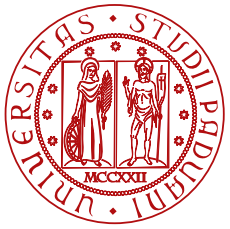

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Control Systems Engineering

Adaptive and Learning-based NMPC Strategies for Quadrotor Control

Master Candidate

Nicolò Alvise Marin

Student ID 2037862

Supervisor

Prof. Angelo Cenedese

University of Padova

Co-supervisor

Prof. Mattia Bruschetta

University of Padova

Academic Year
2022/2023

To my parents,
family and friends.
Thanks to everyone that
supported me during these years.
To everyone that cheered me, listened to
me or was just present when I needed them.
Thanks!

Abstract

Thanks to hardware and software evolution, machine learning implementations have become more viable. Nowadays the possibility to handle precise data acquisition and high computational capabilities is available also for embedded devices used in drones.
This thesis aims at implementing and comparing the results of an Adaptive Nonlinear Model Predictive Controller (Adaptive+NMPC) and two Learning-based NMPC (Lb-NMPC) methods, in a trajectory tracking task for a quadrotor, where different types of disturbances and model mismatches need to be rejected. The Lb-NMPC approaches are respectively based on Gaussian Processes and Neural Networks.

Contents

List of Figures

List of Tables

List of Acronyms

w.r.t. with respect to

s.t. such that

i.c. initial conditions

UAV Unmanned Aerial Vehicle

CoM Center of Mass

CW Clockwise

CCW Counterclockwise

DOF Degree of Freedom

LQR Linear Quadratic Regulator

RHC Receding Horizon Control

MPC Model Predictive Control

NMPC Nonlinear Model Predictive Control

MRAC Model Reference Adaptive Controller

Lb Learning-based

NN Neural Network

FFNN Feedforward Neural Network

MSE Mean Squared Error

GP Gaussian Process

GPR Gaussian Process Regression

VFE Variational Free Energy

MVM Motor Voltage Mismatch

CPT Computational Time

DM Dangerous Manoeuvre

1 Introduction

1.1 Topic Introduction

In recent years, Nonlinear Model Predictive Control (NMPC) has widely spread both in research and industrial contexts. This advanced control technique is in fact able to provide interesting control actions in several high performance scenarios.
NMPC reliability, however, depends on the accuracy of the model description, hence in case of model mismatches the system performances can be subject to considerable uncertainty and can deteriorate significantly.

On the other hand, in the last years, a new approach has been tested in order to mitigate this problem and to try making the control algorithm more robust.
This is the $\mathcal{L}_1$ Adaptive Approach. The idea behind this technique is to complement the basic NMPC with an adaptive controller in order to have an online tool to reject model mismatches.
The structure of this adaptive controller is provided through the union of a Model Reference Adaptive controller (MRAC) and a Low Pass Filter (LPF) in order to create an explicit and easily tunable trade-off between robustness and performance.

Another interesting aspect is provided by the advances of the machine learning techniques, that led to a renewed interest in data-driven control strategies for complex systems, defining a novel research field, namely Learning-based

NMPC.
In this field, there are two main approaches that are of interest for this thesis, the first one is to use Gaussian Processes (GP), known and applied in this kind of context for many years now.
The other one is Neural Networks (NN). NN have been studied for many years, but only recently, thanks to hardware and software evolution, they had a growth in terms of research, with several implementation that aim at learning different elements (e.g. control inputs or model components) in different ways (e.g. disparate types of neural).

In this thesis the objective is trying to establish a fair comparison among the aforementioned approaches for a quadrotor control: $\mathcal{L}_1$-NMPC, GP-NMPC and NN-NMPC. The task will be a trajectory tracking one while trying to compensate for model mismatches.
These methods will be directly inspired from the master thesis of Marco Concetto Bonazza [18], that researched Adaptive-NMPC at UNIPD and the research training on learning-based methods carried out by myself and Filippo Simonetti. This colleague of mine wrote a master thesis [9] on the specific implementation choices that we tested during the previously mentioned research training.

1.2 Thesis Structure

In the following a quick chapter guide is given:

- Chapter 2:

  The mathematical description and formulation of the problem is provided alongside with a table displaying quadrotor parameters used in the simulations.

- Chapter 3:

  The concepts behind Model Predictive Control (MPC) are provided starting from Linear Quadratic Regulator and Receding Horizon Control. After this, the Nonlinear version is presented in terms of concept, core ideas and main elements regarding the Unmanned Aerial Vehicle (UAV) case in analysis.

- Chapter 4:

  Starting from the concept behind adaptive controllers, $\mathcal{L}_1$ approach and its implementation in the control scheme is explained.

- Chapter 5:

  This chapter aims at giving a quick introduction to a learning problem applied to the considered setup. Then both the Gaussian Process and Neural Networks approaches are debated and implemented in the Nonlinear Model Predictive Control (NMPC) structure.

- Chapter 6:

  First the simulation setup and objectives are explained, then the results are presented. In the end a high performance scenario is introduced in order to underline some important differences among the adaptive and learning-based methods.

- Chapter 7:

  In the final chapter a brief summary and comment on the results is provided along with some proposals for possible continuations of this work.

2

Model Definition

In this chapter the quadrotor model used in the discussion and the simulations for this thesis will be introduced. A complete mathematical model will be provided, with a table that displays the model parameters used.

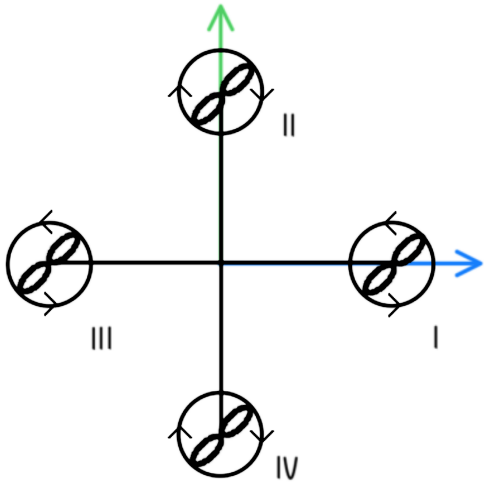

Figure 2.1: Quadrotor + Configuration

2.1 Mathematical Model

In order to implement the control system of the quadrotor, it is necessary to start from the kinematic and dynamic models. These will be implemented in

the following.

The standard quadrotor configuration is a so called + configuration. This name is due to the shape in which the arms are arranged, aligned with the body axes, forming 90°angles. A propeller is placed at the end of each arm as illustrated in Figure 2.1.

It is useful now to introduce the body frame $\mathcal{F}_B$, which has origin O_B in the Center of Mass (CoM) of the platform, and the inertia world frame $\mathcal{F}_W$. It is possible to appreciate these elements in Figure 2.2 (image source [13]).

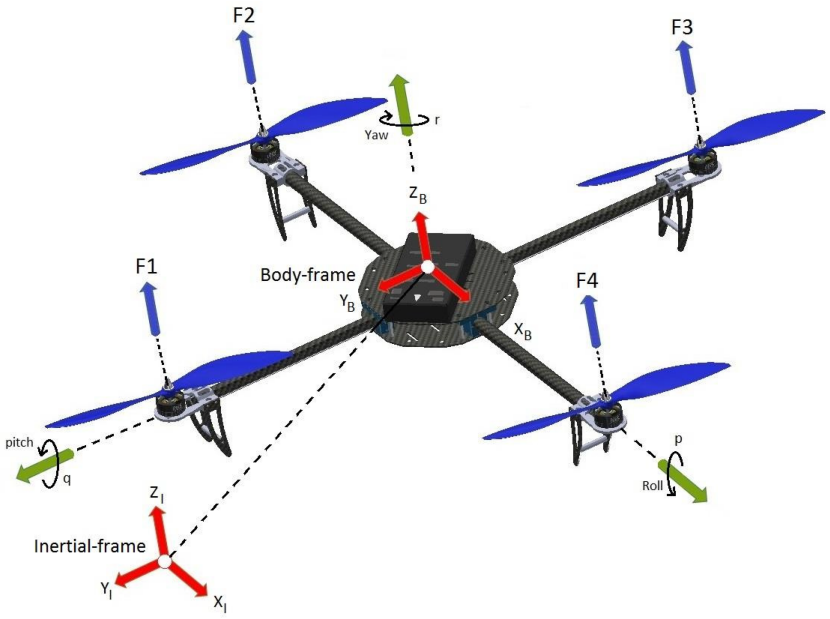

Figure 2.2: Body and World Frame

In order to describe the quadrotor pose, the pair $(\mathbf{p}, \mathbf{R}_{WB}) \in \mathbb{R}^3 \times \mathbb{SO}(3)$ is introduced. Respectively the vector $\mathbf{p}$ and the rotation matrix $\mathbf{R}_{WB}$ are used for the representation of the origin position and the orientation of the body frame with respect to (w.r.t.) the world frame. Other important elements that need to be introduced are the linear velocity $\mathbf{v} \in \mathbb{R}^3$ of the O_B and the angular velocity $\omega_B \in \mathbb{R}^3$ of $\mathcal{F}_B$, both w.r.t. the $\mathcal{F}_W$. The kinematic model of the quadrotor is then given by:

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{v} \\ \dot{\mathbf{R}}_{WB} &= \mathbf{R}_{WB}[\omega_B]_\times \end{aligned} \tag{2.1}$$

For what concerns the dynamic model, if gravity is not considered, most of the remaining forces and torques on the quadrotor are generated by the four propellers. Each propeller spins around its axis (parallel to the z body-frame axis), with a controllable angular speed of ω_i (with $i = 1, 2, 3, 4$ that is the number corresponding to each propeller).
If the propeller spins Counterclockwise (CCW) the velocity vector is pointing towards the positive direction of the axis, while if the propeller spins Clockwise (CW) the vector will be pointing towards the negative direction. The control input will be defined $u_i = \omega_i^2 \in \mathbb{R}$ meaning that the thrust of each propeller is proportional to the square angular spinning velocity of its respective motor.
Several effects will act on the drone thanks to the propellers rotation.
A thrust force w.r.t. the $\mathcal{F}_B$ equal to

$$\mathbf{f}_i = c_{f_i} u_i \hat{\mathbf{k}}_{B_i} \in \mathbb{R}^3 \tag{2.2}$$

where c_{f_i} is the thrust coefficient, $\hat{\mathbf{k}}_{B_i}$ is a versor parallel to the z body-frame axis and overlapped with the rotation axis.
This force generates a force moment equal to

$$\tau_i = \mathbf{p}_i \times \mathbf{f}_i = \mathbf{p}_i \times c_{f_i} u_i \hat{\mathbf{k}}_{B_i} \in \mathbb{R}^3 \tag{2.3}$$

where $\mathbf{p}_i \in \mathbb{R}^3$ is the vector that represents the position of the spinning axis w.r.t. the O_B.
Due to the air friction a drag torque is generated w.r.t. the $\mathcal{F}_B$. This torque opposes the angular velocity of each propeller and it is equal to:

$$\tau_i^d = c_{\tau_i} u_i \hat{\mathbf{k}}_{B_i} \in \mathbb{R}^3 \tag{2.4}$$

where c_{τ_i} is the drag coefficient. This value will be positive if the rotor spins CW and negative if it spins CCW.
By combining (2.2), (2.3) and (2.4) it is possible to obtain the expressions of the

total force $f_c \in \mathbb{R}^3$ and total torque $\tau_c \in \mathbb{R}^3$ applied on the CoM :

$$\begin{aligned} f_c &= \sum_{i=1}^{4} f_i = \sum_{i=1}^{4} c_{f_i} u_i \hat{\mathbf{k}}_{B_i} \\ \tau_c &= \sum_{i=1}^{4} (\tau_i^t + \tau_i^d) = \sum_{i=1}^{4} (c_{f_i} \mathbf{p}_i \times \hat{\mathbf{k}}_{B_i} + c_{\tau_i} \hat{\mathbf{k}}_{B_i}) u_i \end{aligned} \tag{2.5}$$

It is possible to rewrite these equations in a more compact form thanks to the introduction of the control force matrix $\mathbf{F} \in \mathbb{R}^{3\times4}$, the control momentum matrix $\mathbf{T} \in \mathbb{R}^{3\times4}$ and the input vector $\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T \in \mathbb{R}^4$

$$\begin{aligned} f_c &= \mathbf{F}\mathbf{u} \\ \tau_c &= \mathbf{M}\mathbf{u} \end{aligned} \tag{2.6}$$

Assuming that all arms of the quadrotor share the same length L as well as all the propellers share the same thrust and drag coefficients (i.e. $c_f = c_{fi}$ and $c_\tau = c_{\tau i}$ $\forall i = 1, 2, 3, 4$) it is possible to express $\mathbf{F}$ and $\mathbf{M}$ like so:

$$\begin{aligned} \mathbf{F} &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ c_f & c_f & c_f & c_f \end{bmatrix} \\ \mathbf{M} &= \begin{bmatrix} 0 & c_f L & 0 & -c_f L \\ -c_f L & 0 & c_f L & 0 \\ -c_\tau & c_\tau & -c_\tau & c_\tau \end{bmatrix} \end{aligned} \tag{2.7}$$

For simplicity, additional aerodynamic phenomena and gyroscopic and inertial effects will be considered as negligible. Now, thanks to the Newton-Euler equations it is possible to write the dynamics of the drone w.r.t. the $\mathcal{F}_W$:

$$\begin{aligned} m\ddot{\mathbf{p}} &= -mg\hat{\mathbf{k}} + \mathbf{R}_{WB} f_c \mathbf{u} = -mg\hat{\mathbf{k}} + \mathbf{R}_{WB}\mathbf{F}\mathbf{u} \\ \mathbf{J}\dot{\omega}_B &= -\omega_B \times \mathbf{J}\omega_B + \tau_c = -\omega_B \times \mathbf{J}\omega_B + \mathbf{M}\mathbf{u} \end{aligned} \tag{2.8}$$

with g gravitational acceleration, m mass of the quadrotor and $\mathbf{J} \in \mathbb{R}^{3\times3}$ positive definite inertia matrix.

By combining both kinematics (2.1) and dynamics (2.8), the full mathematical model is obtained:

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{v} \\ \dot{\mathbf{R}}_{WB} &= \mathbf{R}_{WB}[\omega_B]_X \\ \dot{\mathbf{v}} &= -g\hat{\mathbf{k}} + \frac{1}{m}\mathbf{R}_{WB}\mathbf{F}\mathbf{u} \\ \dot{\omega}_B &= \mathbf{J}^{-1}(-\omega_B \times \mathbf{J}\omega_B + \mathbf{M}\mathbf{u}) \end{aligned} \tag{2.9}$$

As a conclusion of this section it is worth to point out how the system is underactuated. This makes sense since the quadrotor is coplanar and every coplanar drone is underactuated. In fact full actuation is reach when each of the 6 (DOFs) is accessible through the input, but

$$rank\begin{bmatrix} \frac{1}{m}\mathbf{R}_{WB}\mathbf{F} \\ \mathbf{J}^{-1}\mathbf{M} \end{bmatrix} = rank\left(\begin{bmatrix} \frac{1}{m}\mathbf{R}_{WB}\mathbf{F} & \mathbf{0}_{3x1} \\ \mathbf{0}_{3x1} & \mathbf{J}^{-1} \end{bmatrix}\begin{bmatrix} \mathbf{F} \\ \mathbf{M} \end{bmatrix}\right) = rank\begin{bmatrix} \mathbf{F} \\ \mathbf{M} \end{bmatrix} = 4 < 6 \tag{2.10}$$

It also results that $rank(\mathbf{M}) = 3$, hence rotational DOFs are all controllable independently, but $rank(\mathbf{F}) = 1$, hence it is possible to control only the z component translational Degree of Freedom (DOF). This means that in order to move the drone on the x-y world plan it is necessary to change the quadrotor orientation. In this way the z direction of the body changes towards the desired acceleration direction.

2.2 Model Parameters

All the simulations carried out in the following chapters of this thesis will consider the previous described mathematical model for the quadrotor. In the Table 2.1 are reported the quadrotor parameters that are used in the simulations in Chapter 6.

Element	**Symbol [Unit]**	**Value**
Mass	m $[kg]$	1.5
Arm Length	L $[m]$	0.255
Moment of Inertia	J $[Kg \cdot m^2]$	diag[29, 29, 55]10^{-3}
Saturation Level	u_{sat} $[(rad/s)^2]$	91
Thrust Coefficient	c_f $[Nms^2]$	0.06
Drag Coefficient	c_τ $[Ns^2]$	4.28 10^{-6}

Table 2.1: Quadrotor Parameters

3

Model Predictive Control

Let's consider a continuous time dynamical system represented by the equation

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

where $\mathbf{x}_0 \in \mathbb{R}^n$ is the initial state, $\mathbf{x} \in \mathbb{R}^n$ is the state and $\mathbf{u} \in \mathbb{R}^m$ is the input of the system.
The aim of the optimal control is to find the sequence of inputs ($\mathbf{u}^*(t)$ with $t \in [0, +\infty[$ in the infite horizon scenario) such that (s.t.) the cost function, defined as follows, is minimized:

$$J_\infty(\mathbf{x}, \mathbf{u}) = \int_0^\infty V(\mathbf{x}(t), \mathbf{u}(t))dt \tag{3.2}$$

The problem may not be easy to solve, but it is possible to demonstrate that if $V(\cdot)$ is positive definite and regular enough and $\mathbf{f}(\cdot)$ is regular enough itself, then the control $\mathbf{u}^*(t)$ that minimizes 3.2, stabilizes the origin of the system for initial conditions (i.c.) in a neighborhood of $\mathbf{x}_0$.

3.1 Linear Quadratic Regulator

In a linear time-invariant system, with $\mathbf{A} \in \mathbb{R}^{nxn}$ as state matrix and $\mathbf{B} \in \mathbb{R}^{nxm}$ as control matrix, the minimization problem becomes:

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \quad & \int_0^\infty \mathbf{x}^T(t)\mathbf{Q}\mathbf{x}(t) + \mathbf{u}^T(t)\mathbf{R}\mathbf{u}(t)dt \\ s.t. \quad & \dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ & \mathbf{x}(0) = \mathbf{x}_0 \end{aligned} \tag{3.3}$$

where $\mathbf{Q} \geq 0 \in \mathbb{R}^n$ is the state weight matrix, while $\mathbf{R} > 0 \in \mathbb{R}^n$ is the control cost weight matrix. Both these matrices are symmetric. Let $\mathbf{Q}^{\frac{1}{2}} \in \mathbb{R}^n$ be a matrix s.t. $\mathbf{Q} = \mathbf{Q}^{\frac{1}{2}T}\mathbf{Q}^{\frac{1}{2}}$, if and only if $(\mathbf{A}, \mathbf{B})$ is stabilizable and $(\mathbf{A}, \mathbf{Q}^{\frac{1}{2}})$ is observable it holds that the Algebraic Ricatti Equation (ARE)

$$\mathbf{A}^T\mathbf{P} + \mathbf{Q} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} = 0 \tag{3.4}$$

has a unique solution $\mathbf{P}_\infty > 0$ while if the pair $(\mathbf{A}, \mathbf{Q}^{\frac{1}{2}})$ is detectable $\mathbf{P}_\infty \geq 0$. The feedback law solving 3.3 and making the sytem asymptoticallu stable is given by

$$\mathbf{u}^*(t) = -\mathbf{K}\mathbf{x}(t) \quad with \quad \mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}_\infty \tag{3.5}$$

But this approach is not flawless. In fact, since the control loop is open-loop, the control loses its optimality. This can happen if the true evolution of the system differs from the predicted one even slightly, depending on the accuracy of the model's dynamics. Another problem is that this approach does not work particularly well for nonlinear elements (e.g. systems or cost functions), which could make the problem impossible to solve. Furthermore, even in the linear model scenario, this type of control requires a, not always immediate to implement, specific constraint management.

3.2 Receding Horizon Approach

In order to face these flaws Receding Horizon Control (RHC) was introduced. The optimal control problem shifts from a infinite horizon to a finite one in

order to make numerical approximate solutions possible. Furthermore, to tackle computational difficulties, dynamical systems are written in discrete-time.
The reformulation of 3.3 takes the following form:

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \quad & \sum_{k=0}^{N-1} J(\mathbf{x}(t), \mathbf{u}(t)) + J_N(\mathbf{x}(t)) \\ s.t. \quad & \mathbf{x}(t+1) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \\ & \mathbf{x}(0) = \mathbf{x}(t_{start}) \\ & \mathbf{x}(k) \in \mathcal{X} \\ & \mathbf{u}(k) \in \mathcal{U} \end{aligned} \tag{3.6}$$

where the condition $\mathbf{x}(0) = \mathbf{x}(t_{start})$ implies that the initial condition of the problem must coincide with the system's conditions at the beginning of each control horizon. $J(\mathbf{x}(\cdot), \mathbf{u}(\cdot))$ is the new cost function considered at each time step while $J_N(\mathbf{x}(t))$ is the cost terminal cost function. Finally $\mathcal{X}$ and $\mathcal{U}$ represent respectively the space and control acceptable spaces.
Only the first element of the control sequence that solves the problem (i.e. **u*(1)**, ...,**u*(N)**) is applied to the system, while the rest of the control inputs are discarded. A graphical representation is displayed in Figure 3.1 (source [15]). At the following step, the problem will be solved again with $t_{start} = t_{start} + 1$.

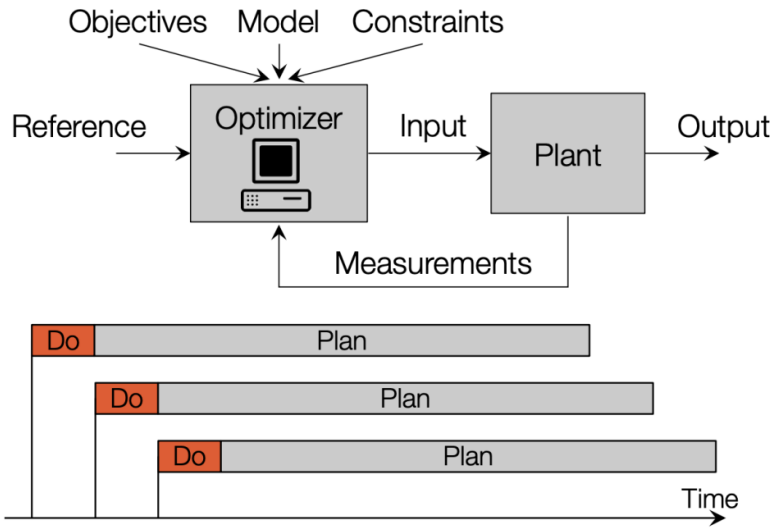

Figure 3.1: Receding Horizon Control Concept

In the following, Algorithm 1 is reported in pseudocode to highlight the main ideas of this approach. RHC practically implements finite horizon Linear Quadratic Regulator (LQR) recursively in order to diminish the effects of model uncertainties.

Algorithm 1 Receding Horizon Control

Require: Control horizon length $N > 0$
  loop
    $\mathbf{x}_0 \leftarrow \mathbf{x}(t_{start})$
    Update $\mathbf{u}^*(\cdot)$ in the interval [t, t+N] with new $\mathbf{x}_0$
    Use first element of $\mathbf{u}^*(t)$
    $t_{start} \leftarrow t_{start} + 1$
  end loop

3.3 MPC

MPC is a particular implementation of RHC where the optimal control law is computed online iteratively over the prediction horizon enforcing state and control constraints. The computation is carried out at each time step, than only the first element of the control series $\mathbf{u}^*$ is applied to the system, while the rest is discarded. A visual representation of this procedure is depicted in Figure 3.2 (source [22]) This control approach is able to achieve considerable performance as well as constraint handling.

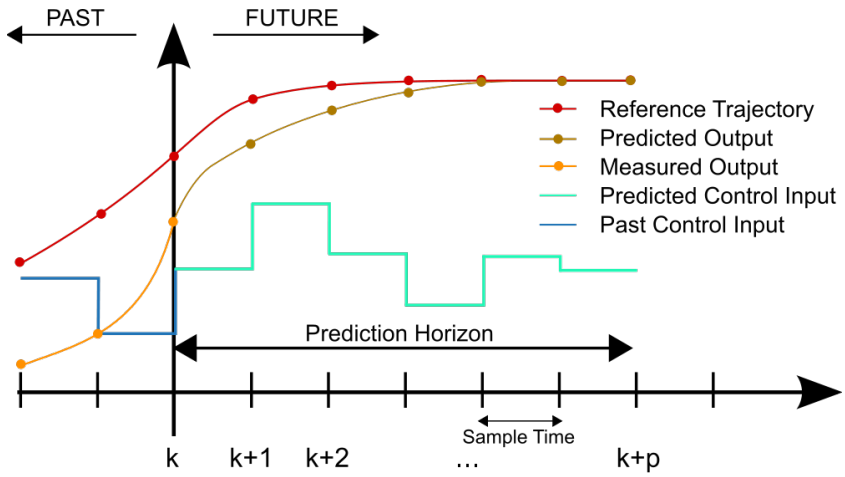

Figure 3.2: Model Predictive Control Visual Example

On the other hand this type of control is very dependant on an accurate model description and does not guarantee stability as well as robustness. Furthermore the constraint introduced may lead the simulation to unfeasibility at some point and the optimization problem needs to be solved in real-time, hence this implies putting particular attention on the processing power of the chosen hardware.

3.4 Nonlinear MPC

Nonlinear MPC or NMPC is a translation of MPC concepts and strategies into the nonlinear world. The approach is conceptually similar but nonlinear dynamic models are used instead of linearized (or linear) ones. This grants meaningful performance improvements. In the past, due to limitations in software and hardware, NMPC industrial applications were basically limited to slow dynamics systems. Nowadays, thanks to improvements in such technical fields and to the introduction of brand new and more efficient algorithms, the applications of NMPC are way more widespread.

3.5 NMPC Implementation Details

While the technical aspects (e.g. the toolbox used) will be discussed in Chapter 6, in this section the implementation details, such as cost function and constraints definitions, will be covered.

3.5.1 System Discretization

The discretization is carried out by MATMPC toolbox that will be describen in Chapter 6. The algorithm selected to carry out this operation is explicit Runge-Kutta, that is one of the state of the art approaches to tackle high nonlinearities.

3.5.2 Cost Function

In order to define a cost function it is necessary to design the state error term

$$\mathbf{e}(k) = \begin{bmatrix} \mathbf{e}_p(k) \\ \mathbf{e}_q(k) \\ \mathbf{e}_v(k) \\ \mathbf{e}_\omega(k) \\ \mathbf{e}_\mathbf{u}(k) \end{bmatrix} \in \mathbb{R}^{17} \tag{3.7}$$

where all the $\mathbf{e}_\cdot(k)$ are the differences between the position, quaternions, linear and angular velocities, controls and the respective references at time instant k. In reality, such an error state definition would be actually meaningless for the quaternion pose representation, making them lose their physical sense.
In this thesis this problem will be overcome by setting to zero the weights related to the difference $\mathbf{e}_q(k)$. This is done since the simulation provided satisfactory results, even without this type of penalization.
Anyway, it is possible to take also the quaternion into account by the modification of the pose error term. For example, if the Frobenius metrics is considered, the error acquires the new meaning of the scaled-down measure of the arc length that connects $\mathbf{q}_{ref}$ and $\mathbf{q}$.

3.5.3 Constraints

In a realistic scenario constraints are everywhere, from the unmovable walls of a room to the actuator limits on the control action. The great advantage in the use of a NMPC is the constraint handling on both the state and the control signals. It is possible to specify them to the controller thanks to simple lower/upper bound relationship.
The simple constraints taken into account in this thesis are:

1. Floor:
   This is a basic state constraint imposing that the drone must not crush on the ground.
   $$0 < z \tag{3.8}$$
2. Saturations:
   Each motor has physical limits that stop it from making a propeller spin over a certain speed.
   $$0 \leq u_i \leq u_{sat} \qquad \forall i = 1, 2, 3, 4 \tag{3.9}$$

where u_{sat} is the maximum control output each motor can produce. Its value is reported in Table 2.1.

3.5.4 Weights and Prediction Horizon

Both of the elements in this subsection are fundamental for the NMPC to work properly. But contrary to the previous considerations, these require specific tuning depending on the task, so they will be briefly introduced here and then numerically made explicit in Chapter 6.

- Weight Matrix:

  $\mathbf{Q} \in \mathbb{R}^{17x17}$ is a diagonal matrix that contains in each diagonal element the weight of the corresponding state and control error. Higher the value, greater the effort of the NMPC to bring the corresponding error to 0. While the terminal Weight Matrix is similar, but does not take into account the controls ($\mathbf{Q}_N \in \mathbb{R}^{13x13}$).
  Hence the cost function at step k will be:

  $$J(\mathbf{e}(k)) = \mathbf{e}^T \mathbf{Q} \mathbf{e} \tag{3.10}$$

  While the terminal cost function is:

  $$J(\mathbf{e}_N) = \mathbf{e}_N^T \mathbf{Q}_N \mathbf{e}_N \tag{3.11}$$

- Prediction Horizon:

  T_h is given by the choice of the number of steps N in a prediction horizon and the time length T_s of each step.

  $$T_h = N T_s \tag{3.12}$$

  Greater is the prediction horizon and the number of steps, better will be the prediction of the system dynamics and heavier will be the computational burden for the controller. Hence a fine tuning is necessary to choose the best trade-off for the task.

3.5.5 Final Problem Formulation

Combining the equations from the previous subsections it is possible to obtain the final formulation of the NMPC optimization problem for the quadrotor:

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \quad & \sum_{k=0}^{N-1} (\mathbf{e}^T \mathbf{Q} \mathbf{e}) + \mathbf{e}_N^T \mathbf{Q}_N \mathbf{e}_N \\ s.t. \quad & 0 < z \\ & 0 \leq u_i \leq u_{sat} \qquad \forall i = 1, 2, 3, 4 \end{aligned} \tag{3.13}$$

4

Adaptive+NMPC

Adaptive techniques are used to act on the controller in order to adapt it to system's variations or to find a more precise model online, starting from an approximated one. Hence it is kind of obvious to understand how some researchers thought that combining a MPC, which presents degrading performance with a non-accurate model, with an adaptive mechanism would lead to performance improvements [5].
In this chapter an adaptive $\mathcal{L}_1$ control scheme will be introduced starting from its core element: a Model Reference Adaptive Controller (MRAC).

4.1 Model Reference Adaptive Controller

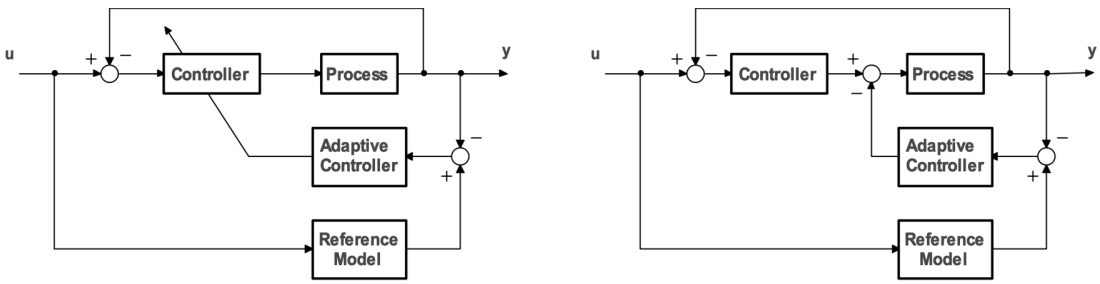

Figure 4.1: Model Reference Adaptive Controller Configurations

MRAC is a control technique that considers the difference between the real system's output and the output obtained from an ideal mathematical model of the system itself. From the result of this comparison the MRAC acts on the

control signal.
The objective is trying to minimize the aforementioned difference. To do so there are two possible types of MRAC to choose from. One that acts directly on the control signal via a sum (Figure 4.1 on the right) and the other that aims at the same result by changing the controller parameters (Figure 4.1 on the left). The first approach will be the one used in the following.
Anyway these types of approaches are sensitive to adaptive gains, that are far from banal to be tuned, hence they tend to produce high-frequency ringing in the control signal, slow convergence rates or large transient errors.

4.2 $\mathcal{L}_1$ Adaptive Approach

In order to prevent these problems a new strategy called $\mathcal{L}_1$ is described in this subsection.
The new structure is composed by the union of the previous adaptive signal with a low pass filter that cuts off the undesired frequencies that may excite unstable modes, regulating the balance between the adaptation contribution and the robustness.
It is possible to prove the stability of the closed loop system with the $\mathcal{L}_1$ control [6]. From this proof derives also the name of the control method. It comes from the fact that the adaptation error bounds depend on several elements among which there is the L1 norm of the filter's transfer function.

4.3 Adaptive+NMPC Cascade

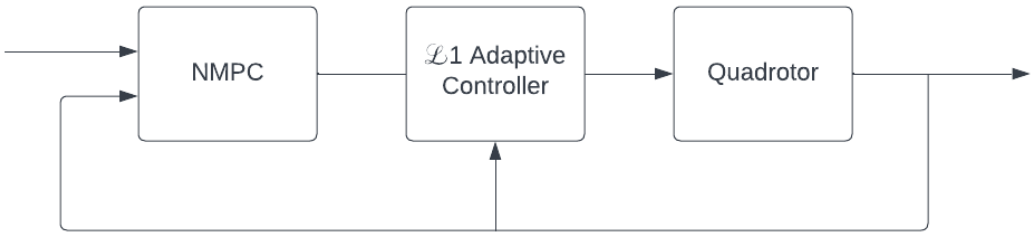

Figure 4.2: $\mathcal{L}_1$+NMPC

In recent years due to the strong dependency from an accurate model of the

NMPC the introduction of a $\mathcal{L}_1$ approach in the control scheme has become a new state of the art control system, able to combine long-term planning and fast adaptation to modeling error and disturbances [14].
Figure 4.2 displays the cascaded control scheme. The quadrotor state is passed through feedback both to the NMPC and to the $\mathcal{L}_1$ block.
On the other hand there are some conceptual problems with this kind of implementation. In fact the adaptive controller does not take into account NMPC constraints while modifying the control output [4]. Furthermore, due to how the implementation is done, there is a loss of the NMPC concept of optimal control.

5

Learning-based NMPC

Thanks to the increasing success of machine learning and due to the constant developments of software and hardware, implementing data driven learning-based approaches in automation systems has become a focus and a trend in the control community. In fact these techniques usually require precise data acquisition and high computational capabilities that were not available a short time ago.
Since NMPC performance deteriorates quickly in case of model uncertainties, there are two main Learning-based (Lb) approaches to complement the NMPC structure in order to marginalize this problem [17]:

1. use machine learning in order to learn unmodeled system dynamics and thus to provide a more refined model to be controlled through NMPC
2. learn directly the NMPC control law

For the purpose of this thesis, the first approach will be the one used to complement the NMPC strategy.

5.1 Introduction to the Learning Problem

As previously mentioned, NMPC results are strictly related with the quality and precision of the model. The problem is that physical models usually are not accurate enough. In fact, in the UAV scenario, phenomena such as aerodynamics or wind effects are often not considered in mathematical models [21].

Thanks to Machine Learning it is possible to learn a more precise model containing the previously mention unmodeled effects, starting from acquired data.
Hence the system becomes:

$$\dot{x} = f(x,u) + g(x,u) \tag{5.1}$$

where the $f(x,u)$ component represents the nominal model while $g(x,u)$ represents the learned dynamics.

In the following some major leaning topics and key decisions for the problem formulation will be introduced.

5.1.1 Learning Methods

For such a task, the two main approaches referenced in literature are Gaussian Process (GP) and Neural Network (NN):

- Gaussian Process:
  given two quantities z_1 and z_2 with $z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \sim \mathcal{N}(\mu, \Sigma)$ (where $\mathcal{N}$ stands for a Gaussian distribution with mean μ and variance Σ), it is possible to obtain an estimation of z_1 from measurements of z_2 thanks to the mathematical relation:
  $$z_1|z_2 \sim \mathcal{N}(E(z_1|z_2), Var(z_1|z_2)) \tag{5.2}$$
  Where $E(\cdot)$ is the expected value and $Var(\cdot)$ is the variance.
- Neural Network:
  having the brain as source of inspiration, a "node" is the basic unit of a NN and it links an input vector $\mathbf{x}$ to an output one $\mathbf{y} = \sigma(\omega^T \mathbf{x})$, where ω in a trainable weight vector and σ is a function (often nonlinear) called "activation function". A group of nodes create a "layer", a NN can be composed of several layers and the output of each node of the previous layer serves as an input of each node of the following one.

In this thesis both the Lb approaches will be considered.

5.1.2 Gray-box and Black-box

The most desirable thing would be carrying out test with an accurate white-box, i.e. the full knowledge of a model, but as it was previously introduced, it is

almost impossible to obtain a perfect accurate model. Thanks to leaning methods it is possible to start from previous knowledge of the model and complete it via learning (gray-box) or learn the full model from data, without any prior knowledge on the model (black-box).

A visual scheme of this distinction is provided by Figure 5.1 (source [20])

Figure 5.1: Concept behind different boxes approach

In this thesis the gray-box approach will be the chosen, since it is also the most common in literature. This is because it makes possible to obtain a fine model with a smaller number of training data, compared to the black-box case. In fact, with sufficient number of data, gray-box and black-box tend to produce similar results [8], but the computational burden of some Lb approaches is strictly related to the number of training data used.

5.1.3 Off-line Learning

In this thesis an off-line approach will be considered. Even though this may be a limiting choice for some scenarios, it is a more simple to implement and straightforward approach, coherent with the aim of this thesis.
Off-line learning consist in acquiring the training data through an UAV flight with only a basic NMPC control strategy. Only after this step, the model will be learned and feed to the system in order to implement a Lb-NMPC strategy.

5.1.4 Continuous and Discrete Formulation

A final decision to be taken is whether to implement the system in a continuous or a discrete formulation [7]:

- Discrete: with this approach the learning is based on the error given by the difference of the nominal speed and the measured one, i.e. the one the system think should be the correct one and the actual speed at which the drone is moving. This approach is considered more efficient, but it is unhandy since it requires to modify the code in case of changes in the NMPC shooting time. Another drawback is that its implementation is complex.

- Continuous: with this approach the learning is based on the error given by the difference of the nominal acceleration (of both the linear and the angular components) and the measured one. Incorporating the learning-based modeling within the accelerations ensures that both the obtained model and the regression process are independent on the time step, hence regression can be accomplished with available data at any frequency. This line of action is more intuitive and easier to implement. For this reason it will be the one chosen for this thesis.

5.1.5 Lb Final Setup

In this subsection all the previously introduced implementation details will be summarize in order to make explicit the final learning setup used in this thesis:

- Both GP and NN will be taken into account
- Gray-box approach
- Off-line learning
- Continuous Implementation

Such an approach should grant a performance improvement for the NMPC by providing a more precise model, even in case of extreme and agile manoeuvres [12].
However the choice of an off-line implementation make the controller robust only for mismatches that are learned during the training, hence if a new mismatch appears during the test flight, these kind of Lb approaches are unable to intervene.
Furthermore, Lb methods tend to introduce high computational burdens for the systems [3], this will imply that a precise trade-off between performance and cost needs to be carefully chosen.

5.2 Gaussian Process

Consider the previously described system

$$\dot{\mathbf{x}} = f(x, u) \tag{5.3}$$

Denoting the model of the system evolution (i.e. nominal dynamics) by

$$\widetilde{\dot{\mathbf{x}}} = \tilde{f}(x, u) \tag{5.4}$$

A physical description is normally used for the derivation of the model, hence the state can be written as:

$$\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]^T \tag{5.5}$$

and where $\mathbf{q}$ are the position and $\dot{\mathbf{q}}$ the velocities of the physical system. From here, it is possible to define the general grey-box model of the system dynamics as follows:

$$\begin{aligned} \widehat{\dot{\mathbf{x}}}(t) &= \dot{\mathbf{x}}(t) + \varphi(t) \\ \begin{bmatrix} \widehat{\dot{\mathbf{q}}} \\ \widehat{\ddot{\mathbf{q}}} \end{bmatrix} &= \begin{bmatrix} \widetilde{\dot{\mathbf{q}}} \\ \widetilde{\ddot{\mathbf{q}}} \end{bmatrix} + \begin{bmatrix} 0 \\ \varphi_{\ddot{q}}(t) \end{bmatrix} \end{aligned} \tag{5.6}$$

where $\varphi_{\ddot{q}}(t)$ is the estimate of the acceleration prediction errors.

5.2.1 Gaussian Process Regression Theory

The idea is to derive an estimator of the mismatch between the measurements and the nominal dynamics, thanks to Gaussian Process Regression (GPR). A different and independent GP will be modeled for each acceleration vector $\ddot{\mathbf{q}}$. At time t_k, each GP takes as input the selected states and the control inputs and outputs $y_k^i = \ddot{q}_k^i - \widetilde{\ddot{q}}_k^i$, where $\ddot{q}_k^i$ is the measured i-th component of $\ddot{\mathbf{q}}$ while $\widetilde{\ddot{q}}_k^i$ is the predicted one.

GPR considers the probabilistic model reported as follows:

$$\mathbf{y}^i = \begin{bmatrix} y_1^i \\ \vdots \\ y_T^i \end{bmatrix} = \begin{bmatrix} \ddot{q}_1^i - \widetilde{\ddot{q}}_1^i \\ \vdots \\ \ddot{q}_T^i - \widetilde{\ddot{q}}_T^i \end{bmatrix} = \begin{bmatrix} \bar{\varphi}_{\ddot{q}}^i(\mathbf{x}_1^{GP}) \\ \vdots \\ \bar{\varphi}_{\ddot{q}}^i(\mathbf{x}_T^{GP}) \end{bmatrix} + \begin{bmatrix} e_1^i \\ \vdots \\ e_T^i \end{bmatrix} = \bar{\varphi}_{\ddot{q}}^i + \mathbf{e}^i \tag{5.7}$$

where $\bar{\varphi}_{\ddot{q}}^i \sim \mathcal{N}(0, K^i)$ is a Gaussian process and $\mathbf{e}^i$ is zero-mean independent gaussian noise with standard deviation σ_n. A kernel function $k^i(\cdot,\cdot)$ defines K^i. Now, let $\mathbf{x}_*^{GP}$ be a general input, its maximum a posteriori estimator is the posterior mean of $\bar{\varphi}_{\ddot{q}}^i(\mathbf{x}_*^{GP})$, since the posterior distribution of $\bar{\varphi}_{\ddot{q}}^i(\mathbf{x}_*^{GP})$ itself is Gaussian.
The closed form expression becomes:

$$\bar{\varphi}_{\ddot{q}}^i(\mathbf{x}_*^{GP}) = \mathbf{k}_*^i \alpha^i \tag{5.8}$$

where

$$\mathbf{k}_*^i = [k^i(\mathbf{x}_*^{GP}, \mathbf{x}_1^{GP}), \ldots, (\mathbf{x}_*^{GP}, \mathbf{x}_T^{GP})]$$
$$\alpha^i = (K^i + \sigma_n^2 \mathbb{I})^{-1} \mathbf{y}^i$$

where $\mathbb{I}$ is the identity matrix of dimension TxT.
Finally, one of the most important steps in GPR is the kernel selection, since it determines a priori the characteristics of $\bar{\phi}_{\ddot{q}}^i$.
In this thesis, the kernel will be the Squared Exponential function:

$$k(x_i^{GP}, x_j^{GP}) = s_f e^{-\frac{(x_i^{GP}, x_j^{GP})^T P^{-1} (x_i^{GP}, x_j^{GP})}{2}} \tag{5.9}$$

where s_f is the signal variance and P is the length-scale matrix of the process.

5.2.2 Variational Free Energy

GPR provides excellent results if many training data are used. The problem is that high number or training data implies high computational burden for the GP. A possible solution is given by Sparse GP Approximations [10].
The technique chosen for this thesis is Variational Free Energy (VFE) [19] approximation. The basic idea behind it is that the training dataset can be compressed in a reduced number of so called inducing points, without loosing information (inducing point assumption).

Considering m inducing points X_u with their corresponding target values f_u and assuming that training values y and test target f_* are independent:

$$p(f, f_*|f_u) = p(f|f_u)p(f_*|f_u) \tag{5.10}$$

Through the choice of these inducing points, VFE aims at minimizing the distance between the exact GP posterior and a variational approximation based on the inducing point themselves.

In Figure 5.2 an example is shown in order to display the effectiveness of VFE. Considered a case of mass mismatch, "Learned Mismatch" is obtained thanks to the GP algorithm with a dataset of 7000 training points, while "Learned Mismatch VFE" is produced via VFE approximation based on only 35 inducing points.
As it is possible to see, the two plot almost overlap, proving the actual effectiveness of VFE.

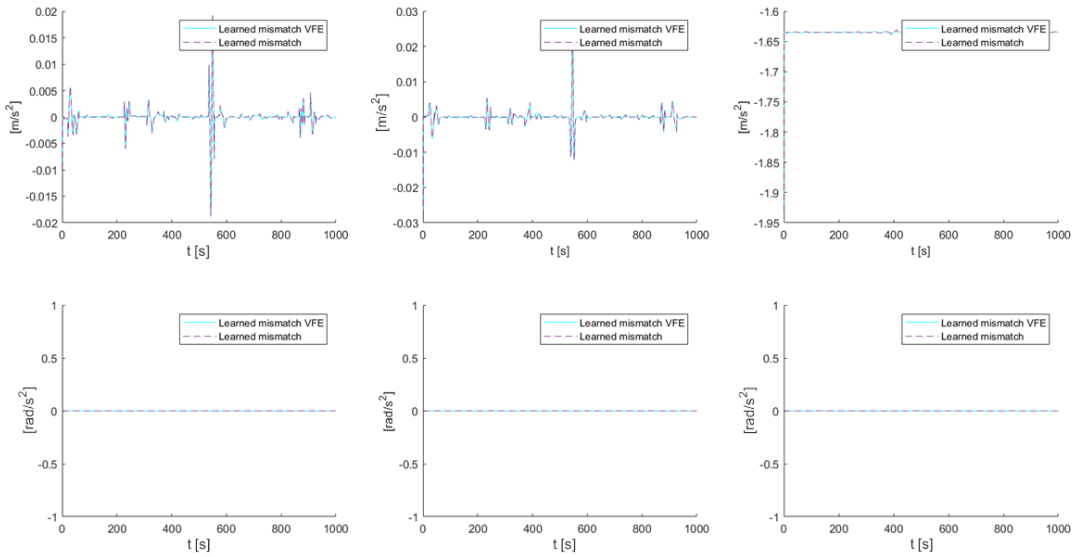

Figure 5.2: Comparison: Learning With and Without VFE. The first row plots are mismatches of linear accelerations along the three directions, while in the second row the angular accelerations are shown

5.2.3 GP-NMPC

In order to implement GP to make the NMPC benefit from the leaning-based component it is sufficient to:

1. Save the training data from a run on the training trajectory and split them in order to keep a small percentage for validation data
2. GP parameters are then obtained maximising the marginal likelihood
3. In the end, training data and GP parameters are used in a new model written in a gray-box formulation like (5.6)

5.3 Neural Networks

Neural Network is a brain inspired technique, where the basic element is called a "node" (or neuron). A group of nodes create a "layer". Each neuron is based on the perceptron algorithm [11], where instead of using a sign function, a non-linear "activation" is chosen (Figure 5.3).

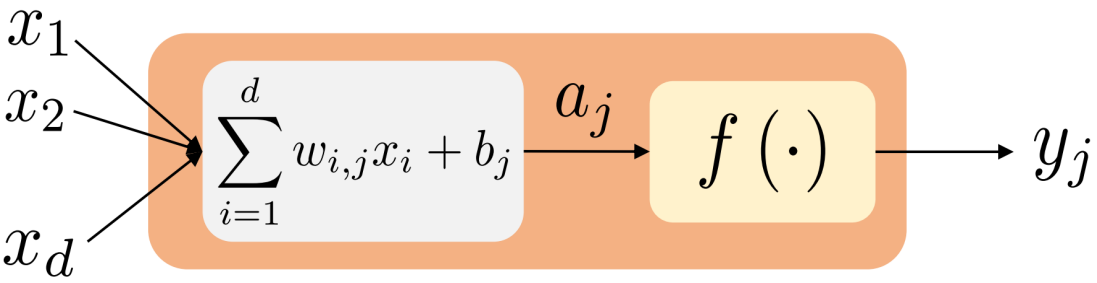

Figure 5.3: Artificial Neuron

In the following a quick theoretical background is provided and structural choices are introduced.
A more in-depth discussion can be found in "Learning based Nonlinear MPC for quadrotor control" [9].

5.3.1 Possible NN Choices

These days Neural Networks are a vast research field in control. For this reason many different types of approaches to NN control have been proposed

in these last few years.
Some of these networks aim at learning different targets [3], some try to exploit previous information to strengthen the learning [1], some other proposals are not about the network structures themselves but about optimization methods in order to avoid the computational complexity when several weights are used [23].
In this thesis project a basic Feedforward NN structure has been selected and the leaning target is the same as the GP in order to guarantee a fair comparison. In the following subsections some more hyperparameter and implementation details about the network at hand are provided.

5.3.2 Feedforward NN Basics

Feedforward Neural Network (FFNN) is the basic network structure, the idea behind it is to stack multiple layers one after the other, hence developing the network in terms of depth. The input vector is fed to a so called hidden layer where each neuron applies an affine transformation ($a = W^{in}x + b^{in}$) and a non-linear activation ($o = f(a)$ applied element-wise). Hence the output is obtained by:

$$y = f(W^{in}o + b^{in}) \tag{5.11}$$

The output is then fed as input to the next hidden layer (Figure 5.4).
The final layer will use an specific output function, chosen depending on the learning problem. In this case the task is regression and a good output function is the Mean Squared Error (MSE) [16]:

$$MSE = \frac{\sum_{i=1}^{n} x_i - \hat{x}_i}{n} \tag{5.12}$$

The training of a FFNN produces as outputs: learned weights and biases. With these new learned elements, the network is able to predict the desired outputs of the regression task, that (in this scenario) are the quadrotor accelerations mismatches.

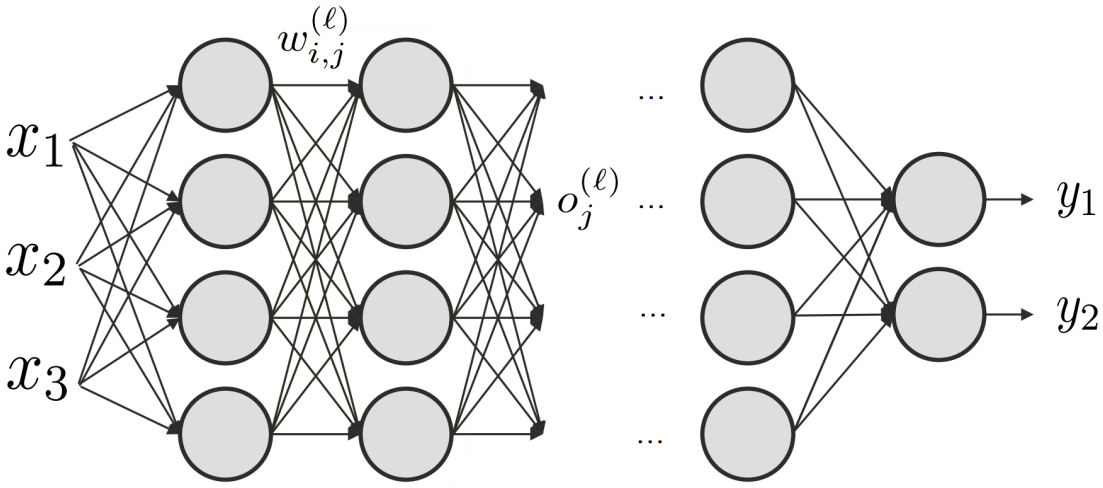

Figure 5.4: Feedforward Neural Network Structure

5.3.3 Network Structure and Specifications

In this subsection several parameter and hyperparameter choices will be introduced.

- **Structure**: 3 layers with 8-16-32 nodes

  Differently from GP the computational weight of the Neural Network is not dependent from the number of training data but by the dimension of the network itself. Theoretically, deeper and wider networks should provide better results, but firstly this is not always true and secondly big networks will cause computational slowdowns or even infeasibility for the setup at hand.

- **Initializer**: narrow-normal

  Choosing the right initializer helps the network to learn easier the weights. For example, initializing all the weights at 0 will make it difficult for the network to learn values different from zero itself.

- **Activation**: Leaky-ReLU

  One of the most crucial choices from which depends the quality of the NN learning.

- **Regularization**: L2 (with a parameter of 10^{-4})

  L2 regularization is one of the most common regularization methods. It imposes the minimization of the squared norm of the weights to the learning process.

- **Training Trajectory**: hand-crafted trajectory with data augmentation

There are several possible picks to train a Neural Network, from random generated to hand-crafted ones. Data augmentation implies the choice of a trajectory and its subsequent expansion (e.g. through some scaling techniques).

- **Normalization**: input normalization

  Normalization is usually carried out in order to help the network learn better equally important mismatches, that usually would have different magnitude scales.

- **Number of Networks**: 2 Networks (one for linear accelerations and one for angular ones)

  While the optimal choice for GP is to learn the mismatch for each acceleration separately, a single NN has proven able to learn multiple mismatches.

5.3.4 NN-NMPC

Similarly to GP, in order to make the NMPC benefit from the mismatches learned from NN it is sufficient to:

1. Save the training data from a basic NMPC run on the training trajectory and split them in order to keep a small percentage for validation data.
2. NN weights and bias are obtained through the network training.
3. In the end, both NN weights and biases are used in a new model written in a gray-box formulation like (5.6).

6 Simulation

In this chapter the simulation setup and results will be introduced. In the first part all the elements that make the final comparison fair (such as trajectory to track, mismatches to reject) are proposed. The same logic will be applied in the Learning-based section, introducing a common training trajectory for both the methods.
The technical setup (e.g. Simulink schemes) will be provided.
In the end the results will be displayed both in the standard, aforementioned case and in a high performance scenario.
The task at hand for these simulation is a trajectory tracking task for the quadrotor, while trying to reject a model mismatch.
For this job three main control schemes are tested and compared. Following the order in which they will be considered, the three approaches are: Adaptive-NMPC, GP-NMPC and NN-NMPC.

6.1 Test Trajectory

The test trajectory is introduced in order to force the quadrotor to perform all the three UAV main manoeuvres: pitch, roll and yaw.
The "Infinity trajectory" is an approximated lemniscate, inclined with respect to the x-y plane (Figure 6.1), obtained thanks to a MATLAB command that generates trajectories based on specified waypoints. The purpose of the modification of the original lemniscate is to introduce a vertical linear acceleration along the

z axis.
The trajectory can be inscribed in a parallelepiped of dimension $5mx8m$ of basis and $5m$ of height and is defined as a fuction of time in order to be completed in exactly 40 seconds.

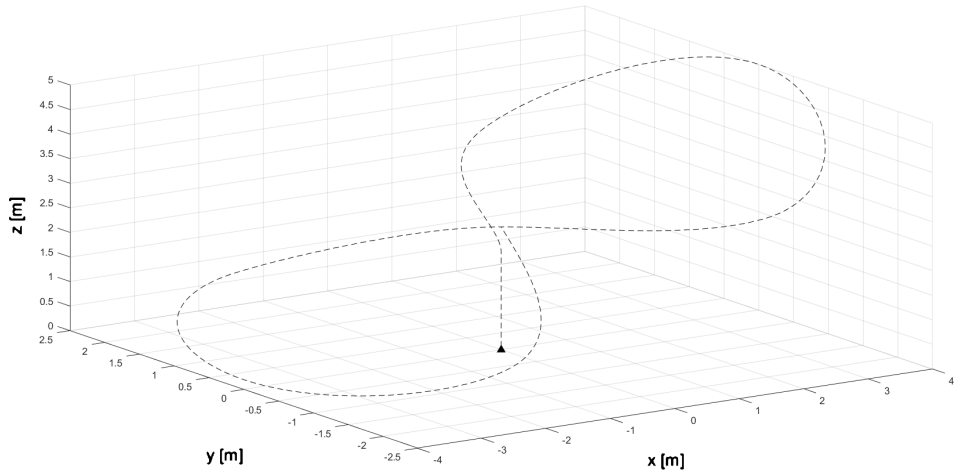

Figure 6.1: Test Trajectory

6.2 Mismatch Introduction

While performing the trajectory tracking task, the quadrotor will have to face some mismatches.
The mismatches are inserted in order to simulate real life scenarios in which the nominal model provided to the NMPC is not the real one due to some uncertainties, wrong data in the datasheet or damaged components.
The two mismatches considered in this thesis are:

- Mass Mismatch:

  A Mass Mismatch is used to simulate the case in which there is an error in the estimated value of the mass (whether it is smaller or grater than the actual one) or to simulate a change of mass, e.g. if a payload is attached to the quadrotor.
  In this thesis a mismatch of $0,25kg$ (i.e. an increase of a sixth of the original mass) is considered.

  In Figure 6.2 a run with basic NMPC, considering the previously described mass mismatch, is plotted.
  The NMPC trajectory is color coded emphasizing the average trajectory

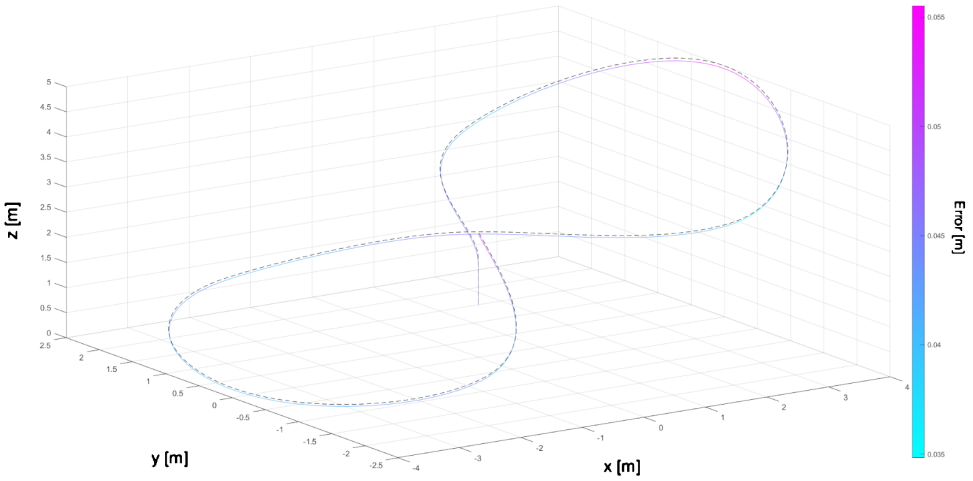

Figure 6.2: NMPC run with Mass Mismatch

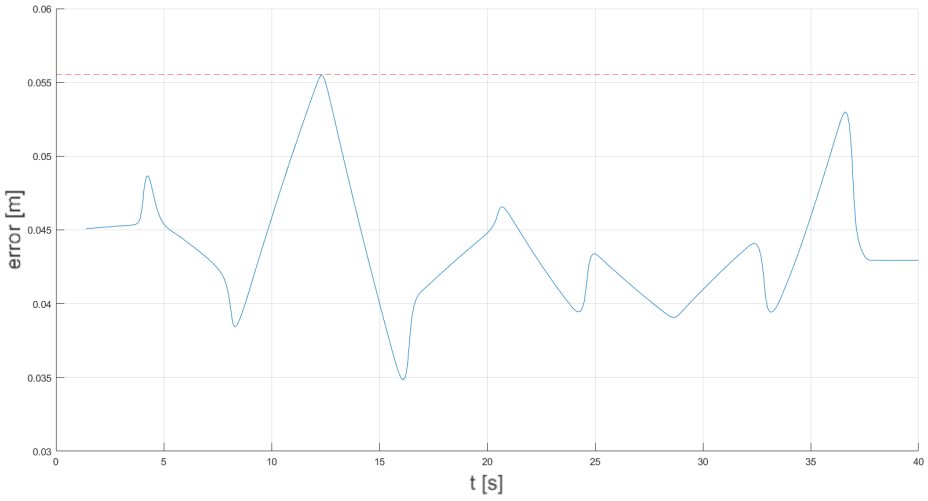

Figure 6.3: Trajectory error of NMPC with Mass Mismatch

error. The color pink represents the parts of the flight where the controller is more distant from the desired trajectory and light blue, the parts where nominal trajectory and flight trajectory are almost the same.
As it is possible to notice, since the NMPC thinks the quadrotor mass is lighter than the actual one, it is producing a thrust smaller than the one required, hence the flight trajectory is below the desired one.
In Figure 6.3 the error variations along the trajectory are plotted. The dashed red line emphasizes the highest value for the error. As it is possible to see in Figure 6.4 the error is greater when the drone gains altitude.

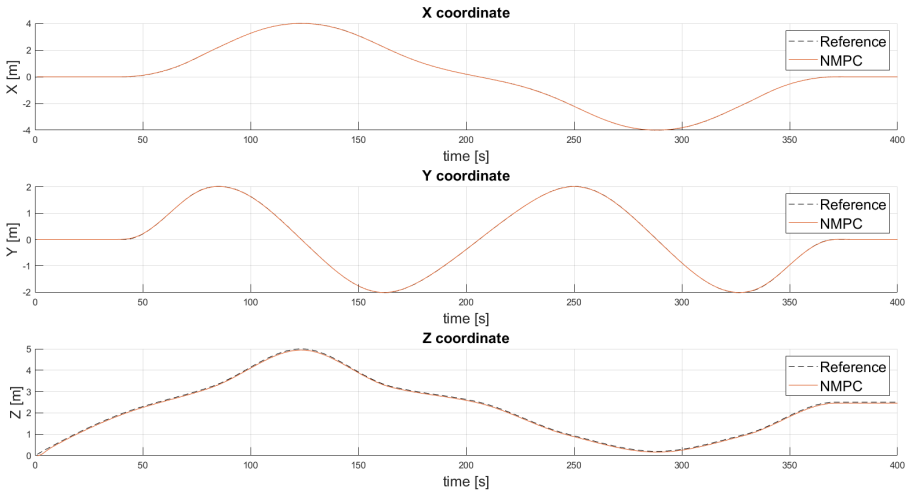

Figure 6.4: Trajectory coordinates of NMPC run with Mass Mismatch

- Motor Voltage Mismatch:

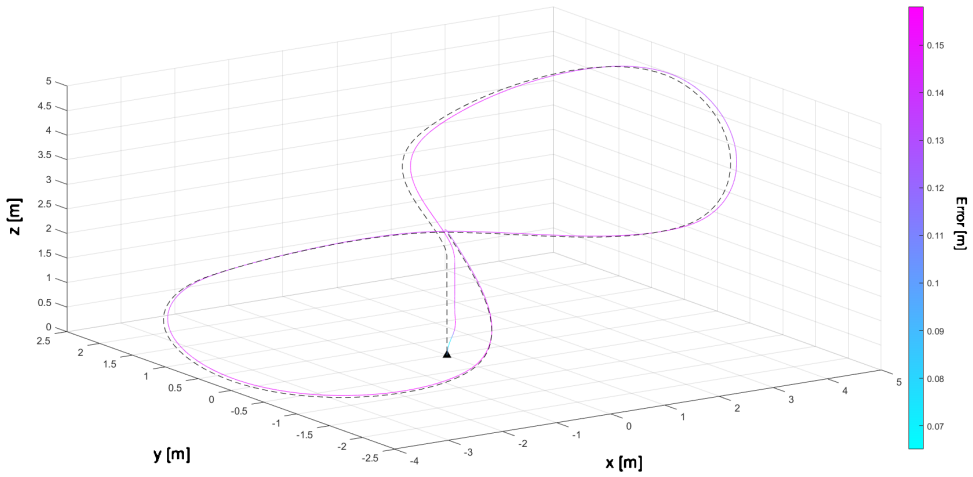

Figure 6.5: NMPC run with Motor Voltage Mismatch

Motor Voltage Mismatch (MVM) is meant to simulate a condition in which a motor does not work at its full potential. A simple example of such a case is a difference between the nominal and the real electrical power a motor is able to handle.
For the purpose of this thesis the mismatch has been modeled by diminishing one of the four propeller control output by 20%, hence causing a thrust deficit that the controller will need to reject.

As it is possible to see in Figure 6.5 the missing knowledge about the lack of power for the first propeller, stops the quadrotor from following the

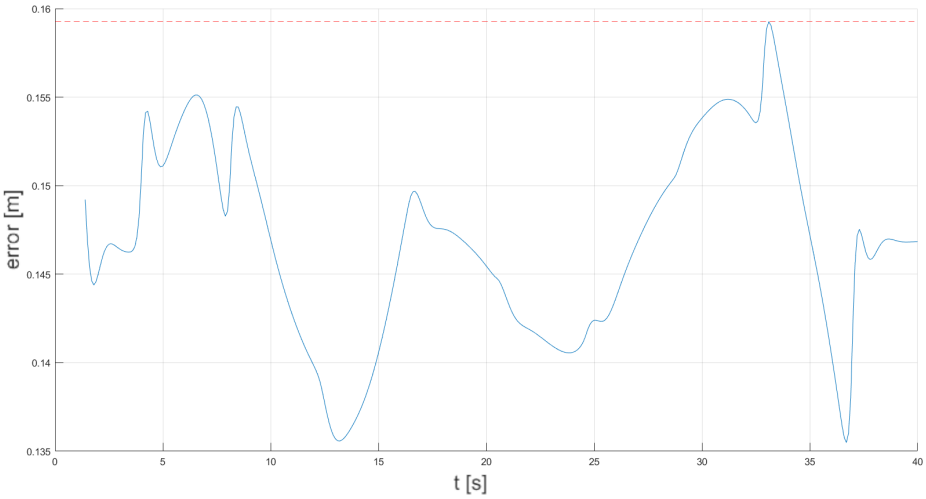

Figure 6.6: Trajectory error of NMPC with Motor Voltage Mismatch

requested trajectory.
As it is possible to notice in Figure 6.6 the error is consistent along the trajectory. This happens since this type of mismatch influences both altitude increases and drone turns as shown in Figure 6.7.

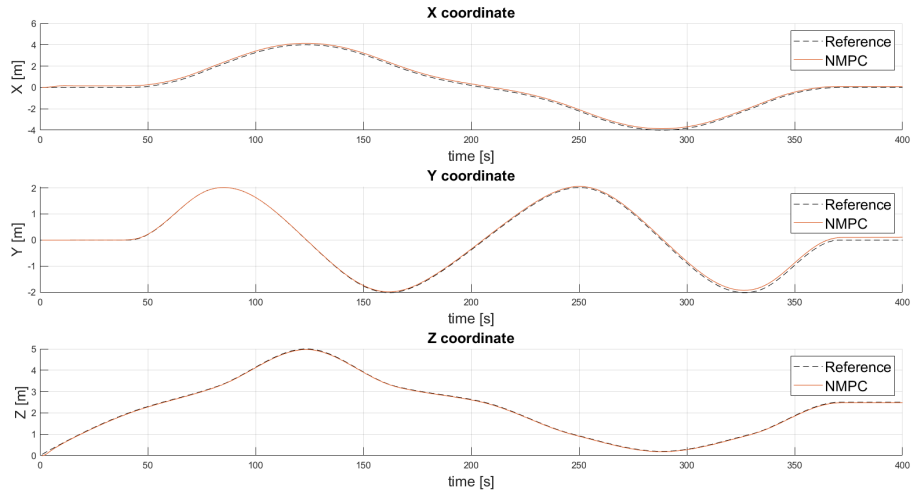

Figure 6.7: Trajectory coordinates of NMPC run with Motor Voltage Mismatch

In this thesis the two mismatches will be considered separately.
The results of the previous runs in terms of average error on the trajectory (i.e. the average distance of the quadrotor from the desired trajectory) are reported in the following Table (6.1)

Mismatch	**Average Error** e_m
Mass Mismatch	$4,38\,cm$
Motor Voltage Mismatch	$14,68\,cm$

Table 6.1: NMPC Mismatches

6.3 Training Setup

In the following subsections two elements fundamental for a correct learning process are introduced.

6.3.1 Training Trajectory

Due to the method's structure, a random trajectory is sufficient to train the model thanks to the GP approach. On the other hand, NN require a more precise training dataset that contains almost all the manoeuvres that the drone might face during its flight.
In order to make the comparison between the two methods fair, a common training trajectory has been chosen (Figure 6.8).

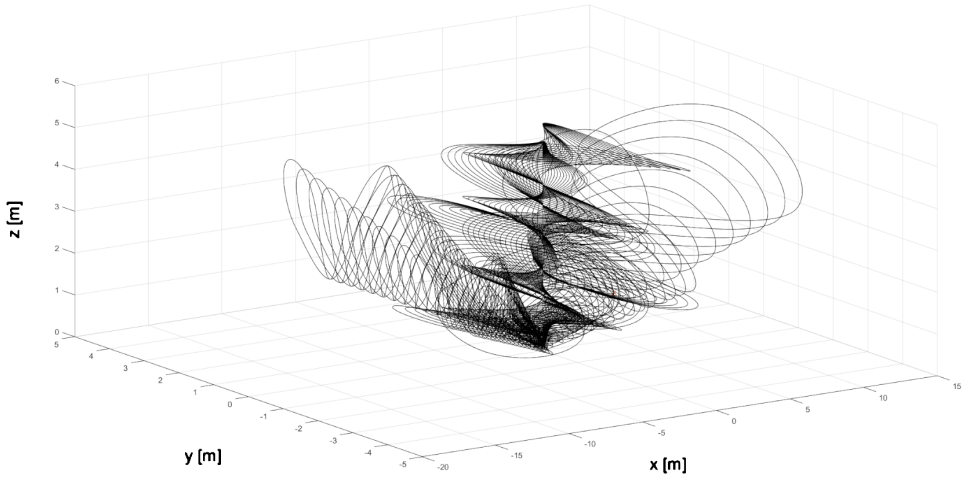

Figure 6.8: Training Trajectory

This trajectory has been created by combining several smaller trajectories,

each one with a specific intent of reproducing a particular manouvre at different acceleration ranges, that the drone might face during its flight.

6.3.2 Training Input Data

The previously required trajectory is needed in order to acquire input data for the training procedure. In order to avoid spurious correlations during the regression process, some elements of the state are not considered. In particular x,y,z and both the linear and the angular velocities are not considered. This is because these state values do not appear in the model definition, hence they do not represent meaningful data to learn from for the Lb methods.
In conclusion, the regression input becomes:

$$[q,\ u] \tag{6.1}$$

where q is the drone pose represented in Euler angles form, while u is the control input vector.

6.4 Simulink Setup

Experiments are carried out using Simulink, a block diagram environment used to design and simulate systems.
The NMPC block is implemented thanks to the MATMPC toolbox explained in the next subsection (6.4.1). The GP and NN components are added directly into the matlab code provided by MATMPC itself, necessary to setup the simulation. A switch is inserted in the simulink scheme in order to enable and disable the adaptive controller intervention (Figure 6.9). This way, the same setup can be useful to carry out all the desired tests with just quick adjustements.

6.4.1 MATMPC

MATMPC [24] is an open source toolbox developed by the Deparment of Information Engineering of the University of Padua. This software, built in MATLAB, for NMPC handling is designed in order to make modeling, controller design and simulation accessible for a wide class of NMPC applications.
All the functions are written in the MATLAB API for C in order to exploit at the

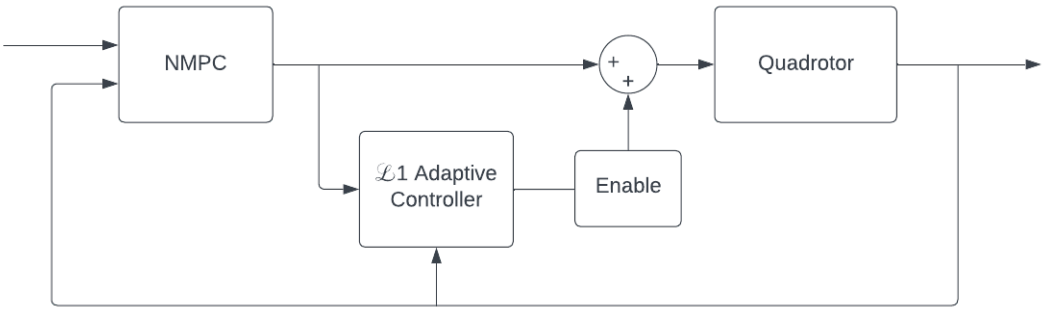

Figure 6.9: Full Control Scheme

same time the accessibility of MATLAB and the great performances of C. The optimal control is discretized via multiple shooting through fixed step Runge-Kutta method. Derivatives are obtained thanks to CasADi [2], state of the art toolbox for these types of needs.
Furthermore, this toolbox provides a NMPC Simulink block that requires the reference and the state as inputs and produces the control signal as output.

6.4.2 Parameters and Tuning

As anticipated in the theoretical chapters, in this subsection technical values will be provided:

1. In order to define the length prediction horizon the knowledge of time length of each step $T_s = 0.1\,s$ (that coincides with the control frequency) and the number of steps in a prediction horizon $N = 10$ are required
2. The weight matrix for the NMPC is given by the following values:
   - [70 70 300] respectively for the x, y and z coordinates.
   - [1 1 10] for the linear velocities.
   - [1 1 1] for the angular velocities.
   - [10^{-4} 10^{-4} 10^{-4} 10^{-4}] for the control inputs.

   an easy intuition behind these numbers is the desire to highly weight the error in the position coordinates to punish the system when the quadcopter is away from the desired trajectory. While the control inputs are very lightly penalized, since in this way the controller is able to push the drone to its limits.
3. The adaptation gain is $A = -0,1$

6.5 Results

In the following sections all simulations results regarding the three methods (adaptive, GP and NN) and the two mismatches (mass and motor voltage) at hand will be presented and discussed.
In subsection 6.5.4 a summary table of the results will be provided and analyzed.

6.5.1 A+NMPC

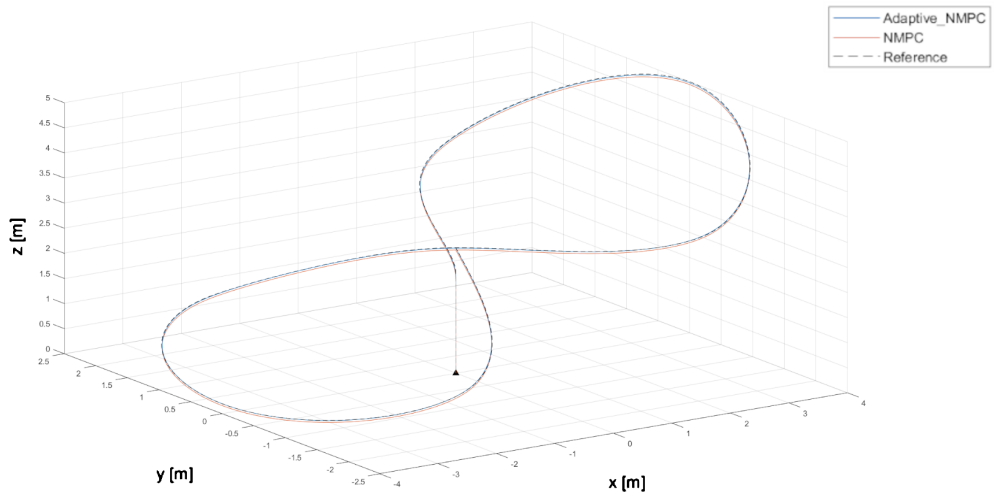

Figure 6.10: Adaptive+NMPC with Mass Mismatch

As it is possible to see in Figure 6.10, the adaptive component is able to reject the mismatch. In fact, the orange line represents the flight of the basic NMPC that is below the desired trajectory, this is because the mass has been increased, hence the required trust is higher than the one provided by the controller. On the other hand the light blue one is the flight with $\mathcal{L}_1$+NMPC and it is almost superimposed perfectly to the requested trajectory.

A similar result is obtained in the case of motor voltage mismatch (Figure 6.11). The only difference is that in this scenario is a little defiance at the beginning of the trajectory.

The reason must be sought in the control signals. Figure 6.12 displays the expected control signals before and after saturation, for the four rotors. Expected control signal means that it is the one produced by the controller without knowing that due to the presence of the mismatch the first rotor will receive a smaller

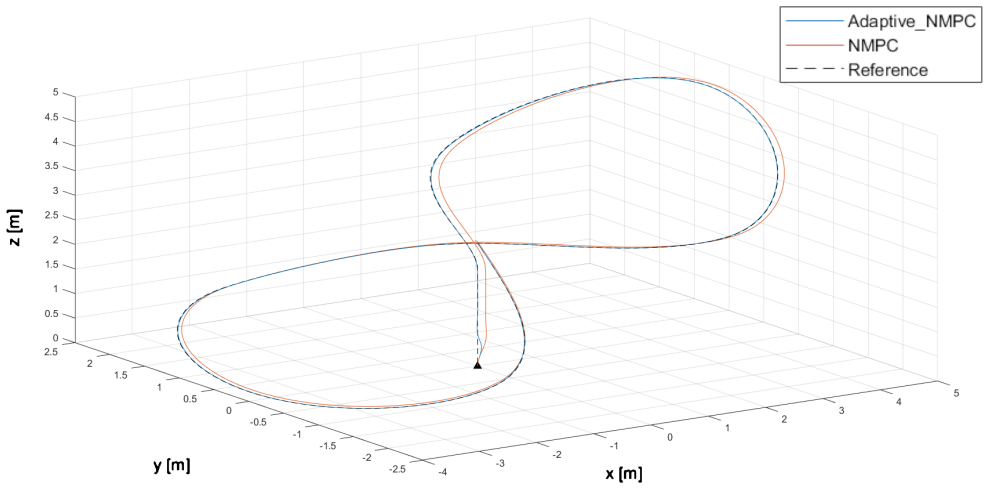

Figure 6.11: Adaptive+NMPC with Motor Voltage Mismatch

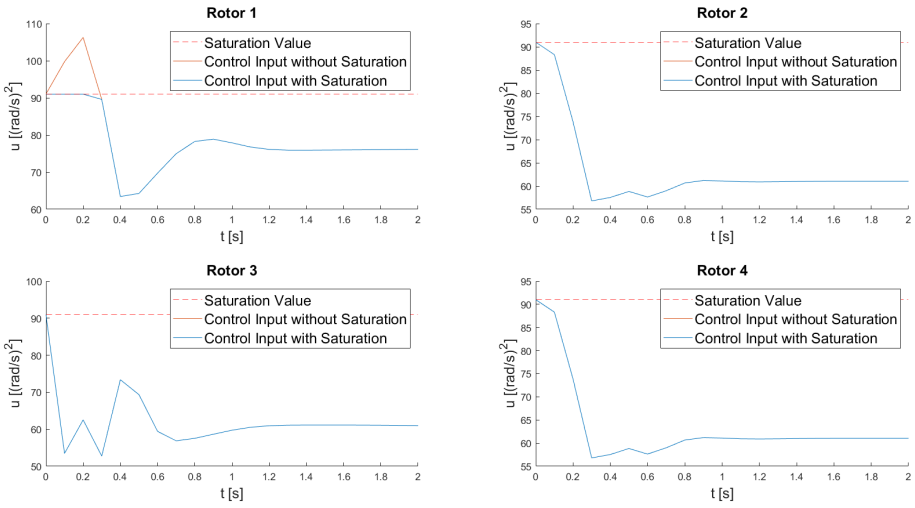

Figure 6.12: Expected Control Signals during the take off phase

amount of power. In this scenario it is possible to notice how the adaptive controller during the first second of flight tries to increase the signal of the first rotor, both trying to compensate for the imbalance and to reach the required altitude fastly. This happens because the Adaptive component does not take into account NMPC constraints.

6.5.2 GP-NMPC

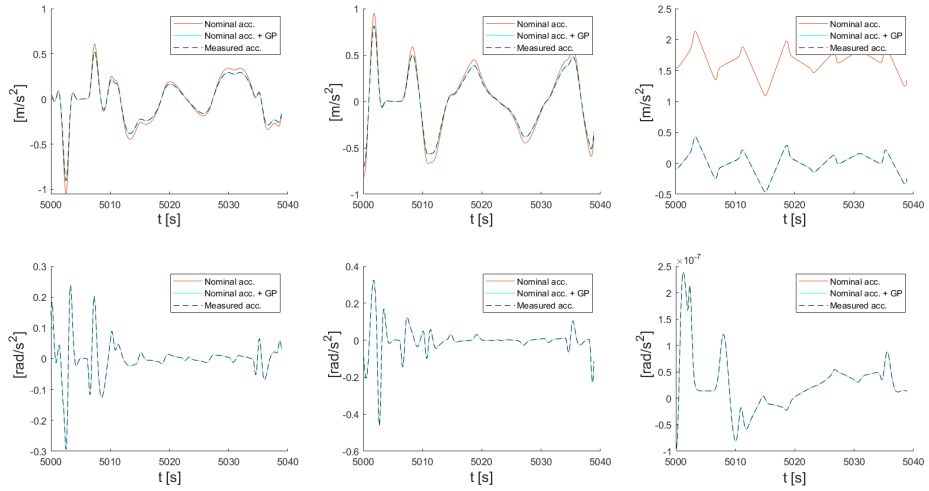

Figure 6.13: Mass Mismatch GP Learned Accelerations. The first row plots are the linear accelerations along the three directions, while in the second row the angular accelerations are shown

In Figure 6.13 the results of the GP learning are displayed in order to showcase the learning goodness. The range depicted (points from $t = [5000, 5040]$) correspond to the test trajectory. The plots in the first row showcase linear accelerations, while in the second row there are the angular ones. The dashed line

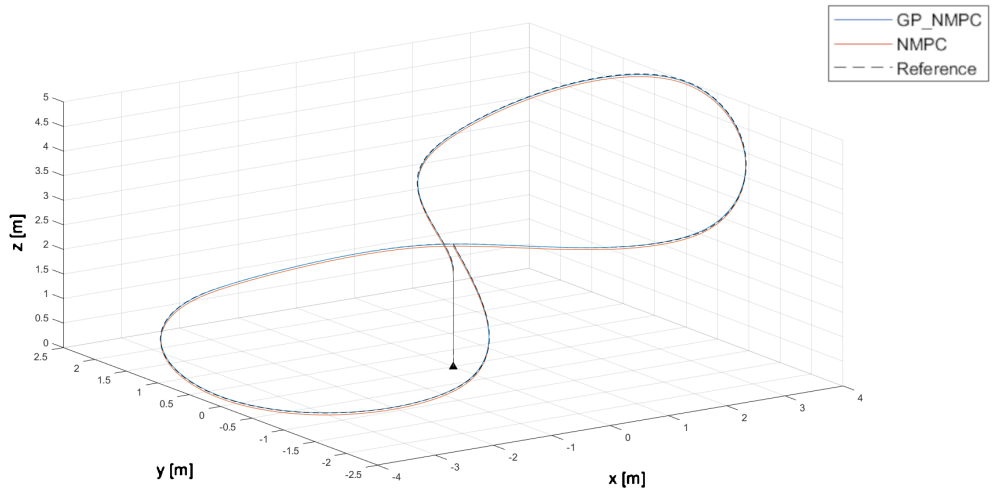

Figure 6.14: GP-NMPC with Mass Mismatch

represents the estimated acceleration in the case the mismatch is assumed to be known (the desired acceleration), the orange line is the actual acceleration of the basic NMPC flight, while the light blue line is the final acceleration given by the sum of the nominal acceleration and the leaned GP mismatch.
As it is possible to notice, a mismatch such as the mass one interferes only with the linear accelerations.

The GP is able to learn perfectly the mismatch and this is also corroborated by the resulting flight shown in Figure 6.14.

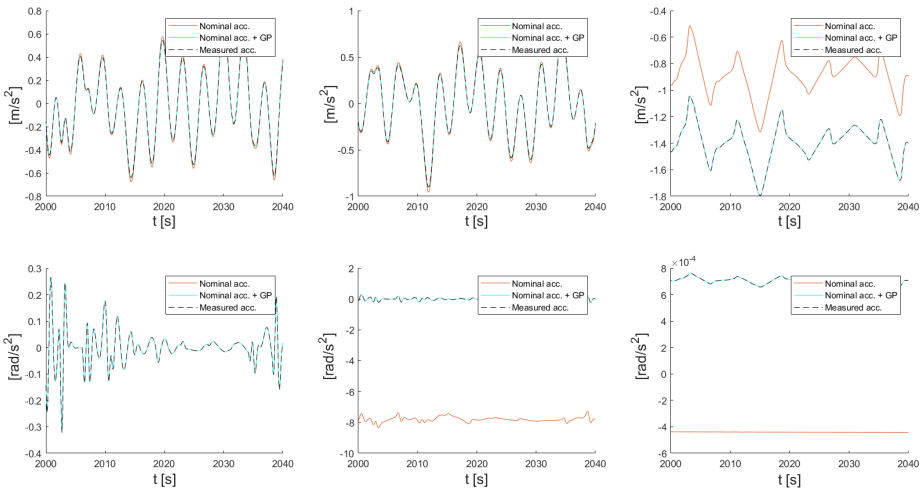

Figure 6.15: MVM GP Learned Accelerations

In Figure 6.15 it is possible to notice how the MVM affect the system kinematics. In this scenario even the angular accelerations present a consistent mismatch between the expected and the nominal one.

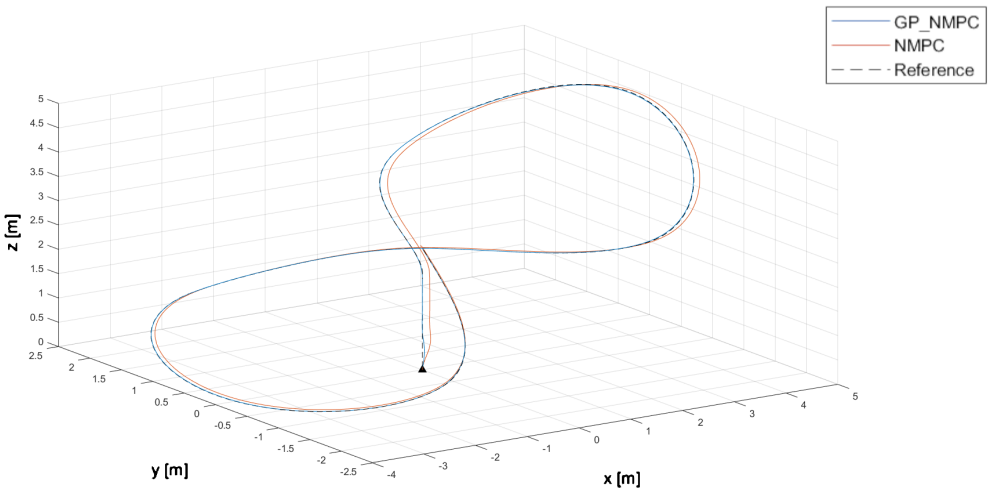

Figure 6.16: GP-NMPC with Motor Voltage Mismatch

Also in this case the GP seems to learn a very accurate description of the model and the NMPC makes good use of this new information in order to track very precisely the requested trajectory (Figure 6.16).

6.5.3 NN-NMPC

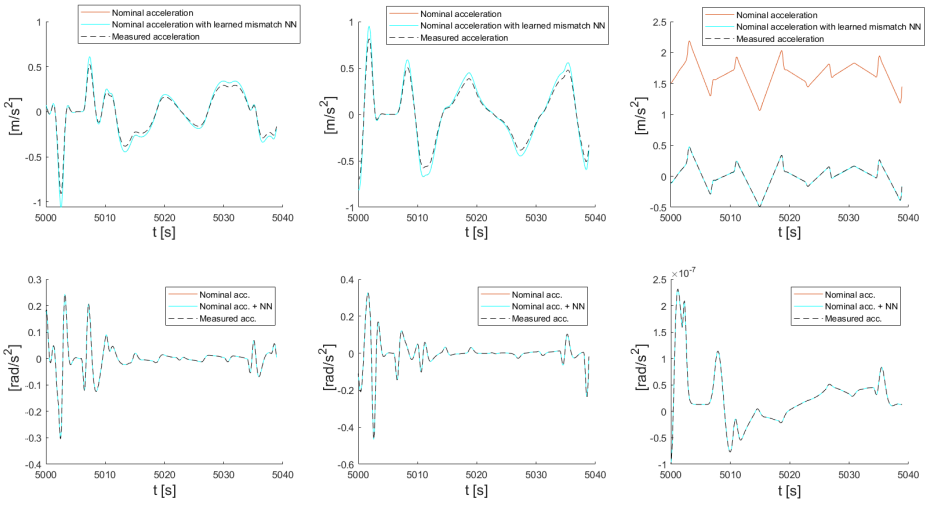

Figure 6.17: Mass Mismatch NN Learned Accelerations

From Figure 6.17 it is immediate to see how Neural Networks learn differently from Gaussian Processes. In fact, being the image structure the same described

in the previous subsection, it is possible to notice how the network used for learning the linear accelerations has leaned with accuracy the mismatch on the third plot (i.e. the z component), while has completely ignored the mismatches on the x and y components (i.e. the first two plots in the first row, where the light blue graph is superimposed with the orange one).
This behaviour is due to the network structure used for this project, resulting in a strong attention on higher magnitude mismatches and less interest in learning the ones reputed (by the network) to be less relevant. It may be compensated with a different network, for example by considering more nodes.

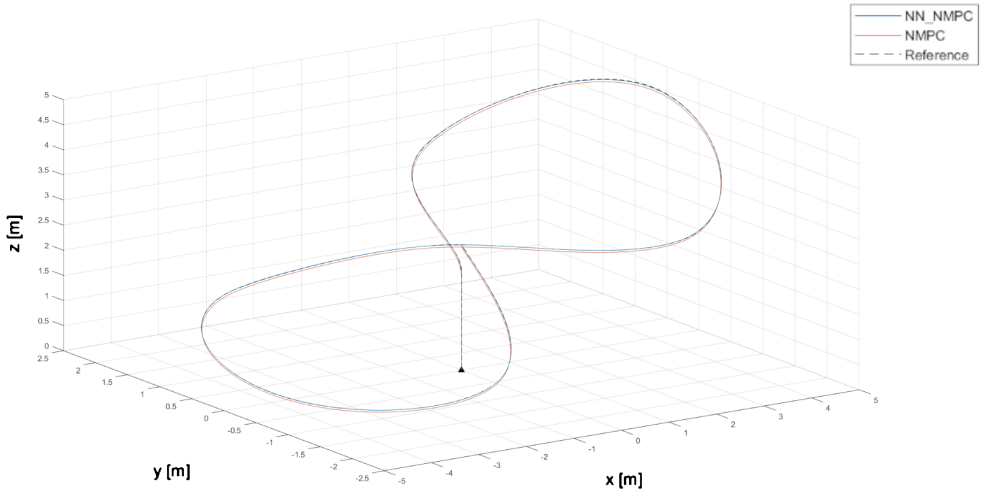

Figure 6.18: NN-NMPC with Mass Mismatch

Nevertheless the results in terms of performance for the trajectory tracking task seem quite noteworthy. The numerical results in terms of average trajectory error will be displayed and discussed in section 6.5.4, but from Figure 6.18 the the network learned component is a significant addition to the original model in order to reject the mismatch.

Figure 6.19 confirms the previous results. In fact, even in the case of MVM, where also mismatches on the angular accelerations come into play, the network still learns with more accuracy the most significant mismatches in terms of magnitude, e.g. the second angular component (i.e. the second plot in row number two).

The result is that even NN is able to provide to the NMPC a mismatch description accurate enough. Figure 6.20 displays the comparison between the two flights of the quadrotor afflicted by the MVM, the controller is the basic

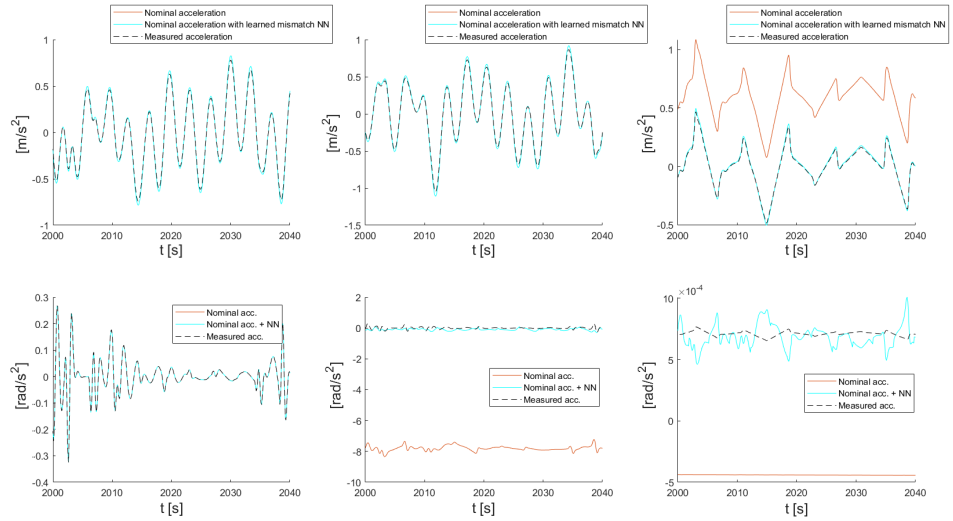

Figure 6.19: MVM NN Learned Accelerations

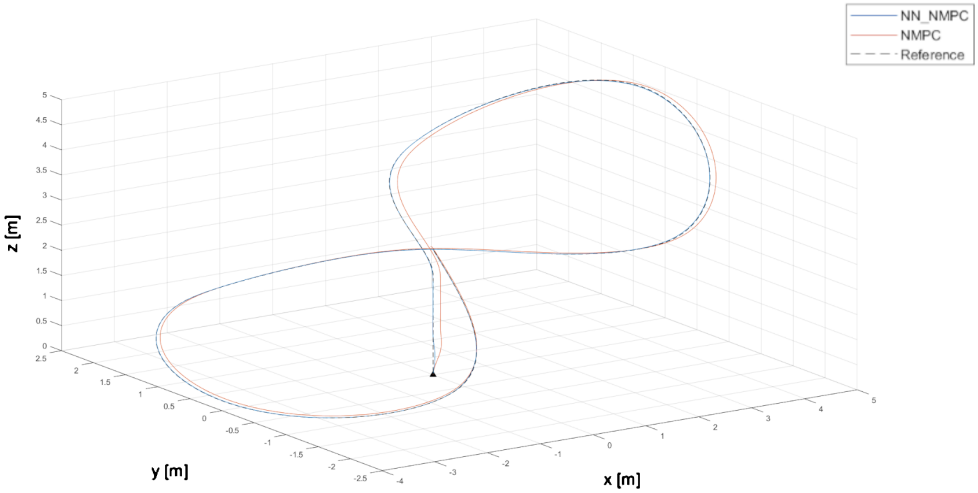

Figure 6.20: NN-NMPC with Motor Voltage Mismatch

NMPC for the orange trajectory and the NN-NMPC approach for the light blue one.

6.5.4 Comparison of the results

Table 6.2 reports the average trajectory error (i.e. the average distance of the quadrotor from the desired trajectory), while the displayed percentages represent the quantity of error reduction carried out by each control method, with respect to the value of the average trajectory error of the standard NMPC.

As it is possible to see from the table, all the three methods are really effective in the track following task and in the mismatch rejection.
NN-NMPC performs slightly better than the other two control strategies, but the difference is smaller than a quarter of a centimeter, hence resulting almost meaningless in case of standard control tasks.

Mismatch	**NMPC** e_m	**A+NMPC** e_m	**GP-NMPC** e_m	**NN-NMPC** e_m
Mass	$4,38\,cm$	$1,09\,cm$	$0,78\,cm$	$0,58\,cm$
		$-75,11\,\%$	$-82,19\,\%$	$-86,76\,\%$
Motor Voltage	$14,68\,cm$	$0,85\,cm$	$0,76\,cm$	$0,72\,cm$
		$-94,21\,\%$	$-94,82\,\%$	$-95,10\,\%$

Table 6.2: Results: Average Trajectory Error

But performance is not the only validation metric to take into account, also computational burden is a crucial aspect in quadrotor control.
In the following, the burden is considered in terms of average Computational Time (CPT) of the controller for each iteration. The simulations are carried out on a PC with 10^{th} generation i7 CPU and a 16 GB RAM.

Control Method	**Average CPT**
A+NMPC	0,58 ms
GP-NMPC	5 ms
NN-NMPC	2 ms

Table 6.3: Results: Average Computational Time

Looking at Table 6.3 is possible to notice how the addition of an adaptive component to the control scheme does not represent a burden in terms of computational time.

On the other hand both the learning-based methods considered in this thesis slow down the process in a significant way, as expected. In fact, the addition of NN make the CPT three times the one with standard NMPC, while the addition of GP slows down the system to almost nine times the basic NMPC CPT.

Considering the chosen task and the tested methods, these simulation results seem to crown Adaptive-NMPC as the best pick so far. In fact, it reaches very competitive values in terms of average trajectory error and by far the smaller average computational time.
Moreover, the adaptive addition to the control loop is more straightforward and easier to tune, compared to the learning-based methods.

6.6 Dangerous Manoeuvre Introduction

As just mention in subsection 6.5.4, the test carried out until now provide very close results in terms of performance. In this scenario the Adaptive-NMPC control scheme seems to be almost the best possible pick, since it achieves similar results with a smaller computational burden and easier procedures for implementation and tuning.
For this reason, in the following, a new type of trajectory tracking task is taken into account.
As previously anticipated and very well documented in literature, Lb-NMPC is often used for agile and high performance manoeuvre.
This is why the Dangerous Manoeuvre (DM) (Figure 6.21) is introduced.

This new trajectory is defined to be completed in $10s$ and requires a quick and aggressive control to the system. The structure of this trajectory can be divided in three main section: a first sudden 4 meters take off, an high speed dive and a final passage near the ground before gaining altitude again.
This structure is obviously a gimmick scenario, but it should not be considered restrictive, since it is conceptually similar to a trajectory that passes close to a wall or near an obstacle, hence a great example of common high performance scenarios.
The learning carried out for this task will be based again on the training trajectory, previously introduced.

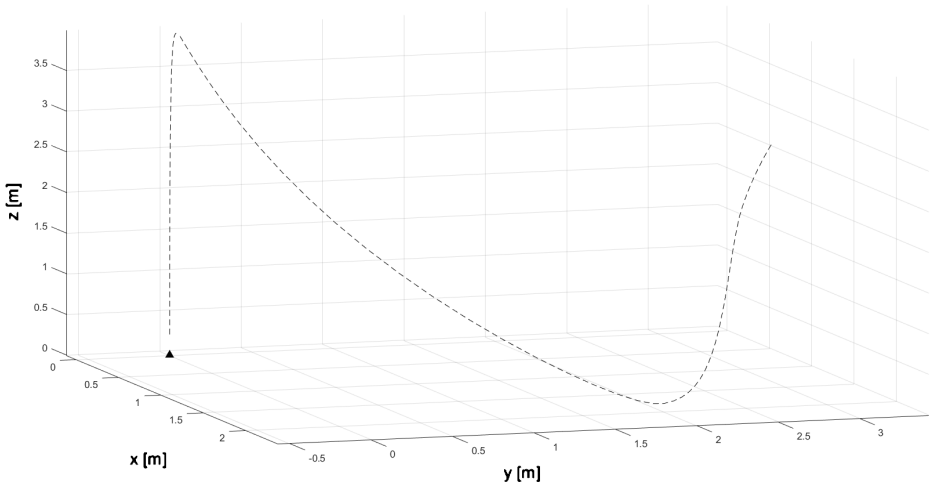

Figure 6.21: Dangerous Manoeuvre Trajectory

6.7 Dangerous Manoeuvre Results

The standard NMPC with the previously described mass mismatch crashes on the floor (Figure 6.22).

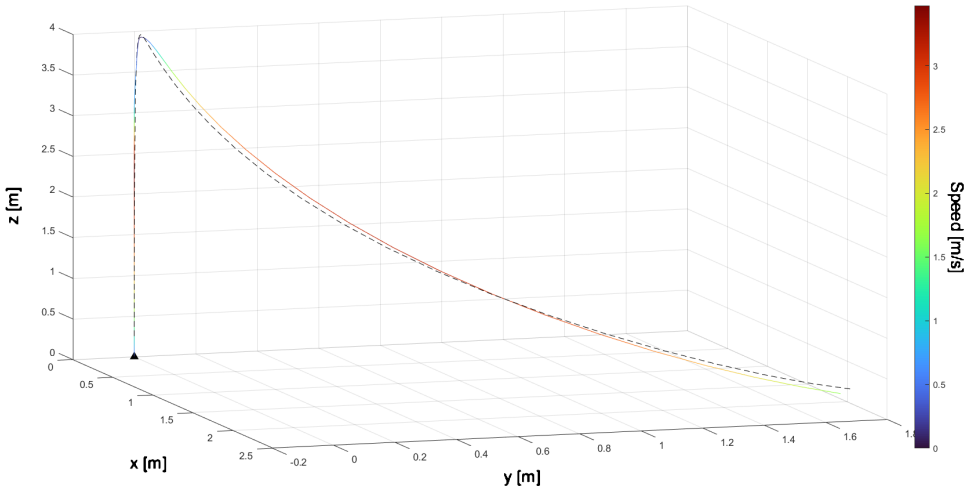

Figure 6.22: DM: NMPC with Mass Mismatch

This is because the Model Predictive Controller is not able to recognize the additional mass that pushes the quadrotor flight below the desired trajectory. Also the simulation with motor voltage mismatch ends up with infeasibility. In this scenario the cause must be sought in another problem, in fact the reason

for this is the incapability to lift off. Since this scenario is less graphically interesting, even if the results are similar, in this section only the mass mismatch will be considered.
In the following subsections the results for Adaptive-NMPC and Lb-NMPC are presented.

6.7.1 A+NMPC

As it is possible to see in Figure 6.23 the cascade of NMPC and adaptive controller still leads to infeasibility.
Also in this scenario the quadrotor crashes in the same spot as for the basic NMPC, but this time the reason is different.

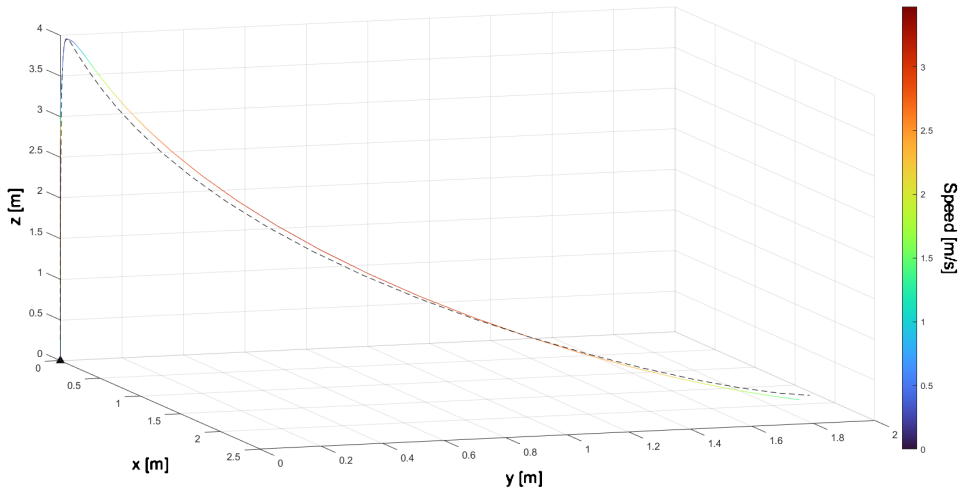

Figure 6.23: DM: Adaptive+NMPC with Mass Mismatch

The crash is caused by the incapability of the adaptive controller of knowing the system constraints. In this case the breached constraint is the control input saturation.

In Figure 6.24 the actual control signal and the control signal before saturation are displayed. The constraint is violated in two time segments. The first one is during the take off task (time segment $[0-1,5]\,s$), the system does not see this violation as a problem since it is just an attempt of the adaptive controller to make the quadrotor lift faster. The second violation is at six seconds, the idea is that the controller is trying to react at the dive, but it acts too late asking a control effort that the drone is not able to produce, hence leading to a fatal crash.

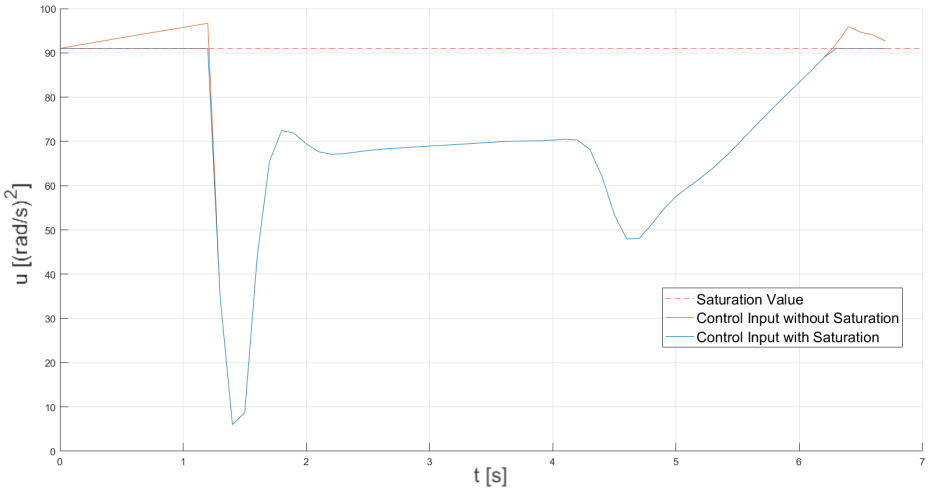

Figure 6.24: DM: Adaptive+NMPC Control Input

6.7.2 GP-NMPC

The Gaussian Process contribution is different from the adaptive controller one and thanks to this difference the drone is able to complete the requested trajectory.

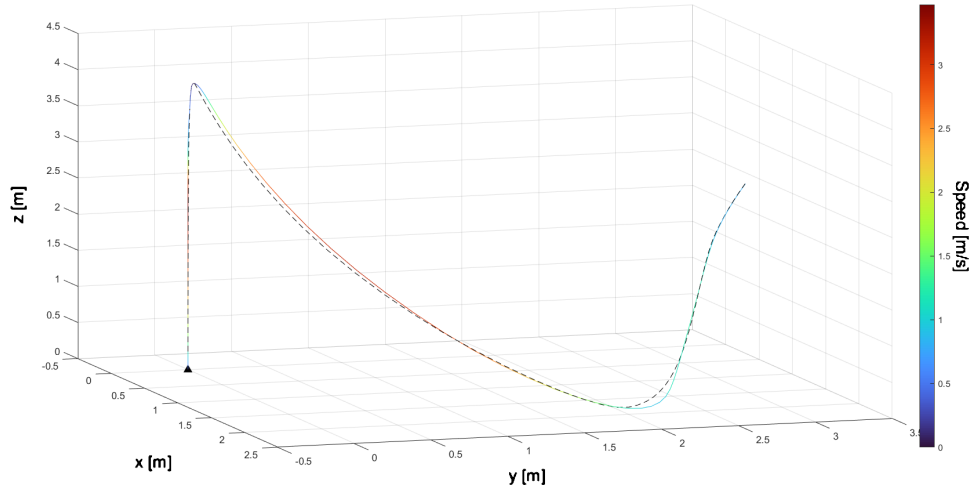

Figure 6.25: DM: GP-NMPC with Mass Mismatch

In fact, GP intervention aims at learning the mismatch and in this way complementing the imprecise model that is used by the basic NMPC. This type of modification is directly made inside the NMPC model, hence, as expected

from theory, no model constraint will be violated.

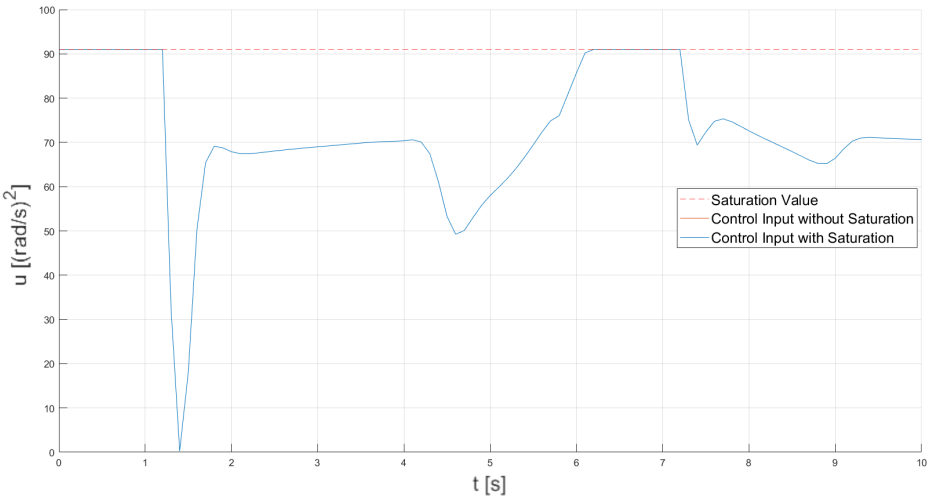

Figure 6.26: DM: GP-NMPC Control Input

It is possible to verify this fact by looking at the control input displayed in Figure 6.26. Differently from Adaptive+NMPC (Figure 6.24), in this graph the control input produced by the controller and the saturated version are exactly the same

6.7.3 NN-NMPC

Also Neural Networks are able to learn an accurate acceleration mismatch, making possible for the drone to complete the trajectory thanks to the NMPC control action.

In Figure 6.27 is also possible to notice that the learning via Neural Networks is different from the Gaussian Process one. In fact it is possible to visually grasp how the dive is more precise in this plot, while the moment in which the quadrotor regains altitude (around coordinate $[1,75; 2; 0]$) is slightly more precise in the GP learning case (Figure 6.25)

Exactly as for GP, also NN acts directly on the model provided to the NMPC, hence the saturation constraints are respected. This is proven by Figure 6.28 where the superposition of the control signal before and after saturation is perfect.

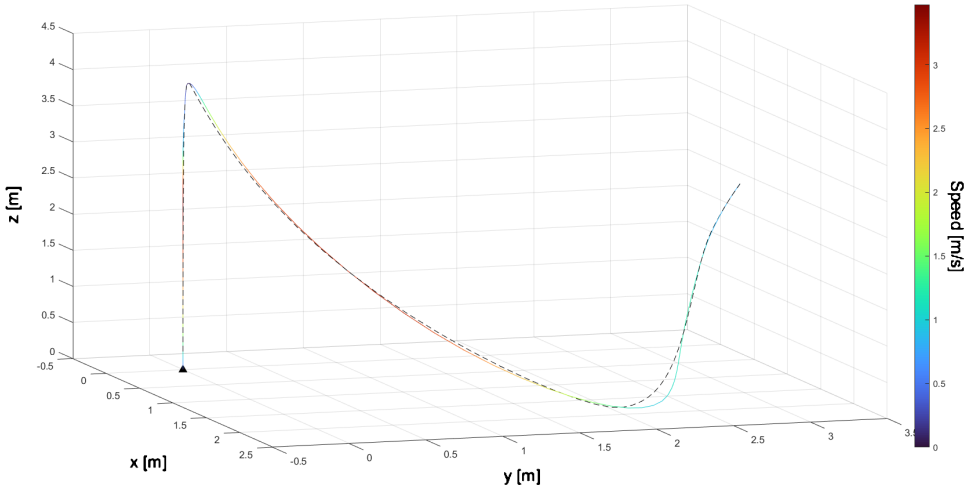

Figure 6.27: DM: NN-NMPC with Mass Mismatch

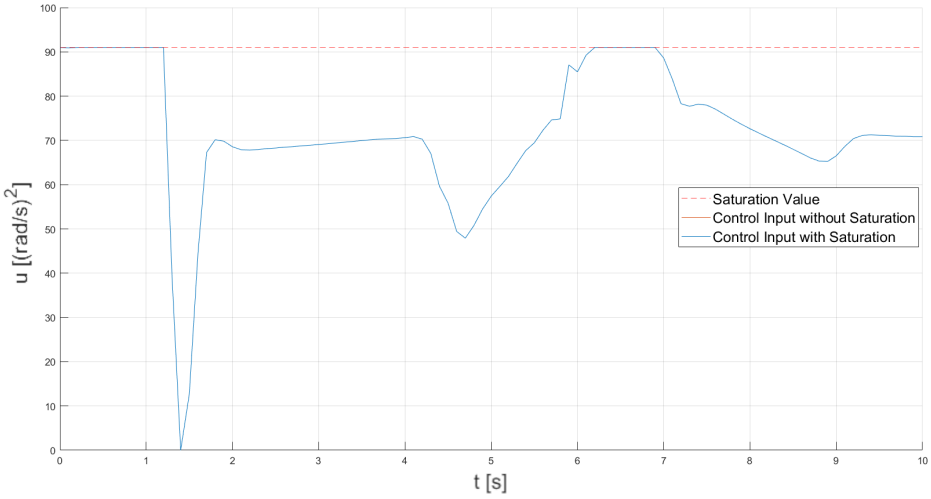

Figure 6.28: DM: NN-NMPC Control Input

6.7.4 Comparison of the results

The dangerous manoeuvre proves several aspects discussed in the theory section.

For such a task, basic NMPC reaches infeasibility since it is not able to recognize that the actual mass of the quadrotor is heavier than expected, this will lead the drone to fly lower than it should and hence to crash when the trajectory reaches heights that are close to the ground.

Adding the adaptive component to the control scheme in theory should im-

prove the results, but the simulation is still infeasible. As previously mentioned in Subsection 4.3, the adaptive controller does not take into account NMPC constraints, hence when the quadrotor gets closer to the ground the adaptive action increases the control input in order to reject the mass mismatch. In this way the control signal reaches saturation and the drone crashes again.
On the other hand, both the learning-based approaches are able to complete the tasks of trajectory tracking and mismatch rejection. Gaussian Processes and Neural Networks are able (in two different ways) to learn the model, hence helping the NMPC in developing a long term control strategy perfectly fitting for the new learned model.

7

Conclusions and Future Works

7.1 Conclusions

With this work, a fair introductory comparison between advanced control techniques has been carried out.
The task was a trajectory tracking for a coplanar quadrotor that needed to reject different types of mismatches during the flight.
The basic control action was carried out by a NMPC that presented a performance deterioration due to the model quality dependence of this type of controller.
In order to overcome this problem, three strategies were proposed: the cascaded addition of a $\mathcal{L}_1$ adaptive controller (Chapter 4) and two learning-based methods (Chapter 5), the first one exploit Gaussian Processes techniques, while the second makes use of Neural Networks.
In a non aggressive scenario, like the one presented in section 6.1 with the "Infinity trajectory", the three proposed approaches performed similarly in terms of average trajectory error both in the case of mass mismatch and the case of motor voltage mismatch.
The main difference here were found in the average computational time at each iteration. In fact the A-NMPC ($CPT = 0,58\,ms$) had a computational time that was almost one third of the one obtained with NN and one ninth of the GP one.
This scenario seemed to be pretty favorable to the adaptive controller but this control technique presented some non-negligible limits.
When, a new, more aggressive test case was introduced in section 6.6, the results

changed.
With this manoeuvre, the standard NMPC reached infeasibility since the difference in mass leaded the quadrotor to a fatal crash. The same outcome happened also with the $\mathcal{L}_1$+NMPC strategy. The cause this time was due to the fact that the adaptive addition to the original control signal did not take into account NMPC constraints, hence the crash happened because saturation was reached while trying to regain altitude.
The outcome was different when the learning-based component were then taken into account. Both Gaussian Processes and Neural Networks showed their ability in learning the rotor acceleration mismatch. Since the learning-based component was directly added inside the model provided to the NMPC, the control respected the original constraints, being able to complete the aggressive trajectory.

In conclusion, depending on the setup and the scenario, different control techniques are proven to be more valid then others. $\mathcal{L}_1$-NMPC is a straightforward method, perfect for the non-aggressive control task considered during this project. While for the dangerous manoeuvre scenario, the more complex (both computationally and in terms of setup) Lb methods prove themselves to be a valid pick when high-performances come into play.

7.2 Future Works

There are many possibilities to expand the work carried out in this thesis.
First of all a more thorough evolution should be performed: here just some model mismatches and disturbances have been investigated, but there would be need for a Monte Carlo evolution in order to provide a significant sensitivity analysis.
Furthermore it is possible to introduce a task of trajectory planning for the NMPC.
From the adaptive controller perspective, new methods for robustness improvement could be researched.
From the learning point of view, an interesting upgrade could lead towards the implementation of on-line learning. In this way the quadrotor should learn mismatches during the flight, being able to adapt to changing model conditions (e.g. partial motor failures during the flight or sudden and in flight change of

mass).
Another interesting research field gravitates towards the choice of the network structure. As mentioned in Subsection 5.3.1, several paper come out each day proposing new approaches and structures for Neural Networks. These implementations could be further investigated in order to improve the current results. A final possibility could be real life, laboratory testing. This kind of approach should validate even further the data presented in this thesis, facing brand new challenges, such as having noisy data to be filtered, several differences between the mathematical and the real model, the necessity to make the code run on a actual microcontroller or even having more unmodeled disturbances, for example the presence of gusts of wind.

References

[1] Saviolo Alessandro, Li Guanrui, and Loianno Giuseppe. "Physics-Inspired Temporal Learning of Quadrotor Dynamics for Accurate Model Predictive Trajectory Tracking". In: *IEEE ROBOTICS AND AUTOMATION LETTERS* (2022).

[2] J. A. E. Andersson et al. "CasADi: A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* (2019).

[3] Kong Yao Chee, Jiahao Tom Z., and M. Ani Hsieh. "KNODE-MPC: A Knowledge-Based Data-Driven Predictive Control Framework for Aerial Robots". In: *IEEE ROBOTICS AND AUTOMATION LETTERS* (2022).

[4] 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* (2018).

[5] Hanover Drew et al. "Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors". In: *IEEE ROBOTICS AND AUTOMATION LETTERS* (2022).

[6] Xargay Enric, Hovakimyan Naira, and Cao Chengyu. "L1-Adaptive Controller for Multi-Input Multi-Output Systems in the Presence of Nonlinear Unmatched Uncertainties". In: *IEEE Xplore* (2021).

[7] Picotti Enrico et al. "Continuous-Time Acceleration Modeling through Gaussian Processes for Learning-based Nonlinear Model Predictive Control". In: *ECC* (2022).

[8] Picotti Enrico et al. "LbMATMPC: an open-source toolbox for Gaussian Process modeling within Learning-based Nonlinear Model Predictive Control". In: *ECC* (2022).

[9] Simonetti Filippo. “Learning based Nonlinear MPC for quadrotor control”. In: *DEI* (2023).

[10] Bianchin Francesco. “Learning-based Nonlinear Model Predictive Control for a Motorcycle Virtual Rider”. In: *DEI* (2022).

[11] Rosenblatt Frank. “The Perceptrona perceiving and recognizing automaton”. In: *Cornell Aeronautical Laboratory* (1957).

[12] Torrente Guillem et al. “Data-Driven MPC for Quadrotors”. In: *IEEE ROBOTICS AND AUTOMATION LETTERS* (2021).

[13] González-Hernández Iván et al. “Real-Time Improvement of a Trajectory-Tracking Control Based on Super-Twisting Algorithm for a Quadrotor Aircraft”. In: *MDPI* (2022).

[14] Pravitra Jintasit et al. “L1-Adaptive MPPI Architecture for Robust and Agile Control of Multirotors”. In: *IEEE Xplore* (2021).

[15] C. Jones, F. Borrelli, and M. Morari. “Course on Model Predictive Control”. In: *Spring Semester* (2015).

[16] Horik Kurt, Stinchcombe Maxwell, and White Halbert. “Multilayer Feedforward Networks are Universal Approximators”. In: *Neural Networks* (1989).

[17] Hewing Lukas et al. “Learning-Based Model Predictive Control: Toward Safe Learning in Control”. In: *Annual Reviews* (2021).

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

[19] Bauer Matthias, Mark van der Wilk, and Rasmussen Carl Edward. “Understanding Probabilistic Sparse Gaussian Process Approximations”. In: *NIPS* (2016).

[20] MCopper. “Difference among Black Box, White Box and Grey Box Testing”. In: *Medium* (2018).

[21] Mehndiratta Mohit and Kayacan Erdal. “Gaussian Process-based Learning Control of Aerial Robots for Precise Visualization of Geological Outcrops”. In: *ECC* (2020).

[22] Galati Rocco and Mantriota Giacomo. “Path Following for an Omnidirectional Robot Using a Non-Linear Model Predictive Controller for Intelligent Warehouses”. In: *MDPI* (2023).

[23] Salzmann Tim et al. “Real-time Neural MPC: Deep Learning Model Predictive Control for Quadrotors and Agile Robotic Platforms”. In: *IEEE ROBOTICS AND AUTOMATION LETTERS. PREPRINT VERSION* (2023).

[24] Chen Yutao et al. “MATMPC - A MATLAB Based Toolbox for Real-time Nonlinear Model Predictive Control”. In: *ECC* (2019).

Acknowledgments

This final section is necessary to give a shout out to everyone that contributed, even in a small part, to this thesis.
The first name that needs to be mentioned is Filippo Simonetti, a great colleague, with which I have shared most of my university experiences during this Master's Degree. It is also important to mention that the research training at the basis of this thesis has been carried out for several months working together.
Secondly I need to mention Prof. Angelo Cenedese, my supervisor for this the thesis, Prof. Mattia Bruschetta (co-supervisor) and Prof. Alessandro Beghi that proposed the idea of a shared project, contacting and connecting all the parties involved.
A final shout out goes to every engineer and PHD student that helped us by giving technical assistance or feedback: Enrico Picotti and Nicola Lissandrini.