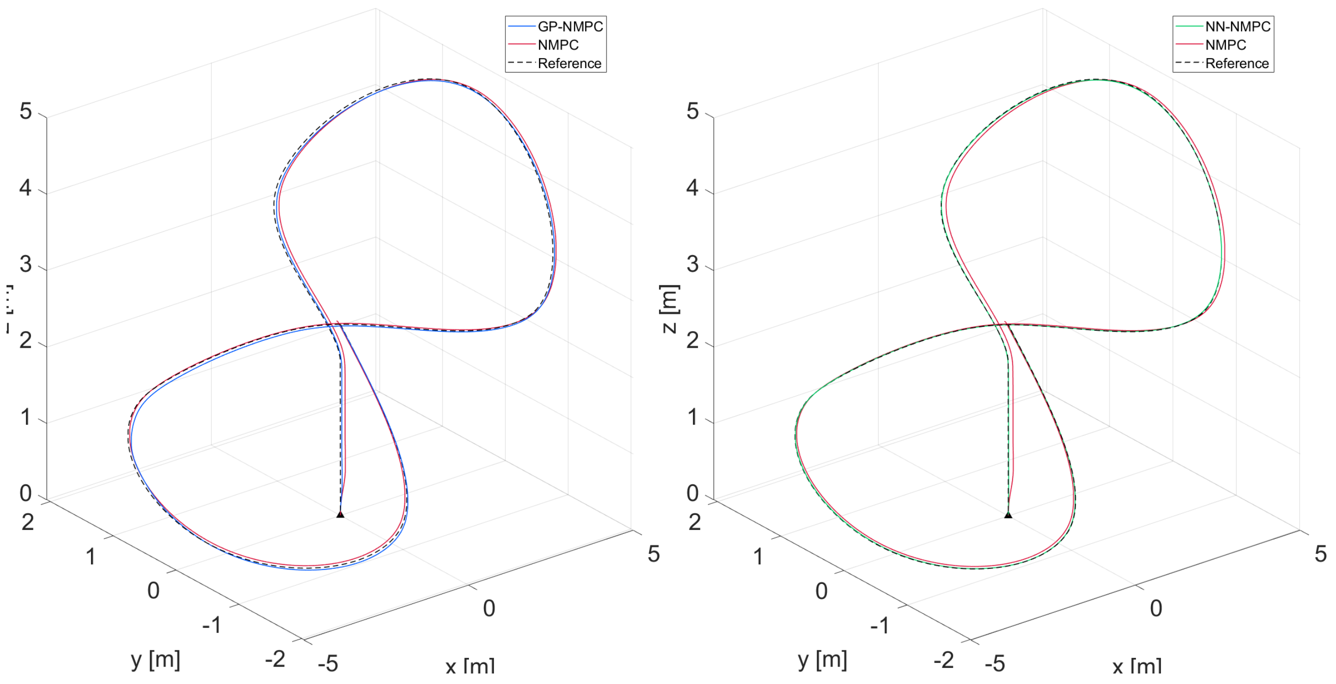


Figure 5.25: GP-NMPC and NN-NMPC simulation results in case of rotor speed mismatch, with grey-box models trained with a zero-phase filtered version of the noisy learning target. While NN-NMPC performs an almost perfect trajectory tracking, the trajcetory obtained with GP-NMPC presents some slight offset with respect to the reference.

Indeed, since the GP regression model takes into account the presence of additive Gaussian noise on the learning target, the introduction of a filter on such noise results is a violation of the model assumptions. One can better understand how the GP prediction is deteriorated also by taking into account the resulting standard deviation of the GP posterior distribution. In fact, while the posterior mean offers a maximum a posteriori estimator of the target value, the posterior standard deviation offers a measure of the confidence of such prediction, as the posterior distribution is itself a Gaussian distribution. In particular, the predicted value is expected not to be more than one standard deviation away from the posterior mean with a 68.3% probability. By looking at Figure 5.26, which shows the posterior standard deviation for the prediction of the z linear component and the y angular component, it can be noticed how such value is consistently higher for the case in which a filtered target has been adopted during the training procedure. Hence, the prediction made by learning on a filtered learning target with the considered GP model is not only less accurate, but also more uncertain. In conclusion, in presence of noisy learning target the NN-NMPC approach should be trained on zero-phase filtered data, while the GP-NMPC grants better results if trained on a noisy version of the target.

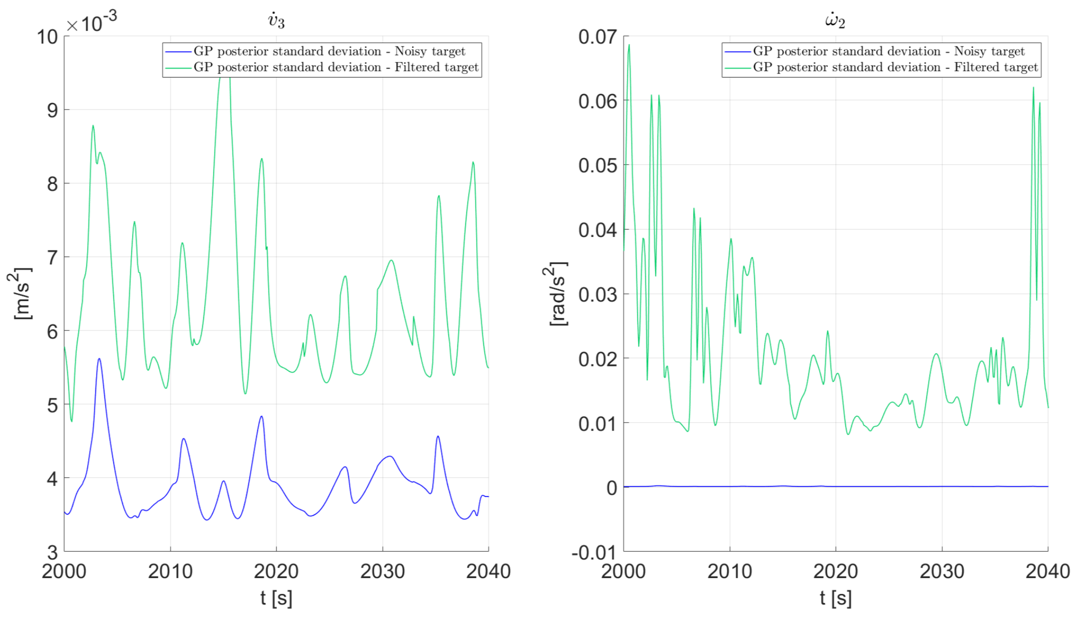


Figure 5.26: Comparison between standard deviation of the posterior distribution of the GP model for the prediction of z linear component and y angular component of the drone acceleration along the test trajectory, in case a noisy learning target or its zero-phase filtered version is used during training.

6 Conclusions

The main scope of this thesis was to investigate the innovative field of Learning-based Nonlinear Model Predictive Control, with focus on the position control of a quadrotor platform. This learning-based advanced control aims at exploiting the high performance of a NMPC approach while dealing with system dynamics that are unpredicted by the nominal prediction model of the controller, thanks to the implementation of data-driven techniques. In particular, GP and NN regression methodologies were here adopted to obtain a correction law for the NMPC prediction model, starting from measurements of the residual error between true and predicted acceleration of the quadrotor platform. Grey-box models derived from the combination of said correction law and quadrotor nominal model were subsequently adopted as novel prediction models for the control algorithm. This control approach was tested in the MATLAB Simulink environment, thanks to an NMPC implementation based on the open-source toolbox MATMPC. The main key points of this research are hence summarized and discussed in the following, before looking into future developments of this work.

6.1 Conclusive remarks

Quadrotor control has been a topic of great interest in robotic research in the past years, thanks to the wide applicability of such mobile platform in customer and industrial contexts. These systems, characterized by nonlinear and fast dynamics, are well suited to be controlled with the NMPC approach. However,

the performance of such control method are highly dependent on the quality of the prediction model adopted by the controller. This notion was also confirmed in the context of this thesis, as it was observed in simulations that an incorrect knowledge of the drone mass or an unexpected speed reduction of one of the drone rotors lead to significant performance deterioration. In order to tackle these undesired effects, learning-based correction laws can be adopted to improve the precision of the controller prediction model. If GP regression models were already a consolidated approach to the problem, in recent years much attention has been given to novel NN-based solutions. In order to deepen the understanding on these approaches, in this work both were implemented in the MATMPC environment and tested on the same problem.
In particular, a GP-based and a NN-based regression models were trained in an off-line fashion on a dedicated trajectory, in order to learn a continuous time correction law for the nominal dynamical model of the considered quadrotor platform. The GP regression offers high flexibility and great generalization capability. However, great attention was necessary to handle the computational burden that is typical of this approach. In fact, since such method cannot be carelessly implemented in time-sensitive scenarios like the Nonlinear Model Predictive Control of a multirotor platform, it was necessary to rely on a popular sparse VFE approximation of the GP model. On the other hand, the NN architectures required challenging design procedures, due to the high number of configuration choice to be made and to the low interpretability that characterize this kind of learning models. Both GP and NN models achieved satisfactory learning results, in terms of accuracy on off-line predictions. In particular, the GP models performed almost perfect dynamical predictions, while NN models achieved good predictions only on the components of the learning target that presented the most significant mismatches. However, it was clear from in-between testing that the learning results not always grant a good understanding of the performance that will be achieved in closed-loop control with Lb-NMPC methods.
After training, Lb-NMPC implementations that adopted the considered learning models were tested in simulation on a trajectory tracking task. These controllers were denominated GP-NMPC and NN-NMPC. Both implementations achieved satisfactory reduction of the position error with respect to the reference trajectory, both in presence of mass mismatch and rotor speed mismatch. In particular, NN-NMPC managed to achieved similar results to GP-NMPC, while granting a lower computational burden.

These Lb-NMPC methods were also tested in non-ideal learning scenarios. First, the presence of a dynamic delay on the control performance was taken into account. In this case, NN-NMPC obtained better results than GP-NMPC in case a delayed control signal data was used as learning information. Moreover, both method achieved good results in case an additive Gaussian noise was introduced on the learning target used during training. In such case, NN-NMPC was shown to benefit from the application of a zero-phase filter on the learning target noisy measurements, while GP-NMPC achieved better performance when the GP models were trained directly on noisy data. This was not surprising, as GP regression models naturally account for the presence of additive Gaussian noise on the learning target.

6.2 Future work

Starting from the promising results delivered in this thesis for the application of GP-NMPC and NN-NMPC to the control of quadrotor platforms, some future developement of this research work can be outlined:

- *Testing on realistic and experimental setups*: throughout this thesis, the considered methods were tested only in fairly simplified simulation environments. A better understanding of the considered technologies can come from testing them in more complex simulation environments, among which the Gazebo robotic simulator stands out distinctly [49]. Following the same idea, the Lb-NMPC approach could be tested also on real world experimental setups.

- *Research on dedicated NN architectures*: NN design is a wide and complex framework, that was only partially explored for the application of interest of this thesis. The promising results that NN-NMPC has achieved in this project can potentially open the door to a research devoted to design dedicated networks for Lb-NMPC applications and for quadrtoror NMPC in particular.

- *On-line learning in Lb-NMPC implementations*: in this thesis project, the adopted learning methods were used to learn the unknown dynamics of the system in an off-line fashion, applying the resulting models to NMPC only after a training procedure. For this reason, the considered Lb-NMPC approaches would not be able to tackle time-varying unmodeled dynamics. Both GP-NMPC and NN-NMPC for on-line learning could be developed. In particular, *continual learning*, namely the problem arising from the necessity to adapt trained learning models to new information while preserving what was previously learned, is a topic of great interest for NN architectures [38].

References

[1] Joel A. E. Andersson et al. “CasADi: a software framework for nonlinear optimization and optimal control”. In: *Mathematical Programming Computation* (2019).

[2] Matthias Bauer, Mark van der Wilk, and Carl Edward Rasmussen. “Understanding probabilistic sparse Gaussian process approximations”. In: *Advances in neural information processing systems* 29 (2016).

[3] Alberto Bemporad, Manfred Morari, and N Lawrence Ricker. “Model predictive control toolbox”. In: *User's Guide, Version* 2 (2004).

[4] Badr Elaamery Beniamino Pozzan and Angelo Cenedese. “Non-Linear Model Predictive Control for autonomous landing of a UAV on a moving platform”. In: *2022 IEEE Conference on Control Technology and Applications (CCTA)* (2022), pp. 1240–1245.

[5] John T. Betts and Joerg M. Gablonsky. “A comparison of interior point and sqp methods on optimal control problems”. In: (2002).

[6] Marco Concetto Bonazza. “Implementation of adaptive nonlinear model predictive control on a PX4-enabled quad-rotor platform”. In: (2023).

[7] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[8] J.C. Butcher. “A history of Runge-Kutta methods”. In: *Applied Numerical Mathematics* 20.3 (1996), pp. 247–260.

[9] C. K. I. Williams C. E. Rasmussen. *Gaussian Processes for Machine Learning*. MIT Press, 2005.

[10] Bárbara Barros Carlos et al. "An efficient real-time NMPC for quadrotor position control under communication time-delay". In: *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE. 2020, pp. 982–989.

[11] Kong Yao Chee, Tom Z Jiahao, and M Ani Hsieh. "Knode-mpc: A knowledge-based data-driven predictive control framework for aerial robots". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 2819–2826.

[12] Yutao Chen et al. "Algorithms and applications for nonlinear model predictive control with long prediction horizon". In: (2018).

[13] Yutao Chen et al. "MATMPC - A MATLAB Based Toolbox for Real-time Nonlinear Model Predictive Control". In: (2019), pp. 3365–3370.

[14] PDP Research Group D. E. Rumelhart J. L. McClelland. *Parallel Distributed Processing: Exporations in the Microstructure of COgnition*. MIT Press, 1986.

[15] Vishnu R Desaraju et al. "Leveraging experience for robust, adaptive nonlinear MPC on computationally constrained systems with time-varying state uncertainty". In: *The International Journal of Robotics Research* 37.13-14 (2018), pp. 1690–1712.

[16] Moritz Diehl, Hans Georg Bock, and Johannes P Schlöder. "A real-time iteration scheme for nonlinear optimization in optimal feedback control". In: *SIAM Journal on control and optimization* 43.5 (2005), pp. 1714–1736.

[17] Moritz Diehl et al. "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations". In: *Journal of Process Control* 12.4 (2002), pp. 577–585.

[18] Hand Joachim Ferreau et al. "Embedded optimization methods for industrial automatic control". In: *IFAC-PapersOnLine* 50.1 (2017), pp. 13194–13209.

[19] Hans Joachim Ferreau et al. "qpOASES: a parametric active-set algorithm for quadratic programming". In: *Mathematical Programming Computation* 6 (2014), pp. 327–363.

[20] E. Fornasini. *Appunti di Teoria dei Sistemi*. Libreria Progetto Padova, 2011.

[21] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.

[22] Fredrik Gustafsson. "Determining the initial states in forward-backward filtering". In: *IEEE Transactions on signal processing* 44.4 (1996), pp. 988–992.

[23] Shih-Ping Han. "Superlinearly convergent variable metric algorithms for general nonlinear programming problems". In: *Mathematical Programming* 11.1 (1976), pp. 263–282.

[24] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. "An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range". In: *Automatica* 47.10 (2011), pp. 2279–2285.

[25] Naira Hovakimyan et al. "L 1 adaptive control for safety-critical systems". In: *IEEE Control Systems Magazine* 31.5 (2011), pp. 54–104.

[26] A. Courville I. Goodfellow Y. Bengio. *Deep Learning*. MIT Press, 2016.

[27] M. Diehl J. B. Rawlings D. Q. Mayne. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[28] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[29] Dimitris Kouzoupis et al. "Towards proper assessment of QP algorithms for embedded model predictive control". In: *2015 European Control Conference (ECC)*. IEEE. 2015, pp. 2609–2616.

[30] S. Krogstad. "Generalized integrating factor methods for stiff PDEs". In: *Journal of Computational Physics* 203.1 (2005), pp. 72–88.

[31] M. Menner L. Hewing K. P. Wabersich. "Learning-Based Model Predictive Control: Toward Safe Learning in Control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3.1 (2020), pp. 269–296.

[32] R. Gamkrelidze L. Pontryagin V. Boltyanskii. *The Mathematical Theory of Optimal Processes*. Inderscience Publishers, 1962.

[33] Haitao Liu et al. “When Gaussian Process Meets Big Data: A Review of Scalable GPs”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.11 (2020), pp. 4405–4423.

[34] H. Hamadah M. G. Forbes R. S. Patwardhan. “Model Predictive Control in Industry: Challenges and Opportunities”. In: *IFAC-PapersOnLine* 48.8 (2015). 9th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2015, pp. 531–538.

[35] Nicoló Alvise Marin. “Adaptive and Learning-based NMPC Strategies for Quadrotor Control”. In: (2023).

[36] 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)*. IEEE. 2020, pp. 10–16.

[37] Fang Nan et al. “Nonlinear MPC for quadrotor fault-tolerant control”. In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 5047–5054.

[38] German I Parisi et al. “Continual lifelong learning with neural networks: A review”. In: *Neural networks* 113 (2019), pp. 54–71.

[39] Enrico Picotti et al. “Continuous- Time Acceleration Modeling through Gaussian Processes for Learning-based Nonlinear Model Predictive Control”. In: (2022), pp. 1228–1233.

[40] Enrico Picotti et al. “LbMATMPC: an open-source toolbox for Gaussian Process modeling within Learning-based Nonlinear Model Predictive Control”. In: (2022), pp. 736–742.

[41] Jintasit Pravitra et al. “ 1-Adaptive MPPI Architecture for Robust and Agile Control of Multirotors”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 7661–7666.

[42] Rien Quirynen, Stefan Vandewalle, and Moritz Diehl. “Numerical simulation methods for embedded optimization”. In: (2017).

[43] Carl Edward Rasmussen and Hannes Nickisch. “Gaussian processes for machine learning (GPML) toolbox”. In: *The Journal of Machine Learning Research* 11 (2010), pp. 3011–3015.

[44] Enrica Rossi et al. “Online nonlinear model predictive control for tethered uavs to perform a safe and constrained maneuver”. In: *2019 18th European Control Conference (ECC)*. IEEE. 2019, pp. 3996–4001.

[45] C. Runge. "Ueber die numerische Auflösung von Differentialgleichungen". In: *Mathematische Annalen* 46 (1895), pp. 167–178.

[46] Tim Salzmann et al. "Real-time neural MPC: Deep learning model predictive control for quadrotors and agile robotic platforms". In: *IEEE Robotics and Automation Letters* 8.4 (2023), pp. 2397–2404.

[47] Alessandro Saviolo, Guanrui Li, and Giuseppe Loianno. "Physics-inspired temporal learning of quadrotor dynamics for accurate model predictive trajectory tracking". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 10256–10263.

[48] Sihao Sun et al. "A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight". In: *IEEE Transactions on Robotics* 38.6 (2022), pp. 3357–3373.

[49] Kenta Takaya et al. "Simulation environment for mobile robots testing using ROS and Gazebo". In: *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE. 2016, pp. 96–101.

[50] Guillem Torrente et al. "Data-driven MPC for quadrotors". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3769–3776.

[51] Stephen J Wright. *Numerical optimization*. 2006.