MASTER PROJECT IN ROBOTICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Towards Real Time Deep Learning Tubes: Trajectory Tracking with a Quadrotor

EPFL Thesis Report

Maxime Boutot

EPFL ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
SCHOOL OF ENGINEERING, SECTION MICROTECHNIQUE

Authors

Maxime Boutot
School of Engineering, Robotics
EPFL Ecole Polytechnique Fédéral de Lausanne

Place for Project

Stockholm, Sweden
Smart Mobility Lab, KTH Royal Institute of Technology

Examiner

Prof. Colin Jones
DAutomatic Control Laboratory
EPFL Ecole Polytechnique Fédéral de Lausanne

Prof. Dimos Dimarogonas
Division of Decision and Control Systems
KTH Royal Institute of Technology

Supervisor

Mr. Emilio Maddalena
Automatic Control Laboratory
EPFL Ecole Polytechnique Fédéral de Lausanne

Mr. Pedro Roque
Division of decision and control system
KTH Royal Institute of Technology

05/03/2021

Abstract

Unmanned Vehicles is a fast-growing robotics area due to their vast application scenario and ability to perform tasks otherwise impossible. A particular branch, Unmanned Aerial vehicles, specifically multirotor, has seen an increasing interest in research due to their mechanical simplicity and agility.

The multiplication of multirotor applications increases the interface with humans and the number of different environments. Some of these applications can be sensitive, and the question of safety arises. The users will need guarantees on the system's behavior. Model Predictive Control in its robust form is particularly suitable to answer these new challenges.

This field of research brought a lot of attention and produced encouraging results. However, translate these results to practical implementation remain an unresolved issue for large and fast dynamics.

This work presents a new lead to overcome this last step using deep learning estimation for tube-MPC. Supported by theoretical basis, some simulations are performed to validate this new approach. Then, the reported hardware experiments shows that it can meet the requirements of real-life implementations.

Acknowledgements

I would like to thank Pedro Roque for his support, encouragement, and feedback throughout this thesis's writing. Your guidance helped me stay motivated and on course. Thanks to Prof. Colin Jones and Emilio Maddalena from EPFL Automatic Control Lab 3 for their confidence, availability, and feedback through this project. I also want to thank Prof. Dimos Dimarogonas and the Smart Mobility Laboratory for their warm welcoming and for allowing me to access the infrastructure to perform experiments during these exceptional times. Finally, a particular thought to my family and friends that supported me all along with my studies.

Contents

Chapter 1

Introduction

Automation of repetitive tasks and missions has always been a paramount concern for human beings. In the last decades, this motivation led to the arising of robotics. Pushed by an active community of scientists and industrial progress, this specific automation area aims to bring smarter machines to work and act aside humans in the same society. Robots can take many shapes and functionalities, from vacuum cleaners to exoskeletons and autonomous cars, passing by industrial arms. A particular branch of robots called Unmanned Vehicles (UV) presents spectacular improvements during the last years due to intensive research. Three main categories raise from it, Unmanned Aerial Vehicle (UAV), Unmanned Ground Vehicles (UGV), and Unmanned Underwater Vehicle (UUV).

Figure 1.0.1: From left to rigth, an example of UAV (Flyability), UGV (BostonDynamics) and UUV (Maracoos)

This work will focus on UAVs as they are becoming more and more present in the market under various applications due to their flexibility, deployment convenience,

and cost-efficiency. Among UAVs, it is possible to distinguish three main categories with (i) fixed wings drones, (ii) multi-copters, and (iii) vertical take of landing (VTOL), presenting an end-less diversity of configurations and possibilities. One can cite the video-making civilian market (DJI, Parrot), agriculture mapping (senseFly), industrial installation inspection (Flyability, Flybotix), and military drones. The latest significant interest is drones for delivery in limited access areas (Zipline, Rigitech), from packages to sensible medical supplies.

Figure 1.0.2: From left to rigth, an example of Fixed wings drones (ZipLine), multicopters (Parrot) and VTOL drones (Rigitech)

In this new dynamic, the question of safety needs to be brought up. Indeed, the multiplication of drones in society leads to closer interactions between humans and UAVs, and the increasing task's sensibility calls for strong guarantees on their behavior. In this work, the utilization of Model Predictive Control (MPC) in its robust form is proposed to meet these challenges. This control technique has recently drawn much attention due to its several advantageous proprieties. (Findeisen et al. 2003) :

- Consideration of nonlinear systems with multiple inputs and outputs.
- Consideration of constraints on states and inputs.
- Rigorous stability and robustness conditions.

However, behind these advantages, some challenging obstacles remain to be overcome in the UAV control framework applied to safety-critical systems subject to uncertainty.

1.1 Background and Motivation

In the described framework, the definition of the control algorithm used to define UAVs' behavior plays a central role. Flying robots will be asked to carry out tasks in a changing and challenging environment where the surrounding can be tough to predict. The usual control approach based on pre-computed policies can have such problems due to the lack of online system knowledge. Moreover, the interactions of the robots with their environment will be subject to constraints. These constraints act for their proper safety as well as the operator safety and physical limitations. Ensuring rigorous stability and robustness conditions while enforcing constraints on the system is a challenge that can be overcome only by a few techniques. One of these techniques draws much attention from the scientific community, Model Predictive Control (MPC). Theorized in the early 60's (E. B. Lee and Markus 1967 J. H. Lee 2011), this control technique inherits from optimal control theory and aims to use a model of the system to predict and optimize the future inputs such as the predicted behavior is optimized. In essence, MPC is built upon the repetitive solution of an optimal control problem computed online, in opposition to many classical control theories based on pre-computed control laws (Faulwasser 2012). To perform such a task, MPC required four main ingredients: a model, a cost objective to be minimized, some constraints and the system state information at each sampling time. It then followed the pattern :

- At a sampling time t obtain a state measurement $\boldsymbol{x_k}$.
- Predict the system behavior for the T next steps using system model, a set of admissible open-loop control actions, and a cost objective to be minimized.
- Apply the optimal open-loop input signal $\boldsymbol{u}$.
- Continue with first step.

The number of future steps taken in the optimization process is called the horizon (T). Intuitively, pushing the horizon to infinite gives all the required knowledge to ensure an optimal response but, in practice, leads to untractable computation time. Tuning the horizon parameter has a significant impact on the performances.
At first, MPC suffered from the lack of computation power and techniques to solve the optimization problem in a satisfying amount of time. Based on the Hamilton-Jacobi-Bellman theory (Dynamic Programming), the optimal input calculation procedure asks for considerable resources. Hence until the end of the '90s, MPC could mostly be

seen in heavy industries such as petrochemical (Qin and Badgwell 2003). Indeed, the slow dynamics of such a system relax time constraints and make the benefits of MPC worth using. However, considering fast dynamics with high dimensional states prone to instability, such as UAVs, asked for more time. In the first decade of this millennium, the progress made allowed to see works as Singh and Fuller 2001 and Richards and How 2005 dealing with MPC for UAV but the challenge of real-time implementation held until Raemaekers 2007.

However, the necessity of large computational power requirements is just one of the challenges presented by MPC. Indeed, the stability of the system is crucial propriety of deploying it in a real system. Unfortunately, it has been shown that stability is not guaranteed for a control policy based on input optimization (Rudolf Kalman 2001). Thus, multiple strategies have been developed throughout the years to enforce the control system's stability, see D. Q. Mayne, J. B. Rawlings, et al. 2000 and H. Chen and Allgöwer 1998b. In the specific framework of the finite horizon MPC for nonlinear system (NMPC), one approach enforces the optimal trajectory to end in a stabilizing invariant set, often taking the neighborhood of the desired setpoint. In addition, it is necessary to give a precise shape to the cost objective, adding a term for the prediction's last step. (W.-H. Chen, O'Reilly, and Ballance 2003 Scokaert and James B. Rawlings 1996 Valluri and Kapila 1998). This approach presents several advantages: recursive feasibility can be shown, and the forecasting horizon can be reduced without severe loss of performances, leading to the reduction of the computational burden.

Nevertheless, even if NPMC research shows strong system stability results, uncertainty can arise and violate these proprieties. A real system deployed in an uncontrolled environment will necessarily present some additional noise or model imprecision. Based on this plant model mismatch, one needs to add additional guarantees to ensure that the system will not suffer a significant performance drop. These considerations add an extra level of complexity. Indeed, even if NMPC presents some inherent robustness proprieties (Allan et al. 2017, James B. Rawlings 2017a), if the uncertainty becomes prepotent and the system is subject to state constraints, the conflict can quickly lead to failure.

For these reasons, robust MPC receives much attention from the research community, and a new technique called feedback MPC emerged James B. Rawlings 2017b. It is well known that, in control, the feedback strategy archives superior performances with respect to the open-loop strategy in a context of external disturbance. Rather than

applying an open-loop input resulting from the optimization problem, the feedback input results from a feedback policy based on the system's actual state and the open-loop input. This policy determination is an arduous task in the nonlinear case, and much of the research effort has been devoted to forms a feedback MPC that sacrifices optimality for simplicity. One approach resulting from these researches is the tube-based approach. Tube MPC is based on the separation between a nominal system driven by an open-loop MPC designed without any disturbance considerations and the full system subject to a feedback policy. A tube is defined around the nominal trajectory as all the possible disturbances realizations on the real system under the given feedback policy's action. The definition of such a tube can be based on previous knowledge, stochastic assumption, or online learning, but it always needs to meet some robust criterion defined later in this work (Fan, J. Nguyen, et al. 2020 Berkenkamp et al. 2017). While the computation and implementation of this tube is doable for linear system D. Mayne and Langson 2001 Chisci, Rossiter, and Zappa 2001, it become intractable for large nonlinear system as UAVs. To overcome such limitation, Mayne and Kerrigan present in David Q. Mayne and Eric C. Kerrigan 2007 a solution based on local linearization and approximation of the tube using the Monte-Carlo method. This method lays the foundation of robust MPC implementation on a real system, and this work will present an extension of it. In parallel, Fan, Agha-mohammadi, and Theodorou 2020 shows that an alternative of the Monte Carlo method for tube estimation is possible using deep-learning quantile regression. This method aims to consider the tube as a stochastic distribution of trajectories around the nominal system and then estimate the quantiles to ensure with a given probability that the system will remain inside the tube. The objective is then to propose a new combination of these approaches to bring robust tube MPC from theoretical research to real-life applications. In the next decades, robots and UVAs will become more present, and the applications more sensible (drugs or blood delivery are some excellent examples). These systems' safety will become a critical question, and robust Model Predictive Control could be the best answer.

1.2 Contribution

In this work, the focus will be on a subcategory of UAVs, which is multi-rotors drones. Their fast and unstable dynamic is an excellent example of a challenging system

for robust MPC. Moreover, quad-copters are more and more present on the market and will continue to grow as they present many advantages such as simplicity of deployment and high maneuverability. With this project, the objective is to contribute to bringing robust tube MPC on real-life application by combining David Q. Mayne and Eric C. Kerrigan 2007 with the quantile regression approach developed by Fan, Aghamohammadi, and Theodorou 2020. This work sets foundations for in-depth research by showing robust behavior and bringing proof of real-life implementation concepts. Moreover, being driven by hardware implementation, this work will concatenate several programming tools to provide a development platform for further research and improvements.

1.3 Outline

This thesis is organized around 5 chapters describe as follow :

- Chapter 2 derives the model of the quad-copter used in this project and on which the controller is based. The references frames are describes as well as the system kinematic and dynamic.
- Chapter 3 gives the description of the control strategy. A brief overview of NMPC is given before going into the robust tube-MPC and this project contribution. Lastly, the deep-learning regression is exposed.
- Chapter 4 describs the experimental setup from trajectory generation to hardware implementation passing by the different tools implemented. A general overview is given as well as precise implementation details on the used solvers ACADOS/CASADI, the simulation environment based on Gazabo and SITL, and the hardware features.
- Chapter 5 show the validity of the presented approach by presenting experiments results for robust behavior and quadrotor implementation.
- Chapter 6 conclude this theses and gives leads for further improvements.

Chapter 2

System description

According to the introduction, the focus will be put on the control of quadcopter UAV. This chapter describes the configuration of such a robot, the mathematical model, and the different frames in which it takes place. Finally, a state place model is derived and, from that, a linear approximation.

2.1 Preliminaries

A quadcopter is an under-actuated aircraft with fixed pitch angle four rotors. Many configurations are then possible, but the most widely used and used in this work is the X configuration due to the increased stability and reactivity of the two motors balancing each ax. In this configuration, the quadrotor's attitude and position can be controlled to desired values by changing the four motors' speed, generating torque thanks to the difference between the four thrusts created by the rotating propellers. These propellers are divided into two groups. In each group, two diagonally opposite motors can be distinguished thanks to their rotation direction:

- front-left and rear-right propellers (numbers 3 and 4 in Figure 2.1.1), rotating clockwise;
- front-right and rear-left propellers (numbers 1 and 2 in Figure 2.1.1), rotating counter clockwise.

Six degrees of freedom are required in describing any time-space motion of a rigid body aircraft. They are three barycenter movements and three angular motions around the barycenter, namely, forward and backward movements, lateral movement, vertical

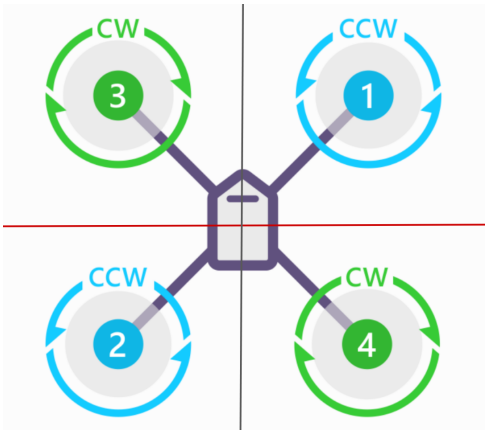
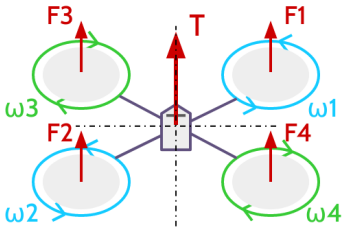


Figure 2.1.1: X configuration of quadcopters, engine indexing and respective propellers rotations

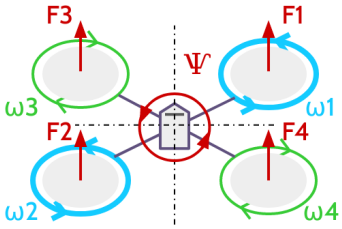
motion, roll, pitch, and yaw motions. Depending on the speed rotation of each propeller, it is possible to identify the four basic actuations of the quadrotor (Figure 2.1.2): the **thrust** caused by the addition of rotors rotation, the **yawing moment** is caused by the unbalanced of the four rotors rotational speeds, the **pitching moment** and **rolling moment** caused by the difference of four rotors thrust, the gravity, the gyroscopic effect, and the yawing moment (gyroscopic effect only appears in the lightweight construction quadrotor).

Because of four inputs and six outputs in a quadrotor, such a quadrotor is considered an underactuated nonlinear complex system. Some assumptions are made in the process of quadrotor modeling: the quadrotor is a rigid body; the structure is symmetric; the ground effect is ignored; the external forces other than the gravity are ignored. These assumptions can seem heavy, but the project's objective is to develop a robust control technique that can be used even in the case of model approximation.

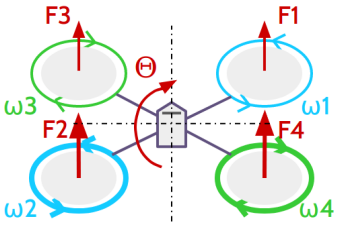
$$F_1 + F_2 + F_3 + F_4 = T \tag{2.1}$$



$$(\omega_1 + \omega_2) > (\omega_3 + \omega_4) \tag{2.2}$$



$$M_2 + M_4 > M_3 + M_1 \tag{2.3}$$



$$M_1 + M_4 < M_3 + M_2 \tag{2.4}$$

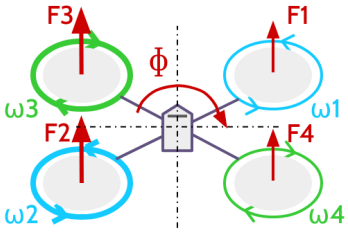


Figure 2.1.2: Basic actuation of the quadcopter. Thrust T (2.1), Yaw Ψ (2.2), Pitch Θ (2.3), Roll Φ (2.4)

Before going into the mathematical description detail, it is necessary to introduce the coordinates to describe the structure's position and orientation. For the quadrotor, it is possible to use two systems. The first is fixed, and the second is mobile. The fixed coordinate system, also called inertial, is a system where the first Newton's law is considered valid. As fixed coordinate system, or ***E-Frame***, we use the O_{ENU} systems, where ENU stands for *East-North-Up*. As we can observe from Figure 2.1, its vectors are directed Nord, East, and opposite to the center of the Earth.

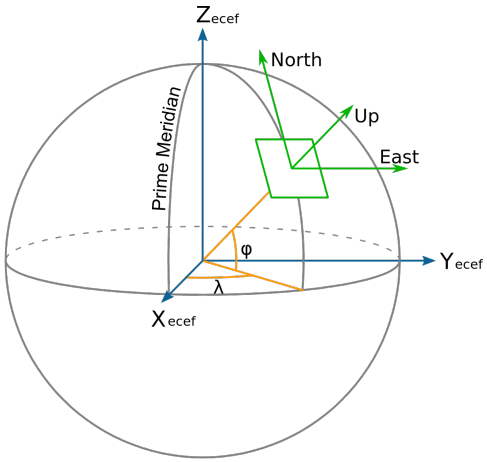


Figure 2.1.3: O_{NED} fixed reference frame.

The mobile reference system that we have previously mentioned is united with the barycenter of the quadrotor. In the scientific literature Bresciani 2008 it is called O_B system or fixed body frame, ***B-Frame***. Figure 2.2 illustrates the two coordinate systems.

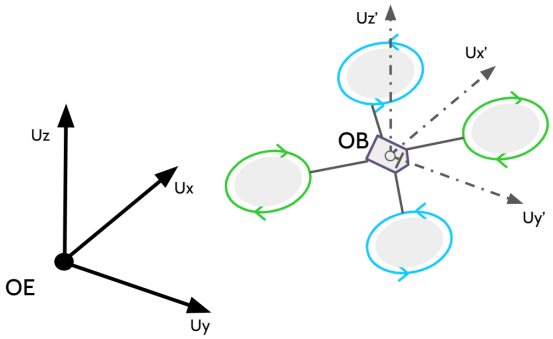


Figure 2.1.4: Fixed E-frame and B-frame mobile frame.

2.2 Euler angles

The Euler angles are three angles introduced by Leonhard Euler to describe a rigid body's orientation using a sequence of consecutive rotations. They are typically denoted as $\phi \in]-\pi, \pi]$, $\theta \in]-\pi/2, \pi/2]$, $\psi \in]-\pi, \pi]$. They can also describe the orientation of a frame relative to another. In the present case, the Euler angle is used to transform the mobile quadcopter body frame to the fixed inertial frame. Many combinations of Euler angle are possible for such description Henderson 1977 but the RPY (*Roll-Pitch-Yaw*) or ZYX convention is used in this work. This convention describes the body frame orientation Θ into the intertial frame using the following sequence of elementary rotation Bresciani 2008 :

$$x^E = (\boldsymbol{R_z}(\psi)\boldsymbol{R_y}(\theta)\boldsymbol{R_x}(\phi)) * x^B \tag{2.5}$$

With $\boldsymbol{R_z}(\psi)$, $\boldsymbol{R_y}(\theta)$,$\boldsymbol{R_x}(\phi)$ define as follow.

$$\boldsymbol{R_z}(\psi) = \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{R_y}(\theta) = \begin{bmatrix} c_\theta & 0 & c_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \quad \boldsymbol{R_x}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix} \tag{2.6}$$

where $c_\psi = cos(\psi)$, $s_\psi = sin(\psi)$, $c_\theta = cos(\theta)$, $s_\theta = sin(\theta)$, $c_\phi = cos(\phi)$, $s_\phi = sin(\phi)$. Thus, the inertial frame coordinates and the body-fixed frame coordinates are linked by the rotation matrix $\boldsymbol{R_\Theta}(\phi, \theta, \psi) \in SO(3)$:

$$x^E = \boldsymbol{R_\Theta} * x^B \tag{2.7}$$

$$\boldsymbol{R_\Theta} = \begin{bmatrix} c_\psi c_\theta & -s_\psi c_\phi + c_\psi s_\theta s_\phi & s_\psi s_\phi + c_\psi s_\theta s_\phi \\ s_\psi c_\theta & c_\psi c_\phi + s_\psi s_\theta s_\phi & -c_\psi s_\phi + s_\psi s_\theta c_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \tag{2.8}$$

This matrix describes the rotation from the body reference system to the inertial reference. Transposing it leads to the description of the rotation from the inertial frame to the body-fixed frame.

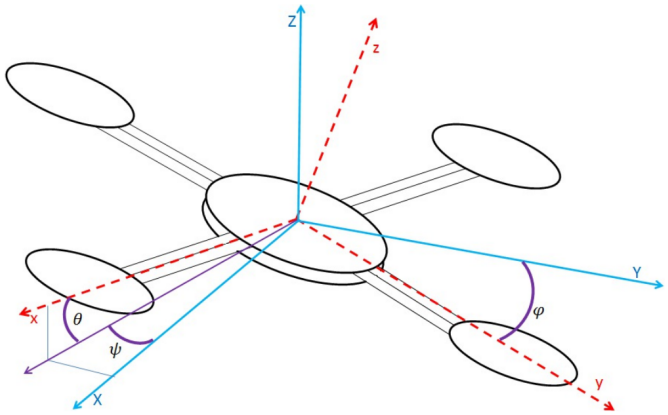


Figure 2.2.1: Euler angles.

As for the quadcopter orientation, it is possible to relate the body frame angular velocity in the inertial frame. It is therefore called Euler rates in the literature Bresciani 2008. The transfer matrice $\boldsymbol{T_\Theta}$ is then computed by evaluating the impact of individual angular variations on the rotation matrix :

$$\omega^B = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \boldsymbol{R_x(\phi)}^{-1} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \boldsymbol{R_x(\phi)}^{-1} \boldsymbol{R_y(\theta)}^{-1} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} = \boldsymbol{T_\Theta}^{-1} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.9}$$

$$\boldsymbol{T_\Theta}^{-1} = \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & c_\theta s_\phi \\ 0 & -s_\phi & c_\theta c_\phi \end{bmatrix} \quad \boldsymbol{T_\Theta} = \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi / t_\theta & c_\phi / c_\theta \end{bmatrix} \tag{2.10}$$

Then, T represent the transformation from angular velocity to Euler Rate and T^{-1} does the inverse operation. Note that $t_\psi = tan(\psi)$, $t_\theta = tan(\theta)$ and $t_\phi = tan(\phi)$.

2.3 Quadrotor mathematical model

This section aims to give a better understanding of the kinematic and dynamic models of the quadrotor to provide a sufficiently reliable model for simulating and developing of the control algorithms. The quadcopter is assumed to be a 6 DOF rigid body, and thus Newton-Euler equations can be used. The development will take place in the two frames explained in section 2.1.1. However, the rotational motion equation will be expressed in the body frame for the following reason : :

- The inertia matrix is time-invariant.
- Advantage of body symmetry can be taken to simplify the equations.
- Measurements taken on-board are easily converted to body-fixed frame.
- Control forces are almost always given in body-fixed frame.

2.3.1 Kinematic

From the literature Bresciani 2008, the kinematics of a generic 6DOF rigid-body can be describe as :

$$\dot{\boldsymbol{\xi}} = \boldsymbol{J_\Theta \nu} \tag{2.11}$$

where $\dot{\boldsymbol{\xi}}$ is the generalized velocity vector in the inertial frame, $\boldsymbol{\nu}$ is the generalized velocity vector in the body frame and $\boldsymbol{J_\Theta}$ in the transformation matrix.

$\boldsymbol{\xi}$ is composed of the quadcopter linear position $\Gamma^E \in \mathbb{R}^3[m]$ and angular orientation $\Theta^E \in \mathbb{R}^3[rad]$ such as :

$$\boldsymbol{\xi} = [\boldsymbol{\Gamma^E} \quad \boldsymbol{\Theta^E}] = [x \quad y \quad z \quad \phi \quad \theta \quad \psi]^T \tag{2.12}$$

Similarly, $\boldsymbol{\nu}$ is composed of quadcopter linear velocity $\boldsymbol{V^B}[m.s^{-1}] \in \mathbb{R}^3$ and angular rate $\boldsymbol{\omega^B}[rad.s^{-1}] \in \mathbb{R}^3$:

$$\boldsymbol{\nu} = [\boldsymbol{V^B} \quad \boldsymbol{\omega^B}] = [u \quad v \quad w \quad p \quad q \quad r]^T \tag{2.13}$$

In addition, J_Θ is composed of the two euler matrices R_Θ and T_Θ developed in section

2.1.2:

$$\boldsymbol{J_\Theta} = \begin{bmatrix} \boldsymbol{R_\Theta} & \boldsymbol{0_{3x3}} \\ \boldsymbol{0_{3x3}} & \boldsymbol{T_\Theta} \end{bmatrix} \tag{2.14}$$

Thus, the kinematic of the quadrotor is given by :

$$\begin{cases} \dot{x} = u[c_\psi c_\theta] - v[-s_\psi c_\phi + c_\psi s_\theta s_\phi] + w[s_\psi s_\phi + c_\psi s_\theta s_\phi] \\ \dot{y} = u[s_\psi c_\theta] + v[c_\psi c_\phi + s_\psi s_\theta s_\phi] - w[-c_\psi s_\phi + s_\psi s_\theta c_\phi] \\ \dot{z} = -u[-s_\theta] + v[c_\theta s_\phi] + w[c_\theta c_\phi] \\ \dot{\phi} = p + q[s_\phi t_\theta] + r[c_\phi t_\theta] \\ \dot{\theta} = q[c_\phi] - r[s_\phi] \\ \dot{\psi} = q[s_\phi / t_\theta] - r[c_\phi / c_\theta] \end{cases} \tag{2.15}$$

2.3.2 Dynamic

Concerning the dynamic part, the literature Bresciani 2008 gives the following generic expressions for a 6 DOF rigid body describe by Newton-Euler equations in the body-fixed frame:

$$\begin{cases} m(\dot{\boldsymbol{V}^B} + \boldsymbol{\omega}^B \wedge \boldsymbol{V}^B) = \boldsymbol{F}^B & \text{Newton's law} \\ \boldsymbol{I}\dot{\boldsymbol{\omega}^B} + \boldsymbol{\omega}^B \wedge \boldsymbol{I}.\boldsymbol{\omega}^B) = \boldsymbol{\tau}^B & \text{Euler's equation} \end{cases} \tag{2.16}$$

Where $\boldsymbol{F}^B$ and $\boldsymbol{\tau}^B$ are the vectors $\in \mathbb{R}^3$ containing, respectively, the total forces and total torques applied to the body frame. $\boldsymbol{I} \in \mathbb{R}^{3\times3}$ is the diagonal inertia matrix with $[I_x \quad I_y \quad I_z]$ as diagonal vector and m the mass of the quadcopter. It is important to state that two assumptions have been made in the previous statement. First, the body frame's origin should be coincident with the quadrotor's barycenter, and second, the body frame has to be aligned with the body principal axes of inertia to keep the diagonal shape of $\boldsymbol{I}$. Both of these assumptions were made to keep the equations simpler.
$\boldsymbol{F}^B$ and $\boldsymbol{\tau}^B$, the force and torque acting on the system, is composed of :

$$\boldsymbol{F}^B = mg\boldsymbol{R}_\Theta^T.\boldsymbol{u_z} - f_t\boldsymbol{u_z'} \tag{2.17}$$

$$\boldsymbol{\tau}^B = \boldsymbol{\tau_m} \tag{2.18}$$

Where g is the gravitational acceleration, f_t the total trust produce by the propellers

and τ_m the torque generated by the difference of rotors speed. Note that the parasite forces or torque such as wind effect and gyroscopic moments have not been taken into account in the development. Indeed, the objective will be to develop a control strategy robust enough to not suffer from these approximations.
Using (2.17) and (2.18) in (2.16) gives the following description of the system's dynamic :

$$\begin{cases} \dot{u} = rv - qw + g[s_\theta] \\ \dot{v} = pw - ru - g[s_\phi c_\theta] \\ \dot{w} = qu - pv - g[c_\phi c_\theta] + \frac{ft}{m} \\ \dot{p} = \frac{I_y - I_z}{I_x} rq + \frac{\tau_x}{I_x} \\ \dot{q} = \frac{I_z - I_x}{I_y} rq + \frac{\tau_y}{I_y} \\ \dot{r} = \frac{I_x - I_y}{I_z} rq + \frac{\tau_z}{I_z} \end{cases} \tag{2.19}$$

However, another approach would be to describe the quadrotor position's dynamic using the inertial frame as developed in T. Nguyen et al. 2017. This allows having an use-full alternative form of the dynamic model used in the state-space model to simplify the control and implementation study. Rewriting Newton's equation in the E-frame gives :

$$m\ddot{\boldsymbol{\Gamma}}^{\boldsymbol{E}} = \boldsymbol{R_\Theta}.\boldsymbol{F}^{\boldsymbol{B}} = mg\boldsymbol{u_z} - f_t\boldsymbol{R_\Theta}.\boldsymbol{u_z}' \tag{2.20}$$

with $\boldsymbol{R_\Theta}$ defined in (2.8). Then, the new description of the dynamic is :

$$\begin{cases} \ddot{x} = \frac{f_t}{m}[s_\psi s_\phi + c_\psi s_\theta s_\phi] \\ \ddot{y} = \frac{f_t}{m}[-c_\psi s_\phi + s_\psi s_\theta c_\phi] \\ \ddot{z} = -g + \frac{f_t}{m}[c_\theta c_\phi] \\ \dot{p} = \frac{I_y - I_z}{I_x} rq + \frac{\tau_x}{I_x} \\ \dot{q} = \frac{I_z - I_x}{I_y} rq + \frac{\tau_y}{I_y} \\ \dot{r} = \frac{I_x - I_y}{I_z} rq + \frac{\tau_z}{I_z} \end{cases} \tag{2.21}$$

2.3.3 State-space model

The definition of the state-space model is the last part of the system description and will summarize the previous sections. Such model is express by :

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{2.22}$$

One way to define the state of the system would be to use $\boldsymbol{x}$ the state of the quadrotor written as :

$$\boldsymbol{x} = [x \quad y \quad z \quad \dot{x} \quad \dot{y} \quad \dot{z} \quad p \quad q \quad r \quad \phi \quad \theta \quad \psi]^T \in \mathbb{R}^{12} \tag{2.23}$$

and $\boldsymbol{u}$ the input to the system such as :

$$\boldsymbol{u} = [f_t \quad \tau_x \quad \tau_y \quad \tau_z]^T = [f_t \quad \boldsymbol{\tau^B}]^T \in \mathbb{R}^4 \tag{2.24}$$

Thus, using (2.21) and (2.15), the state space model can be expressed as :

$$\begin{cases} \dot{x} = \dot{x} \\ \dot{y} = \dot{y} \\ \dot{z} = \dot{z} \\ \ddot{x} = \frac{f_t}{m}[s_\psi s_\phi + c_\psi s_\theta s_\phi] \\ \ddot{y} = \frac{f_t}{m}[-c_\psi s_\phi + s_\psi s_\theta c_\phi] \\ \ddot{z} = -g + \frac{f_t}{m}[c_\theta c_\phi] \\ \dot{p} = \frac{I_y - I_z}{I_x} rq + \frac{\tau_x}{I_x} \\ \dot{q} = \frac{I_z - I_x}{I_y} pr + \frac{\tau_y}{I_y} \\ \dot{r} = \frac{I_x - I_y}{I_z} pq + \frac{\tau_z}{I_z} \\ \dot{\phi} = p + q[s_\phi t_\theta] + r[c_\phi t_\theta] \\ \dot{\theta} = q[c_\phi] - r[s_\phi] \\ \dot{\psi} = q[s_\phi / t_\theta] - r[c_\phi / c_\theta] \end{cases} \tag{2.25}$$

2.3.4 Reduced model

Due to implementation constraints that will be develop later one, a reduced model is presented. This model will be referenced as :

$$\dot{\boldsymbol{x}} = \boldsymbol{f_r}(\boldsymbol{x}, \boldsymbol{u}) \tag{2.26}$$

and the state and input are redefine as :

$$\boldsymbol{x} = [x \quad y \quad z \quad \dot{x} \quad \dot{y} \quad \dot{z} \quad \phi \quad \theta \quad \psi]^T \in \mathbb{R}^9 \tag{2.27}$$

and $\boldsymbol{u}$ the input to the system such as :

$$\boldsymbol{u} = [f_t \quad p_x \quad q_y \quad r_z]^T = [f_t \quad \boldsymbol{\omega^B}]^T \in \mathbb{R}^4 \tag{2.28}$$

Thus, using (2.21) and (2.15), the state space model can be expressed as :

$$\begin{cases} \dot{x} = \dot{x} \\ \dot{y} = \dot{y} \\ \dot{z} = \dot{z} \\ \ddot{x} = \frac{f_t}{m}[s_\psi s_\phi + c_\psi s_\theta s_\phi] \\ \ddot{y} = \frac{f_t}{m}[-c_\psi s_\phi + s_\psi s_\theta c_\phi] \\ \ddot{z} = -g + \frac{f_t}{m}[c_\theta c_\phi] \\ \dot{\phi} = p + q[s_\phi t_\theta] + r[c_\phi t_\theta] \\ \dot{\theta} = q[c_\phi] - r[s_\phi] \\ \dot{\psi} = q[s_\phi / t_\theta] - r[c_\phi / c_\theta] \end{cases} \tag{2.29}$$

2.3.5 Linearization

As suggested in the project objectives, the robust MPC feedback policy will be based on a linearized version of the quadcopter dynamic, reducing the overall computational cost. This approximation will take place at each solving step to not be limited by the validity region of the linearization. Thus, for each operating point (x_L, u_L) the linear

system take the form of :

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} \tag{2.30}$$

with :

$$A = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}_{(x=x_L)} \qquad B = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}_{(u=u_L)} \tag{2.31}$$

Chapter 3

Controller design

This chapter constitutes the core of this work as it describes the theoretical approach. From the dynamic described in chapter 2, the objective is to present a new way to implement robust MPC control for real-time application based on tube MPC and deep learning quantile regression. This challenge has to be overcome while keeping theoretical consistency.

3.1 Nonlinear Model Predictive Control

To set the basis of this work, a formal description of non-linear MPC (NMPC) with finite time horizon T is presented. The stability is discussed for a system without disturbances to introduce notations and tools for further development. Moreover, the NMPC formulation will be used for trajectory generation purposes, as discussed in section 4.2.

3.1.1 Formulation

The target system subject to NMPC control takes the form :

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{3.1}$$

With f as described in the previous chapte **??** and discretized with a sampling period δ and subject to the following constraints:

$$\boldsymbol{x_t} \in \mathbb{X}, \quad \boldsymbol{u_t} \in \mathbb{U}, \quad \forall t \geq 0 \tag{3.2}$$

where $\boldsymbol{x_t} \in \mathbb{X} \subset \mathbb{R}^{n_x}$. is the state of the system, $\boldsymbol{u_t} \in \mathbb{U} \subset \mathbb{R}^{n_u}$ is the input applied to the system. In this work $\mathbb{U}$ and $\mathbb{X}$ take the form :

$$\begin{aligned} \mathbb{X} &= \{\boldsymbol{x} \in \mathbb{R}^{n_x} \mid x_{min} \leq x \leq x_{max}\} \\ \mathbb{U} &= \{\boldsymbol{u} \in \mathbb{R}^{n_u} \mid u_{min} \leq u \leq u_{max}\} \end{aligned} \tag{3.3}$$

where $x_{min}, x_{max}, u_{min}, u_{max}$ are given constant vectors and $\leq$ element wise.
Let $T \in \mathbb{N}$ denote the planing horizon of the NMPC problem. In order to distinguish clearly internal variable from system state, the notation $\boldsymbol{x_{k|t}}$ is used. It denotes the variable $\boldsymbol{x_k}$ for $k = 0, ..., T$ within the NMPC problem at time t. Then, $x_{.|t}$ denotes the set of variables $\{\boldsymbol{x_{k|t}}\}_{k=0}^{T}$. The finite horizon NMPC control can now be defined as the repeated solution of the open-loop optimal control problem taking the form of :

$$\min_{\boldsymbol{u}_{.|t} \in \mathbb{U}} J(\boldsymbol{x}_{.|t}, \boldsymbol{u}_{.|t}) \tag{3.4}$$

with :

$$J(\boldsymbol{x}_{.|t}, \boldsymbol{u}_{.|t}) = \sum_{k=0}^{k=T} F(\boldsymbol{x_{k|t}}, \boldsymbol{u_{k|t}}) \tag{3.5}$$

$$\forall k = 0, ..., T \tag{3.6}$$

subject to:

$$\begin{aligned} &\boldsymbol{x_{k+1|t}} = f(\boldsymbol{x_{k|t}}, \boldsymbol{u_{k|t}}), \\ &\boldsymbol{u_{k|t}} \in \mathbb{U}, \quad \forall k \in [0, T], \\ &\boldsymbol{x_{k|t}} \in \mathbb{X}, \quad \forall k \in [0, T], \\ &\boldsymbol{x_{0|t}} = \boldsymbol{x_t} \end{aligned} \tag{3.7}$$

With J the cost function to be minimized and F the stage cost. F is often chosen as quadratic for efficiency and simplicity purpose taking the form H. Chen and Allgöwer 1998b:

$$F(\boldsymbol{x_{k|t}}, \boldsymbol{u_{k|t}}) = \boldsymbol{x_{k|t}}^T Q \boldsymbol{x_{k|t}} + \boldsymbol{u_{k|t}}^T R \boldsymbol{u_{k|t}}, \tag{3.8}$$

$Q \in \mathbb{R}^{n_x * n_x}$, $R \in \mathbb{R}^{n_u * n_u}$ both being positive definite are symmetric weighting matrices. Note the initial condition, the system model is initialized by the actual system state, that is in general assumed to be measured or must be estimated. Let $\boldsymbol{x}^*_{.|t}$ and $\boldsymbol{u}^*_{.|t}$ be minimizer of the problem at time t and thus the solution of the NMPC problem.
The closed-loop control is set to $\boldsymbol{u_t} = \boldsymbol{u_{0|t}}^*$ and apply to the system. Then the system state is measure and the NMPC is solve again.

3.1.2 Stability

Ensure stability for the finite horizon closed loop NMPC as it can be done for infinite horizon is unfeasible with the presented formulation (Findeisen 2004). The basic idea to overcome this limitation is to approximate an infinite horizon prediction by adding an additional terminal cost $E(\boldsymbol{x_{T|t}})$ into the finite horizon formulation and impose a final constraint on $\boldsymbol{x_{T|t}}$ as describe by H. Chen and Allgöwer 1998a :

$$\min_{\boldsymbol{u}_{.|t} \in \mathbb{U}} J(\boldsymbol{x}_{.|t}, \boldsymbol{u}_{.|t})$$

(3.9)

with :

$$J(\boldsymbol{x}_{.|t}, \boldsymbol{u}_{.|t}) = \sum_{k=0}^{k=T} F(\boldsymbol{x_{k|t}}, \boldsymbol{u_{k|t}}) + E(\boldsymbol{x_{T|t}}) \tag{3.10}$$

$$\forall k = 0, ..., T \tag{3.11}$$

subject to:

$$\begin{aligned}
&\boldsymbol{x_{k+1|t}} = f(\boldsymbol{x_{k|t}}, \boldsymbol{u_{k|t}}),\\
&\boldsymbol{u_{k|t}} \in \mathbb{U}, \quad \forall k \in [0, T],\\
&\boldsymbol{x_{k|t}} \in \mathbb{X}, \quad \forall k \in [0, T],\\
&\boldsymbol{x_{T|t}} \in \mathbb{X}_f,\\
&\boldsymbol{x_{0|t}} = \boldsymbol{x_t}
\end{aligned} \tag{3.12}$$

To give an intuition, the idea is to be sure that the trajectories of the closed-loop system remain within some terminal region $\mathbb{X}_f$ for the time interval $[t + T, \infty)$. $\mathbb{X}_f$

is constructed such as a local state feedback law $\boldsymbol{u} = \kappa(\boldsymbol{x})$ asymptotically stabilize the nonlinear system in $\mathbb{X}_f$ and renders $\mathbb{X}_f$ invariant.

Definition 3.1.1 (Positive invariant set) *A closed set $\mathbb{A}$ is positive invariant for the system $\boldsymbol{x_{t+1}} = f(\boldsymbol{x_t})$ if $\boldsymbol{x} \in \mathbb{A}$ implies $f(\boldsymbol{x_t}) \in \mathbb{A}$.*

Thus, due to the invariant nature of $\mathbb{X}_f$, it suffices to impose an additional terminal constraint $\boldsymbol{x_{T|t}} \in \mathbb{X}_f$ to ensure that the trajectory indeed remains in $\mathbb{X}_f$ for infinity.

It is common practice to define the local state feedback law as $\boldsymbol{u} = K_{LQR}\boldsymbol{x}$ with K_{LQR} being the LQR stabilizing gain computed offline. The terminal cost P can be extracted from this local state feedback law as solution of the Riccati equation.
Details of stability proof have been voluntary omitted and interested reader could refer to H. Chen and Allgöwer 1998b and Lazar and Spinu 2015 for terminal set computation and terminal cost necessary proprieties.

3.1.3 Inherent robustness

NMPC in its stable formulation shows some inherent stability propriety for vanishing small perturbations. Proof and developments can be found in James B. Rawlings 2017a, Allan et al. 2017 , Yu, Reble, et al. 2011 . Even if this work does not go into inherent stability details, it is important to note that it exists and can be used for experiments. However, for real-life applications where strong guarantees are needed, it is preferable to design a robust controller i.e., able to cope with bounded uncertainty (James B. Rawlings 2017a).

3.2 Tube based Model Predictive Control

Tube-based model predictive control constitutes an implementable form of feedback MPC addressing disturbances in the controlled system. In this section, the main features and results of this technique are described as well as the limitations and how this work presents a new way to use it.

3.2.1 Description

The system to be controlled is described as :

$$\boldsymbol{x_{t+1}} = f(\boldsymbol{x_t}, \boldsymbol{u_t}) + \boldsymbol{w_t} \tag{3.13}$$

With f as described in the previous chapter 2.22, discretized with sampling period of δ, $\boldsymbol{x_t} \in \mathbb{R}^{n_x}$ the system state, $\boldsymbol{u_t} in \mathbb{R}^{n_u}$ the control input and $\boldsymbol{w_t}$ an additive disturbance assumed to lie in a set $\mathbb{W}$, compact and containing the origin. This system is constrained according to $\boldsymbol{x_t} \in \mathbb{X}$ and $\boldsymbol{u_t} \in \mathbb{U} \quad \forall t$.
From this, one can define a nominal system :

$$\boldsymbol{z_{t+1}} = f(\boldsymbol{z_t}, \boldsymbol{v_t}) \tag{3.14}$$

This nominal system does not take into account the disturbance and is a useful tool for further developments. $\boldsymbol{z_t}$ is the nominal state and $\boldsymbol{v_t}$ nominal control input both under constraints such as $\boldsymbol{z_t} \in \mathbb{Z}$ and $\boldsymbol{v_t} \in \mathbb{V}$.
The main objective is then to design a controller to steer the system's initial state to the neighborhood of an equilibrium state of the nominal system against every admissible disturbance. This has to be done respecting the state constraints $\boldsymbol{x_t} \in \mathbb{X}$ and input constraints $\boldsymbol{u_t} \in \mathbb{U}$. Two main issues have to be overcome, robust stability (stability for every admissible disturbance sequence) and robust performance (adequate performance for every admissible disturbance sequence).
The NMPC formulation has to be modified for a feedback form to deal with these problems. In a more general consideration, feedback forms give best performances in control theory as the system's actual state (disturbance included) is taken into account into the generated command. In feedback MPC, the control sequence is replaced by a policy $\boldsymbol{\mu} = \{\mu_0, \mu_1 ...\}$ i.e., a sequence of control law and aim to ensure that the deviation of the real system from the nominal one is much less than for the open-

loop MPC. Feedback MPC is an active field of research as the computation of these policies is usually prohibitive (D. Q. Mayne, E. C. Kerrigan, et al. 2011). Two main categories arise for robust implementation. Min-max feedback MPC is one of those two and provides satisfying results but are impossible to implement, the computational cost increasing exponentially with system's dimension (Qin and Badgwell 2003). The second one is the subject of this work and is called Tube-MPC.
Tube MPC is an implementable form of feedback MPC aiming to build a tube centered on the nominal system's trajectory and whose cross-section is a robust positively invariant set for the system.

Definition 3.2.1 (Robust positive invariant set) *A closed set $\mathbb{Y}$ is Robust positive invariant for the system defined in 3.13 if*

$$\boldsymbol{x_t} \in \mathbb{Y} \quad \Rightarrow \quad \boldsymbol{x_{t+1}} \in \mathbb{Y} \quad \forall \boldsymbol{w_t} \in \mathbb{W}, \quad \forall t \quad \mathbb{N}^+. \tag{3.15}$$

To give an intuition, a system under open-loop or feedback control, trying to follow a given trajectory while being subject to uncertainty, will create a bundle (or a tube) of alternative trajectory around it. Each of those corresponds to a particular realization of the noise. Robust tube-MPC is all about shaping this tube (i.e., the possible trajectories) to ensure constraints satisfaction inside it. The shape of the tube depends on the initial state and the chosen feedback policy. However, as explained, finding the right policy is challenging and extracts the tube's shape even more in non-linear systems.
D. Q. Mayne, E. C. Kerrigan, et al. 2011 propose an approach based on an ancillary NMPC controller used to maintain the disturbed system close to the nominal one by minimizing the cost of the deviation. There is no state or input constraints for this second NMPC but a terminal cost pushing the final state toward the robust invariant set. This technique gives strong guarantees on robust behavior, and the flexibility of tuning possibilities helps to reject disturbance more effectively. However, even if presented as computationally efficient, this method requires solving two consecutive online NMPC. The example presented in D. Q. Mayne, E. C. Kerrigan, et al. 2011 is based on slow dynamics compared to the fast dynamic of a UAV explaining the good results. However, in the framework of this project, the computational burden of the aforementioned technique is too heavy.

Going back for a more general approach, it is interesting to define :

$$\boldsymbol{e_{t+1}} = \boldsymbol{x_{t+1}} - \boldsymbol{z_{t+1}} \tag{3.16}$$

Then, let the input $\boldsymbol{u_t}$ of the system 3.13 be:

$$\boldsymbol{u_t} = \boldsymbol{v_t} + \kappa(\boldsymbol{e_t}) \tag{3.17}$$

It is now important to give the definition of robust control invariant set as in Yu, Maier, et al. 2013.

Definition 3.2.2 (Robust Control invariant set) *A set $\Omega \subset \mathbb{X} \subset \mathbb{R}^{n_x}$ is robust control invariant set for the system 3.16 if there exists a control law $\kappa(.)$ with $\kappa(.) + \boldsymbol{v_k} \in \mathbb{U} \subset \mathbb{R}^{n_u}$ such as for all $\boldsymbol{x_{t_0}} \in \Omega$ and all $\boldsymbol{w_t} \in \mathbb{W}$, $\boldsymbol{x_t} \in \Omega$ for all $t \geq t_0$*

Furthermore, if Ω can be defined as robust control invariant set for 3.16, Ω can also be considered as robust invariant set for the real system 3.13 (Yu, Maier, et al. 2013). According to Gao et al. 2014 and D. Q. Mayne, Seron, and Raković 2005, with Ω robust invariant for the system 3.13 then, if the initial state $\boldsymbol{x_0}$ start sufficiently close to the nominal state $\boldsymbol{v_0}$, the control law 3.17 will keep the trajectory within the robust positively invariant set Ω centred at the predicted nominal trajectory, i.e the tube, for all admissible sequence $\boldsymbol{w_k}$:

$$\boldsymbol{x_0} \in \{\boldsymbol{z_0}\} \oplus \Omega \Rightarrow \boldsymbol{x_k} \in \{\boldsymbol{z_k}\} \oplus \Omega, \quad \forall \boldsymbol{w_k} \in \mathbb{W}, \quad \forall k \geq 0 \tag{3.18}$$

This also suggest that if the a nominal solution can be found for the nominal system under the tighten constraints :

$$\mathbb{Z} = \mathbb{X} \ominus \Omega \quad \mathbb{V} = \mathbb{U} \ominus \kappa(\Omega) \tag{3.19}$$

then the control law $\boldsymbol{u_t} = \boldsymbol{v_t} + \kappa(\boldsymbol{e_t})$ will ensure constraint satisfaction for the real system.

Note. According to Gao et al. 2014 $\oplus$ is define as the Minkowski sum and $\ominus$ the Ponttryagin difference. The Minkowski sum of two polytopes $\mathbb{P}$ and $\mathbb{P}$ is a polytope:

$$\mathbb{P} \oplus \mathbb{Q} = \{x \in \mathbb{R}^n : x + q \in \mathbb{P}, \quad \forall q \in \mathbb{Q}\} \tag{3.20}$$

And the Pontryagin difference of two polytopes $\mathbb{P}$ and $\mathbb{P}$ is a polytope:

$$\mathbb{P} \ominus \mathbb{Q} = \{x + q \in \mathbb{R}^n : x \in \mathbb{P}, \quad \forall q \in \mathbb{Q}\} \tag{3.21}$$

Concerning the choice of κ, the choice of a stabilizing gain K is a choice often made in the literature (Yu, Maier, et al. 2013, Gao et al. 2014).

$$\boldsymbol{u_t} = \boldsymbol{v_t} + K\boldsymbol{e_t} \tag{3.22}$$

It is interesting to note that even D. Q. Mayne, E. C. Kerrigan, et al. 2011, proposing an ancillary MPC controller, highlight the promising potential of a time changing gain K such as:

$$\boldsymbol{u_t} = \boldsymbol{v_t} + K_t\boldsymbol{e_t} \tag{3.23}$$

The approach is chosen in this work and more details can be found later.

3.2.2 Nominal trajectory

The nominal system 3.14 constitutes a noise-less approximation of the real system and will be used as a nominal trajectory. The formulation of the controller meets the NMPC formulation discussed below with :

$$\min_{\boldsymbol{v_{.|t}} \in \mathbb{U}} J(\boldsymbol{z_{.|t}}, \boldsymbol{v_{.|t}}) \tag{3.24}$$

with :

$$J(\boldsymbol{z_{.|t}}, \boldsymbol{u_{.|t}}) = \sum_{k=0}^{k=T} F(\boldsymbol{z_{k|t}} - \boldsymbol{r_{t+k}}, \boldsymbol{v_{k|t}}) + E(\boldsymbol{z_{T|t}} - \boldsymbol{r_{t+T}}) \tag{3.25}$$

$$\forall k = 0, ..., T \tag{3.26}$$

subject to:

$$\begin{aligned}
&\boldsymbol{z_{k+1|t}} = f(\boldsymbol{z_{k|t}}, \boldsymbol{v_{k|t}}),\\
&\boldsymbol{v_{k|t}} \in \mathbb{V}, \quad \forall k \in [0, T],\\
&\boldsymbol{z_{k|t}} \in \mathbb{Z}, \quad \forall k \in [0, T],\\
&\boldsymbol{z_{T|t}} \in \mathbb{Z}_f,\\
&\boldsymbol{z_{0|t}} = \boldsymbol{z_t}
\end{aligned} \tag{3.27}$$

With $\mathbb{V}, \mathbb{Z}$ and $\mathbb{Z}_f$ compact sub-sets of respectively $\mathbb{U}, \mathbb{X}$ and $\mathbb{X}_f$. The choice of theses tightened sets $\mathbb{V} \subset \mathbb{U}$, $\mathbb{Z} \subset \mathbb{X}$ and $\mathbb{Z}_f \subset \mathbb{X}_f$ will be discuss later. $r_.$ denotes the discrete reference trajectory sample with a period δ. Finally $E(.)$ is chosen as discussed in the NMPC section.

3.2.3 Stabilazing gain for Tube MPC

The stabilizing gain used in the feedback policy :

$$\boldsymbol{u_t} = \boldsymbol{v_t} + K\boldsymbol{e_t} \tag{3.28}$$

to maintain the state of the uncertain system close to the nominal trajectory can be defined in several ways. It is crucial to keep in mind that this choice of gain molds the shape of the robust invariant set Ω i.e. the shape of the tube and the tightened constraints of the nominal system defined as 3.19
Yu, Maier, et al. 2013 propose a development starting from the Lipschitz propriety of the system continuous dynamic in the set $\mathbb{X}$ to extract the associate constants. As a reminder, the Lipschitz constant comes from the following equation for a given continuous function $g(x)$:

$$\|g(x_1) - g(x_{x_2})\| \leq L \|x_1 - x_{x_2}\| \tag{3.29}$$

The minimum value of L satisfying 3.29 is the Lipschitz constant. Base on this development, the authors present a way to compute the gain K and the set Ω offline, giving good online performances. However, it is highly conservative, and the computation of this constant is not straightforward as the number of dimension increase. The example presented holds for a system in $\mathbb{R}^2$, but the dynamic of the quadcopter lives in $\mathbb{R}^{12}$.
In the framework of this work, the system f as defined in chapter 2 2.22 can be linearized around a given equilibrium (x_L, u_L) point using the jacobian computation with sampling period δ. Let :

$$\boldsymbol{x_{t+1}} = A\boldsymbol{x_t} + B\boldsymbol{u_t} \tag{3.30}$$

with:

$$A = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}_{(x=x_L)} \qquad B = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}_{(u=u_L)} \tag{3.31}$$

$\boldsymbol{x_t} \in \mathbb{R}^{n_x}$ state of the system, $\boldsymbol{u_t} \in \mathbb{R}^{n_u}$ control input and A and B assumed to be controllable, it exist a stabilizing gain K_{LQR} result of the linear quadratic regulator defined as :

$$J = \sum_{k=0}^{\infty} (\boldsymbol{x_t}^T Q \boldsymbol{x_k} + \boldsymbol{u_t}^T R \boldsymbol{u_k}) \tag{3.32}$$

with $Q \in \mathbb{R}^{n_x * n_x}$, $R \in \mathbb{R}^{n_u * n_u}$ the gain K_{LQR} is define such as

$$\boldsymbol{u_t} = K_{LQR} \times \boldsymbol{x_t} \tag{3.33}$$

$$K_{LQR} = (R + B^T P B)^{-1} (B^T P A) \tag{3.34}$$

let P solution of the Riccati equation :

$$P = A^T P A - (A^T P B)(R + B^T P B)^{-1} (B^T P A) + Q \tag{3.35}$$

giving :

$$\boldsymbol{x_{t+1}} = A\boldsymbol{x_t} + B(K_{LQR}\boldsymbol{x_t}) = (A + BK_{LQR})\boldsymbol{x_t} \tag{3.36}$$

is asymptotically stable (James B. Rawlings 2017c). Using this propriety of our system, the feedback stabilizing gain K is chosen as K_{LQR}. The same approach is described in Gao et al. 2014 and can be compared as the linear version of tube MPC.(D. Q. Mayne, Seron, and Raković 2005).

The question of the equilibrium point around which the system is linearized can be discussed. Indeed, for computation efficiency reason, a right choice could be to find K_{LQR} off-line, taking as equilibrium point the hovering point of the quadcopter:

$$\boldsymbol{x_L} = 0 \in \mathbb{R}^{12}, \quad \boldsymbol{u_L} = [-m * g, 0, 0, 0] \tag{3.37}$$

With g gravity and m mass of the system. However, this choice is highly conservative and presents some risks as the linearization is only valid around the equilibrium point. In the framework of this work, the linearization is performed at each step of the trajectory i.e., for each sampling period δ to tackle this issue. The Riccati equation has to be recomputed to obtain $K_{LQR|t}$.

Finally, the feedback policy for the robust tube MPC controller use in this project is :

$$\boldsymbol{u_t} = \boldsymbol{v_t} + K_{LQR|t} \times (\boldsymbol{x_t} - \boldsymbol{z_t}) = \boldsymbol{v_t} + K_{LQR|t}\boldsymbol{e_t} \tag{3.38}$$

3.2.4 DeepLearning tube estimation

The objective is now to define a way to compute the tube to guarantee robust constraint satisfaction for the controlled system. Based on the previous choice of $K_{LQR|t}$ as feedback gain, inspiration from the linear version can be interesting. Indeed, for a linear system, an approximation of the robust invariant set can be computed using the algorithm described in Rakovic et al. 2005. Such an algorithm proved its efficiency but only for low dimension systems and, in the project's situation, the dimension of the dynamic makes it intractable.
Gao et al. 2014, uses an approach with fixed K_{LQR} and develops a computation of the robust invariant set using the Lipschitz constant associated with the system. However, for the same reason mention in subsection 3.2.3, this technique is not implemented in real life scenario.
We propose a more straightforward procedure to determine the tube by estimating it using deep learning quantile regression. To give an intuition, the idea is to make the system run multiple times in simulation under the action of the predefined feedback policy but with a nominal system constrain by $\mathbb{Z} = \mathbb{X}$ and $\mathbb{V} = \mathbb{U}$. The system will be under disturbance, and a bundle of trajectories will be created around the reference one. This bundle constitute an inner approximation of the tube as described earlier and can be seen as a Monte-Carlo approximation of it. As explained by D. Q. Mayne, E. C. Kerrigan, et al. 2011, an exact solution is not required in many cases, and this kind of approximation gives satisfying results even if transgression can occur. However, the approach takes in their work only takes into account the maximum spreading on the full trajectory and tightened these constraints using this fixed value forbidding time varying tubes and giving a conservative approach.
With deep-learning quantile regression, the objective is to compute offline a time-varying tube for the system under the action of the chosen feedback policy while ensuring that the real system will stay inside the tube with a probability α. This approach has roots in the work of Fan, Agha-mohammadi, and Theodorou 2020.

Let the coupled system be:

$$\begin{aligned} \boldsymbol{x_{t+1}} &= f(\boldsymbol{x_t}, \boldsymbol{u_t}) + \boldsymbol{w_t} \\ \boldsymbol{z_{t+1}} &= f_z(\boldsymbol{z_t}, \boldsymbol{v_t}) \\ \boldsymbol{\omega_{t+1}} &= f_\omega(\boldsymbol{\omega_t}) \\ P(d(\boldsymbol{x_t}, \boldsymbol{z_t}) \geq \boldsymbol{\omega_t}) &\geq \alpha, \quad t \in \mathbb{N}^+ \end{aligned} \tag{3.39}$$

with $\omega_t \in \mathbb{R}^{n_x}$ being the tube width with all the elements greater than 0. This coupled system define a tube around the nominal trajectory within which the real system will stay with a probability greater than $\alpha \in [0, 1]$. The distance function $d(\boldsymbol{x_t}, \boldsymbol{z_t})$ is taken as $\|\boldsymbol{x_t} - \boldsymbol{z_t}\|_2$.
Let $\Omega_\omega(t) \subset \mathbb{X}$ be a set :

$$\Omega_{\omega_t} = \{x \in \mathbb{X}, d(x, z_t) \leq \omega_t\} \tag{3.40}$$

According to the last definitions,3.39 define a sequences of sets $\{\Omega_{\omega_t}\}_{t=0}^{T}$ that form a tube around the nominal trajectory. The objective is then to learn it.
Deep learning quantile regression allow to do it, given data collected by making the system travel though reference trajectory. The objective is to learn f_ω. For a given data point $\{\boldsymbol{x_t}, \boldsymbol{u_t}, \boldsymbol{x_{t+1}}, \boldsymbol{z_t}, \boldsymbol{v_t}, \boldsymbol{z_{t+1}}, t\}$ let $\omega_t = d(x_t, z_t)$ be the input tube width to f_ω and $\boldsymbol{\omega_{t+1}} = d(K_{LQR|t} x_{t+1}, \boldsymbol{z_{t+1}})$ the candidate output. The candidate tube width at $t+1$ must be less than the estimate of the tube width at $t+1$ i.e $\omega_{t+1} \leq f_\omega^\theta(\omega_t)$. To train the network f_ω^θ the following check loss function is used :

$$\begin{aligned} L_\omega^\alpha(\theta, \delta) &= L^\alpha(\boldsymbol{\omega_{t+1}}, f_\omega^\theta(\boldsymbol{\omega_t})) \\ L^\alpha(y, r) &= \begin{cases} \alpha \mid y - r \mid \\ (1-\alpha) \mid y - r \mid \end{cases} \end{aligned} \tag{3.41}$$

An additional point is that it is possible to play with alpha to vary the acceptable margin. Details about deep learning quantile regression are omitted for brevity but can be found in Rodrigues and Pereira 2020 ,Koenker and Bassett 1978, Taylor 1999 Fan, Aghamohammadi, and Theodorou 2020. Once the training procedure is done, the time varying tube $\{\Omega_{\omega_t}\}_{t=0}^{Tt}$ is used to tightened the nominal constraints such as :

$$\begin{aligned} \mathbb{Z} &= \{\boldsymbol{z} \in \mathbb{R}^{n_z} \mid \boldsymbol{z_{min}} + \boldsymbol{\omega_t} \leq \boldsymbol{z} \leq \boldsymbol{z_{max}} - \boldsymbol{\omega_t}\} \\ \mathbb{V} &= \{\boldsymbol{v} \in \mathbb{R}^{n_v} \mid \boldsymbol{v_{min}} + K_{LQR|t}\boldsymbol{\omega_t} \leq \boldsymbol{v} \leq \boldsymbol{v_{max}} - K_{LQR|t}\boldsymbol{\omega_t}\} \end{aligned} \tag{3.42}$$

Chapter 4

Experimental Validation

This chapter describes the experimental setup used to validate the selected approach. Some first validation steps are performed with a custom Python package integrating simulation environment, MPC algorithm, Quadcopter model, and trajectory generation. Then, based on Robot Operating System (ROS), custom-developed nodes in Python/c++ and Gazebo simulator engine, some high fidelity simulations are performed. Finally, the objective is to run some tests on the hardware quadrotor. The workflow is sequenced in three steps :

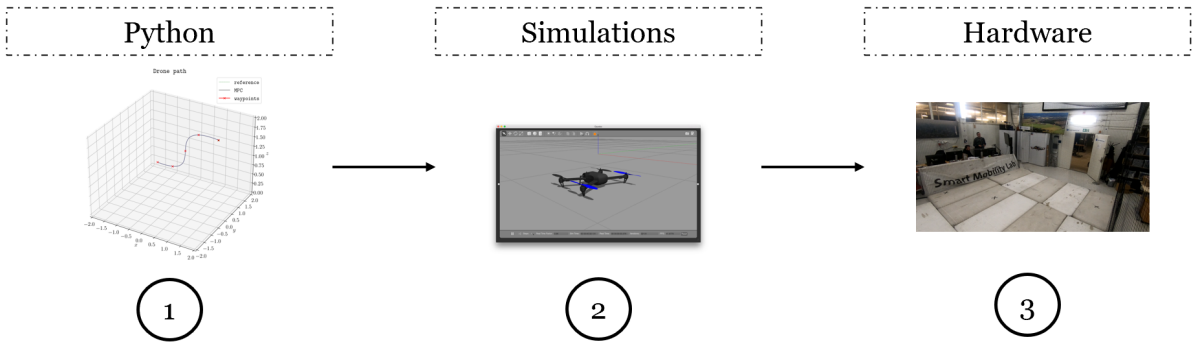


Figure 4.0.1: 1. Python validation step, 2. High fidelity simulation, 3.Hardware validation

The objective is to make the hardware system run in real-time with a control frequency, at least equal to 50Hz. The developed framework should allow generating trajectories that a quadrotor can follow. In a mindset of continuity for this project all the tools and custom package are chosen or developed to be flexible and reusable for further developments.

4.1 Simulation setup

4.1.1 Introduction to ROS

Robot Operating System (ROS) is a flexible framework for robotic research, based on a collection of tools, libraries, and conventions that simplify creating complex and robust behavior. This tool is a reference in robotic research and industry. ROS was designed with open-source in mind, intending that users would choose the configuration of tools and libraries that interacted with the core of ROS so that users could shift their software stacks to fit their robot and application area. This state of mind is producing results as most of the available libraries and tools have been developed by the community apart from the core structure.
ROS implements a compliant inter-machine/inter-process communication architecture. Different processes are called nodes and communicate to each other through topics. These topics are created and handle by a master and are implemented this way:

1. A first node, called publisher, advertises the master that he will publish data.
2. The master add an open topic on which the node will publish
3. A second node, called subscriber, subscribe to the created topic through the master
4. From this point, every time that the publisher will publish need data, a callback function will be called in the subscriber.

A node can also advertise a service which is more an action that returns a unique response. Note that the node can be developed in Python or C++ without any compatibility issue as ROS provides an abstraction layer with its communication system.

This simple architecture provides end-less possibility and flexibility. Moreover, nodes can be hosted by different machines as long as each node can communicate with the main master. This feature allows fast deployments from simulation to real-life systems with distributed setups.

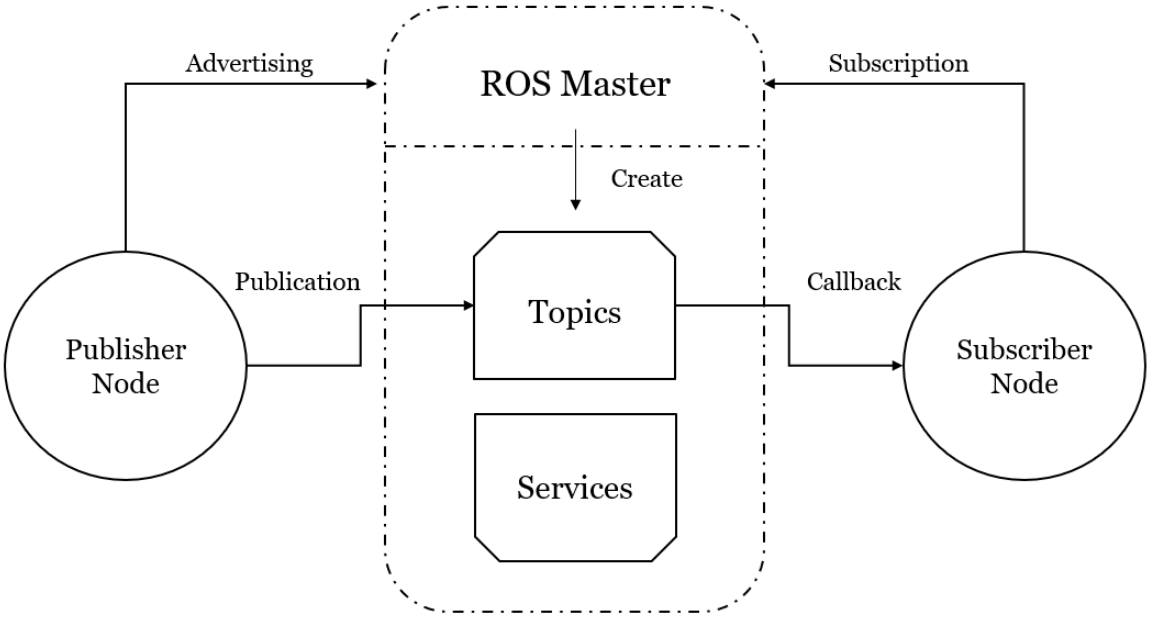


Figure 4.1.1: Fondations of ROS inter-process communication features.

Several versions of ROS exist, but this work uses ROS Melodic, and all the package developed will only be supported by this distribution. Note that this distribution implements Python 2.

4.1.2 GAZEBO simulation tool

In order to perform high fidelity simulations, this work uses Gazebo. Gazebo is a 3D dynamic simulator with the ability to accurately and efficiently simulate robots in complex indoor and outdoor environments. This engine provides physics simulation at a much higher degree of fidelity, a suite of sensors, and interfaces usually used to design and test robot behavior. Each model to be used in the simulation is defined with Unified Robot Description Format (URDF) that traduce kinematic and dynamic of the robot in an XML format. For this project, a generic multi-copter descriptor of the Iris quadcopter is used.

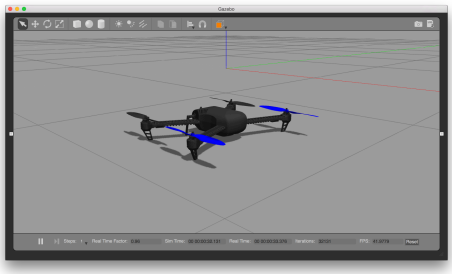


Figure 4.1.2: Iris model in Gazabo simulator

4.1.3 PX4 Autopilot

In order to make a quadcopter or any UAV fly, in simulation or real life, it is mandatory to use flight control software that will interact with available sensors to evaluate the state, control the motors, integrate user input and other vital functionalities. Developing such software can be done to obtain basic features but can be too time-consuming and prone to errors unrelated to this project's main objectives.
In this work, the open-source flight controller PX4 is used to deploy the developed control features. PX4 is hugely spread in industry, research and is proven to be efficient while providing a flexible set of tools. Multiple types of UAVs can be controlled from it, and, among the ecosystem, some ready-to-use hardware is available, facilitating the integration and deployment. Moreover, some stable and robust libraries have been developed to interface this software with ROS.

PX4 for multicopter implement the following control architecture :

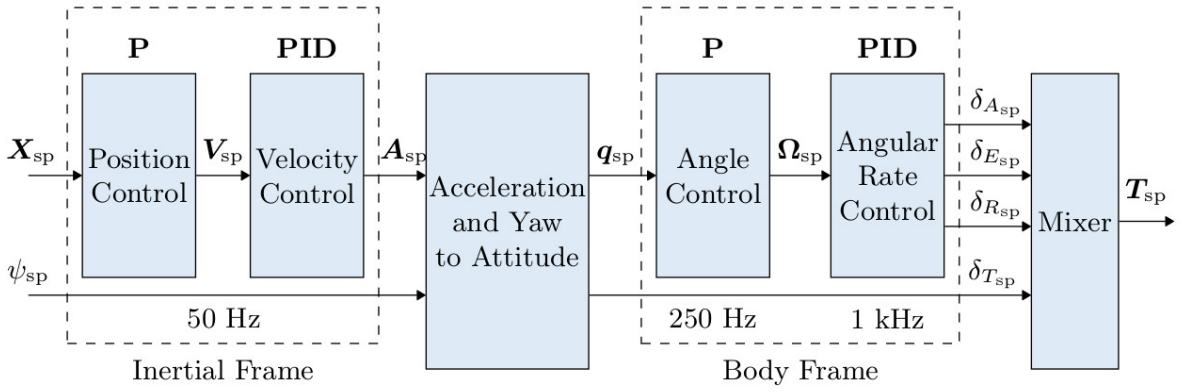


Figure 4.1.3: PX4 on-board cascade controller. $*_{sp}$ variables are references, and depending on the flight mode, the command is branched on one of these variables.

This controller is based on a cascade PID scheme, with each subpart has to be tuned to optimize overall behavior. In figure 4.4.3, one block is responsible for the transformation between inertial and body frame, and the last one transforms thrust and body rate as motors input. Depending on the flight mode, the user can branch his command on $*_{sp}$ variables.

- Position mode, keep a target set point $\boldsymbol{X_{sp}}, \psi_{sp}$
- Off-board mode is used to test the custom MPC controller. MPC return u^* and

can be branch on $\boldsymbol{\Omega_{sp}}, \delta T_{sp}$.

However, in chapter 2, the input system of the model used by the MPC controller is the vector composed of vertical thrust ans 3 axis torques:

$$\boldsymbol{u} = [f_t \quad \tau_x \quad \tau_y \quad \tau_z]^T = [f_t \quad \boldsymbol{\tau^B}]^T \in \mathbb{R}^4 \tag{4.1}$$

As shown in the figure 4.1.3, the torque can not be used as PX4 input. The choice has been made to use the angular rate as the input. Indeed solving the MPC optimization problem at t gives optimal input sequence $\boldsymbol{u^*} = \{\boldsymbol{u_t^*}, \boldsymbol{u_{t+1}^*}, ..., \boldsymbol{u_{t+T-1}}^*\}^T$ with T the horizon. From this optimal control sequence result an optimal state sequence $\boldsymbol{x^*} = [\boldsymbol{x_t^*}, \boldsymbol{x_{t+1}^*}, ..., \boldsymbol{x_{t+T}^*}]^T$. Extracting the optimal state's angular rate at t+1 gives a good approximation of the torque part of the input. The thrust is kept as the one present in u_t^* :

$$u_t^* = [f_t, \tau_{x|t}, \tau_{y|t}, \tau_{z|t}] \in \mathbb{R}^4 \tag{4.2}$$

Gives :

$$\boldsymbol{x_{t+1}^*} = [x_{t+1}^*, y_{t+1}^*, z_{t+1}^*, \dot{x}_{t+1}^*, \dot{y}_{t+1}^*, \dot{z}_{t+1}^*, p_{|t+1}^*, q_{|t+1}^*, r_{|t+1}^*, \phi_{t+1}^*, \theta_{t+1}^*, \psi_{t+1}^*]^T \in \mathbb{R}^{12} \tag{4.3}$$

Then :

$$[\delta_{Tsp} \quad \boldsymbol{\Omega_{sp}}]^T = [f_t, \quad p_{|t+1}^*, \quad q_{|t+1}^*, \quad r_{|t+1}^*]^T \in \mathbb{R}^4 \tag{4.4}$$

See 2.22 The necessity to use angular velocity as system input as implication on the feedback policy presented in the tube approach. Indeed, the linearization and computation of the stabilizing gain have to be done for the reduced dynamic f_r presented in 2.26 such as :

$$u_t^r = v_t^r + K_{LQR|t}(x_t^r - z_t^r) \tag{4.5}$$

with:

$$K_{LQR|t} \in \mathbb{R}^{4*9} \quad x_t^r, z_t^r \in \mathbb{R}^9 \tag{4.6}$$

such as $x_{r|k}, z_{r|k}$ *taking the form:*

$$\begin{aligned}
\boldsymbol{x_t^r} &= [x_{t+1}, y_{t+1}, z_{t+1}, \dot{x}_{t+1}, \dot{y}_{t+1}, \dot{z}_{t+1}, \phi_{t+1}, \theta_{t+1}, \psi_{t+1}]^T \in \mathbb{R}^9 \\
\boldsymbol{z_t^r} &= [x^*_{t+1}, y^*_{t+1}, z^*_{t+1}, \dot{x}^*_{t+1}, \dot{y}^*_{t+1}, \dot{z}^*_{t+1}, \phi^*_{t+1}, \theta^*_{t+1}, \psi^*_{t+1}]^T \in \mathbb{R}^9 \\
v_k^r &= [f_t, \quad \omega^*_{x|t+1}, \quad \omega^*_{y|t+1}, \quad \omega^*_{z|t+1}]^T \in \mathbb{R}^4 \\
u_k^r &= [\delta_{Tsp} \quad \boldsymbol{\Omega_{sp}}]^T \in \mathbb{R}^4
\end{aligned} \tag{4.7}$$

$\boldsymbol{x_t^r}$ being the estimated state of the real system, $\boldsymbol{z_t^r}$ the nominal state, v_k^r the nominal optimal input and u_k^r the system input to PX4.
Concerning the state estimation, PX4 Estimation and Control Library use an Extended Kalman Filter, a non-linear extension of Kalman Filter presented by R. Kalman 1960, Kálmán and Bucy 1961. This tool fuses inputs from different sensors to compute a precise state estimation from noisy signals. The fusion uses a combination of IMU, Gyroscope, Magnetometer, Barometer, and external references (vision-based for example).

Remark 1. PX4 is based on quaternion attitude description. Thus transformation tools need to be used to obtain Euler angles (ROS provides these features but can easily be implemented).

4.1.4 Overview

The simulation setup is summarize in the figure 4.1.4:

Gazebo acts as a simulation engine; the drone model lives in the simulated world and takes motor inputs from PX4 autopilot while returning simulated sensors values. PX4 usually communicates with a protocol named MAVLink and needs an extra abstraction level to be integrated into ROS. The plug-in used is named MAVRos, it publishes/subscribes to all the necessary topics on ROS Master and make the conversion between ROS messages and MAVLink messages. For testing purpose, the autopilot is switched to the off-board mode and the extra custom node implementing MPC publish the target thrust and angular rate for the simulated system.

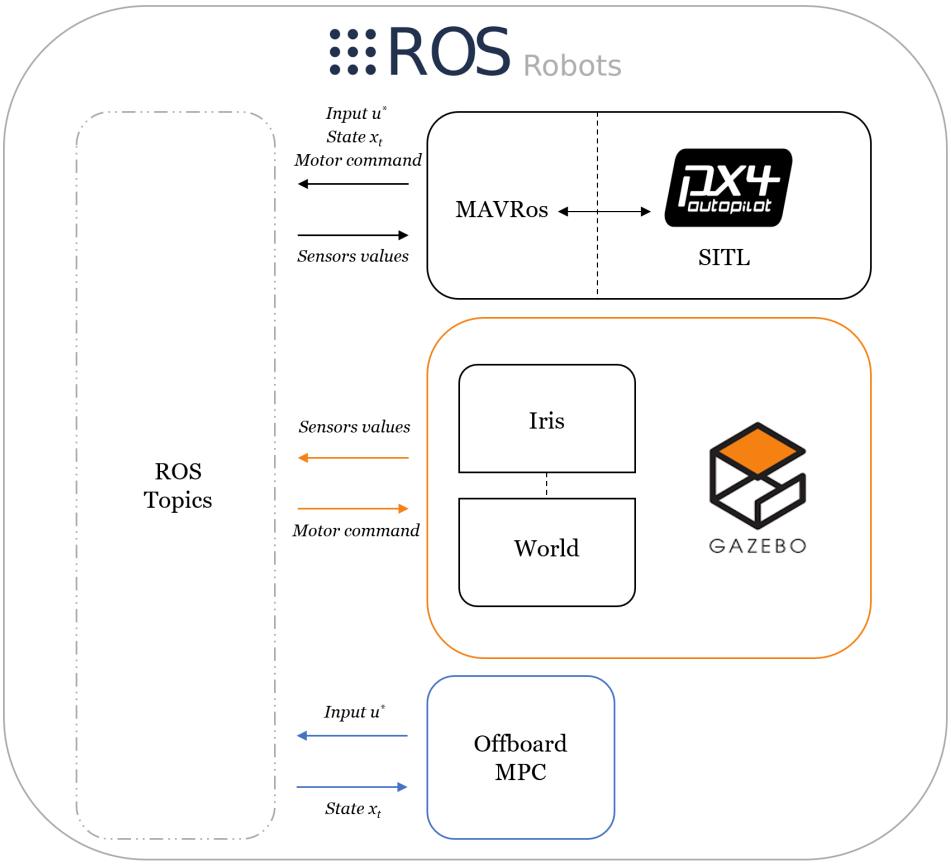


Figure 4.1.4: Overview of the simulation setup. Based on ROS communication features, three elements are display. Gazebo simulation engine, PX4 flight control software and its MAVRos interface and the custom offboard MPC.

4.2 Solvers

MPC optimization problems require powerful solvers. The solver's computational speed is a significant factor for simulation and hardware validation as the target control frequency is set at 50Hz. Two frameworks have been tested to research the best performances, the first framework is named CasADI, and the second one is acados.

4.2.1 CasADI

acados is an open-source software tool for numerical optimization, started as an academic project by Joel Andersson and Joris Gillis at KU Leuven. Mainly focus on optimal control, CasADI is now a widely spread tool that defines a whole framework of variable types, function definitions, and solver implementations. CasADI is available in C++, Python and MATLAB/Octave. The authors present the different languages

As presented in the workflow, the first step is based on Python validation, CasADI Python API is the best documented and more stable one. It runs on Python 2 and 3.

4.2.2 acados

acados is a software package for the efficient solution of optimal control and estimation problems. Build as the successor of ACADO software developed at KU Leuven it provides a collection of computationally efficient building blocks tailored to optimal control and estimation problems. acados is built on top of the acados formalism and provides cross-compatibility. User interfaces exist for Python and MATLAB/Octave and can describe optimal control problems to generate self-contained C code that can be readily deployed on embedded platforms. This last feature is critical in the real-time application framework as the generated C code should be better optimized for fast computation.

4.2.3 Comparison

The two frameworks can be compare with two main criterions :

- Computation efficiency.
- Deployment convenience.

Being driven by the 50 Hz control frequency objective, computational efficiency is crucial in the solver's choice. As it can be expected, acados present a better performance of about one order of magnitude. Indeed, this framework generates a C-optimized SQP-RTI solver from the problem defined in Python by the user. This solver can then be used as an external library to obtain :

These performances are obtained for the following NMPC problem:

Parameter	Value
Q	[1, 1, 1, 1, 1, 1, 1, 1, 1 ,1 ,1]
R	[1,1,1,1]
P	P_{LQR}
Horizon	15
solver type	QP solver

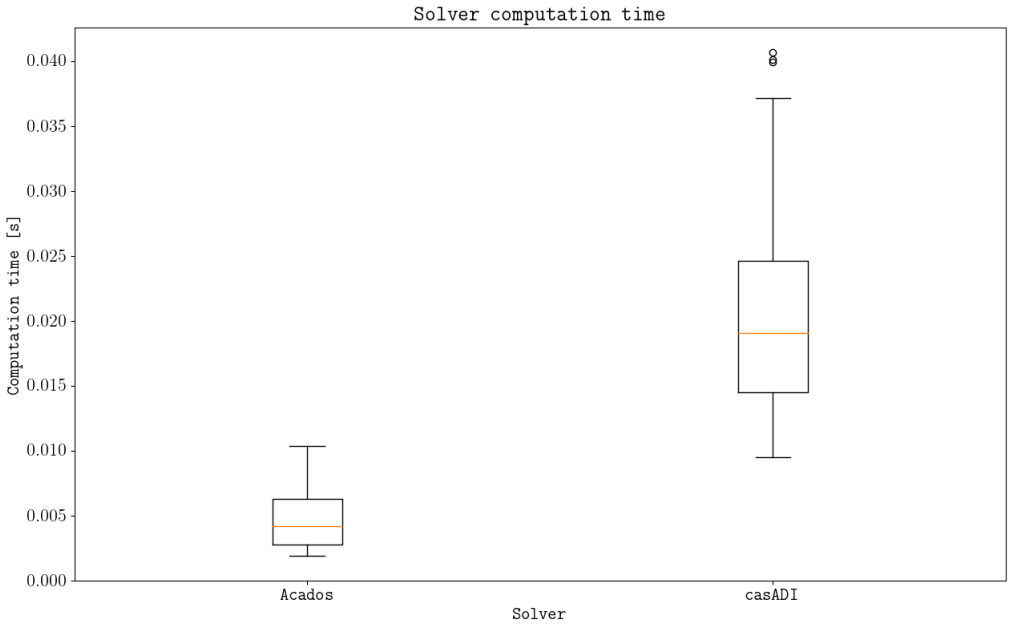


Figure 4.2.1: Performances comparison of acados and acados solver on the same NMPC problem.

The difference between the two solvers can vary depending on the problems' nature, but this gives a good idea about raw performances and why acados have been selected for the real-time controller.
However, CasADI still presents a significant advantage in deployment convenience as it can run on Python 2. Indeed, this project simulation and hardware experiments are based on ROS Melodic running this Python version. For this reason, CasADI will still be used for tasks with no critical time constraints as trajectory generation. Concerning the deployment of acados, as it only runs on Python 3, the associate ROS node has been developed in C++ using a pre-generated c solver.

4.3 Trajectory generation.

Trajectory generation is one of the bases of the experimental setup. The objective is to provide a finite set of points, a maximum traveling speed, and a discretization step to obtain a trajectory exploiting the quadrotors' full dynamic capabilities. In this project, the problem is separated into two parts. A first polynomial trajectory is computed based on polynomial segments defined between given waypoints. The transition between each segments is done ensuring continuity of the position, velocity,

acceleration, and jerk. The result is a discrete trajectory with a sampling period δ that will serve as a reference for a long horizon NMPC (where horizon T equal the total length of the polynomial reference). Indeed, the second step is based on a non-linear MPC controller using the generated reference to obtain a full final trajectory exploiting the quadrotor dynamic. The usage of NMPC also allows integrating constraints in trajectory generation.

The trajectory generation has been developed in an independent custom Python

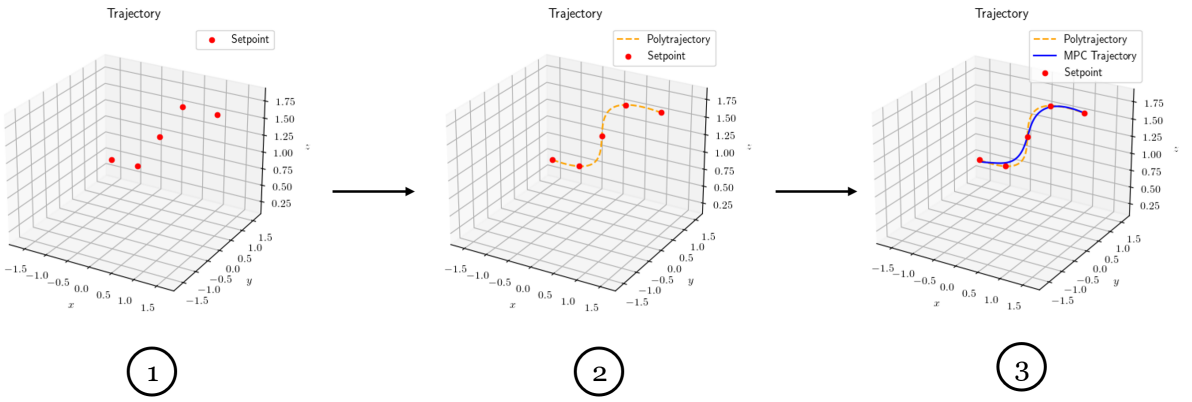


Figure 4.3.1: 1. Python validation step, 2. High fidelity simulation, 3.Hardware validation

package that can reuse in other types of work where a trajectory based on the system's dynamic is required.

4.3.1 Polynomial trajectory

The method describes here is inspired by the work of **traj_poly_2007** and aims to generate a polynomial trajectory to be followed by a quadrotor. A sequence of N_{wp} waypoints in 3D space $\{\boldsymbol{x_{ref|k}}\}_{k=0}^{N_{wp}}$ is used to generate a minimum-snap polynomial path passing through each of those waypoints. Between each waypoint, the trajectory segment is composed of independent polynomials, $P(t)$ with $t \in [0, T]$ and T the estimate travel time. A cost function $J(T)$ is define from this polynomial and its N_d first derivative :

$$J(T) = \int_0^T c_0 P(t)^2 + c_1 P^{'}(t)^2 + ... + c_{N_d} P^{N_d}(t)^2 dt = \boldsymbol{p}^T Q(T) \boldsymbol{p} \tag{4.8}$$

In this expression, $\boldsymbol{p}$ is a vector of the N_d coefficients of a single polynomial. Playing with the associate value modify the weight of the derivative in the cost function. The construction of the Hessian matrix Q is omitted for brevity,but follows from

differentiation of the square of the polynomial with respect to each of its coefficients. M polynomial segments can be jointly optimized by concatenating their cost matrices in a block-diagonal fashion to obtain a fully optimized trajectory :

$$J_{total} = \begin{bmatrix} \boldsymbol{p}_1 \\ . \\ . \\ \boldsymbol{p}_M \end{bmatrix}^T \begin{bmatrix} Q_1(T_1) & & & \\ & . & & \\ & & . & \\ & & & Q_M(T_M) \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_1 \\ . \\ . \\ \boldsymbol{p}_M \end{bmatrix} \tag{4.9}$$

From this, constraints needs to be added to assign specific values of velocity, acceleration, jerk or snap at each waypoints. Exact derivation can be found in **traj_poly_2007**. The result is a quadratic optimization of the coefficients of each polynomial segment. This is solved using CasADI and results in optimized polynomial trajectory that can be sampled with a period δ. This sampled trajectory $\{\boldsymbol{r_t}\}_{t=0}^{T_{total}}$ passing by all the defined waypoints $\{\boldsymbol{x_{ref|k}}\}_{k=0}^{N_{wp}}$ in a time $T_{total} = T_1 + T_2 + ... + T_M$ is used as reference for the NMPC problem .

4.3.2 NMPC trajectory

Taking the polynomial trajectory $\{\boldsymbol{r_t}\}_{t=0}^{T_{total}}$ generated above, a NMPC problem is used ensure its feasibility for the target system i.e. a quadrotor. Moreover, this additional step allows to exploit the full potential if the system dynamic, and playing with the weights in the stage cost could be used to navigate between aggressive and soft behavior. The NMPC horizon is taken as the full reference trajectory length T_{total}.

$$\min_{\boldsymbol{u}_. \in \mathbb{U}} J(\boldsymbol{x}_., \boldsymbol{u}_.) \tag{4.10}$$

with :

$$J(\boldsymbol{x}_., \boldsymbol{u}_.) = \sum_{k=0}^{k=T_{total}} F(\boldsymbol{x_k} - \boldsymbol{r_k}, \boldsymbol{u_k}) + E(\boldsymbol{x_{T_{total}}} - \boldsymbol{r_{T_{total}}}) \tag{4.11}$$

$$\forall k = 0, ..., T_{total} \tag{4.12}$$

subject to:

$$\begin{aligned} \boldsymbol{x_{k+1}} &= f(\boldsymbol{x_k}, \boldsymbol{u_k}), \\ \boldsymbol{u_k} &\in \mathbb{U}, \quad \forall k \in [0, T_{total}], \\ \boldsymbol{x_k} &\in \mathbb{X}, \quad \forall k \in [0, T_{total}], \\ \boldsymbol{x_{T_{total}}} &\in \mathbb{X}_f, \\ \boldsymbol{x_0} &= \boldsymbol{r_0} \end{aligned} \tag{4.13}$$

With, f target system dynamic and $\mathbb{X}_f$ taken as $\{r_{T_{total}}\}$. This last equality constraint can be difficult to satisfy in practice. To avoid to run into infeasible, slack variables are added according to acados documentation. ((*acados — acados documentation* 2021) NMPC result in optimal state $\boldsymbol{x_t^*}$ and $\boldsymbol{u_t^*}$ optimal input with $t \in [0, T_{total}]$. $\boldsymbol{x_t^*}$ constitute an optimal discrete trajectory that can be serve to the online controller of the real system rewrote as $\boldsymbol{r^*} = \{r_0^*, r_1^*, ..., r_{T_{total}}^*\}$.

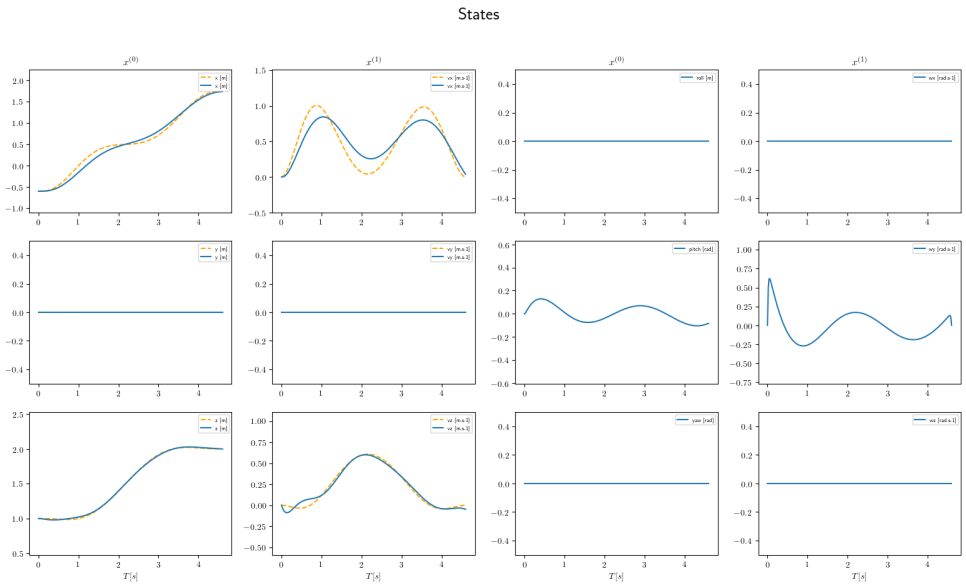


Figure 4.3.2: Example of optimal trajectory for the quadrotor dynamic described in chapter 2. In yellow the polynomial reference and in blue the NMPC output trajectory x_t^*

4.4 Deep-learning for tube estimation.

As described in chapter 3, a key point is to estimate the tube using deep-learning quantile regression. The neural network is developed using Python 3 jupyter notebook and is based on Pytorch framework. This framework is widely spread for this kind of development and allows to quickly deploy complex architecture while being able to control and modify each subpart.
No GPU was available at SML for training. Thus Google Colab solution have been used.

4.5 Experimental setup

4.5.1 Hardware quadcopter

The experiment support on which the algorithm has been tested is presented in the figure 4.5.1. This quadcopter is based on an srd370 frame with an embedded computer mRo Pixracer. The board is part of the PX4 ecosystem and implement all the features described in the last section, interfacing a set of sensors fused by an extended Kalman filter, implementing a low-level control loop, and the communication tools. Built-in wifi allows to communicate with external devices, and a FrSky receptor is used to get user input from the radio command. The main hardware characteristics of the board are described as :

- STM32F4 MCU 180MHz Cortex M4 with FPU and 256Kb SRAM
- ICM-20608 Accel / Gyro (4 KHz) / MPU9250 Accel / Gyro / Mag (4 KHz)
- HMC5983 magnetometer with temperature compensation

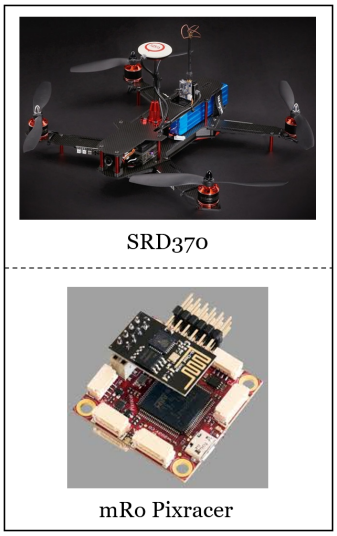


Figure 4.5.1: Quadrotor srd370 frame and associated mRo Pixracer

The full setup with battery pack is measure to weight 10.090Kg.

4.5.2 Motion capture

Figure 4.5.2: Oqus 400 camera for Qualisys motion capture

A motion capture setup is used to get the quadcopter state with high accuracy. The setup is based on Qualisys products composed of 12 cameras (10 Oqus 400 and 2 Oqus 310+) made for cinema motion capture. They provide high quality tracking at the ground level. However, when going up, the performances drops limiting the possible flying window. Due to this limitation and to the lab disposition the flying window is about $2m * 2m * 2m$.

The position obtains with the setup is fed into PX4 kalman filter through a dedicate ROS topic publish by a compatible node running on the Qualisys computer.

4.5.3 Overview

The hardware setup is summarize in th below figure:

In this setup, the basis of the simulation tools is kept. ROS is still used for its inter-machine communications feature that allows running ROS master and the off-board controller on the main computer while controlling the srd370 drones and gathering inputs from the qualisys motion capture system (running on another machine). PX4 is running on the PixRacer on-board computer, and MAVROS plays the role of interface with ROS master. All these components are linked through the same network.

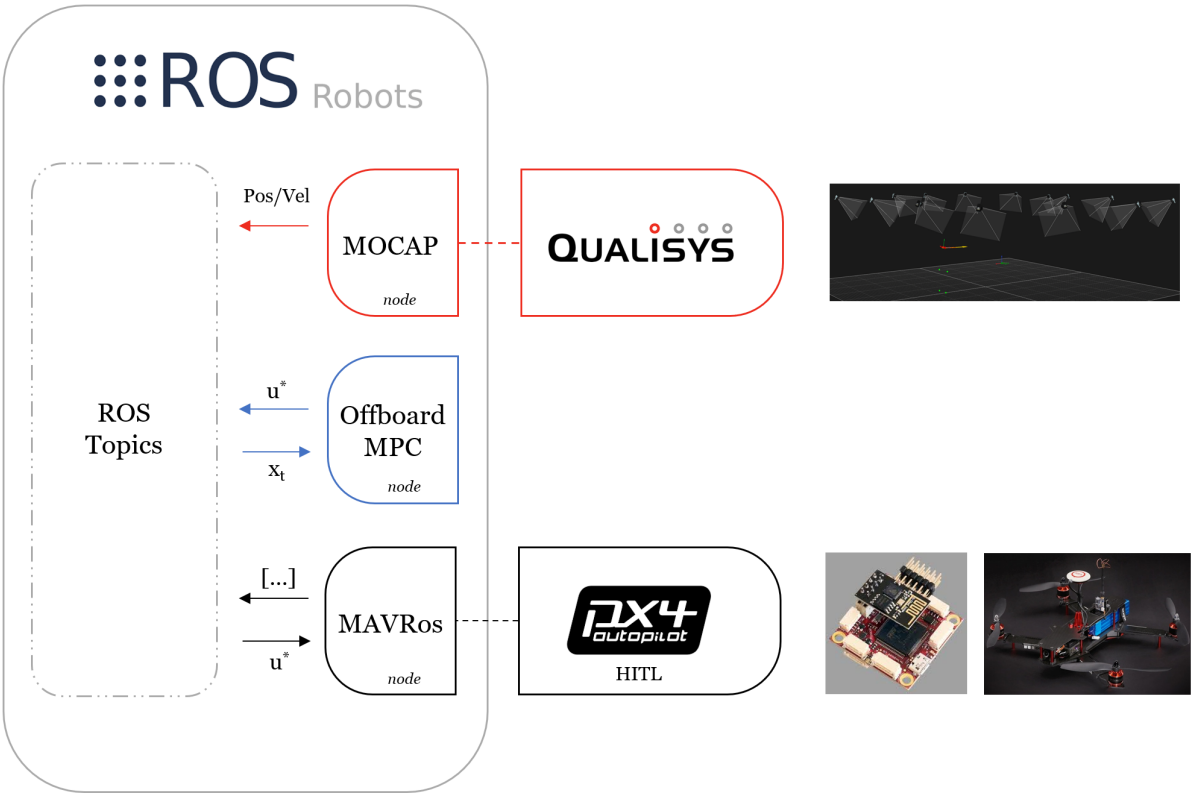


Figure 4.5.3: Overview of the simulation setup. Based on ROS communication features, three elements are display. Qualisys motion capture system, PX4 flight control software and its MAVRos interface runing on emdedded pixRacer and the custom offboard MPC.

Chapter 5

Result

5.1 Python fundamental validation

First experiments are performed to validate basic features of NMPC implementation using acados solver, Python, and quadrotors model. Further developments take roots in these initial experiments, acados problem formulation is tested, and different NMPC parameters (cost weight, constraints, terminal cost, terminal constraints, and horizon). System sampling period is taken as $\delta = 0.02$ s and control frequency target as $50Hz$. However, this target frequency is not a key factor as the priority is on NMPC setup exploration.

5.1.1 Setpoint

Setpoint problem is the first step performed in this work. The NMPC formulation in its stable version describes in 3.9 is use with:

$$\begin{aligned} F(\boldsymbol{x_{k|t}} - \boldsymbol{r}, \boldsymbol{u_{k|t}}) &= (\boldsymbol{x_{k|t}} - \boldsymbol{r})^T Q(\boldsymbol{x_{k|t}} - \boldsymbol{r}) + \boldsymbol{u_{k|t}}^T R \boldsymbol{u_{k|t}}, \\ E(\boldsymbol{x_{T|t}} - \boldsymbol{r}, \boldsymbol{u_{k|t}}) &= (\boldsymbol{x_{k|t}} - \boldsymbol{r})^T P(\boldsymbol{x_{k|t}} - \boldsymbol{r}) \end{aligned} \tag{5.1}$$

$P \in \mathbb{R}^{12\times12}$ is the LQR terminal cost derived from the Algebraic Riccati Equation, $Q \in \mathbb{R}^{12\times12}$ and $R \in \mathbb{R}^{4\times4}$ chosen for satisfying performances and $\boldsymbol{r} \in \mathbb{R}^{12}$ the target point. Details can be found in :

Parameter	Value
Q	diag([5, 5, 1.5, 0.01, 0.01, 0.01, 0.01, 0.01, 0.1, 0.2, 0.2, 0.2])
P	diag([0.5, 1, 1, 1])
x_{max}	[1, 1, 1.5, 1, 1, 1, 1, 1, 1, 0.4363, 0.4363, 3.1415]
x_{min}	[-1, -1, -1.5, -1, -1, -1, -1, -1, -1, -0.4363, -0.4363, -3.1415]
u_{max}	[30, 1.5, 1.5, 1.5]
u_{min}	[0, -1.5, -1.5, -1.5]
x_{term}	[0.2, 0.2, 0.4, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1745, 0.1745, 0.1745]
T	15

The choice of x_{term} such as:

$$\mathbb{X}_f = \{x \in \mathbb{R}^{n_x} \mid -x_{term} \leq x_{k|t} - r \leq x_{term}, \quad \forall k \in [0, T] \quad \forall k \in \mathbb{N}^+\} \tag{5.2}$$

is quite large but allows a large enough set to ensure that the problem remains feasible for a small horizon. The reference remains closer from the system for the trajectory following, allowing to tighten this terminal set.

Figure 5.1.6 shows the full state of the quadrotor going from $x_0 = [0, 0, 0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ to $r = [0.4, 0.4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ and 5.3.6 the error evolution of the position. In figure 5.1.6, utilization of slack variables is highlighted.

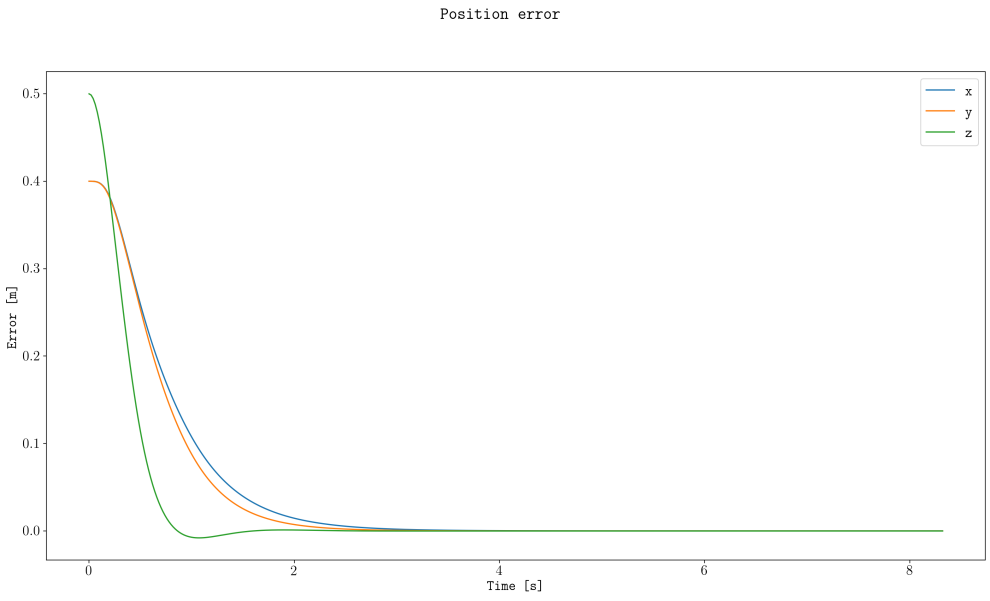


Figure 5.1.1: Position error plot for setpoint control problem solved with NMPC

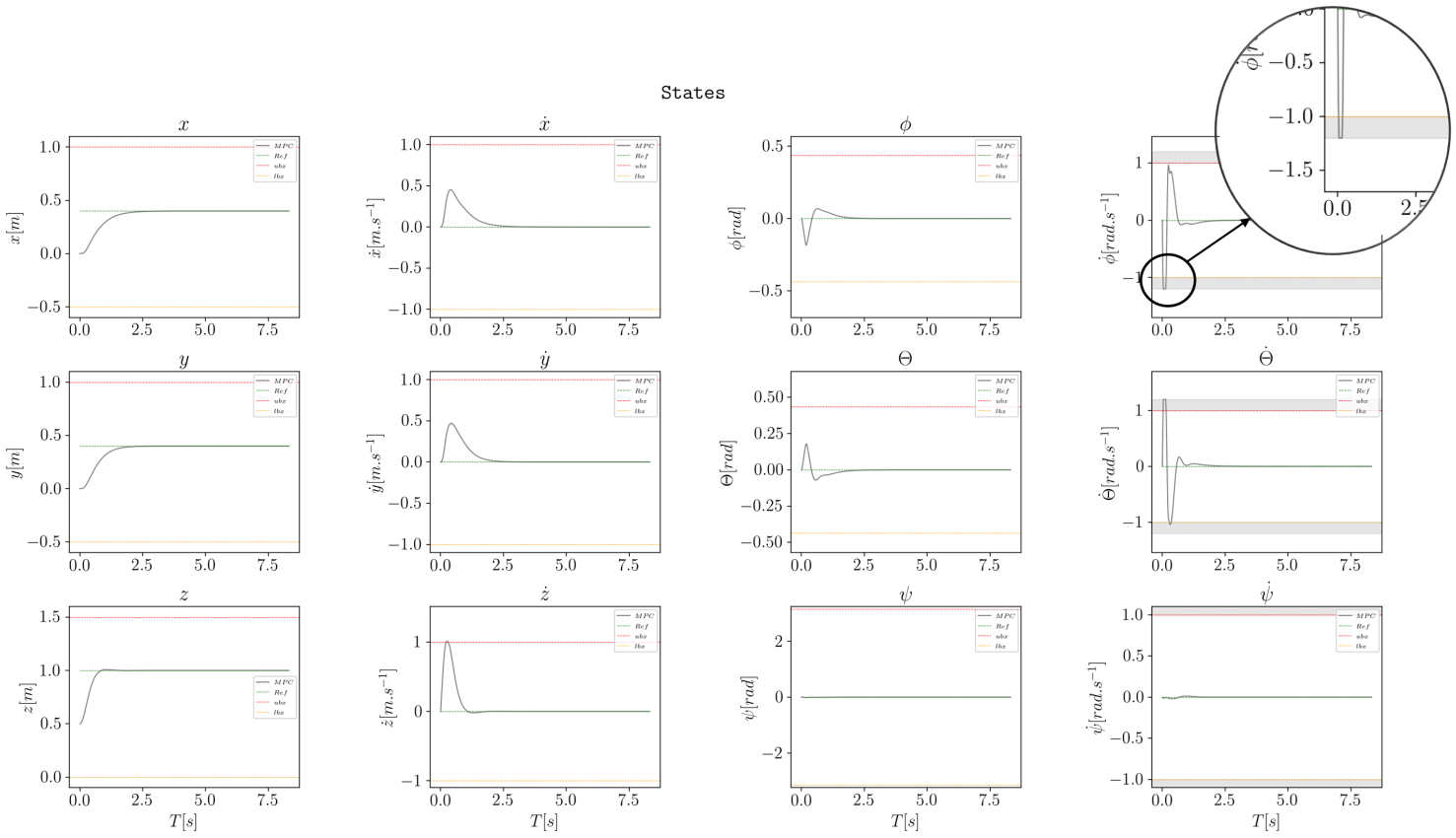


Figure 5.1.2: Evolution of simulated quadrotor states. Up-right corner a zoom showing the effect of slack variables.

5.1.2 Trajectory

Trajectory following is the core problem for this work. Based on the generated trajectory $\boldsymbol{r^*} = \{r_0^*, r_1^*, ..., r_{T_{total}}^*\}$ sampled with a period δ the cost functions are rewrote as:

$$
\begin{aligned}
&F(\boldsymbol{x_{k|t}} - \boldsymbol{r^*_{k+t}}, \boldsymbol{u_{k|t}}) = \\
&\quad (\boldsymbol{x_{k|t}} - \boldsymbol{r^*_{k+t}})^T Q (\boldsymbol{x_{k|t}} - \boldsymbol{r^*_{k+t}}) \\
&\quad + \boldsymbol{u_{k|t}}^T R \boldsymbol{u_{k|t}}, \\
&E(\boldsymbol{x_{T|t}} - \boldsymbol{r^*_{T+t}}, \boldsymbol{u_{k|t}}) = \\
&\quad (\boldsymbol{x_{k|t}} - \boldsymbol{r^*_{T+t}})^T P (\boldsymbol{x_{k|t}} - \boldsymbol{r^*_{T+t}})
\end{aligned}
\tag{5.3}
$$

$P \in \mathbb{R}^{12\times4}$ is the LQR terminal cost compute from Riccati equation, $Q \in \mathbb{R}^{12\times12}$ and $R \in \mathbb{R}^{4\times4}$ chosen for satisfying performances and $r_{k+t} \in \mathbb{R}^{12}$ the time varying target point. Details can be found in :

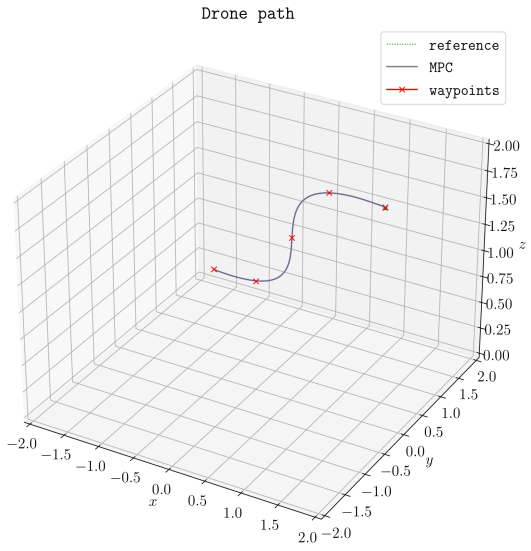


Figure 5.1.3: simple trajectory following problem using NMPC on quarotors model.

Parameter	Value
Q	diag([5, 5, 1.5, 0.01, 0.01, 0.01, 0.01, 0.01, 0.1, 0.2, 0.2, 0.2])
P	diag([0.5, 1, 1, 1])
$\boldsymbol{x_{max}}$	[1, 1, 1.5, 1, 1, 1, 1, 1, 1, 0.4363, 0.4363, 3.1415]
$\boldsymbol{x_{min}}$	[-1, -0.5, 0.0, -1, -1, -1, -1, -1, -1, -0.4363, -0.4363, -3.1415]
$\boldsymbol{u_{max}}$	[30, 1.5, 1.5, 1.5]
$\boldsymbol{u_{min}}$	[0, -1.5, -1.5, -1.5]
$\boldsymbol{x_{term}}$	[0.05, 0.05, 0.05, 0.01, 0.01, 0.01, 0.2, 0.2, 0.2, 0.1745, 0.1745, 0.1745]
T	15

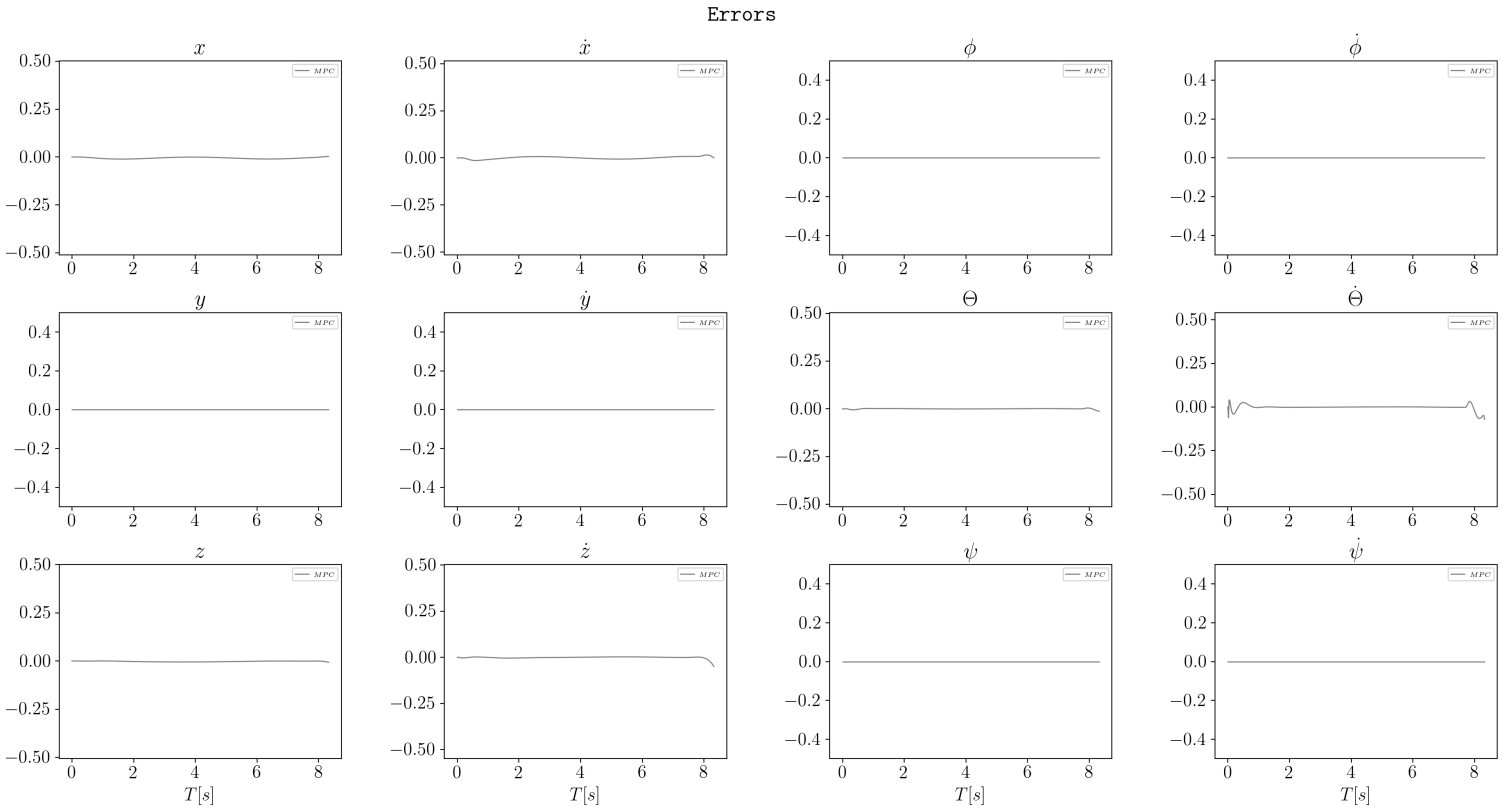


Figure 5.1.4: Error evolution on the full quadrotor state. The system is shown following a trajectory using simple NMPC formulation.

5.1.3 Inherent stability

Inherent
stability is interesting to observe. Indeed, even under external noise, the stable version of NMPC can still produce good performances. In order to simulate it, we change the dynamic such as :

$$\boldsymbol{x_{t+1}} = f(\boldsymbol{x_t}, \boldsymbol{u_t}) + \boldsymbol{w_t} \tag{5.4}$$

and let w_t follow a state dimension dependant uniform probability centered on 0:

$$W \sim U(0, w^j) \quad j \in [0, 12] \tag{5.5}$$

with w^j noise range associated with the j^{th} state dimension and :

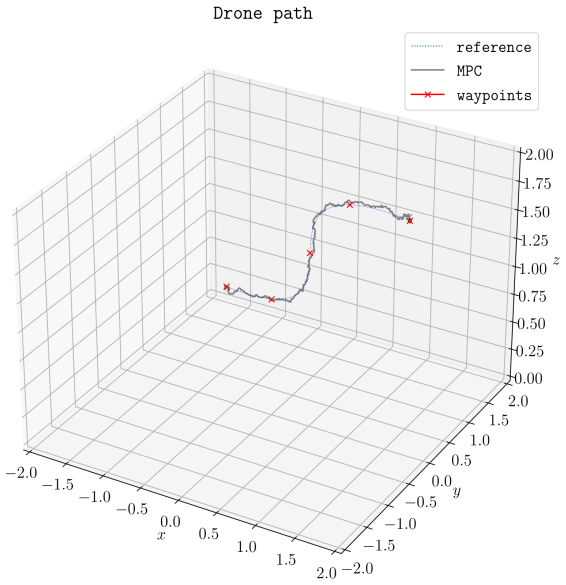


Figure 5.1.5: Additive noise on simple trajectory following problem using NMPC on quarotors model.

$$\boldsymbol{w} = [0.005, 0.005, 0.005, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.005, 0.005, 0.005] \tag{5.6}$$

The noise can seem small but considering that sampling time of $\delta = 0.02$, it remains a good way to evaluate the robustness of NMPC.
Terminal weight gain is kept as the solution of the riccati equation for the LQR problem. For the rest, the same formulation as for simple trajectory following problem is used, and details can be found in :

Parameter	Value
Q	diag([5, 5, 1.5, 0.01, 0.01, 0.01, 0.01, 0.01, 0.1, 0.2, 0.2, 0.2])
P	diag([0.5, 1, 1, 1])
x_{max}	[1, 1, 1.5, 1, 1, 1, 1, 1, 1, 0.4363, 0.4363, 3.1415]
x_{min}	[-1, -0.5, 0.0, -1, -1, -1, -1, -1, -1, -0.4363, -0.4363, -3.1415]
u_{max}	[30, 1.5, 1.5, 1.5]
u_{min}	[0, -1.5, -1.5, -1.5]
x_{term}	[0.05, 0.05, 0.05, 0.01, 0.01, 0.01, 0.2, 0.2, 0.2, 0.1745, 0.1745, 0.1745]
T	15

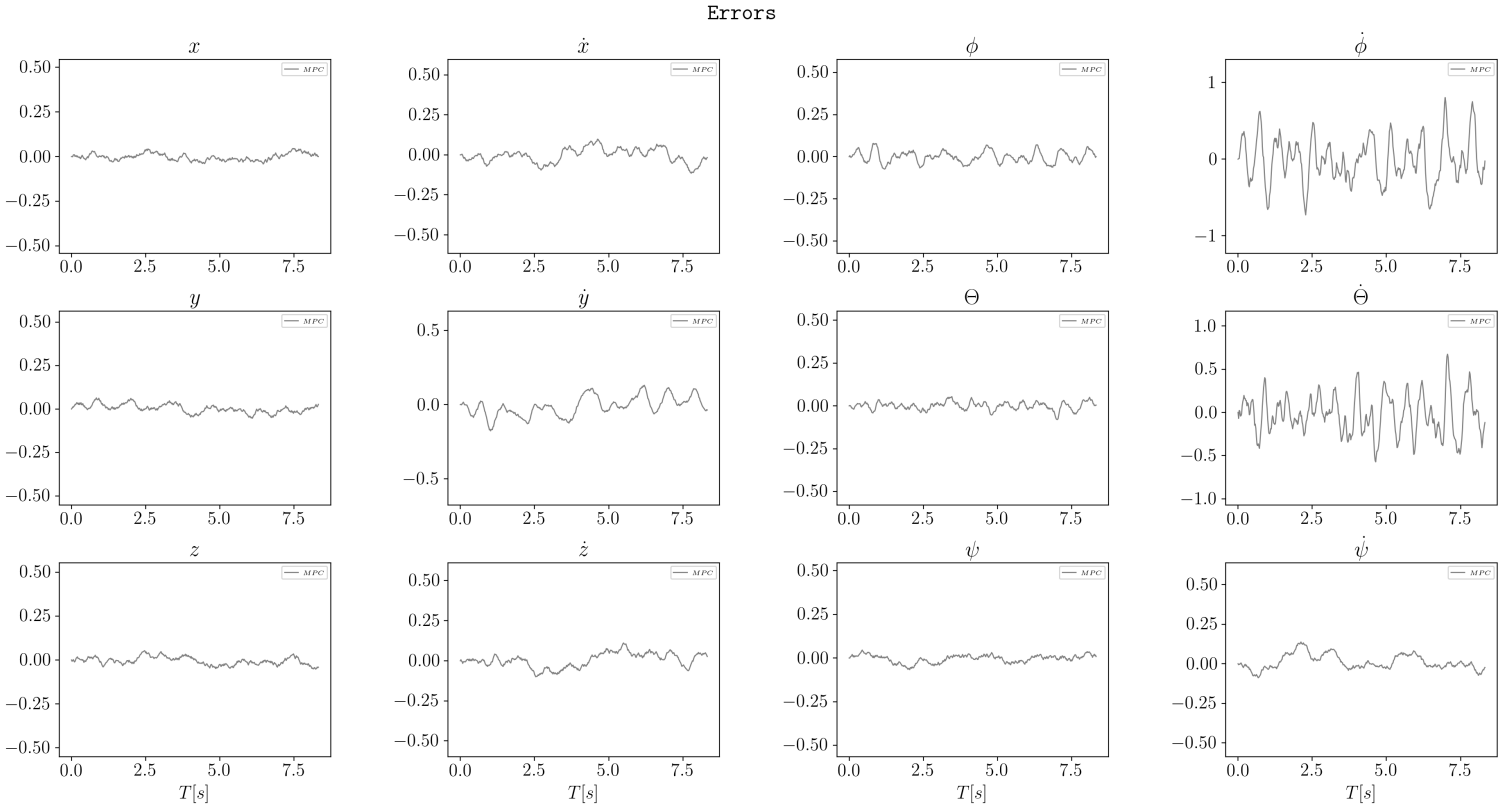


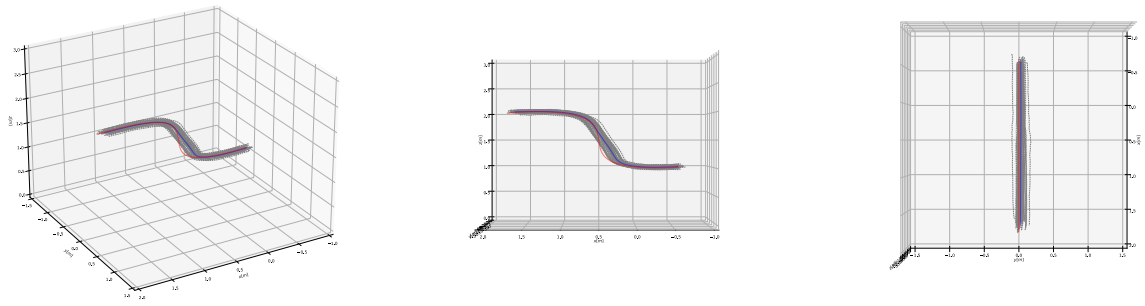
Figure 5.1.6: Error evolution on the full quadrotor state. The system under the action of additional noise is shown following a trajectory using simple NMPC formulation.

5.2 Experimental validation

After the first python tests, the objective was to validate the experimental setup by implementing the same NMPC formulation on simulated quadrotors and real-life systems. This step is crucial and challenging. Indeed, as described in chapter 2, quadrotors present fast dynamic, which implies high control frequency but also large state dimension, which leads to increasing solving time for MPC.
This section demonstrates that these challenges have been overcome with success. All the setups have been described in the last chapter, and the objective here is to show the results. All the tests are performed using the described trajectory generation, which produces a step-like function. The shape has been chosen on purpose and according to the limited flying area available. It allows observing the quadrotor reaction in a situation of sharp variation.

5.2.1 Gazebo simulation

The first natural step is to transfer the basic python tests to high-fidelity simulations. The results can be seen in figure 5.2.1, and it is immediately possible to observe the large error in the transition phase as the NMPC struggles to follow the sharp reference trajectory. Fine-tuning could increase the position error cost, but it could also lead to instability or increasing computation time.



(a) Isometric (b) Side (c) Top

Figure 5.2.1: Different views of 50 Gazebo simulated quadrotor travel along reference trajectory. Control based on NMPC

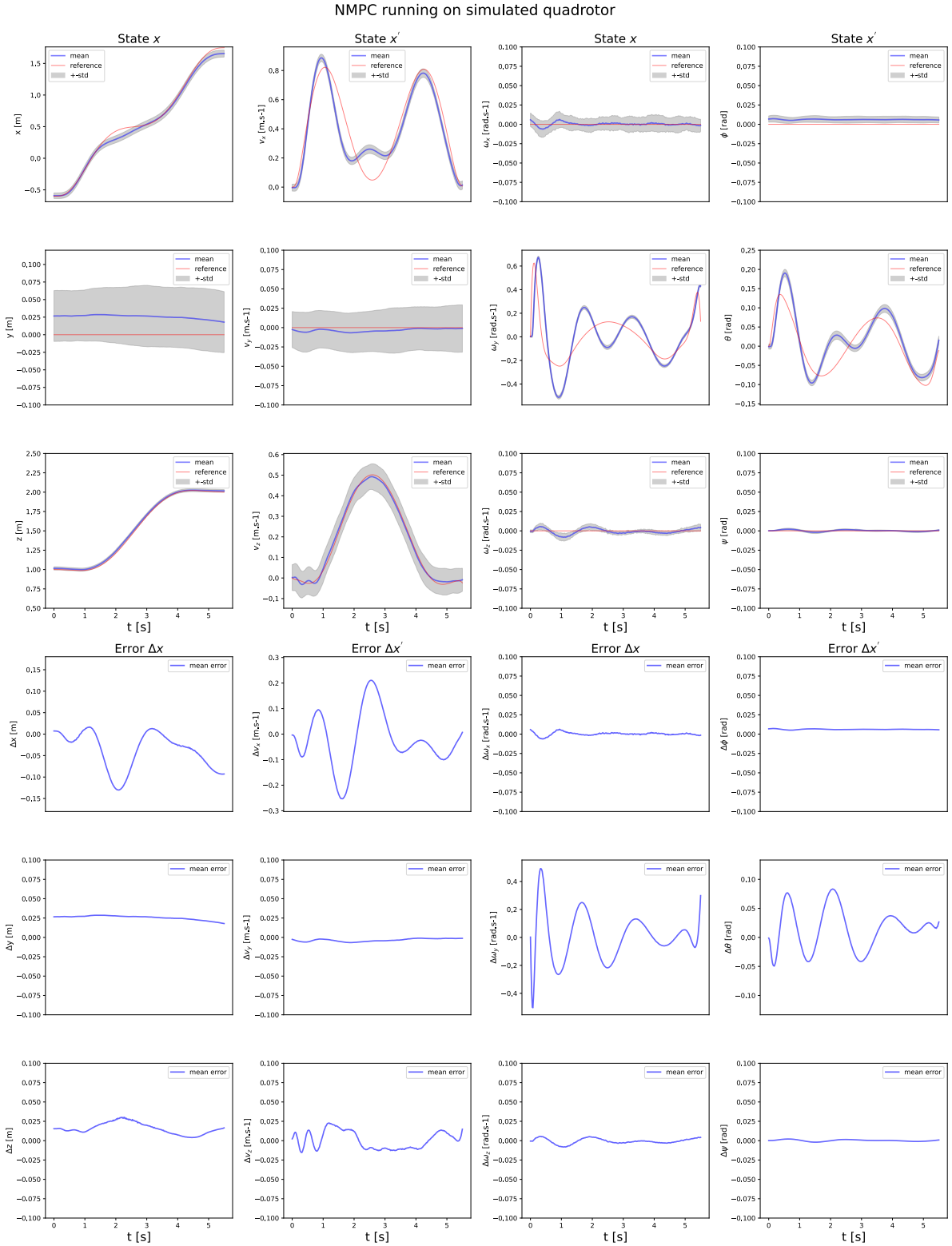
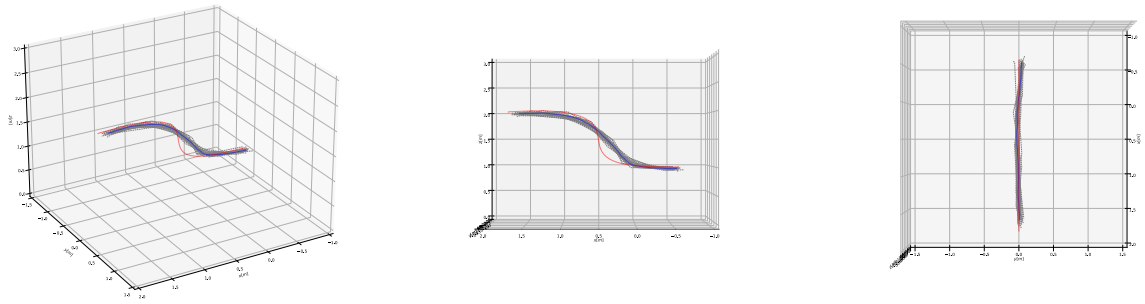


Figure 5.2.2: Summary of 50 Gazebo simulated quadrotor travel along reference trajectory. Control based on NMPC. The upper part show the state evolution and the lower part the average error between the system and the reference.

5.2.2 Hardware

After a successful implementation in ROS simulation, the setup has been transferred to a real-life system. Even if a standard deviation is much more present and the transition phase is worse than in simulation, these results are expected. Indeed, the hardware implementation brings an additional complexity challenging to reproduce even with a high fidelity simulation engine. However, both experiments go in the same way and show that the developed experimental setup can provide results.



(a) Isometric (b) Side (c) Top

Figure 5.2.3: Different views of 50 quadrotor travel along reference trajectory. Control based on NMPC.

Both these experiments shows the inherent robustness of NMPC technique.

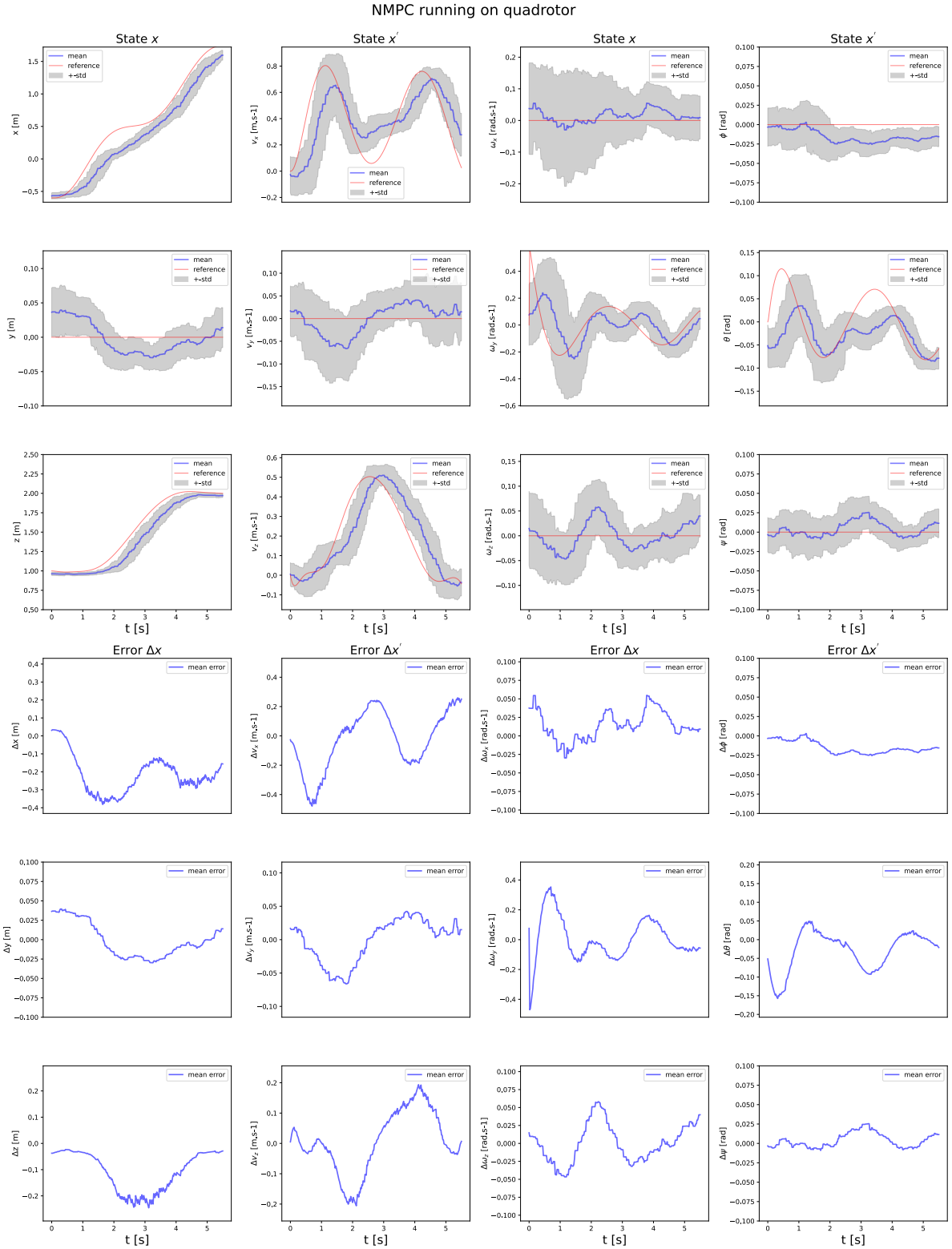


Figure 5.2.4: Summary of 50 quadrotor travel along reference trajectory. Control based on NMPC. The upper part show the state evolution and the lower part the average error between the system and the reference.

5.3 Tube estimation

The developed approach in this work is based on the tube estimation for a given controlled system under a chosen feedback policy. To perform such estimation, deep learning has been chosen for its faculty not to assume noise distribution and for the possibility to compute quantile within which the system is guaranteed to stay with a probability α.
As for any deep learning problem, data are needed. The experimental setup has been used in simulation and hardware to generate multiple traveling around the reference trajectory, creating a representative dataset of the tube to be learned.
It is important to remember that the tube is extremely dependent on the feedback policy. Thus, the dataset is generated using a middle ground controller (see D. Q. Mayne, E. C. Kerrigan, et al. 2011). This controller has the same formulation as the final tube MPC; an NMPC controller is implemented on the nominal system giving an optimal input that is modified using the feedback policy based on time-varying $K_{LQR|t}$ and the difference between the nominal system and the real one. However, the adapted constraints on the nominal system are omitted as the tube is not estimated yet(see chapter 3). Thus, it can not be considered as robust.

5.3.1 Data-set

In order to test our approach in challenging conditions, a new trajectory is generated. This trajectory stays in a 2D plan but flies close to the upper bound of the y constraint. To put some context, it is possible to imaging a real trajectory for a delivery drone flying close to a building in urban environment.
As a first step, this generation is implemented in a Python framework to keep it simple and validate the approach.
5.3.1 and 5.3.2 show this trajectory and the 50 travel of a simulated quadrotor around it. It is now easy to visualize the tube generated by the noise's different realization and how, without tighten constraint, the quadrotor violates the y bound.
Note: The simulated system takes the form:

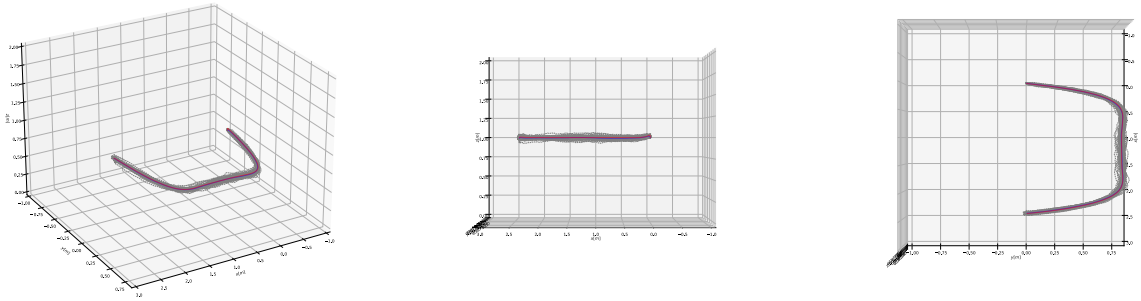
$$\boldsymbol{x_{t+1}} = f(\boldsymbol{x_t}, \boldsymbol{u_t}) + \boldsymbol{w_t} \tag{5.7}$$

and the additive noise w_t following a state dimension dependant uniform law centered on 0:

$$W \sim U(0, w^j) \quad j \in [0, 12] \tag{5.8}$$

with w^j noise range associated with the j^{eme} state dimension and :

$$\boldsymbol{w} = [0.005, 0.005, 0.005, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.005, 0.005, 0.005] \tag{5.9}$$



(a) Isometric (b) Side (c) Top

Figure 5.3.1: Different views of 50 Python simulated quadrotor travel along reference trajectory. Control based on intermediate controller. On these plot the trajectory is made such as the real system violate the constraints. Our tube-MPC should correct it.

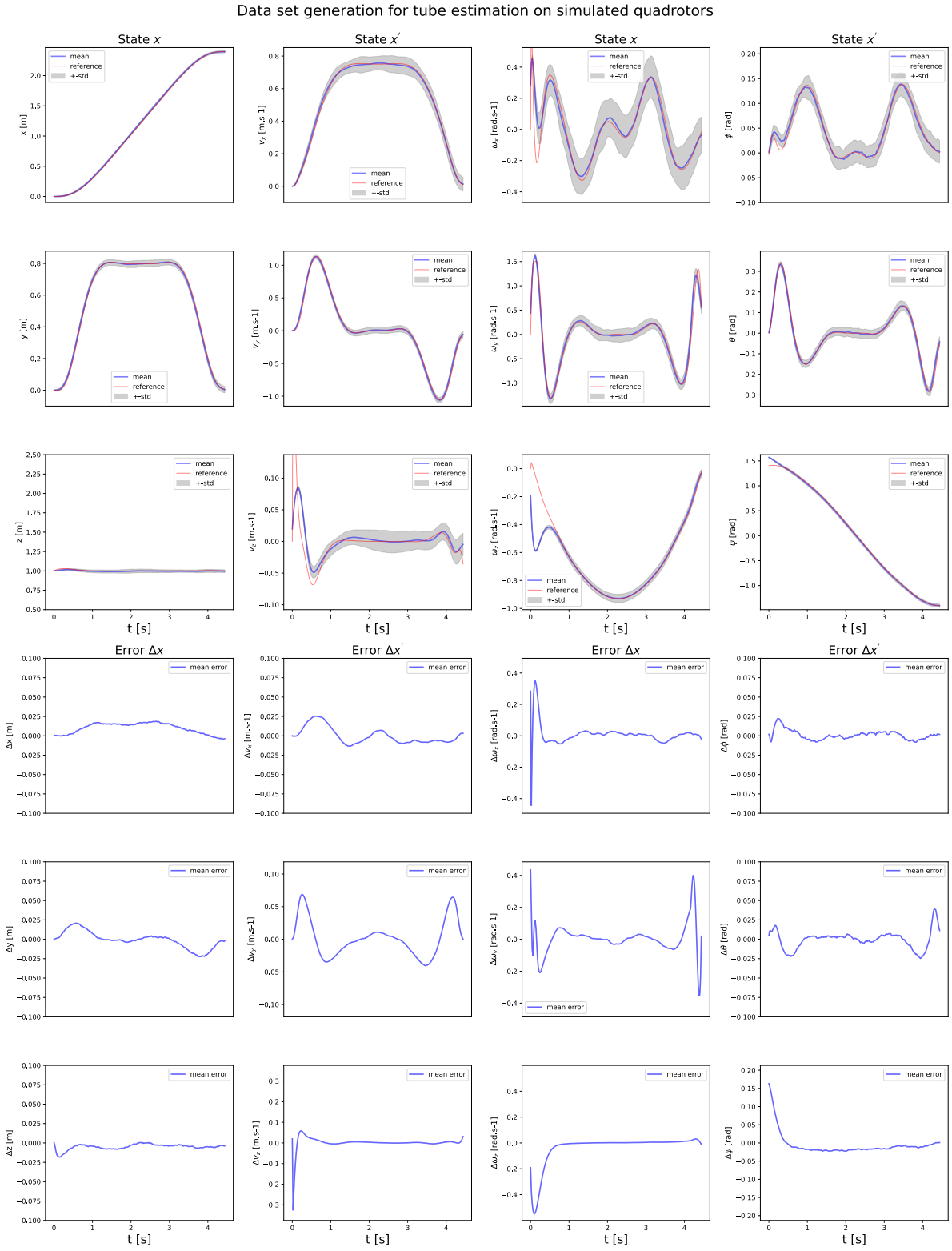


Figure 5.3.2: Summary of 50 Python simulated quadrotor travel along reference trajectory. Control based on NMPC. The upper part show the state evolution and the lower part the average error between the system and the reference. Taking a closer look to the y allows to observe constraints violation.

5.3.2 Training

Param	Value	Param	Value
Hidden Layers	2	Layers width	64
Epochs	5000	Batch Size	30
Activation function	RelU	Dropout	0.2
Optimizer	Adam	Batch norm	True

According to the last section, data set for tube estimation can be generated. The objective is to use these data to perform a quantile regression as in Fan, Agha-mohammadi, and Theodorou 2020 and obtain an inner approximation of the robust tube for tube MPC. However, as mentioned before, the estimation is done off-line, which constitutes a significant difference with the reference paper. In the work presented by Fan, Agha-mohammadi, and Theodorou 2020 they formulated the deep learning estimation directly into the MPC formulation introducing complexity for the solver, which is very difficult to run in real-time systems with fast dynamics, such as the quadrotor. In this work, the choice has been made to keep a simple architecture mainly due to time constraints, as it can be seen in figure 5.3.7.

The described neural network takes a randomized data point ω_t as input and, using the two hidden layers, will learn f_ω to output ω_{t+1}. Three outputs with different α values can be observed and correspond to the different quantile to estimate. In order to build the tube, only $\alpha = 0.02$ and $\alpha = 0.98$ are taken into account.

The training results 5.3.4 show performances under the desired values. Many variations of the architecture have been tried to archive better performances without success. Further work implementing cutting-edge architecture could be interesting. However, the obtained results can be considered as sufficient to validate our approach. Note that validation steps are not applied as no further prevision will be made by the learn model. The objective is to learn the tube and to use it as parameter of the MPC module.

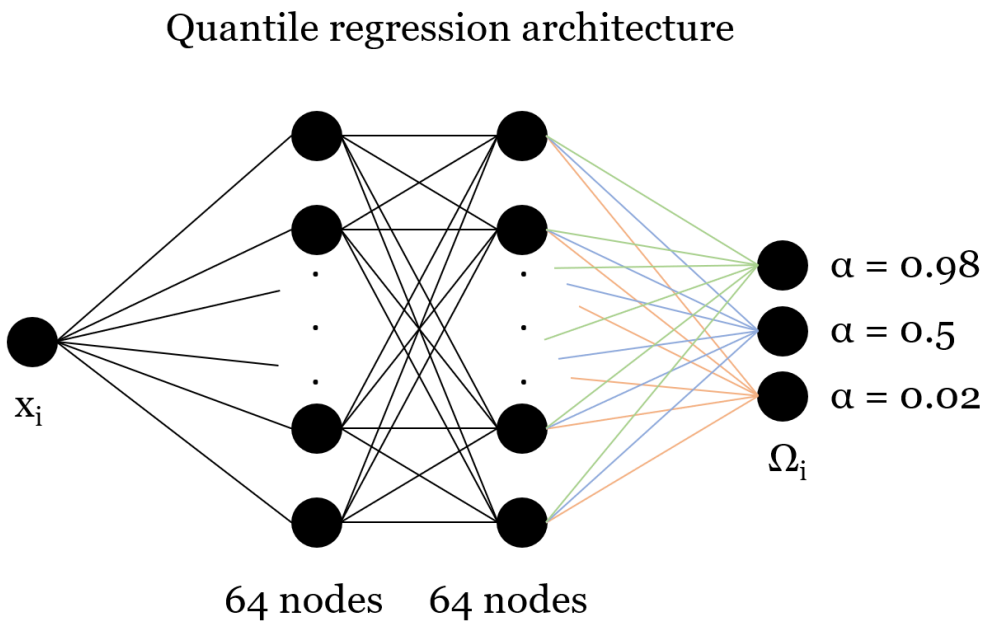


Figure 5.3.3: Deep learning regression architecture

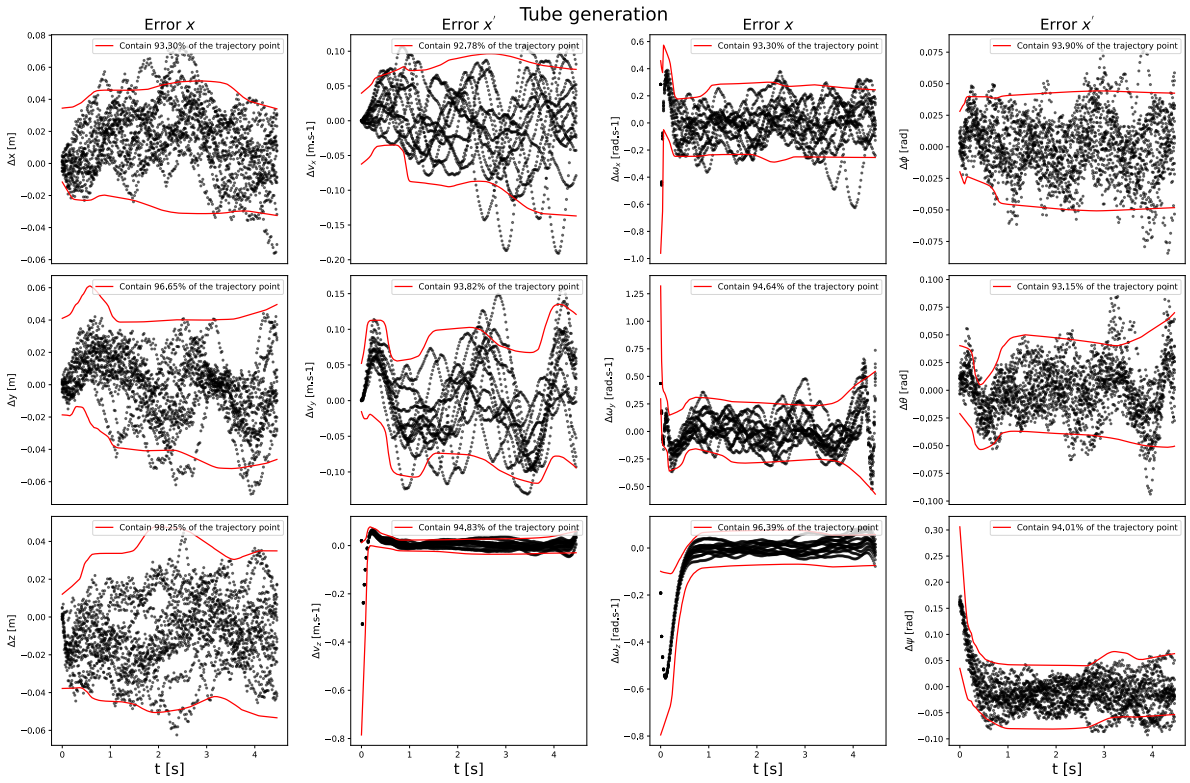


Figure 5.3.4: Estimated tube after training procedure.

5.3.3 Results

Once the tube is estimated, the nominal constraints are tightened according to 3.42, and the simulation is used to verify the quadcopter's robust behavior. This section shows the results of our approach in three different situations.
The first one is described in the tube estimation section. The trajectory pushes the quadrotor close to the upper bound of y, simulating a wall presence. In the last section, tube estimation have been discussed; this section is showing the result of the tube integration in our robust MPC.
The second and third situations are more general but aim to prove that they can be implemented on real systems. Indeed, the assumed objective was to demonstrate that this approach can bring robust MPC to a real fast and complex system. Thus, the two last part shows our approach work on high fidelity simulator and real quadrotor.
The last step would be to put the real system in a position of constraints violation, but due to the heavy time constraints on this project, it will remain as future work.

Constraints activation

In this experiment, it is possible to observe the effect of tightening constraints on the controlled system. After tube estimation, the noise is taken into account, and the nominal component of our robust MPC adds an additional margin to avoid the real system colliding with the wall. At the same time, the feedback policy based on stabilizing K_{LQR} keeps the system in the tube with a probability $\alpha = 0.98$. These two components united, this control approach can be considered robust. The noise has been modified to highlight this robust behavior when the system comes closer to the wall. The uniform distribution remains in the range of the initial one, keeping the validity of our estimation, but only the positive part is kept. This simulates an extreme situation where the robustness will need to guaranty the system's safety.

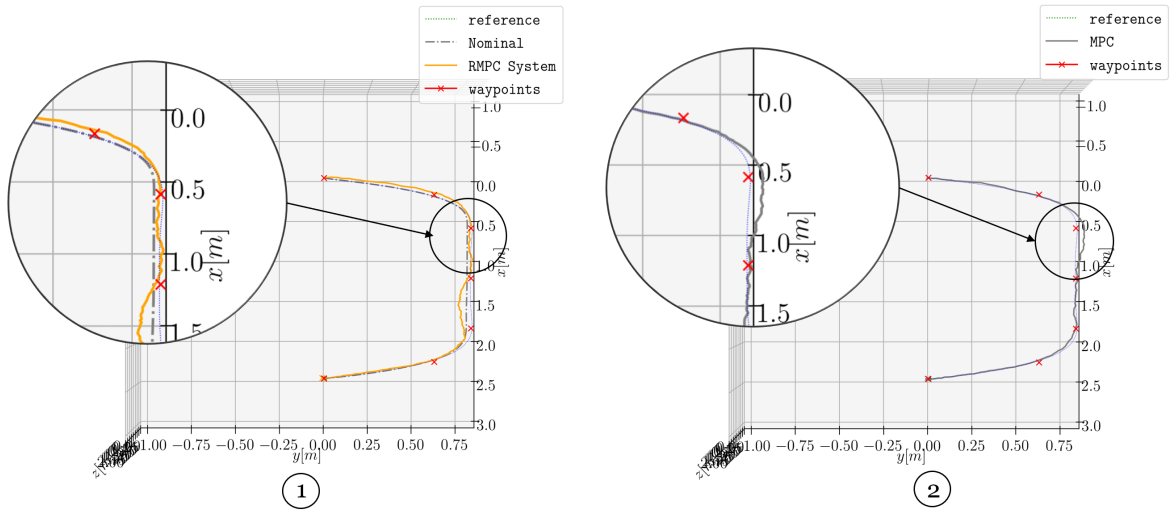


Figure 5.3.5: Comparison of tube MPC with estimated tube of the left (1) and NMPC on the right (2) under the same conditions. The tube MPC shows robust behavior.

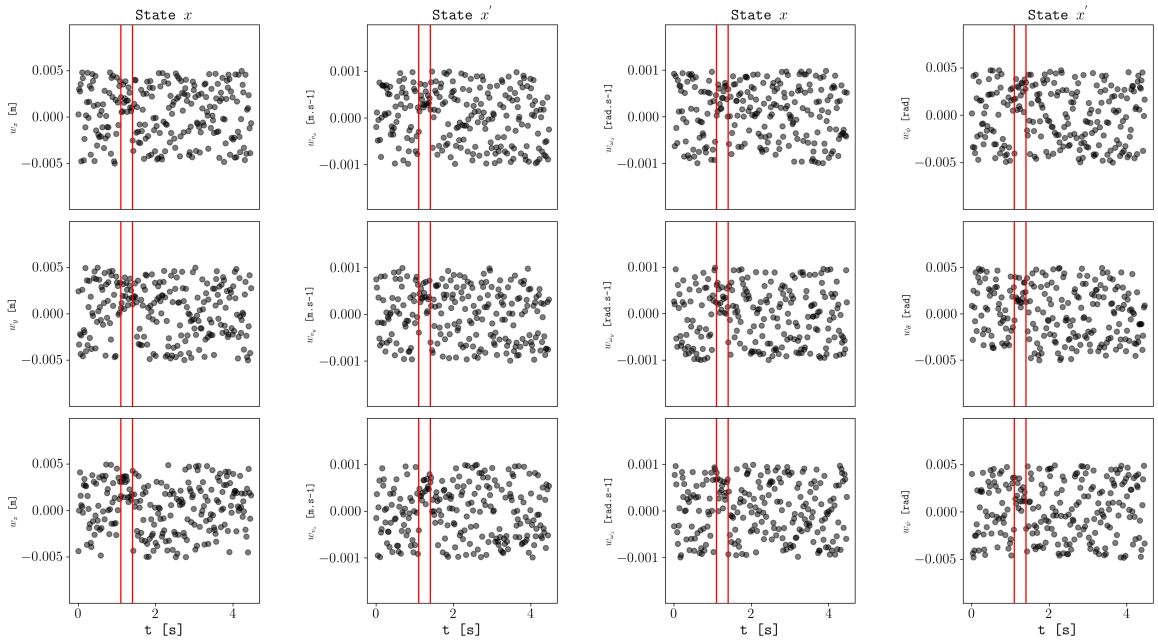
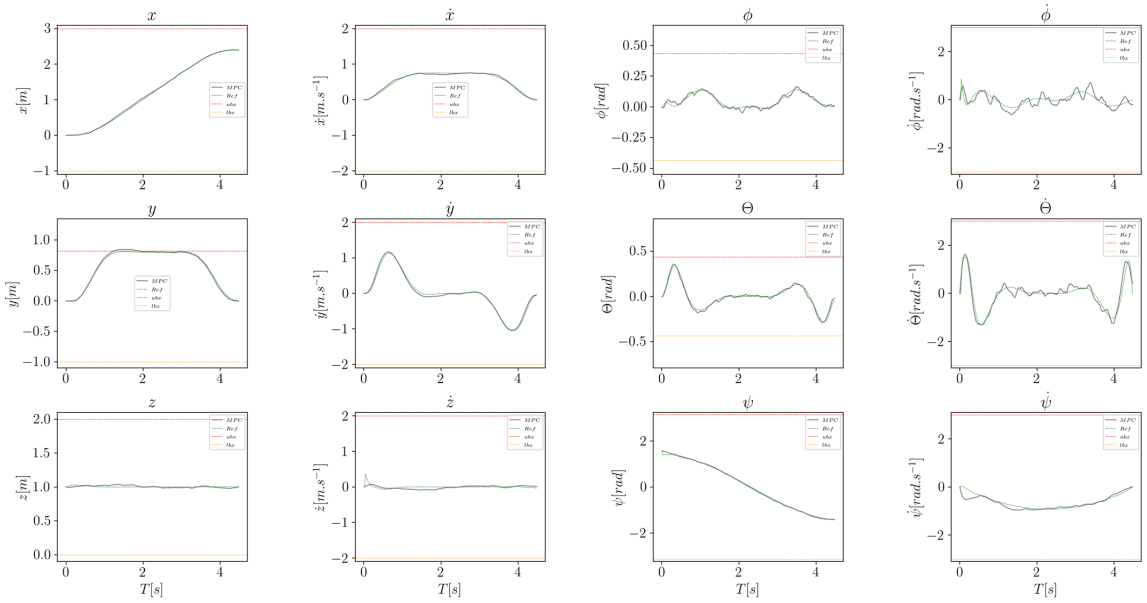


Figure 5.3.6: Noise applied on both systems. Note that the noise is amplified before coming closer to the wall to show the effect of our approach.

NMPC under noise.



Tube MPC with estimated tube under noise.

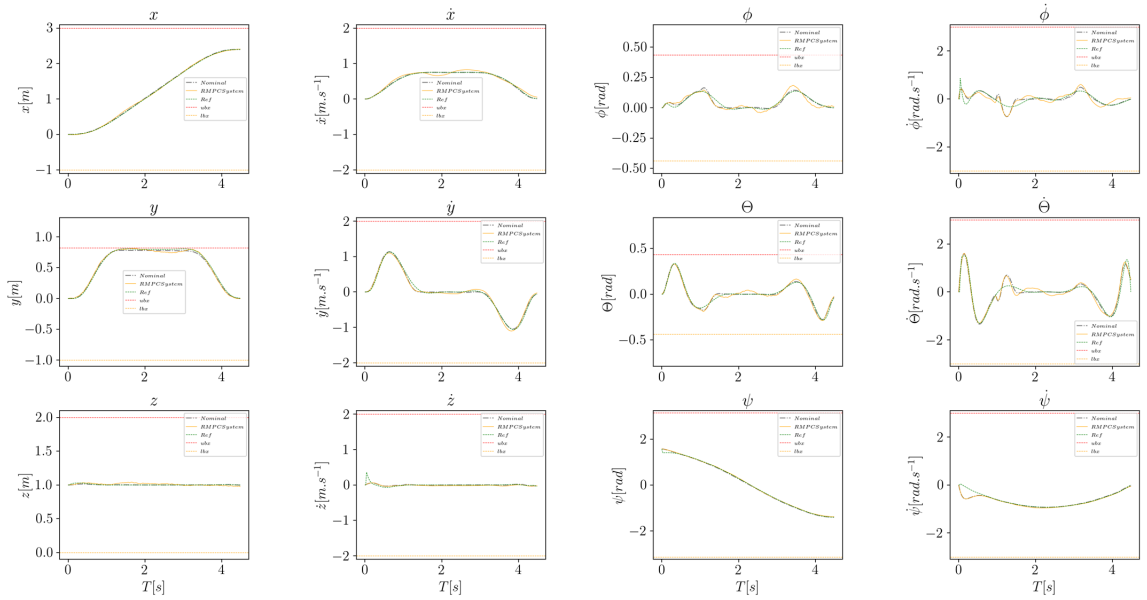
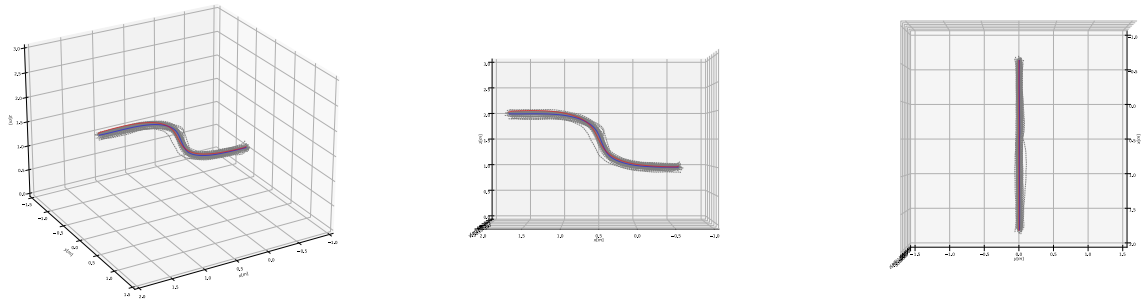


Figure 5.3.7: State evolution of both NMPC and tube MPC with estimated tube under the same noise conditions.

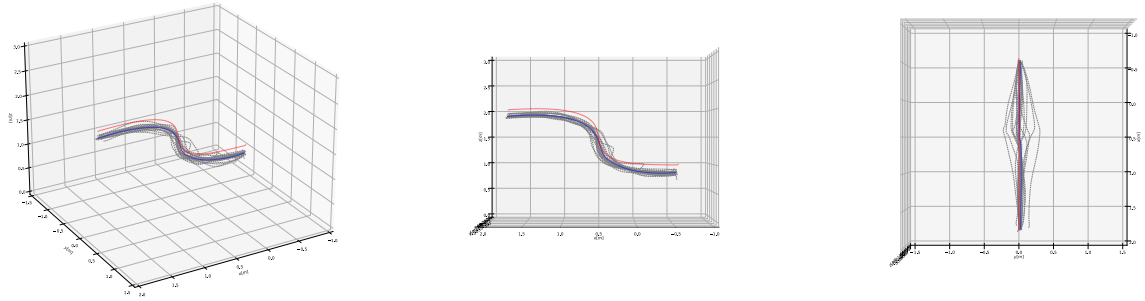
Simulation and hardware

Based on the discussed tube generation, the exact same procedure is applied for the step-like trajectory. A tube has been generated and is integrated into the controller. This experience's objective is not to show robust propriety as the quadcopter travel far from constraints boundaries. The idea here is to show that our approach can be implemented on a real system while keeping a high control rate and satisfying performances. For Gazebo simulation, the performances have been improved as the position error is now lower than for NMPC trajectory tracking. For the hardware experiment, the system mainly suffers from the sub-optimal initial condition and seems to struggle to avoid errors in the y axis. H



(a) Isometric (b) Side (c) Top

Figure 5.3.8: Different views of 50 Gazebo simulated quadrotor travel along reference trajectory. Control based on Tube MPC with estimated tube.



(a) Isometric (b) Side (c) Top

Figure 5.3.9: Different views of 50 quadrotor travel along reference trajectory. Control based on Tube MPC with estimated tube.

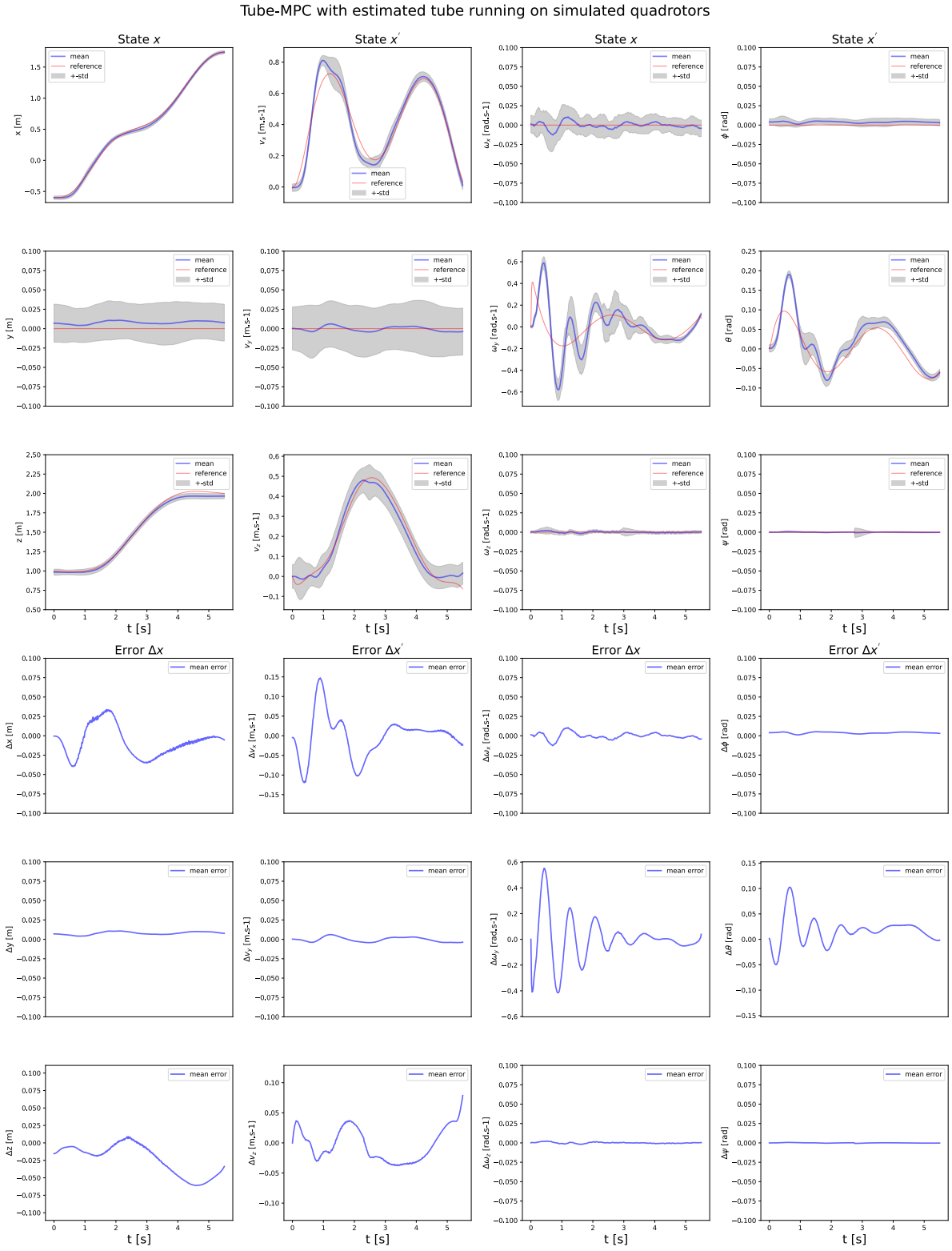


Figure 5.3.10: Summary of 50 Gazebo simulated quadrotor travel along reference trajectory. Control based on Tube MPC with estimated tube. The upper part show the state evolution and the lower part the average error between the system and the reference.

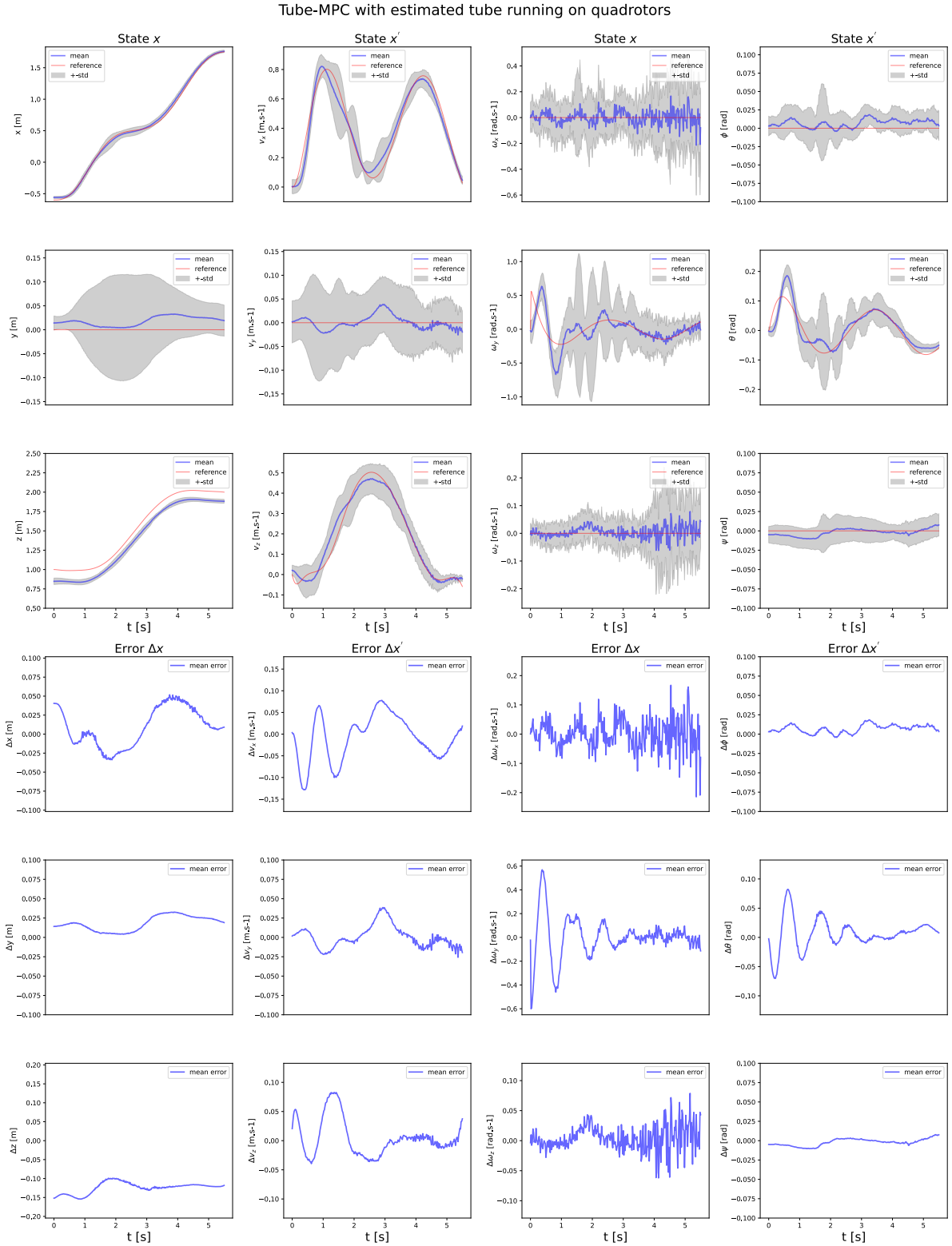


Figure 5.3.11: Summary of 50 quadrotor travel along reference trajectory. CControl based on Tube MPC with estimated tube. The upper part show the state evolution and the lower part the average error between the system and the reference.

Chapter 6

Conclusions

The work presented in this thesis contribute to bringing robust tube MPC on real-life application by combining Tube MPC technique deep learning quantile regression for RPI estimation for systems with fast and complex dynamic- a quadrotor in this case.
It is shown that the controlled system uses the learned tube to avoid constraint violation in a robust manner, using offline generated tube and online updated feedback policy.
It is essential to highlight that a lot of work has been done to provide experimental support for validation and future experiments. Keeping in mind flexibility and modularity, this work's results are also a development platform for future improvement. This developed platform proved its efficiency as several experiments have been performed with success from simulation to real drones while keeping consistency on the results.

6.0.1 Future Work

Even if the work provides promising results, it only constitutes the first steps of tube estimation for robust MPC control on a real complex system. As the project's objectives were centered on experimental setup and validation, some additional theoretical developments need to be done to reinforce the bases of the controller derivation. A central point would be stability proof for such MPC formulation in indiscreet time, which is not trivial but central to safety-critical application. In addition, some performance validations have to be done as, due to time constraints, robust behavior has not been shown on high fidelity simulation or hardware support. However,

promising results on Python simulation and first implementation on the real system shows that these test could provide the same kind of outcome.
The tube estimation gave satisfying results. However, there is a big room for improvement and many things to try. From live estimation with GPU hardware to multi-step forecasting using recurrent neural networks, this field is growing extremely fast, and this work demonstrated an interest in exploring such techniques.
Concerning the experimental setup, fine-tuning and code optimization could provide better performances essentially on a no-linear solver. Going this way would allow to allocate time to other tasks such as tube forecasting or any further development while keeping a high control frequency.

Bibliography

acados — acados documentation (2021). URL: https://docs.acados.org/ (visited on 02/24/2021).

Allan, Douglas A., Bates, Cuyler N., Risbeck, Michael J., and Rawlings, James B. (2017). "On the inherent robustness of optimal and suboptimal nonlinear MPC". In: *Systems & Control Letters* 106, pp. 68–78. ISSN: 0167-6911.

Berkenkamp, Felix, Turchetta, Matteo, Schoellig, Angela P., and Krause, Andreas (2017). *Safe Model-based Reinforcement Learning with Stability Guarantees*. arXiv: 1705.08551 [stat.ML].

Bresciani, Tommaso (2008). "Modelling, Identification and Control of a Quadrotor Helicopter". In: pp. 119–125.

Chen, H. and Allgöwer, F. (1998a). "A Quasi-Infinite Horizon Nonlinear Model Predictive Control Scheme with Guaranteed Stability□□This paper was not presented at any IFAC meeting. This paper was accepted for publication in revised form by Associate Editor W. Bequette under the direction of Editor Prof. S. Skogestad." In: *Automatica* 34.10, pp. 1205–1217.

— (1998b). "Nonlinear Model Predictive Control Schemes with Guaranteed Stability". In: *Nonlinear Model Based Process Control*. Ed. by Ridvan Berber and Costas Kravaris. Dordrecht: Springer Netherlands, pp. 465–494.

Chen, Wen-Hua, O'Reilly, John, and Ballance, Donald J. (2003). "On the terminal region of model predictive control for non-linear systems with input/state constraints". In: *International Journal of Adaptive Control and Signal Processing* 17.3, pp. 195–207.

Chisci, L., Rossiter, J. A., and Zappa, G. (2001). "Systems with persistent disturbances: predictive control with restricted constraints". In: *Automatica* 37.7, pp. 1019–1028.

Fan, David D., Agha-mohammadi, Ali-akbar, and Theodorou, Evangelos A. (2020). *Deep Learning Tubes for Tube MPC*.

Fan, David D., Nguyen, Jennifer, Thakker, Rohan, Alatur, Nikhilesh, Agha-mohammadi, Ali-akbar, and Theodorou, Evangelos A. (2020). *Bayesian Learning-Based Adaptive Control for Safety Critical Systems*. arXiv: 1910.02325 [eess.SY].

Faulwasser, Timm (Oct. 2012). "Optimization-based Solutions to Constrained Trajectory-tracking and Path-following Problems". PhD thesis. Otto von Guericke University Magdeburg, pp. 10–19.

Findeisen, Rolf (2004). "Nonlinear Model Predictive Control: A Samples-Data Feedback Perspective". PhD thesis.

Findeisen, Rolf, Imsland, Lars, Allgower, Frank, and Foss, Bjarne A. (2003). "State and Output Feedback Nonlinear Model Predictive Control: An Overview". In: *European Journal of Control* 9.2, pp. 190–206.

Gao, Yiqi, Gray, Andrew, Tseng, H. Eric, and Borrelli, Francesco (2014). "A tube-based robust nonlinear predictive control approach to semiautonomous ground vehicles". In: *Vehicle System Dynamics* 52.6, pp. 802–823.

Henderson, D. (1977). "Shuttle Program. Euler angles, quaternions, and transformation matrices working relationships". In:

James B. Rawlings David Q. Mayne, Moritz M. Diehl (2017a). *Model Predictive Control: Theory, Computation, and Design, 2nd Edition*. Nob Hill Publishing, LLC, pp. 252–262.

— (2017b). *Model Predictive Control: Theory, Computation, and Design, 2nd Edition*. Nob Hill Publishing, LLC, pp. 200–202.

— (2017c). *Model Predictive Control: Theory, Computation, and Design, 2nd Edition*. Nob Hill Publishing, LLC, pp. 11–26.

Kalman, R. (1960). "A new approach to linear filtering and prediction problems" transaction of the asme journal of basic". In:

Kalman, Rudolf (2001). "Contribution to the Theory of Optimal Control". In: *Bol. Soc. Mat. Mexicana* 5.

Kálmán, R. and Bucy, R. (1961). "New Results in Linear Filtering and Prediction Theory". In: *Journal of Basic Engineering* 83, pp. 95–108.

Koenker, Roger and Bassett, Gilbert (1978). "Regression Quantiles". In: *Econometrica* 46.1, pp. 33–50.

Lazar, Mircea and Spinu, Veaceslav (2015). "Finite-step Terminal Ingredients for Stabilizing Model Predictive Control". In: *IFAC-PapersOnLine* 48.23, pp. 9–15.

Lee, E. B. and Markus, L. (1967). *Foundations of optimal control theory [by] E. B. Lee [and] L. Markus*. Wiley New York, x, 576 p.

Lee, Jay H. (June 4, 2011). "Model predictive control: Review of the three decades of development". In: *International Journal of Control, Automation and Systems* 9.3, p. 415.

Mayne, D. Q., Kerrigan, E. C., Wyk, E. J. van, and Falugi, P. (2011). "Tube-based robust nonlinear model predictive control". In: *International Journal of Robust and Nonlinear Control* 21.11, pp. 1341–1353.

Mayne, D. Q., Rawlings, J. B., Rao, C. V., and Scokaert, P. O. M. (2000). "Constrained model predictive control: Stability and optimality". In: *Automatica* 36, pp. 789–814.

Mayne, D. Q., Seron, M. M., and Raković, S. V. (2005). "Robust model predictive control of constrained linear systems with bounded disturbances". In: *Automatica* 41.2, pp. 219–224.

Mayne, D.Q. and Langson, Wilbur (2001). "Robustifying Model Predictive Control of Constrained Linear Systems". In: *Electronics Letters* 37, pp. 1422–1423.

Mayne, David Q. and Kerrigan, Eric C. (2007). "TUBE-BASED ROBUST NONLINEAR MODEL PREDICTIVE CONTROL1". In: *IFAC Proceedings Volumes* 40.12, pp. 36–41.

Nguyen, Thinh, Prodan, Ionela, Stoican, Florin, and Lefèvre, Laurent (July 2017). "Reliable nonlinear control for quadcopter trajectory tracking through differential flatness". In: *IFAC-PapersOnLine* 50.

Qin, S.Joe and Badgwell, Thomas A. (2003). "A survey of industrial model predictive control technology". In: *Control Engineering Practice* 11.7, pp. 733–764.

Raemaekers, A.J.M. (Dec. 2007). "Design of a model predictive controller to control UAVs". PhD thesis. Royal Melbourne Institute of Technology.

Rakovic, S. V., Kerrigan, E. C., Kouramas, K. I., and Mayne, D. Q. (2005). "Invariant approximations of the minimal robust positively Invariant set". In: *IEEE Transactions on Automatic Control* 50.3, pp. 406–410. DOI: 10.1109/TAC.2005.843854.

Richards, Arthur and How, Jonathan (2005). "Decentralized Model Predictive Control of Cooperating UAVs". In: 4, 4286–4291 Vol.4.

Rodrigues, F. and Pereira, F. C. (2020). “Beyond Expectation: Deep Joint Mean and Quantile Regression for Spatiotemporal Problems”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.12, pp. 5377–5389.

Scokaert, Pierre O. M. and Rawlings, James B. (1996). “Infinite Horizon Linear Quadratic Control with Constraints”. In: *IFAC Proceedings Volumes* 29.1, pp. 5905–5910.

Singh, L. and Fuller, J. (2001). “Trajectory generation for a UAV in urban terrain, using nonlinear MPC”. In: 3, 2301–2308 vol.3.

Taylor, James W. (1999). “A Quantile Regression Approach to Estimating the Distribution of Multiperiod Returns”. In: *The Journal of Derivatives* 7.1. Publisher: Institutional Investor Journals Umbrella, pp. 64–78.

Valluri, S. and Kapila, V. (1998). “Stability analysis for linear/nonlinear model predictive control of constrained processes”. In: 3.

Yu, Shuyou, Maier, Christoph, Chen, Hong, and Allgöwer, Frank (2013). “Tube MPC scheme based on robust control invariant set with application to Lipschitz nonlinear systems”. In: *Systems & Control Letters* 62.2, pp. 194–200.

Yu, Shuyou, Reble, Marcus, Chen, Hong, and Allgöwer, Frank (2011). “Inherent Robustness Properties of Quasi-infinite Horizon MPC”. In: *IFAC Proceedings Volumes* 44.1, pp. 179–184.

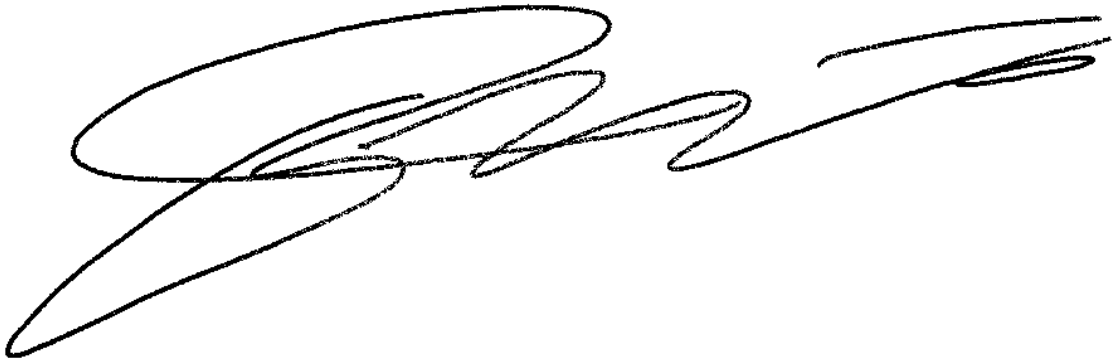


Figure 6.0.1: Maxime Boutot