ÉCOLE CENTRALE DE NANTES

MASTER CORO-IMARO
"CONTROL AND ROBOTICS"

2020 / 2021

Master Thesis Report

Presented by

Elie Hatem

On July 23, 2021

Making Flips With Quadrotors In Constrained Environments

Jury

Evaluators:	Dr. Olivier Kermorgant	Associate Professor (ECN)
	Dr. Damien SIX	Robotics Engineer (CNRS)
Supervisor(s):	Dr. Sébastien Briot	Researcher (CNRS)
	Dr. Isabelle Fantoni	Research Director (CNRS)

Laboratory: Laboratoire des Sciences du Numérique de Nantes LS2N

Abstract

Within the rapidly growing aerial robotics market, one of the most substantial challenges in the quadrotor community is performing aggressive maneuvers, especially multi-flip maneuvers. A proper physical definition of the issue is not addressed by the current approaches in the field and several key aspects of this maneuver are still overlooked. It can be shown, in particular, that making a flip with a quadrotor means crossing the parallel singularity of the dynamic model. The aim of the master thesis is to explore the possibility of defining aggressive trajectories for quadrotors on the basis of their dynamic model degeneracy analysis and to adapt various strategies to control the robot in a closed loop. In addition, the possibility of performing the aggressive maneuvers in constrained environments will also be investigated. Therefore, the analysis will be extended from the previous studies to create general feasible trajectories that will allow quadrotors to perform aggressive multi-flip maneuvers while passing through a constrained environment and while guaranteeing a satisfactory degree of robustness to the uncertainties of the dynamic model.

Keywords: quadrotors, parallel robots, aggressive maneuvers, multi-flips, constrained environment.

Acknowledgements

I would like to express my special thanks and gratitude to my supervisors Dr. Sébastien Briot and Dr. Isabelle Fantoni who gave me the opportunity to work on this wonderful project which encapsulates control theory, dynamics and quadrotors, which are all subjects that are very interesting for me. This project has allowed me to perform research on all of these topics and I am now more knowledgeable thanks to my supervisors. Moreover, I would like to thank them for believing in my capabilities and giving me the confidence and the support when I needed it.

Moreover, I would like to thank Julian Erskine, Damien Six and Shiyu Liu for their help throughout my master thesis. I have learned alot about quadrotors, ROS and coding in general from them.

My profound gratitude to my girlfriend Glysa, who has been with me for more than 6 years. Thank you for all the love, support and comfort that you have given me during these stressful 2 years, and thank you for choosing me everyday.

I would like to thank my family as well: my parents Naji and Yolla, my sister Rebecca, my uncle Fadi, his wife Lara and my aunt Bernadette. And, my aunts Nada and Hoda. They have provided me with the emotional and economical support from the very beginning and they gave me the opportunity to travel and study for this Master's degree. They have always been proud and encouraging.
I would not be where I am today if it wasn't for them.

Finally, I would like to thank everyone who will take the time to read my master thesis report and to the future researchers that may be reading it in order to improve my work. I hope that the report is clear enough to read and I will always be available if you have any questions on my work.

Notations

$I_{k\times k}$	identity matrix of size $k \times k$
I_{xx}, I_{yy}, I_{zz}	diagonal terms of the inertia matrix
ϕ, θ, ψ	roll, pitch and yaw angles respectively
l	arm length of a quadrotor
u_1	total thrust input of a planar quadrotor
u_2	total torque input of a planar quadrotor
u, T	total thrust input of the 3D quadrotor (depending on the model used)
$\boldsymbol{\tau}$	vector of the torque inputs
$\omega_x, \omega_y, \omega_z$	angular rates inputs with respect to the x, y and z axes respectively
$T_s, \Delta t$	sample Time
N	prediction horizon
m	control horizon
J	objective function
M_i	motor i
b	thrust factor
d	drag force
Ω_i	speed of motor i
f_i	force resulting from rotor i, perpendicular to the rigid body

Abbreviations

UAV	unmanned aerial vehicle
CoG	center of gravity
MPC	model predictive control
NMPC	nonlinear model predictive control
HLC	high level commander
SMC	sliding mode control
MIMO	multi-input multi-output
OCP	optimal control problem
SQP	sequential quadratic program
QP	quadratic program
NLP	a nonlinear program
RTI	real-time iteration
API	application programming interface
ODE	ordinary differential equations
KF	Kalman filter
EKF	extended Kalman filter
SIL	software-in-the-loop

List of Figures

List of Tables

Contents

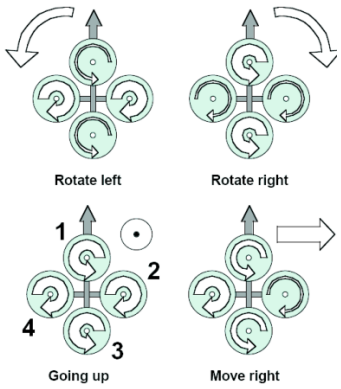
Introduction

The aim of this section is to provide a general summary of the robotic platform that is used for this master thesis and to illustrate the main objective of the research work. In specific, in the sections below, quadrotors and parallel robots are briefly presented.

The quadrotor platform

A quadrotor is a type of unmanned aerial vehicle (UAV) with four rotors and six degrees of freedom. Typically, drones have a small size and low inertia which allows them to be controlled by simple flight control systems. It is typically designed in a cross-configuration such that the electronics are held in the center of the platform and the rotors are placed at the borders. An example of a real quadrotor, namely the DJI Phantom, is shown in figure 1a. The quadrotor is typically built in a way such that a pair of opposite rotors rotates in a clockwise direction, whereas the other pair rotates in a counter-clockwise direction. The attitude and the position of the drone are controlled by changing the spinning speed of the rotors, as shown in figure 1b.

(a) A DJI Phantom quadcopter (UAV)[1]



(b) Representation of the concept of a quadrotor. The width of the arrows is proportional to the angular speed of the propellers.[16]

Figure 1: A commercial quadrtotor platform with a representation of the quadrotor concept.

The distinctive mechanical design of the quadrotor permits the actuation system to control all of the six degrees of freedom even though it is under-actuated. This is due to the fact that the rotational and translational dynamics are tightly coupled. Thus, all the translational and rotational motions can be carried off by properly controlling the magnitude and direction of the spinning speed of the rotors.

[1] https://en.wikipedia.org/wiki/Quadcopter#/media/File:Quadcopter_camera_drone_in_flight.

Over the last few years, quadrotors have gained a large popularity in academia and in the industry. This is due to several reasons, such as:

1. Quadrotors are very simple to design and they can be easily assembled using relatively cheap components.

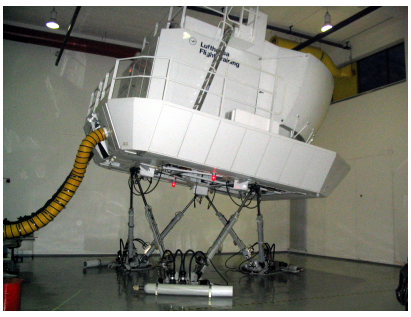
2. As quadrotors became more and more affordable and dependable, the number of real-world applications for quadrotors has grown significantly. They are being used for aerial photography, agriculture, surveillance, inspection tasks, in addition to many other uses as well.

3. Quadrotors are quite agile and maneuverable during flight, especially when compared to other types of UAVs.

However, one of the main challenges in the quadrotors community is the capability to design control and planning methods that will allow the quadrotors to carry out aggressive maneuvers. The fast dynamics associated with typically small dimensions of such agile quadrotors, along with several aerodynamic effects that will become crucial during aggressive flight maneuvers, are just a few of the main problems that are faced during the system control design. Moreover, accurate tracking of the provided trajectory is a big issue in the case of aggressive maneuvers when the rotors are commanded high speeds and accelerations, which will cause rotors to become saturated and may also cause delays.

Parallel manipulators

A parallel manipulator is a mechanical system that consists of two connected platforms, the fixed platform and the moving platform. The latter is linked to the fixed platform thanks to at least two serial chains that are working in parallel. When compared to serial manipulators, parallel manipulators are more accurate and rigid. In addition, the ability to install the motors next to the fixed platform is a very important feature for them. Moreover, parallel manipulators can be used in a wide variety of applications that demand precision and high payload combined with high speed.[17]



(a) Gough-Stewart used for a flight-simulator application.[2]

(b) The "PAR4" 4 degrees of freedom, high-speed, parallel robot prototype.[3]

Figure 2: Two examples of parallel robots.

jpg, accessed on 01/08/2021.

[1] https://en.wikipedia.org/wiki/Stewart_platform#/media/File:Simulator-flight-compartment.jpeg, accessed on 01/08/2021.

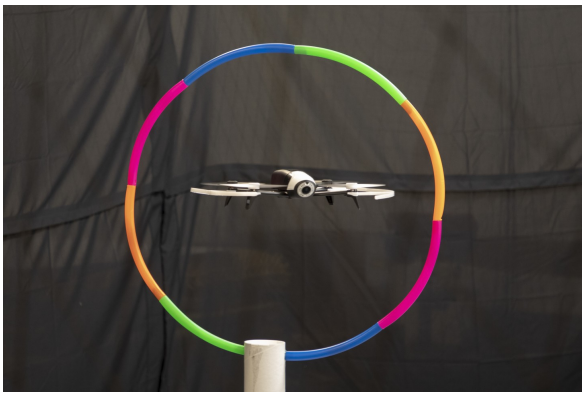
[2] https://en.wikipedia.org/wiki/Parallel_manipulator#/media/File:Prototype_robot_parall%C3%A8le_PAR4.jpg, accessed on 01/08/2021.

However, parallel manipulators are subject to singularities which can lead to big problems in the robot workspace in case they were not handled correctly. Thus, the study of the singular configurations of parallel manipulators is very important. Because, even just before reaching a singularity, the performance of the parallel manipulator will decrease dramatically. Moreover, the robot may loose the ability of moving in a certain direction, gain uncontrollable motions and the mechanism could even break. The main difference between serial and parallel manipulators is that singularity configurations may also appear inside the workspace of the robot (depending on the dimensions of the robot) and not just at the boundaries of the robot workspace, which can significantly decrease the area of the robot workspace. As a result, many works have been developed by robotics researchers in order to allow parallel manipulators to safely cross these singularities by using trajectory planning and specific control methods.

The goal of this thesis

This master thesis lies at the intersection of parallel robotics and aerial robotics. The two fields may seem very different from each other. However, quadrotors can be seen as a particular case of a parallel manipulator. In fact, a parallel manipulator is made up of a wrench system, applied by the robot limbs on the moving platform. And, this wrench system will define the motion of the moving platform. In the same manner, each propeller in a quadrotor can be considered as a limb of a parallel robot and the moving platform to be controlled can be considered as the body of the drone. Specifically, the goal of this master thesis is to study a distinct class of aggressive maneuvers for quadrotors, namely flip maneuvers. By doing flip maneuvers, full rotations around one or more axes of the body of the quadrotor can be done. In addition, the quadrotor should also be able to perform the flip maneuvers in constrained environments.

(a) Quadrotor performing a triple flip.[18]



(b) Quadrotor going though a loop.[1]

Figure 3: Representation of the issues to be tackled in the master thesis.

[1] `https://newatlas.com/drones/muscle-signals-drone-control/#gallery:2`, accessed on 01/08/2021.

Outline of the work

The rest of the report is structured as follows:

Chapter 1 provides an overview of the state of the art in the control of quadrotors and will later on focus on the main control method that will be used, namely Model Predictive Control (MPC). Moreover, a literature review of MPC applications on quadrotors will be presented. Furthermore, an overview on the software used to design a MPC controller will be presented. Finally, the state of the art in flipping maneuvers will be presented.

Chapter 2 provides a detailed explanation of the quadrotor dynamics for the planar (2D) and 3D quadrotors. Moreover, an Extended Kalman Filter (EKF) is then designed for the planar quadrotor case to be used in the presence of noisy measurements (states) and noisy control inputs. Finally, simulation results using MPC to reach a single waypoint and to follow circular trajectories with and without noise are presented.

Chapter 3 focuses on the trajectory generation of a flip trajectory where different optimization problems with different objective functions and initial conditions will be used in order to find the optimal flipping trajectory that satisfies the dynamic constraints of the quadrotor which will be used in the experimentation phase. Moreover, simulations with a MPC are then performed using the optimal flip trajectory.

Chapter 4 focuses on Software-in-the-Loop simulations that were performed using ROS2 and Gazebo, in addition to the experimentation results.

Chapter 1

State of the Art

In the following sections of this chapter, the general control architecture of a quadrotor, differential flatness, different potential control approaches (linear and nonlinear), and the main control method that will be used to control a quadrotor, namely Model Predictive Control (MPC) will be explained.

1.1 General Control Architecture

Recently, many researchers have developed interest in the control of quadrotors. As a result, various control approaches have been proposed. The most known control architecture [19] consists of three nested control loops, as shown in figure 1.1, in order to generate the suitable motor commands to follow the desired signal. This controller is known as the cascaded controller. This strategy assumes that the attitude dynamics of a quadrotor are much faster than the translational dynamics.

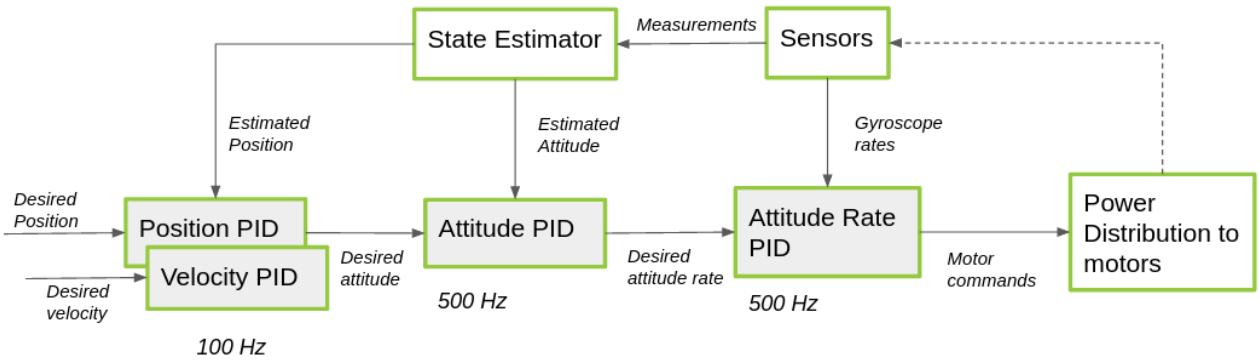


Figure 1.1: Cascaded PID controller that is present on the crazyflie 2.1 quadrotor [1].

In this case, the position and velocity controller, attitude controller and attitude rate controller are all PID controllers. And, it is evident that the attitude dynamics (who's attitude and attitude rate controllers operate at a frequency of 500Hz) are considered to be much faster than the translational dynamics (who's position and velocity controllers operate at a frequency of 100Hz).

Moreover,

Attitude Rate controller directly controls the attitude rate. It receives the gyroscope rates after they have been filtered and uses the error between the desired attitude rate and current attitude rate, and outputs the motor commands which are then directly sent to the power distribution.

Attitude controller directly controls the attitude of the drone. It takes the estimated attitude from the state estimator and uses the error between the desired attitude and the current attitude to output the desired attitude rate.

Position and Velocity controller is the most outer-loop of the cascaded PID controller. It receives the position or the velocity input from the high level commander which are then handled to output the desired attitude.

Furthemore, it should be noted that the controllers above can be of any type. However, in the case of the crazyflie 2.1, PID controllers were used.

1.2 Differential Flatness

In the quadrotor community, a well-established finding is that the dynamic model of a quadrotor is differentially flat. Moreover, the control design problem in nonlinear systems will be considerably simplified. Precisely, a system with state $\mathbf{x} \in \mathbb{R}^n$ and input $\mathbf{u} \in \mathbb{R}^m$ is considered to be *differentially flat* if there exists a set of *flat outputs* $\mathbf{y} \in \mathbb{R}^m$ which have the following form:

$$\mathbf{y} = \mathbf{y}(\mathbf{x}, \mathbf{u}, \dot{\mathbf{u}}, ..., \mathbf{u}^{(p)}) \tag{1.1}$$

With,

$$\begin{cases} \mathbf{x} = \mathbf{x}(\mathbf{y}, \dot{\mathbf{y}}, ..., \mathbf{y}^{(q)}) \\ \\ \mathbf{u} = \mathbf{u}(\mathbf{y}, \dot{\mathbf{y}}, ..., \mathbf{y}^{(r)}) \end{cases} \tag{1.2}$$

As a result, the new set of variables is required to be a function of the state, the input and the derivatives of the input. Moreover, this set should also have the same dimensions as the control input. In this manner, it is possible to rewrite both the state and the input in function of the flat outputs and the derivatives of the flat outputs. This is a very useful property in under-actuated systems where $m < n$, such as quadrotors, because, it will allow to generate trajectories in the lower dimensional space m, then this trajectory will be mapped into the full dimensional space n. An example of this is shown in figure 1.2 below:

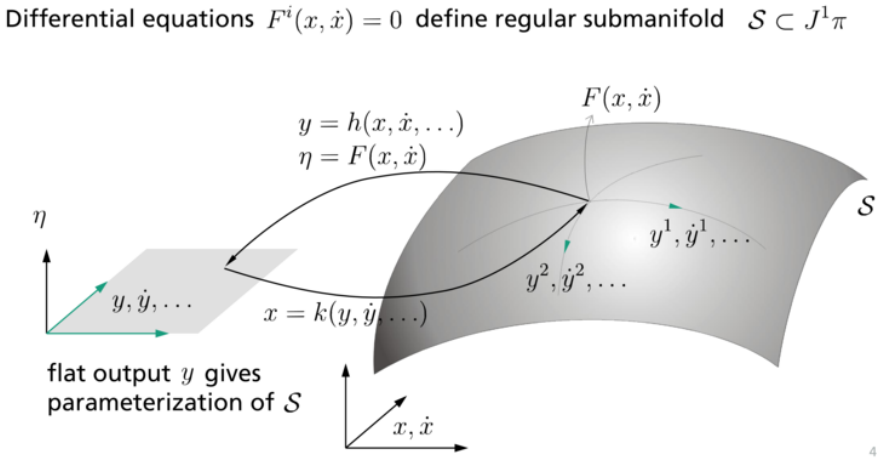


Figure 1.2: Mapping between different dimensional spaces as a result of differential flatness. [2]

Another well known example of systems is a car, in which the under-actuation is the result of the non-holonomic constraints that are imposed by the wheels. So, for a car, a generated trajectory for (x, y) position of the rear-wheels is enough to specify all the viable trajectories of the system. Formal proofs that the quadrotor system is differentially flat can be found in [20], and [19] for the full model with first-order aerodynamics. The standard choice of flat outputs for the quadrotor is the coordinates of the center of mass and the yaw angle:

$$\mathbf{y} = \begin{bmatrix} x & y & z & \psi \end{bmatrix}^\intercal \tag{1.3}$$

Consequently, the problem of generating a feasible trajectory for a quadrotor then tracking it can be dimensionally decreased from a 6-dimensional space to a 4-dimensional space. By reason of the tight coupling between the rotational and translational dynamics, then defining a trajectory in function of the flat outputs $\mathbf{y}$ is sufficient to properly define the full dynamics $\mathbf{x}$.

1.3 General Control Approaches

1.3.1 Method of Linearization

By using extreme assumptions, it is feasible to apply linear control techniques in order to control a quadrotor ([21], [22]). Particularly, this can be made by doing a linearization of the full dynamic model around an equilibrium point $\overline{\mathbf{x}}$ and by using the assumption that the vehicle is only capable of oscillating lightly around the hover point. It is very easy to observe that a feasible equilibrium is provided by a configuration where the center of mass is at a random position $\overline{\mathbf{r}}$ and all the other elements of the state are set to zero. So, the nominal input $\mathbf{u} = \overline{\mathbf{u}}$ to sustain such equilibrium can be assessed as the thrust that is required to compensate the gravity force:

$$\overline{\mathbf{u}} = \begin{bmatrix} f \\ \boldsymbol{\tau} \end{bmatrix} = \begin{bmatrix} mg \\ \mathbf{0_{3\times 1}} \end{bmatrix} \tag{1.4}$$

At this stage, the complete non-linear dynamics that have the form :

$$\dot{\mathbf{x}} = \overline{\mathbf{f}}(\overline{\mathbf{x}}, \overline{\mathbf{u}}) \tag{1.5}$$

can now be linearized around the hover point $(\overline{\mathbf{x}}, \overline{\mathbf{u}})$ as shown below.

$$\dot{\mathbf{x}} = \left[\frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\right]_{(\bar{\mathbf{x}},\bar{\mathbf{u}})} \mathbf{x} + \left[\frac{\partial \mathbf{f}(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\right]_{(\bar{\mathbf{x}},\bar{\mathbf{u}})} \mathbf{u} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{1.6}$$

It can be demonstrated that both matrices $\mathbf{A}$ and $\mathbf{B}$ can be used to determine a linear system that is both controllable and observable [21]. Thus, any control technique that is linear can now be used on the quadrotor in order to keep it around a desired equilibrium point, such as optimal LQR/LQG [23, 24] control or simple PD or PID controller [25, 26].

1.3.2 Nonlinear Control Methods

In order to perform more complex tasks and follow aggressive trajectories, nonlinear control methods are required. A comprehensive literature review on this topic is beyond the scope of this work and several works can be found, such as [27], which provides a general overview on nonlinear control of quadrotors. However, some nonlinear control methods deserve to be mentioned due to their extensive use and applications:

Sliding Mode control It is a control technique that is nonlinear presenting exceptional attributes of robustness,accuracy, easy tuning and execution. The aim of SMC systems is to drive the system states to a specific surface in the state space, called *"sliding surface"*. Upon reaching the sliding surface, sliding mode control allows the states to remain on the close neighborhood of the sliding surface. Therefore, the sliding mode control consists of a controller design with two parts. The first part contains the design of a sliding surface in order for the sliding motion to fulfill design requirements. The second deals with selecting a control law that makes the switching surface interesting with respect to the system state [28]. There exist two main benefits of sliding mode control. Firstly, the behavior of the dynamics of the system can be changed according to a specific selection of the sliding function. Secondly, the response of the closed loop system becomes completely insensitive to some special uncertainties. This principle goes beyond bounded model parameter uncertainties, interference and non-linearity. In a practical sense, SMC allows the control of nonlinear processes that are affected by external noise and heavy model uncertainties. The most important principles of SMC are shown in the following

significant references [28, 29, 30]. Researchers have also studied the problems appearing in the practical execution of this class of techniques. [31] Interested readers can refer to the book [32] which presents a very modern overview of the most promising current line of theoretical and practical research in the domain.

Backstepping control The main idea is to divide the system into successive subsystems and to apply a recursive algorithm which will stabilize each subsystem after the other [33]. However, this method is not robust, but it is computationally fast. In order to handle disturbances, Fang et al. [34] implemented an integral backstepping control law, in which the integral term was shown to reduce steady state errors and the response time of the system greatly.

Adaptive control This method is required when the parameters that are characterizing the system contain errors or are unknown. This type of control algorithms contains a parameter adaptation law, which is enclosed in the control to track the desired trajectory of the system, even if the model of the system is not completely known. For instance, Diao et al. [35] obtained good performance even though the inertial parameters of the quadrotor and the aerodynamic coefficients were not perfectly known. This method is convenient in some cases, such as the existence of unpredictable wind [36] or pick-and-place applications with small loads.

In the next section, the main control method that will be used in the master thesis will be explained.

1.4 Model Predictive Control

1.4.1 General Idea

There exist "open-loop" methods [37] in which the control input sequence $\mathbf{u}(t)$ is designed using a model of the system and a set of constraints. However, the problem with this approach is that modeling errors and noise are not taken into consideration. So, these inputs will not necessarily generate the desired response from the system. Because of that, a *"closed-loop"* strategy is required in order to cancel out these errors. So, an approach that can be used is called *"Model Predictive Control"* (MPC). This method is also known as *"receding horizon control"* [3] because the *"prediction horizon"* (which is a finite horizon) translates forward by one time step after the current optimization problem is solved . In short, MPC is a *feedback control* algorithm which uses a model of the system to predict the future outputs of the system and it solves an optimization problem on-line in order to select an optimal control.

Basic strategy The basic strategy of MPC is the following:

- At time step k, the system model and an optimizer will be used in order to design a sequence of control inputs

$$\mathbf{u}(k|k), \mathbf{u}(k+1|k), \mathbf{u}(k+2|k), \mathbf{u}(k+3|k), ..., \mathbf{u}(k+N|k)$$

  starting from the current state $\mathbf{x}(k)$ over a prediction horizon N.

- Only the first step of the sequence of control inputs will be applied on the system.

- The processes above are then iterated for time $(k+1)$ at state $\mathbf{x}(k+1)$.

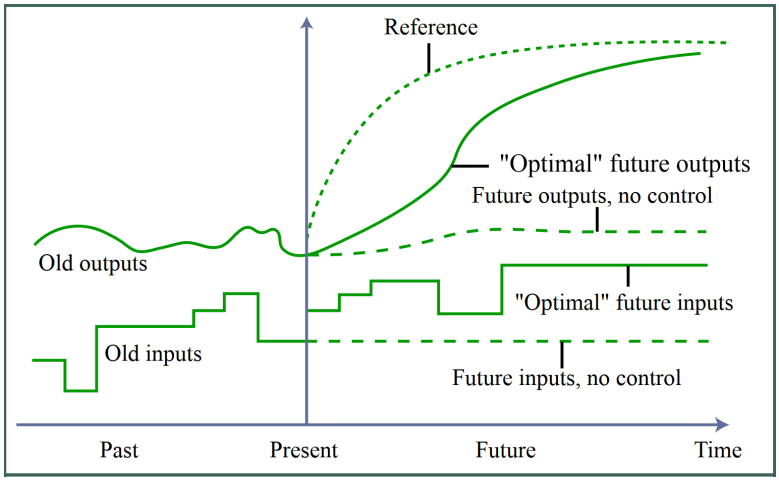


Figure 1.3: Basic idea of MPC [3].

It should be noted that the control algorithm of MPC is based on numerically solving an optimization problem at each time step. And, it is a constrained optimization in general.

Advantages and drawbacks of MPC There are several advantages when using MPC:

- MPC is able to control multi-input multi-output (MIMO) systems which might have interactions between their inputs and outputs.

- MPC explicitly accounts for the constraints that are imposed on the system. So, it does not just design a controller to keep the system away from the constraints.

- MPC can easily handle nonlinear dynamics and time-varying plant dynamics, because the controller is explicitly a function of the model of the system which can be modified in real-time.

The main drawback of MPC is that it usually requires a powerful and fast processor with a large memory in order to properly solve the problem at hand. This is because MPC solves an optimization problem at each time step.
There have been many commercial applications of MPC starting from the early 1970s in the process industry. The table 1.1 below shows the different companies that have used MPC in different industries in 2014.

Table 1.1: The number of MPC applications in different industries in 2014 [15].

Application	Aspen	Honeywell	Adersa	CCI	Pavilion	Total
Refining	950	300	290	-	15	1555
Chemicals	437	55	12	21	25	550
Food	-	-	48	-	14	62
Pulp paper	21	39	-	-	3	63
Gas and air	11	13	-	24	-	48
Polymer	5	-	-	-	22	27
Utilities	7	9	-	6	-	22
Other	39	-	51	6	-	96
Total	1470	416	401	57	79	2423

However, as computational power has increased throughout the years thanks to the advancement of technology, there has been a renewed curiosity in applying this control approach to systems with fast dynamics for which the computational complexity is significantly larger when compared to the industrial applications for which computational complexity was not a concern (since MPC was applied on systems with slow dynamics in that case).

1.4.2 Design Parameters

The different parameters that can be tuned in a MPC controller ([38, 39, 40]) are the following:

- The sample time T_s.

- The prediction horizon N.

- The control horizon m.

- The constraints.

- The weights.

Choosing the proper values for the parameters stated above is very important since they affect the performance of the controller and the computational complexity of the MPC algorithm.

Sample Time T_s The sample time determines the rate at which the controller executes the control algorithm.

- If T_s is too large, then when disturbances occur, the controller will be unable to react to the disturbances quickly enough.
- If T_s is too small, the controller will have much faster reaction times to disturbances and set-point changes. However, this comes at the cost of an excessive computational load.

In order to find reasonable balance between controller performance and computational effort, the general recommendation is to have between 10% and 25% of the minimum desired closed-loop response time .

Prediction horizon N The prediction horizon N (sometimes referred to with the variable p, however, N will be used instead for the remaining of this report) is the number of predicted future time steps of the system. It shows how far the controller predicts into the future. Thus, a prediction horizon must be chosen in such a way that it covers the significant dynamics of the system. However, it should be noted that a large prediction horizon should not be selected, since unexpected phenomenons could occur that may affect the dynamics of the system, which will cause a waste of computational power. The general recommendation is to increase N until additional increases will have little impact on the performance. The maximum N is the number of control intervals needed for the open-loop step response of the system to become infinite. However, having $N > 50$ is hardly ever required unless T_s is very small.

Control horizon m The control horizon is the set of future actions which will lead to the predicted plant output. It represents the number of control moves until time step m. After the first m steps, the remaining inputs will remain constant as shown in figure 1.3 for the *"Optimal"* future inputs. Moreover, each control input element in the control horizon is a free variable that will be computed by the optimizer. Thus, the smaller the control horizon, the fewer the computations. However, setting $m = 1$ may not give the best possible output for the system. And, similarly to the prediction horizon, if the control horizon is increased, this will lead to better predictions at the cost of increasing the computational complexity. Moreover, the general recommendation for the control horizon is to keep it much smaller than the prediction horizon, because:

- A smaller control horizon m will lead to less variables to optimize in the QP that will be solved at each control input interval. This will encourage quicker computation times.
- If delays are considered, then having $m < N$ is mandatory. Otherwise, some control input elements within the control horizon may not have any effect on the plant outputs before the prediction horizon ends, which will lead to a QP Hessian matrix which is singular.
- Having a small value for m encourages having an internally stable controller. However, this is not guaranteed.

Constraints A model predictive controller can integrate constraints on the inputs, the rate of change of the inputs and the outputs. In addition, the constraints can be *"soft"* constraints or *"hard"* constraints. *"Soft"* constraints represent constraints that are allowed to be violated if the MPC deems it to be necessary in order to find a solution for the control problem at hand. In addition, it should be noted that *hard* constraints are a set of constraints that cannot be violated by the MPC. However, applying hard constraints on both the inputs and outputs at the same time may cause conflicts to occur between the constraints, which may lead to an

unfeasible solution. Moreover, the general recommendation is to use soft constraints on the outputs and to avoid having hard constraints on both the inputs together with the rate of change of the inputs.

Weights Model Predictive Control could have many goals. A possible goal could be to have the outputs converge to their set-points as fast as possible. Another goal could be to have smooth control inputs in order to avoid aggressive control maneuvers. So, in order to achieve a balanced performance between these two competing goals, the input rates and the outputs can be weighted relatively to each other. It is also possible to adjust relative weights within the input rates and the outputs.

1.4.3 Basic formulation of a MPC problem

For a given set of plant dynamics which is first assumed to be linear:

$$\begin{cases} \mathbf{x}(k+1) = A\mathbf{x}(k) + B\mathbf{u}(k) \\ \mathbf{z}(k) = C\mathbf{x}(k) \end{cases} \tag{1.7}$$

And a cost function as follows:

$$J = \sum_{j=0}^{N} \{\|\mathbf{z}(k+j|k)\|_{R_{zz}} + \|\mathbf{u}(k+j|k)\|_{R_{uu}}\} + F\mathbf{x}(k+N|k)) \tag{1.8}$$

With:

- $\|\mathbf{z}(k+j|k)\|_{R_{zz}}$ is the weighted L^2-norm of the state, so it is expressed as follows:

$$\|\mathbf{z}(k+j|k)\|_{R_{zz}} = \mathbf{z}(k+j|k)^\intercal R_{zz}\mathbf{z}(k+j|k)$$

- $\|\mathbf{u}(k+j|k)\|_{R_{uu}}$ is the weighted L^2-norm of the control input sequence, so it is expressed as follows:

$$\|\mathbf{z}(k+j|k)\|_{R_{zz}} = \mathbf{z}(k+j|k)^\intercal R_{zz}\mathbf{z}(k+j|k)$$

- $F\mathbf{x}(k+N|k))$ is a terminal cost function which can also have weights if required.

It should be noted that if $N \to \infty$, and there are no additional constraints on $\mathbf{z}$ and/or $\mathbf{u}$, then the problem falls back to the discrete LQR problem [41]. Moreover, when limits are added on $\mathbf{x}$ and/or $\mathbf{u}$, then the general solution cannot be found anymore in analytical form, and it has to be solved numerically.
Furthermore, solving for a very long sequence of control inputs is irrelevant if the model used for the computations is expected to be erroneous or there are disturbances applied on the system, since only the first element of the optimized control sequence will be implemented. This is why MPC is generally designed by using a relatively small N.

Typical problem statement For a finite N and with $F = 0$ the problem can be expressed as follows:

$$\begin{aligned} \min_{u} \quad & J = \sum_{j=0}^{N} \{\|\mathbf{z}(k+j|k)\|_{R_{zz}} + \|\mathbf{u}(k+j|k)\|_{R_{uu}}\} \\ \text{s.t.} \quad & \mathbf{x}(k+j+1|k) = A\mathbf{x}(k+j|k) + B\mathbf{u}(k+j|k), \\ & \mathbf{x}(k|k) \equiv \mathbf{x}(k), \\ & \mathbf{z}(k) = C\mathbf{x}(k+j|k), \\ & |\mathbf{u}(k+j|k)| \leq u_{max} \end{aligned} \tag{1.9}$$

Interested readers can refer to appendix A where it is demonstrated how equation (1.9) can be converted to the form of a quadratic program.

1.4.4 Different MPC Methods

The MPC methods that are mainly used are the following:

- Linear time-invariant MPC.
- Adaptive MPC.
- Gain-Scheduled MPC.
- Non-linear MPC.

The method to be used is chosen based on the complexity of the system at hand and on the required goals to be reached.

In case of linear systems The method to be used is called linear time-invariant MPC. The constraints must be linear and the cost function must be quadratic. These characteristics will result in a convex optimization problem [42] which has a global optimum. An example of a convex function is shown in figure 1.4 below.

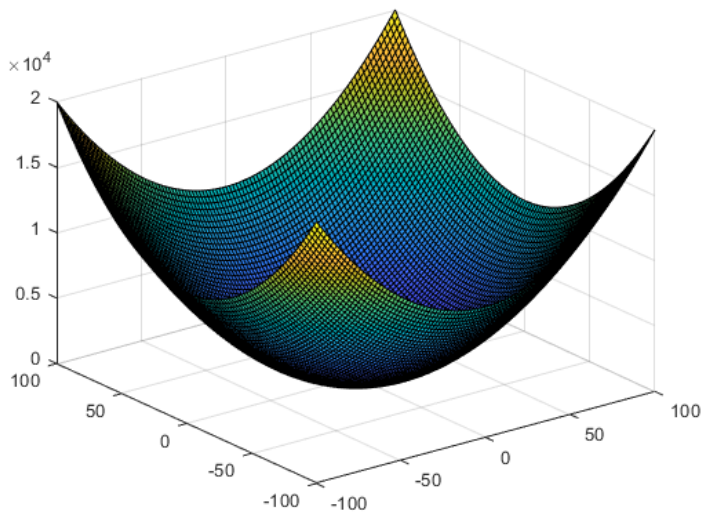


Figure 1.4: Representation of a convex function.

In case of a nonlinear systems If the system is linearizable, then both Adaptive MPC and Gain-Scheduled MPC can be used in this case. But the constraints are still linear and the cost function is still quadratic. In this case, the nonlinear function can be linearized around an operating point, which will result in a linear function that approximates the nonlinear system well near the operating point. However, it should be noted that the linear function will not work well outside the operating region. This is why it is interesting to find multiple linearized models, with each model representing the nonlinear function well around it's operating point.

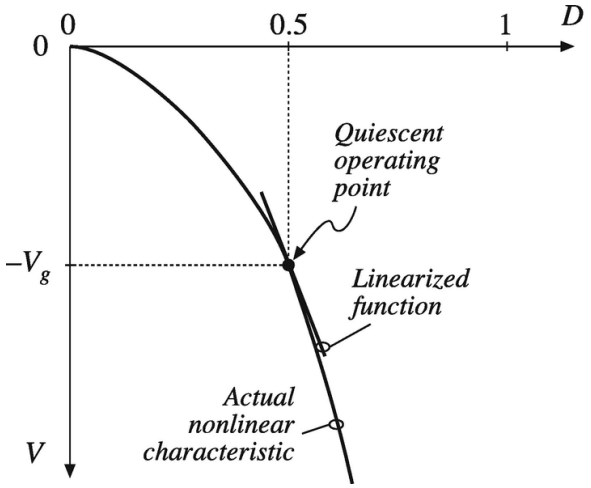


Figure 1.5: Example of the linearization of a nonlinear function around an operating point [4].

Adaptive MPC In this case, a linear model is computed on-line as the operating conditions change. And, the internal plant model used by the MPC is updated with the corresponding linear model at each time step. Moreover, it should be noted that the optimization problem remains the same across different operating points. In other words, the number of elements in the state and the constraints remain the same across different operating conditions [43].

Gain-Scheduled MPC In this case, linearization of the nonlinear model is done offline at the operating points of interest. Then, a linear MPC controller is designed for each operating point. However, it should be noted that unlike Adaptive MPC, each controller is now independent from the other. In other words, the number of elements in the state and the constraints are different across different operating conditions. It should also be noted that for this method, an algorithm must be designed to switch between the predefined MPC controllers for different operating conditions. Moreover, Gain-Scheduled MPC also uses more memory than Adaptive MPC [44].

Nonlinear MPC If the system is nonlinear and cannot be linearized, and both the constraints and the cost function are also nonlinear, then nonlinear MPC can be used in that case. It is the most powerful method of all the different methods mentioned earlier, since it uses the most accurate representation of the plant. Thus, predictions are more accurate. However, nonlinear MPC is the most difficult method to solve in real-time. Because, in that case the problem becomes a non-convex optimization problem. Thus, the cost function may have many local minima, and finding the global minimum may be hard in that case. An example of a non-convex function is shown in figure 1.6 below.

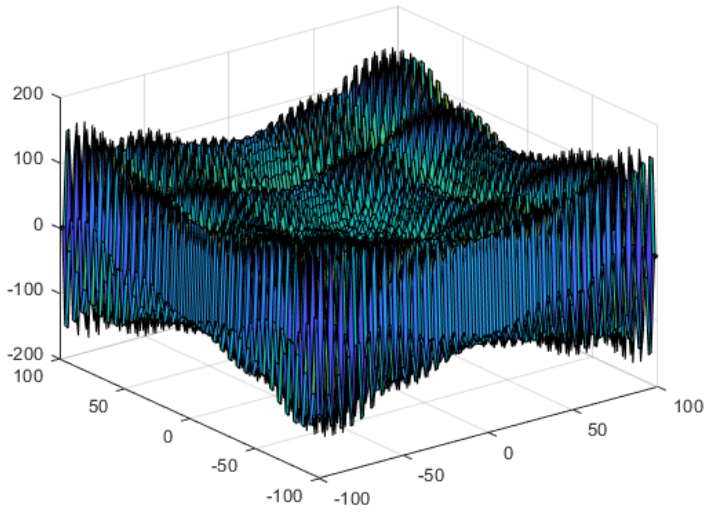


Figure 1.6: Representation of a non-convex function.

1.4.5 Strategies to Improve Computational Time

As demonstrated in (A.2), a MPC problem is formulated as a QP problem that tries to minimize a quadratic cost function in general. Moreover, MPC computations become more complex as the number of state elements, the number of constraints and the MPC parameters increase. Moreover, as stated in section 1.4.1, if the MPC controller is running on applications with slow dynamics, then computational complexity is not a concern. However, if MPC is running on applications with fast dynamics, then the computational complexity becomes crucial. In addition, it is important to note that the matrices that are stored in the processor of the system for MPC computations grows with the increasing number of optimization variables. Thus, this could cause a memory problem. So, to reduce the complexity and computational time of MPC, the following methods can be used:

Order reduction techniques They are used to discard states that do not contribute to the dynamics of the system [45]. And, using order reduction techniques will also reduce the memory usage of the controller.

Explicit MPC Instead of solving the optimization problem online for the current state, *Explicit* MPC solves it offline for each value of the state $\mathbf{x}$ within a given range [46]. Thus, for each state value within a given range, the Explicit MPC pre-computes the optimal solution. And, this solution consists of linear functions that are piece-wise affine and continuous is $\mathbf{x}$. An example of Explicit MPC applied on a one-dimensional system is shown in figure 1.7 below.
As can be observed from figure 1.7, the constraints cut the solution space into different regions. And, each region maps into a unique solution. Another example of an Explicit MPC applied on a two-dimensional system is shown in figure 1.8 below.

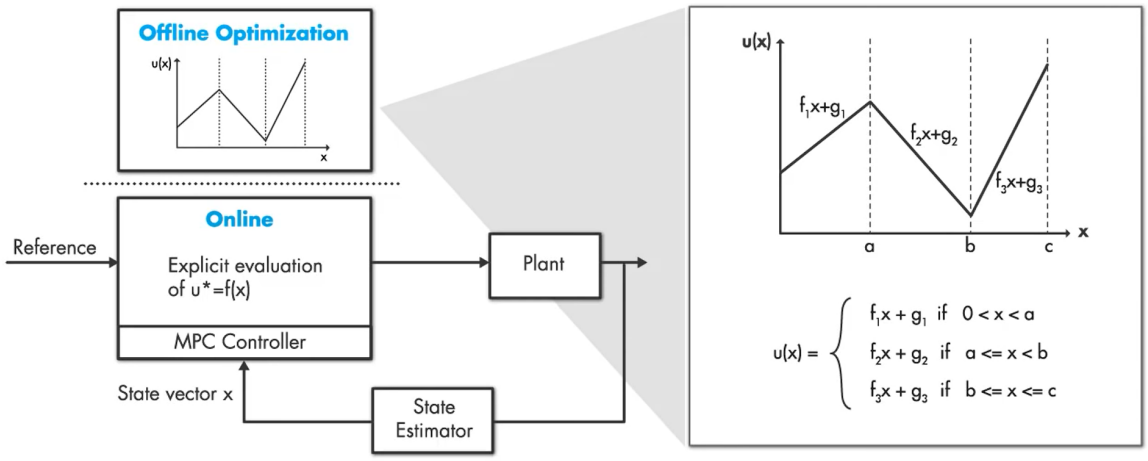


Figure 1.7: Example of an Explicit MPC controller applied on a one-dimensional system [5].

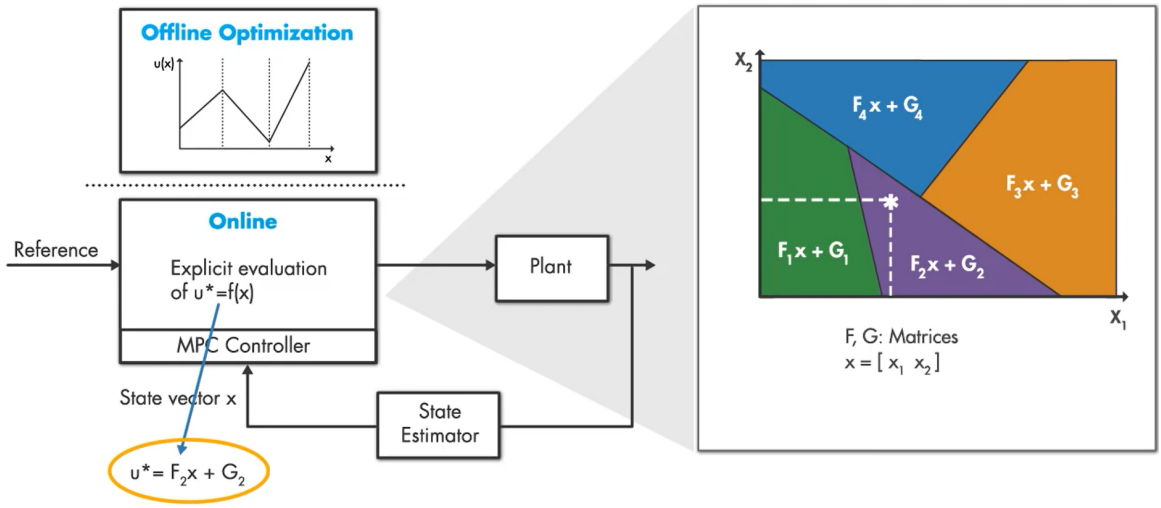


Figure 1.8: Example of an Explicit MPC controller applied on a two-dimensional system [5].

As can be observed from figure 1.8, the Explicit MPC finds the region that the current state lies in and evaluates the linear function that creates the current control input.
Thus, it can be concluded that if the iterative optimization process is reduced to linear function evaluations, then this will greatly simplify the online computations. However, if there is a large number of regions that the state can lie in, then searching for the current state region could sometimes be time consuming. Also, having many regions could cause memory problems for the processor. Thus, the number of regions can be reduced by merging some regions together. However, thee computed solution in that case is not optimal anymore [47].

Sub-optimal Solution It is another method to improve the computational time of the MPC controller. In a MPC problem, the optimal solution is very unpredictable and can drastically change between each time step. In addition, the computational time may exceed the sample time T_s. So, it is mandatory to make sure that a solution can be found within T_s and that there still exists additional time for other tasks that need to be executed [48]. A simple example is provided in figure 1.9 below where a maximum value for the iterations is determined, and it is taken as 5 for illustration purposes.
As can be seen from figure 1.9, the optimal solution is reached in 10 iterations. However, the controller will stop at iteration 5 and take the sub-optimal solution. Moreover, it is important to note that the sub-optimal solution still satisfies all the constraints of the optimization problem.

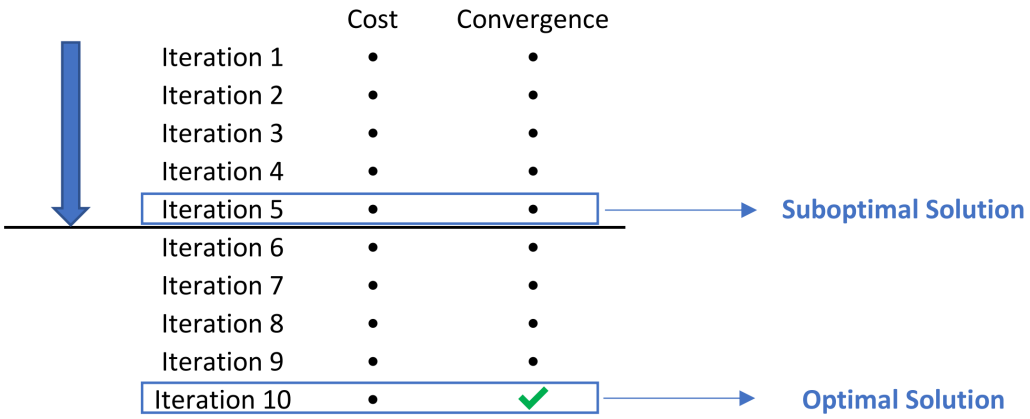


Figure 1.9: Example showing how the sub-optimal solution is found when the maximum number of iterations is set to 5.

The only question that remain is how to determine the maximum number of iterations. This can be done by testing the algorithm on the hardware directly and identifying the execution time used by each iteration. Then, the maximum number of iterations is chosen such that the total execution time does not exceed the sample time of the controller.

1.4.6 MPC applications on quadrotors

There have been numerous applications of MPC on quadrotors. The main MPC applications on drones can be separated into two different categories:

1.4.6.1 Centralized MPC

In this type of application, the MPC controller is a single structure that encapsulates both the translational dynamics and the rotational dynamics. It takes the trajectory waypoints as inputs, and directly outputs the control inputs required to control the quadrotor. This type of implementation of the MPC controller produces the least amount of errors. However, this comes at a cost of a larger computational load.

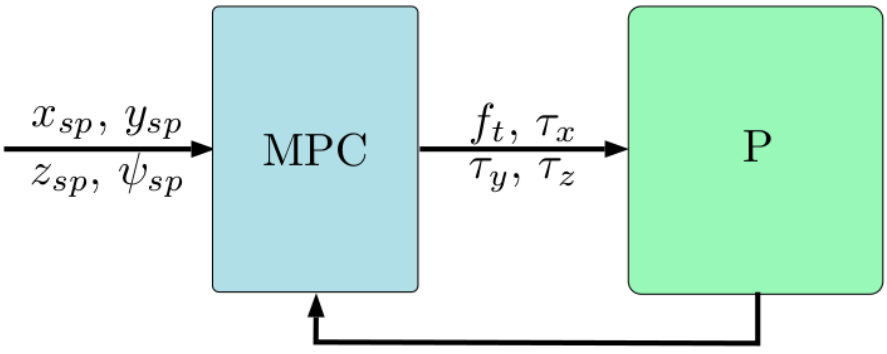


Figure 1.10: Example of a centralized MPC control loop [6].

As can be seen in figure 1.10, which represents a centralized control loop structure for a basic MPC, the right block is the plant which represents the quadrotor dynamics. The MPC block takes as input the desired trajectory, and outputs control inputs to allow the quadrotor to track the trajectory.
It should be noted that different versions of the MPC controller shown in figure 1.10 can exist. For example, another version of the MPC could be that the state variables of roll and

pitch angles can be added as outputs. This is mainly implemented in order to ensure that the quadrotor does not take extreme values in roll and pitch angles, which can potentially destabilize it, because no restrictions are being used for these angles otherwise.

1.4.6.2 Non-centralized MPC

In this type of application, the state-space of the system will be split to obtain two different controllers in a master-slave structure. This more simplified structure allows the implementation of MPC in different embedded systems, each one with less computational load when compared to the complete structure of a centralized MPC. For example, the simplified slave structure can be used to control the attitude of the quadrotor, whereas the master structure can be used to control the position and direction of the quadrotor. In addition, this splitting allows different sample times for the master and slave loops [49]. Furthermore, an implementation of non-centralized MPC could be of having both master and slave structures being MPC controllers. An example of this non-centralized MPC implementation is shown in figure 1.11 below:

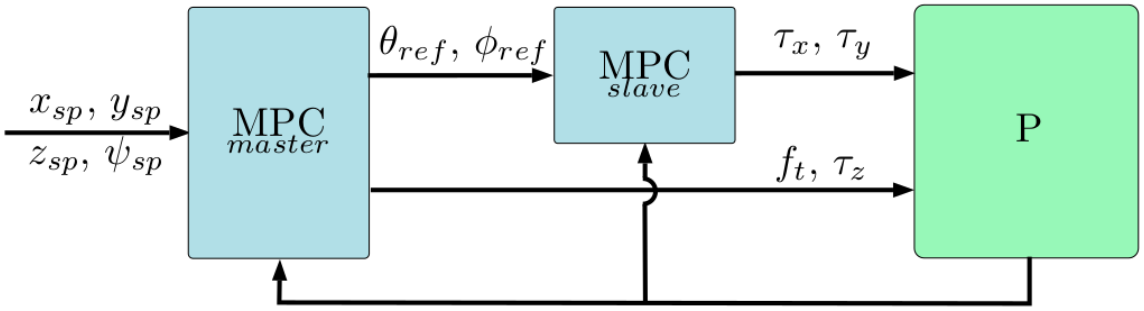


Figure 1.11: Example of a non-centralized MPC-MPC control loop [6].

Another implementation is using a MPC controller for the master structure and pairing it with the slave structure that is using a controller with a different control law, such as a PD-P controller, in order to further reduce the computational load. An example of this non-centralized MPC implementation is shown in figure 1.12 below:

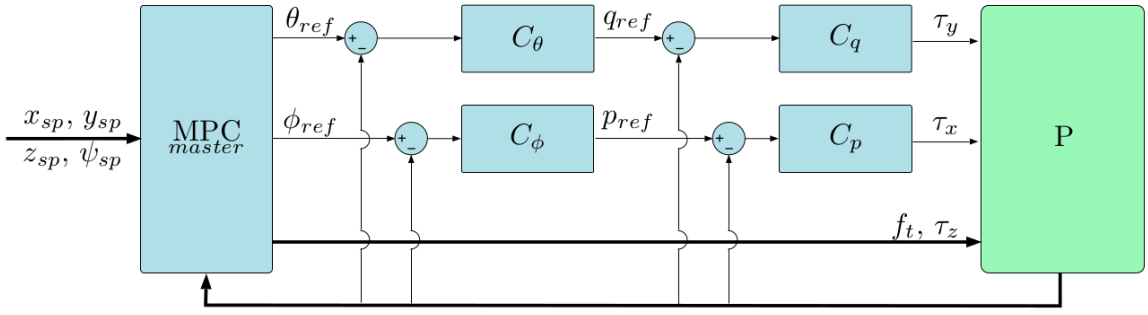


Figure 1.12: Example of a non-centralized MPC-PD-P control loop [6].

However, since the control law in this master thesis will be entirely focused on MPC controllers, and since the task will be to follow aggressive trajectories in real-time, then using a non-centralized MPC is a good method to consider, since this will result in a lower computational load on the processor. In other words, it is preferable to follow a trajectory with slightly less accuracy if it results in a faster computation time, which is crucially needed in real-time experimentation.
In the next section, the software used to implement the MPC controller is presented.

1.5 acados – Fast Embedded Optimal Control Solvers

1.5.1 Difference between acados and other Embdedded OCP Solvers

A big role in bringing MPC to real-time applications is played by the implementation of efficient embedded optimal control methods. This gave rise to software packages such as:

- `MPT`: for explicit MPC [50].
- `qpOASES`: an active-set solver for quadratic programming [51].
- `FORCES`: an interior-point solver for quadratically constrained QP and NLPs with optimal control structure [52].
- `ACADO Code Generation tool`: for tailored SQP based NMPC solvers [53].

One challenge in developing software for embedded optimal control lies in the trade-off between flexibility, memory usage and speed. Moreover, many of the software packages mentioned above are based on *automatic code generation*. This allows to have self-contained and efficient routines. However, the size of the problem and the choice of algorithms are usually fixed for one specific optimal control instance, which will result in a loss of flexibility. In addition, generated code is often hard to read, thus hard to debug. Finally, it is hard to predict if an automatic code optimization strategy is beneficial or counterproductive for some types of problems. This means that in some cases, a programmer can create faster and more efficient code.
For these reasons, `acados` does not rely on automatic code generation to perform linear algebra operations. Instead, the high-performance linear algebra package `BLASFEO` is used.
Furthermore, other embedded optimal control software use global data in order to not sacrifice execution speed and memory. The resulting code-base is difficult to understand, maintain and extend. However, `acados` organizes the code in a modular way, with formal interfaces between the different algorithmic components. This makes it easier to interchange solvers, routines and libraries needed for the embedded control algorithm.
Finally, `acados` uses `CasADi` modeling language instead of `Mathematics`, `sympy` or `MATLAB Symbolic Toolbox`. Many of these software make use of expression trees to represent mathematical functions, which could lead to a large code size, high memory usage and slow evaluation of high-order derivatives for non-trivial models. Whereas `CasADi` is based on expression graphs. This usually leads to smaller and faster code which make it more suitable for embedded applications. In addition, it is free and open-source software. In summary, `acados` offers the following:

- It contains efficient optimal control algorithms implemented in C.
- It has a modular architecture enabling rapid prototyping of solution algorithms.
- It interfaces to `C++`, `Python` and `MATLAB`.
- It uses the high-performance linear algebra package `BLASFEO`.
- It is compatible with `CasADi` expressions.
- It is deployable on a variety of embedded devices.
- It is a free and open-source software.

1.5.2 Algorithm Components for Embedded Nonlinear Optimal Control

1.5.2.1 Nonlinear Optimal Control

The nonlinear optimal control problems that are typically solved by nonlinear MPC (NMPC) have the following form in continuous-time:

$$\begin{aligned} \min_{x(\cdot),z(\cdot),u(\cdot)} \quad & \int_{t=0}^{T} l(x(t),z(t),u(t))dt + M(x(T)) \\ \text{s.t.} \quad & x_0 = \bar{x}_0 \\ & 0 = f(\dot{x}(t),x(t),z(t),u(t)) \qquad t \in [0,T], \\ & 0 \geq g(x(t),z(t),u(t)) \qquad t \in [0,T] \end{aligned} \tag{1.10}$$

With:

- l: Lagrange term or running cost.
- M: Mayer term or terminal cost.
- x: differential states with $x \in \mathbb{R}^{n_x}$.
- z: algebraic variables with $z \in \mathbb{R}^{n_z}$.
- u: control inputs with $u \in \mathbb{R}^{n_u}$.
- f: Dynamics (as implicit differential-algebraic equations).
- g: Nonlinear path constraints.

1.5.2.2 Multiple Shooting Discretization

In `acados`, the nonlinear OCP is discretized with a **multiple shooting** approach. The following elements are introduced:

- Time grid: $[t_0, t_1, \ldots, t_N]$ with $t_k < t_{k+1},\ k = 0, \ldots, N-1$
- Discretized state variables: $x_0, x_1, \ldots, x_N$
- Algebraic variables $z_0, z_1, \ldots, z_{N-1}$
- Controls $u_0, u_1, \ldots, u_{N-1} \Rightarrow$ piece-wise constant parametrization is used.

The numerical simulation routine on each time interval $[t_k, t_{k+1})$ can be written as:

$$\begin{bmatrix} x_{k+1} \\ z_k \end{bmatrix} = \phi_k(x_k, u_k), \quad k = 0, 1, \ldots, N-1$$

By performing multiple shooting discretization on the continuous-time optimal control problem formulation in equation (1.10), this will result in the formulation of the nonlinear program (NLP) as shown below.

1.5.2.3 General Form of the Resulting Nonlinear Program

The general form of the resulting nonlinear program that can be handled by `acados` is:
The resulting (general form) NLP problem becomes:

$$\begin{aligned}
\min_{\substack{x_0,\dots,x_N \\ u_0,\dots,u_{N-1} \\ z_0,\dots,z_{N-1} \\ s_0,\dots,s_N}} \quad & \sum_{k=0}^{N-1} l_k(x_k, u_k, z_k) + M(x_N) + \sum_{k=0}^{N} \rho_k(s_k) \\
\text{s.t.} \quad & \begin{bmatrix} x_{k+1} \\ z_k \end{bmatrix} = \phi_k(x_k, u_k) && k = 0, 1, \dots, N-1, \\
& 0 \geq g_k(x_k, z_k, u_k) - J_{s,k} s_k && k = 0, 1, \dots, N-1, \\
& 0 \geq g_N(x_N) - J_{s,N} s_N, \\
& 0 \leq s_k && k = 0, 1, \dots, N-1.
\end{aligned} \tag{1.11}$$

It should be noted that that (1.11) includes slack variables s_k which can be reformulated as control inputs u_k. This allows the use of *soft* constraints. But, the slack variables s_k cannot be included in the dynamics of the system. And, can only enter the cost linearly and quadratically, using the following function:

$$\rho_k(s_k) = \sum_{i=1}^{n_{s_k}} \alpha_k^i s_k^i + \beta_k^i s_k^{i^2} \tag{1.12}$$

with $\alpha_k^i \in \mathbb{R}, \beta_k^i > 0$. Slack variables s_k can also be used to model piece-wise quadratic, and possibly asymmetric cost functions.
In addition, `acados` is capable of exploiting the structure of the widely used linear and nonlinear least-squares functions, and can also handle general nonlinear functions.
Moreover, within the constraint function g_k in (1.11), `acados` is able to consider:

- Simple bounds on x_k and u_k.
- Linear constraints on x_k and u_k.
- Nonlinear constraints on x_k and u_k.

So, g_k could be formulated as follows:

$$g_k(x_k, z_k, u_k) = \begin{bmatrix} J_{bx,k} x_k \\ J_{bu,k} u_k \\ C_{x,k} x_k + C_{u,k} u_k \\ g_{nonl}(x_k, z_k, u_k) \end{bmatrix} \tag{1.13}$$

where $J_{bx,k}$ and $J_{bu,k}$ are selection matrices that determine which components of x_k and u_k have simple bounds.
The NLP in (1.11) can be solved by any general-purpose NLP solver, like `IPOPT` [54]. But, `acados` will use embedded optimal control methods to solve the NLP which are better suited in a real-time setting.
Interested readers can refer to appendix B for a more detailed explanation on sequential quadratic programming (SQP) and real-time iterations using `acados`.

1.5.2.4 Python Interface Overview

Before testing the MPC controller in ROS2 and Gazebo, the MPC controller was first designed using the acados Python interface. And, simple simulations were performed by making use of a Runge-Kutta of order 4 integrator that is already implemented in acados.

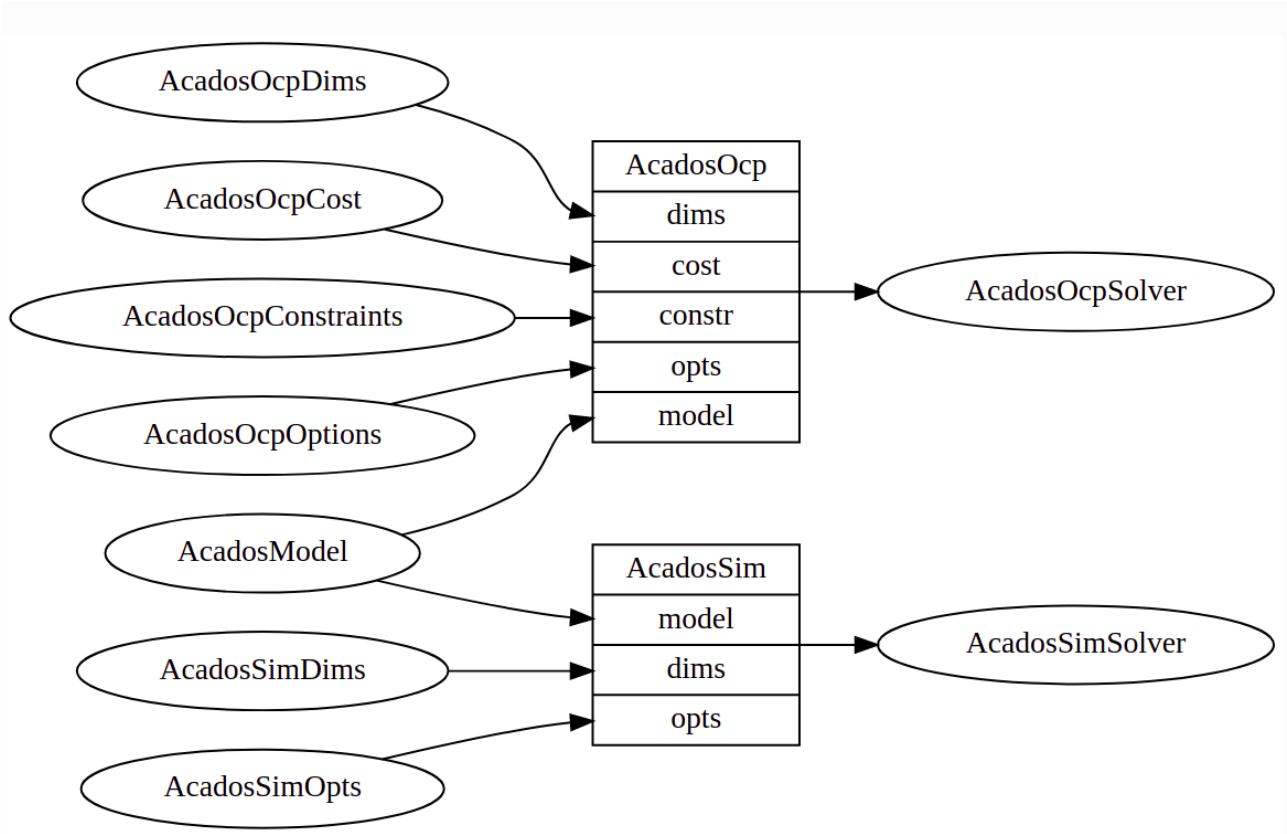


Figure 1.13: Overview of the Python API classes in acados [7]

As can be seen from figure 1.13, there exists two main classes in the acados python interface. Namely, `AcadosOcpSolver` and `AcadosSimSolver`.
In order to create the optimal control problem solver class (`AcadosOcpSolver`), the dynamic model of the system, along with the dimensions of the OCP, the cost function, the constraints and the solver options must be clearly defined. Then, the OCP is solved, which results in the control inputs. Then, a simulator class can be used in order to apply the resulting control inputs on the dynamics and compute the new state of the system. So, in order to create the `AcadosSimSolver` class, the dynamic model of the system must also be used, along with the dimensions and the solver options.

1.6 Flip Maneuvers with Quadrotors

The aim of this section is to explain the physics of the flip maneuver, in addition to the major difficulties that relate to the implementation of flipping maneuvers with quadrotors. Furthermore, a review of the existing limitations that the current approaches have are shown and the existing scientific literature is presented. The content of this chapter were thoroughly covered by the previous master students Marco Orsingher [55] and Antonio Marino [56] for which their papers were used as references for this section.

1.6.1 Physics of a quadrotor flip

In order to perform the tedious process of multi-flip maneuvers with a quadrotor, high angular velocity and large angle attitude control are needed. More conventionally, the multi-flip trajectory can be expressed as a rotation of $2n\pi$ about a flipping axis **a**, with n the number of desired flips to be carried out by the quadrotor. Physically, the maneuver can be divided into 4 different phases, which are shown in figure 1.14.

Climb phase The quadrotor must accelerate in a vertical direction as much as possible. This will ensure that the required height to perform the multi-flip maneuver is achieved. In fact, while performing the multi-flip maneuver, the thrust forces cannot balance out the weight of the drone. This will cause the drone to start falling. The duration of this phase is based on the number of flips that must be performed by the quadrotor. This number will be chosen by the user.

Multi-flip phase This stage begins the moment the vehicle starts rotating and continues until the flipping angle reaches the chosen value of $2n\pi$. In this phase, the quadrotor hits the maximum altitude because of the required high angular velocity and starts falling because of the gravity effect. Generally, all the focus is on following the desired closed-loop attitude, and the vertical position is not controlled. the time spent in this phase depends on the desired number of flips n and on the maximum angular velocity achievable by the quadrotor.

Descent phase In order for the quadrotor not to crash into the ground, the main idea is to set the maximum thrust reference to the actuation system. This will balance out the gravity force. The higher the number of required flips, the faster the quadrotor will fall due to gravity effects. Thus, this phase ends when the descent is fully balanced out. The user may also decide what the duration of this phase is if the physics of the problem allow a slower descent.

Re-stabilization phase In this phase, the altitude of the quadrotor is adjusted to a desired value. Moreover, this phase ends when a new control signal is provided by the user.

1.6.2 Control Approaches for Multi-Flip Maneuvers

The research community has tackled the multi-flip problem numerous times on quadrotors and different methods have been suggested to achieve up to triple flips both indoor and outdoor. The existing research can be stored into three different methods: hybrid systems theory, open-loop iterative learning and closed-loop attitude control. This section gives an inclusive overview and suitable references on the state of the art in flipping maneuvers with quadrotors.

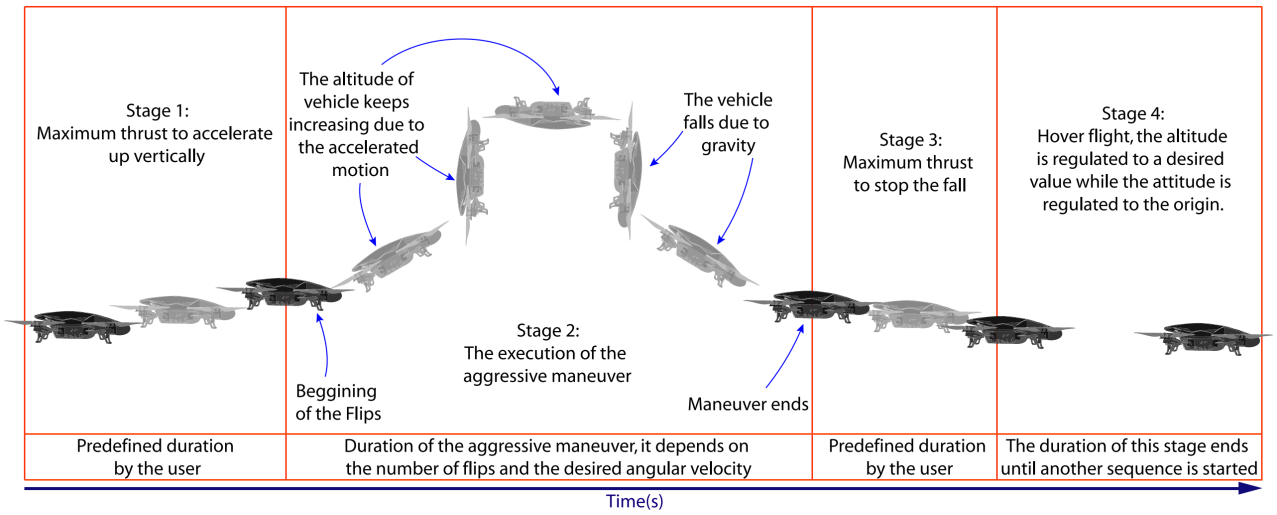


Figure 1.14: Representation of the different phases needed for performing multi-flip maneuvers[8].

1.6.2.1 Hybrid Systems Theory

A convenient approach for dealing with complex systems is to divide them into a set of individual modes. Each mode will have its own continuous dynamics equations. Then, they will be analyzed by using hybrid systems theory. Gillula et al. ([9, 57]) suggested using this strategy for quadrotor multi-flips. The main idea is to breakdown the flipping maneuver into a sequence of individual phases and then use an appropriate technique from hybrid systems theory to generate provable transitions between the modes. By doing this procedure, the maneuver will be able to be performed safely. The separate modes that are considered in this work are shown in figure 1.15, which should be right from right to left. When compared to the different phases shown in section 1.6.1, the climb phase is called the *"impulse"*, the multi-flip phase is the *"drift"* and the descent and the re-stabilization phases are merged together and are called *"recovery"*.

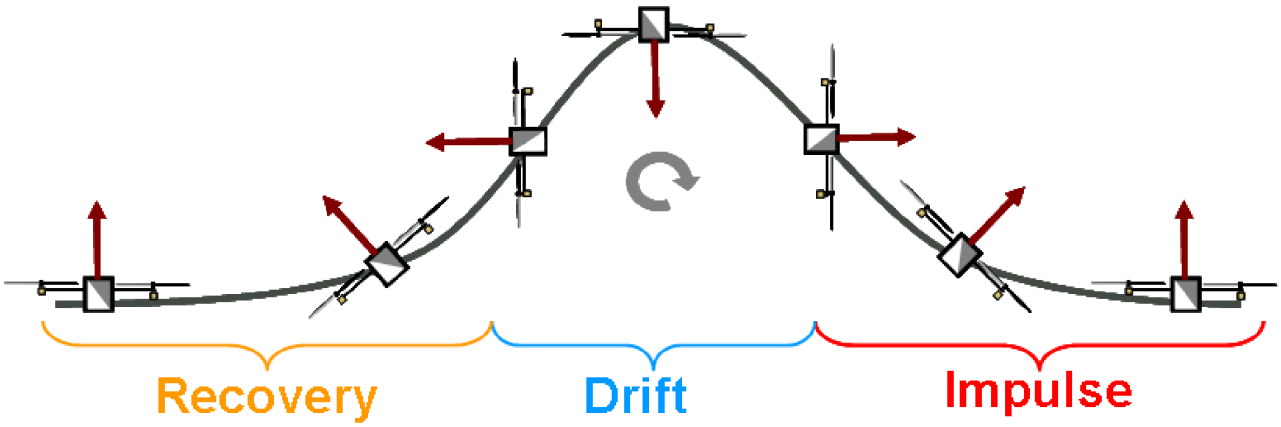


Figure 1.15: Representation of the individual phases during a back-flip (from right to left)[9]

The tool used for generating feasible transitions between the individual modes is called the *"theory of reachable sets"*. By considering a system characterized by the following dynamics:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{d}) \tag{1.14}$$

with $\mathbf{x}$ the state of the system, $\mathbf{u} \in U$ the control input of the system and $\mathbf{d} \in D$ a bounded noise applied on the system. In that case, two different types of reachable sets are established:

1. A *capture set* which represents all the states for which, for any possible value of the noise $\mathbf{d}$, the control input $\mathbf{u}$ is still capable of driving the state into a desired state region in a finite time horizon t.

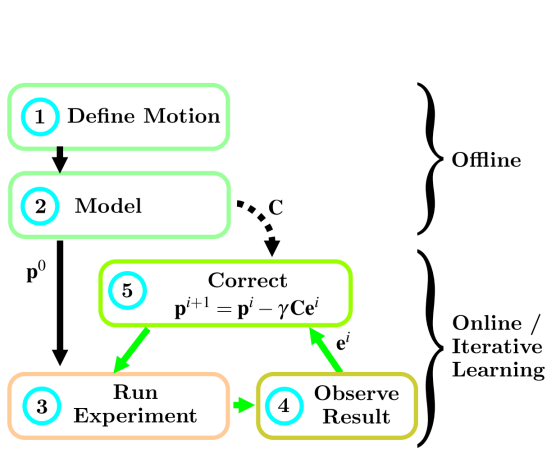
2. An *avoid set* which represents all the states for which, for any possible value of the control input $\mathbf{u}$, the noise $\mathbf{d}$ will drive the system into an undesired set. Thus, the roles of $\mathbf{u}$ and $\mathbf{d}$ are reversed when compared to the capture set.

Generating quadrotor maneuvers using reachable sets is achievable by using a backward approach. The main idea consists of starting from the final desired set, the dynamics of the n-th mode can be run in reverse to build a capture set for that phase. This capture set becomes the target region for the $(n-1)$-th node and the procedure will be repeated between every 2 modes. By applying this procedure, feasible transitions between a chain of individual modes can be built. From a mathematical perspective, the problem is formulated as a differential battle between the control and the noise. In fact, the aim of the control input $\mathbf{u}$ is to keep the state of the system away from the region of undesired sets which the noise $\mathbf{d}$ is attempting to drive the state of system into (*avoid set*) and also to reach a desired set. Whereas the noise attempts to drive the state of the system out of it (*capture set*). This method has been proven to be successful with a 1.1 kg quadrotor which was able to perform a back-flip outdoors. However, the main disadvantage of this method is that it necessitates a long procedure of generating such capture sets and avoid sets.

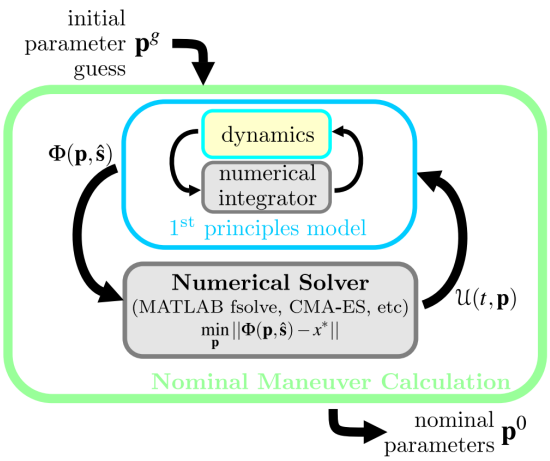
1.6.2.2 Open-Loop Iterative Learning

One of the main issues that is related to performing aggressive maneuvers with quadrotors is that having an exact physical modeling of the quadrotor during an intense flight procedure is very complicated. In addition, generating reference trajectories for aerobatic flight is not an easy operation. Beginning from these factors, Lupashin et al. ([58, 59, 10]) designed a simple open-loop iterative learning method to perform multi-flip maneuvers without performing any aerodynamic modeling nor trajectory planning. The workflow of this method is shown in figure 1.16 on the left.
The first step which is needed is to determine the desired motion in a parameterized manner, so that it is possible to optimize vector $\mathbf{p}$ over such parameters. $\mathbf{p}$ contains the time and the accelerations needed during each phase of the flip maneuver. The second procedure which is needed to initialize the iterative learning algorithm is to calculate the nominal motion and to build the correction matrix. Starting from a simplified model of the quadrotor and a coarse initial guess $\mathbf{p}^g$, then a numerical solver is utilized to calculate the nominal parameters $\mathbf{p}^0$ in order to execute the multi-flip maneuver. In fact, assuming that the desired final state $\mathbf{x}^*$ can be attained with the control input $\mathbf{u}(\mathbf{p}, t)$ and by referring to $\mathbf{\Phi}(\mathbf{p}, \hat{\mathbf{s}})$ as the final state of the nominal system after the maneuver is executed.(with the vector $\hat{\mathbf{s}}$ representing physical constants), then the nominal set of parameters $\mathbf{p}^0$ is the solution of the following optimization problem, as shown in figure 1.16:

$$\mathbf{p}^0 = \arg\min \|\mathbf{\Phi}(\mathbf{p}, \hat{\mathbf{s}}) - \mathbf{x}^*\| \tag{1.15}$$



(a) The global work-flow of the algorithm



(b) The global work-flow of the algorithm

Figure 1.16: The open-loop iterative learning workflow for performing multi-flips with quadrotors[10].

Moreover, by denoting $\mathbf{\Phi}(\mathbf{p}^0, \hat{\mathbf{s}})$ as the final state of the nominal model obtained using the nominal set of parameters calculated earlier, then the correction matrix for the iterative learning algorithm is expressed as follows:

$$\boldsymbol{C} = \left(\frac{\partial \mathbf{\Phi}(\mathbf{p}^0, \hat{\mathbf{s}})}{\partial \mathbf{p}} \right)^{-1} \tag{1.16}$$

In the end, steps 3 to 5 in figure 1.16a will then run online in an iterative manner until a proposed convergence criterion is satisfied. For $i = 0 \dots n$, the motion is carried out on the quadrotor itself while using the current set of parameters $\mathbf{p}^i$ and the final measured error $\mathbf{e}^i$. Subsequently, the parameters are updated according to the following law:

$$\mathbf{p}^{i+1} = \mathbf{p}^i - \gamma \mathbf{C} \mathbf{e}^i \tag{1.17}$$

with γ being a step size which is defined by the user. This method is very simple does not require complex computations. But, it is by definition an open-loop method. Thus, there are no guarantees that the maneuver will be performed successfully. In fact, a triple flip maneuver has been done during experiments with this method, however the repeatability of this result is a big problem. In addition, "creating a parameterized input trajectory formulation that results in a robust learning strategy remains a difficult and tedious task" [10].

1.6.2.3 Closed-Loop Attitude Control

The most known strategy for obtaining great results in flipping maneuvers of quadrotors is to use closed-loop control just for the attitude part of the platform, thus the position of the center of mass during the flipping maneuver is ignored. Many research works can be recognized under this context, and all of them follow the same ideology: a proper reference on attitude or angular velocity is planned and then tracked in closed-loop with a nonlinear control law. Each paper is distinguished from the others based on the strategy that is chosen to build the control law: attractive ellipsoid method [8], geometric control [60], energy-based control[61] , direct Lyapunov method [62]. Particularly, Chen et al. ([11], [63] ,[64]) generated interesting series of papers within this framework that should be mentioned because they applied the method on the same physical platform of this thesis (Crazyflie 2.0); moreover, significant theoretical contributions have been provided in these papers to greatly comprehend the issue of quadrotor flipping. Particularly, three elements have been investigated:

1. A new depiction of the flipping angle, called ϕ_g, such that a flip around a random axis $\mathbf{a}$ through the center of mass of the quadrotor can be determined. This has permitted to execute multiple flips about the body-fixed axis $\mathbf{b}_1$, and another general axis, rotated by 45° on the ($\mathbf{b}_1$,$\mathbf{b}_2$) plane.

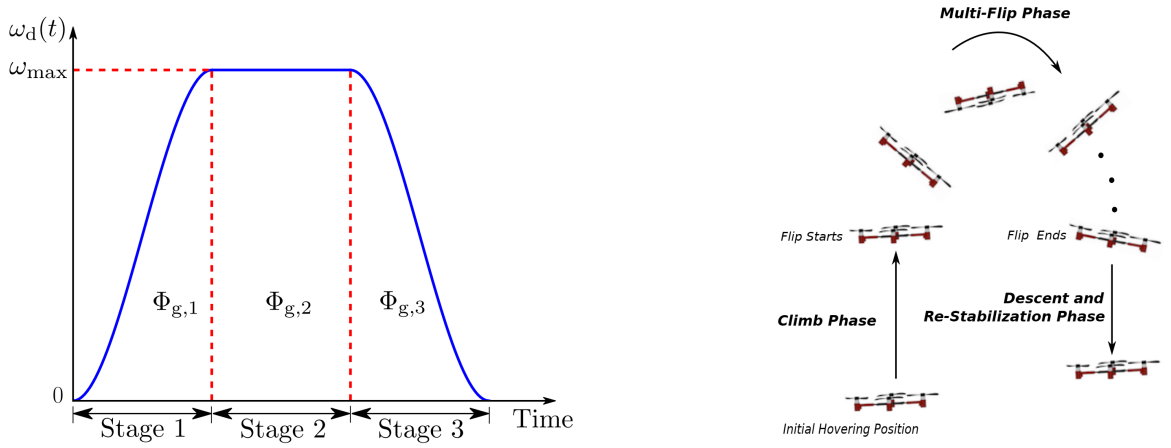
2. A meticulous angular speed planning approach using cubic splines to produce the reference signal $\boldsymbol{\omega}_d$ in such a way that the dynamics of actuation are respected by ensuring that w_{max} is never surpassed. Particularly, the reference is made out of three distinct stages, corresponding to the physical stages presented in figure 1.17a. In fact, throughout the *climb phase* a rising-speed section $\boldsymbol{\omega}_{d,1}$ is needed to reach $\boldsymbol{\omega}_{max}$, then a stage of constant-speed at $\boldsymbol{\omega}_{d,2} = \boldsymbol{\omega}_{max}$ is held while the quadrotor is flipping (*multi-flip phase*), and in the end, the angular velocity $\boldsymbol{\omega}_{d,3}$ is reduced to zero within the *descent* and *re-stabilization phase*. The use of cubic functions ensures that the angular accelerations is kept continuous during all the process. Other works, such as [8], suggested equivalent shape, however linear functions were used instead of splines. Considering that the time derivative of a linear function is just a constant, this resulted in undesirable jumps in the angular reference. Considering the desired number of rotations accumulated with each stage $\Phi_{g,i}1$, the time of each stage is changed based on the current flipping angle ϕ_g:

$$\boldsymbol{\omega}_d(t) = \begin{cases} \boldsymbol{\omega}_{d,1} & \text{if} \quad \phi_g \leq \Phi_{g,1} \\ \boldsymbol{\omega}_{d,2} & \text{if} \quad \Phi_{g,1} < \phi_g \leq \Phi_{g,1} + \Phi_{g,2} \\ \boldsymbol{\omega}_{d,3} & \text{if} \quad \Phi_{g,1} + \Phi_{g,2} < \phi_g \leq \Phi_{g,1} + \Phi_{g,2} + \Phi_{g,3} \end{cases} \tag{1.18}$$

where:

$$\Phi_{g,1} + \Phi_{g,2} + \Phi_{g,3} = 2n\pi \tag{1.19}$$

in order to carry-out the flip.



(a) The reference of the angular velocity.

(b) The stages of the multi-flip maneuver.

Figure 1.17: The reference of the flipping speed and the different flipping phases. [11]

3. Three distinct control methods to track the reference of the angular speed, which are then compared in terms of the root mean square error. First, a linear time-invariant (LTI) has been proposed [11] as follows. Provided the error of the angular velocity $\mathbf{e}_\omega = \boldsymbol{\omega} - \omega_d\mathbf{a}$ and the error of the angular acceleration $\mathbf{e}_\alpha = F(s)\boldsymbol{\omega} - \dot{\omega}_d\mathbf{a}$, with F(s) a filter expressed as $F(s) = \lambda s(s+\lambda)^{-1}$, $\lambda > 0$, the consequence control law is:

$$\boldsymbol{\tau} = -\boldsymbol{K}_{\omega}\mathbf{e}_{\omega} - \boldsymbol{K}_{\alpha}\mathbf{e}_{\alpha} \tag{1.20}$$

In the subsequent works, a backstepping control approach [63] has been examined to solve the latency issue which is the result of the slow response of the actuators, in addition to an adaptive control law to estimate unknown parameters on the fly [64]. In the two cases, these methods achieved better performance when compared to the LTI controller. However, they were not able to successfully reduce the oscillation of the angular velocity within the descent and re-stabilization phase.

Regardless, this series of papers are by far the most successful and comprehensive work available in literature on performing multi-flip maneuvers with quadrotors.

1.6.3 Shortcomings of Current Strategies

The control approaches for multi-flip maneuvers that have been presented in section 1.3 demonstrate that the implementation of flipping maneuvers is currently addressed as a more complicated version of the classical attitude control problem. But, no attention has been given on the consequent Cartesian path of the center of mass of the quadrotor, nor on an accurate depiction of the climb phase. In fact, within the first stage, the maximum thrust is established in order to acquire sufficient height. However, none of the works presented above deals with comprehensive altitude analysis in order to give an approximation of the necessary duration for the climb phase. This is mainly done heuristically in experiments. This problem, combined with the fact that the attention is put only on attitude planning and control, results in a totally erratic trajectory for the quadrotor. Thus, it is important to be able to predict the consequent trajectory of the quadrotor during a flipping maneuver. Because, this will help to keep the quadrotor safe, particularly in complex outdoor settings.

Chapter 2

MPC Design and Simple Simulations with acados

In this chapter, the equations of motions of the planar quadrotor and the 3D quadrotor will be presented. And, an extended Kalman filter (EKF) will be designed for the planar case to test the controller in the presence of disturbance. Then, the MPC controller design will be presented. Finally, the MPC controller will by making the quadrotor follow a circular trajectory.

2.1 System Modeling

In this section, the equations of motion of both the planar quadrotor and the 3D quadrotor will be presented and explained. However, the notion of position and orientation will be explained first. The two main references used for this section are [12] and [65].

2.1.1 Notion of Position and Orientation

The configuration of a vehicle, namely the *pose*, includes the parameters that pertmits to describe the mobile frame R_M with respect to the working frame R_O. In addition, there are several methods to describe the rotation of a rigid body in space, for instance, Euler angles, Cardan angles, [66], etc. The main distinction between the Euler angles and the Cardan angles is that Cardan angles express rotations around three different axes (e.g. $x-y-z$, or $x-y''-z''$), whereas Euler angles utilize the same axis for both the first and third elemental rotations(e.g., $z-x-z$, or $z-x'-z''$)
A common representation of the pose of a vehicle in 3D Euclidean space is the following:

$$q = [\ x \quad y \quad z \quad \psi \quad \theta \quad \phi\ ]^\intercal \tag{2.1}$$

The pose can be described as the transformation from the working frame related to the environment R_O to the mobile frame R_M based on the four following procedures [12]:

- The translation $\overrightarrow{OM}$ from R_O to R_M: $(M, \overrightarrow{s_0}, \overrightarrow{n_0}, \overrightarrow{a_0})$.
- The rotation $(\psi, \overrightarrow{a_0})$ where ψ is the yaw angle from $R_O(M, \overrightarrow{s_0}, \overrightarrow{n_0}, \overrightarrow{a_0})$ to $R_1(M, \overrightarrow{s_1}, \overrightarrow{n_1}, \overrightarrow{a_0})$.
- The rotation $(\theta, \overrightarrow{n_1})$ where θ is the pitch angle from R_1 to $R_2(M, \overrightarrow{s_2}, \overrightarrow{n_1}, \overrightarrow{a_1})$.
- The rotation $(\phi, \overrightarrow{s_2})$ where ϕ is the roll angle from R_2 to $R_M(M, \overrightarrow{s_2}, \overrightarrow{n_2}, \overrightarrow{a_2}) = R_M$

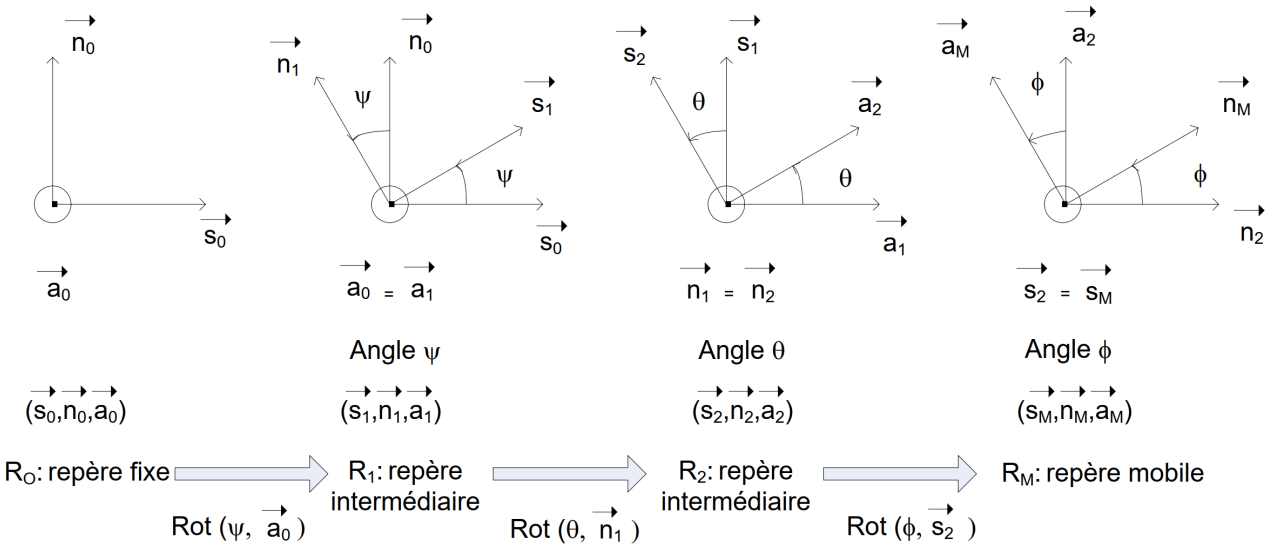


Figure 2.1: Representation of the Euler Transformations [12]

The related change of coordinates matrices are expressed as follows:

$$L = {}^{O}A_1 = Rot(\psi, \overrightarrow{a_0}) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$T = {}^{1}A_2 = Rot(\theta, \overrightarrow{n_1}) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.3}$$

$$R = {}^{2}A_M = Rot(\phi, \overrightarrow{s_2}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{2.4}$$

with L, T and R denoting rotation matrices of roll, pitch and yaw respectively. Hereafter, $Rot(\psi, \overrightarrow{a_0})$, $Rot(\theta, \overrightarrow{n_1})$ and $Rot(\phi, \overrightarrow{s_2})$ will denote the corresponding change of coordinate matrices. Thus, the transition matrix from the mobile frame to the fixed frame is then expressed as follows:

$$\begin{aligned} LTR = {}^{O}A_M &= Rot(\psi, \overrightarrow{a_0})Rot(\theta, \overrightarrow{n_1})Rot(\phi, \overrightarrow{s_2}) \\ &= \begin{bmatrix} \cos\psi\cos\theta & -\sin\psi\cos\phi + \cos\psi\sin\theta\sin\phi & \sin\psi\sin\theta + \cos\psi\sin\theta\cos\phi \\ \cos\theta\sin\psi & \cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi & -\sin\phi\cos\psi + \sin\psi\sin\theta\cos\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \end{aligned} \tag{2.5}$$

2.1.2 Inputs

The principal forces and torques that the quadrotor is subject to, are generated by the rotations of the propellers.

The forces The forces that are acting on the quadrotor are the weight and the consequent lift which is created by the 4 rotors:

$$\overrightarrow{F_\xi} = \overrightarrow{P} + \sum \overrightarrow{f_i} \tag{2.6}$$

with the weight is expressed as $\overrightarrow{P} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix}$

And, $\overrightarrow{f_i}$ is the lift resulting from every individual motor M_i given that the motors M_1 and M_3 are rotating in the counter-clockwise direction and the motors M_2 and M_4 are rotating in the clockwise direction. So, $\overrightarrow{f_i}$ is expressed as $\overrightarrow{f_i} = \begin{bmatrix} 0 \\ 0 \\ f_i \end{bmatrix} = b \begin{bmatrix} 0 \\ 0 \\ \Omega_i^2 \end{bmatrix}$

$f_i = b\Omega^2; i = 1, \ldots, 4$
Ω_i: rotational velocity of motor M_i
b: Thrust coefficient

The consequent lift is the full thrust of the UAV and is expressed by $u = \sum \overrightarrow{f_i}$. It is lying on the same line as the u_z axis of the R_U frame where the final expression of $\overrightarrow{F_\xi}$ is made of two parts . The first part (the weight) is expressed in the fixed frame and the second part (the total thrust u) is expressed in the mobile frame:

$$\overrightarrow{F_\xi} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + b \begin{bmatrix} 0 \\ 0 \\ \Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2 \end{bmatrix} = {}^{O}\begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + b \, {}^{U}\begin{bmatrix} 0 \\ 0 \\ \Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2 \end{bmatrix} \tag{2.7}$$

The torques Regarding the torques, they are also divided into two parts. The first part is the result of the relations between the velocities of the rotors. Whereas the other part is generated by the gyroscopic effects that are resulting from the change of attitude direction of the rotors. In fact, a rotation around the u_x or u_y axis coupled to the rotation of the rotors that is executed around the u_z axis, generates a torque respectively on u_x or u_y axis.
By taking $\tau = \begin{bmatrix} \tau_1 & \tau_2 & \tau_3 \end{bmatrix}^\intercal$ to be the consequent torque of the quadrotor. It is divided into two parts, namely τ_a which contains the rotation torques (roll,pitch and yaw) on the 3 axes $(\tau_\phi, \tau_\theta, \tau_\psi)$ and the torque τ_G which is the result of the gyroscopic effects of the rotors, which is usually neglected in many works.

$$\tau = \tau_a + \tau_G \tag{2.8}$$

with:

- $\tau_a = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} l(f_4 - f_2) \\ l(f_3 - f_1) \\ \sum \tau_{M_i} \end{bmatrix}$
  $\sum \tau_{M_i} = \sum d\Omega_i^2 = d(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2); i = 1, \ldots 4$: motor torque about the axis.
  d: drag coefficient .
  l: distance between the center of gravity (CoG) of the quadrotor and the axis of motor M_i.

- $\tau_G = \overrightarrow{\omega} \times J_r \begin{bmatrix} 0 \\ 0 \\ \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4 \end{bmatrix} = \begin{bmatrix} -qJ_r(\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4) \\ pJ_r(\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4) \\ 0 \end{bmatrix}$

  $\overrightarrow{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$: angular velocities in the body frame. Ω_i: spinning speeds of motors M_i. J_r: rotor inertia.

Thus, the total torque $\boldsymbol{\tau}$ is expressed as follows:

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \tau_a + \tau_G = \begin{bmatrix} l(f_4 - f_2) - qJ_r(\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4) \\ l(f_3 - f_1) + pJ_r(\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4) \\ d(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \end{bmatrix} \tag{2.9}$$

System Inputs

Thus, there exists two types of forces and torques that are acting on the quadrotor platform: the translation force F_ξ which is called the thrust u, and the torque τ which represents all the torques about the distinct axes $(\tau_\phi, \tau_\theta, \tau_\psi)$.

Rather than controlling the velocities of the motors, the control in torque and thrust is synthesized, from which motor velocities are effortlessly inferred. Thus, the quadrotor has four control inputs, which are the inputs of its motors:

$$\begin{aligned} u_1 &= u \text{ (thrust)} \\ u_2 &= \tau_\phi \\ u_3 &= \tau_\theta \\ u_4 &= \tau_\psi \end{aligned}$$

The various torques and forces which are taken into account are illustrated in figure 2.2.

2.1.3 Dynamical equations

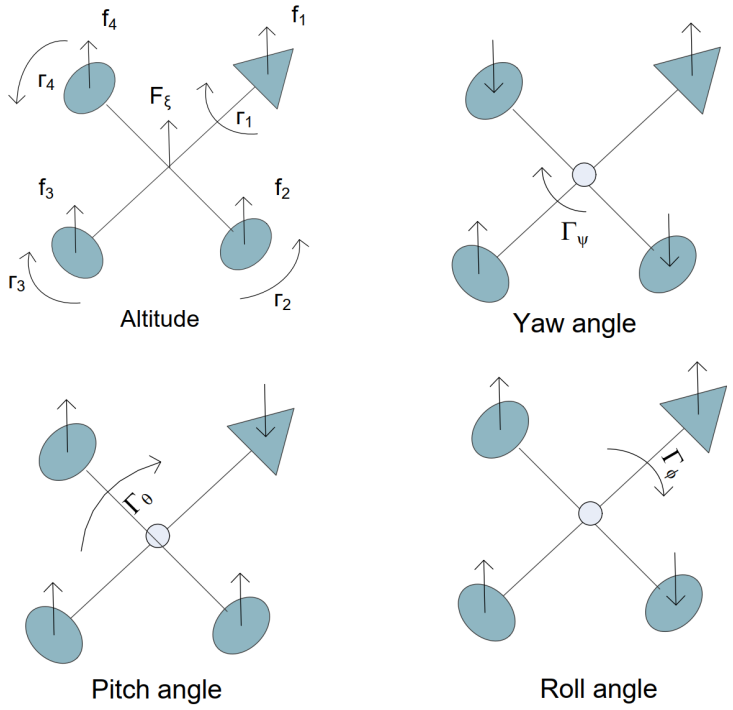


Figure 2.2: Model of the quadrotor represented with motor torques and Euler angles [12].

The equations of motion presented in the next subsection are based on the following assumptions:

- The quadrotor has a rigid structure.
- The quadrotor has a symmetrical structure.
- The center of gravity (CoG) and the fixed frame at the center of the body are assumed to be coincident.
- The propellers of the quadrotor are assumed to be rigid.
- The thrust and drag forces are assumed to be proportional to the square of the spinning speed of each propeller.

2.1.4 Planar Quadrotor

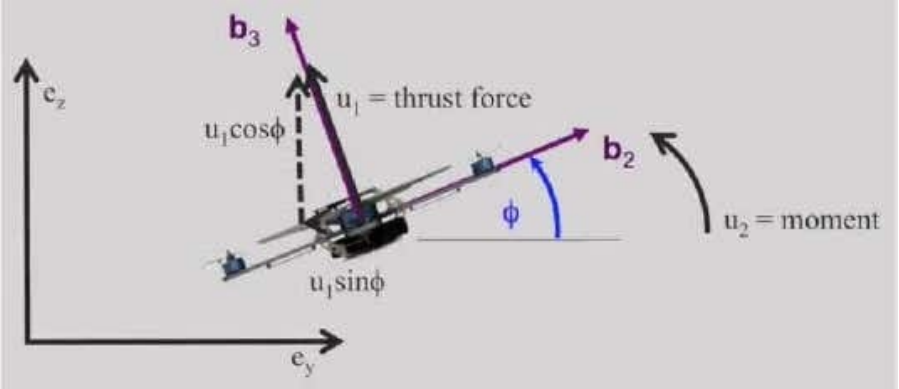


Figure 2.3: Representation of a planar drone. [13]

The equations of motion of the planar quadrotor are:

$$\begin{cases} \ddot{y} = -\frac{u_1}{m} \sin\phi \\ \ddot{z} = -g + \frac{u_1}{m} \cos\phi \\ \ddot{\phi} = \frac{u_2}{I_{xx}} \end{cases} \tag{2.10}$$

With:

- u_1: Total thrust applied on the planar quadrotor.
- u_2: Torque applied by the planar quadrotor along the x-axis.

The dynamic model in equation (2.10) assumes that the quadrotor is free to maneuver in the y-z plane. This means that the torque control input u_2 is only applied along the x-axis. Moreover, the equations of $\ddot{y}$ and $\ddot{z}$ represent the translational dynamics of the planar quadrotor, and they are controlled by u_1. Furthermore, the equation of $\ddot{\phi}$ represents the rotational dynamics of the planar quadrotor, and it is controlled by u_2. Finally, it should be noted that the total thrust u_1 is nothing else than the sum of the two thrust forces generated by the two rotors in the planar quadrotor.

2.1.5 3D Quadrotor

In the case of the 3D quadrotor, the equations of motion can be expressed in different ways. They can either be expressed using the Euler angles ϕ, θ, ψ, which represent the roll, pitch and yaw angles respectively, or they can be expressed using quaternions. Both expressions will be presented below using the Newton-Euler formalism instead of the Lagrange-Euler formalism, since it is easier to implement when writing code.

2.1.5.1 Equations of Motion with Euler Angles

Translational Model

Based on Newton's second law:

$$\sum \overrightarrow{F} = m\overrightarrow{a} \tag{2.11}$$

with

- $\overrightarrow{F} = \overrightarrow{F}_{\xi} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}$: vector of external forces.
- $\overrightarrow{a} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}$: accelerations vector.
- m: mass of the quadrotor.

So, by applying Newton's second law on the quadrotor problem at hand:

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = {}^{O}\begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + {}^{U}\begin{bmatrix} 0 \\ 0 \\ u \end{bmatrix} = {}^{O}\begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + b\,{}^{U}\begin{bmatrix} 0 \\ 0 \\ \Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2 \end{bmatrix}$$

Where u is the total thrust resulting from the 4 rotors. If u is then expressed in the world frame, the equations of the translational model can be expressed as follows [12]:

$$\begin{cases} m\ddot{x} = (\sin\psi \sin\phi + \cos\psi \sin\theta \cos\phi)u \\ m\ddot{y} = (-\sin\phi \cos\psi + \sin\psi \sin\theta \cos\phi)u \\ m\ddot{z} = \cos\theta \cos\phi u - mg \end{cases} \tag{2.12}$$

Rotational Model

Based on the Newton-Euler equations, for a rotating frame [67]:

$$I\dot{\omega} + \omega \times I\omega = \tau \tag{2.13}$$

with

- $\boldsymbol{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$
- $\boldsymbol{\eta} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$
- $\boldsymbol{\tau} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix}$

Thus, equation (2.13) becomes:

$$\begin{bmatrix} I_{xx}\dot{p} \\ I_{yy}\dot{q} \\ I_{zz}\dot{r} \end{bmatrix} + \begin{bmatrix} qr(I_{zz} - I_{yy}) \\ pr(I_{xx} - I_{zz}) \\ pq(I_{yy} - I_{xx}) \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} \tag{2.14}$$

If equation 2.9 is used, then input vector $\boldsymbol{\tau}$ can be substituted to acquire the final model of the rotation in the body frame:

$$\begin{cases} \dot{p} = qr\frac{(I_{yy}-I_{zz})}{I_{xx}} + \frac{l}{I_{xx}}(f_4 - f_2) - \frac{1}{I_{xx}}qJ_r(\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4) \\ \dot{q} = pr\frac{(I_{zz}-I_{xx})}{I_{yy}} + \frac{l}{I_{yy}}(f_3 - f_1) + \frac{1}{I_{yy}}pJ_r(\Omega_1 - \Omega_2 + \Omega_3 - \Omega_4) \\ \dot{r} = pq\frac{(I_{xx}-I_{yy})}{I_{zz}} + \frac{1}{I_{zz}}d(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \end{cases} \tag{2.15}$$

However, it can be inferred that the translational dynamics in (2.12) and the rotational dynamics in (2.13) are highly nonlinear when they are expressed using Euler angles. And, this will result in high computational loads when they are included in a controller that solves an optimization problem at each time instant such as MPC. For this reason, it is simpler to express the dynamics of the quadrotor using quaternions. Quaternions are highly efficient for analyzing situations where rotations in $\mathbb{R}^3$ are considered. A quaternion is a 4-tuple, which is a more brief and simple representation of orientation when compared to a rotation matrix. Its geometric meaning is also more clear as the rotation axis and angle can be easily retrieved from it. In addition, quaternions allow the easy composition of rotations. This is due to the fact that composition of quaternions takes only sixteen multiplications and twelve additions [68]. Thus, using quaternions will yield equations that are linear, which will result in ordinary differential equations (ODE's) that are easier to solve with MPC. In other words, when quaternions are used with a quadratic cost function, such as a linear least-squared cost function, the optimization problem becomes convex, which has a global minimum, as shown in figure 1.4.

2.1.5.2 Equations of Motion with Quaternions

The equations of motion of a quadrotor expressed with quaternions are [65]:

$$\text{Translational model: } \begin{cases} \dot{\boldsymbol{p}}_i &= \boldsymbol{v}_i \\ \dot{\boldsymbol{v}}_i &= \frac{T}{m} \begin{bmatrix} 2(q_w q_y + q_x q_z) \\ 2(q_y q_z - q_w q_x) \\ 1 - 2(q_x^2 + q_y^2) \end{bmatrix} + \boldsymbol{g} \end{cases} \tag{2.16}$$

$$\text{Rotational model: } \begin{cases} \dot{\boldsymbol{q}}_i &= \frac{1}{2} \begin{bmatrix} 0 \\ \boldsymbol{\omega}_i \end{bmatrix} \otimes \boldsymbol{q}_i \\ \dot{\boldsymbol{\omega}}_i &= \boldsymbol{I}_i^{-1} \boldsymbol{\tau}_i - \boldsymbol{I}_i^{-1} (\boldsymbol{\omega}_i \times \boldsymbol{I}_i \boldsymbol{\omega}_i) \end{cases} \tag{2.17}$$

The 2 equations in (2.16) represent the first order kinematics and the second order dynamics of the translational model of the system respectively. Similarly, the 2 equations in (2.17) represent the first order kinematics and the second order dynamics in the rotational model of the system respectively. Moreover, this model will be used for the remainder of this report for performing simulations in both the acados interface, and in ROS2/Gazebo.
It is clear that the elements of the body torques vector $\boldsymbol{\tau}$ are not being computed using this model. This is because there exists a direct linear mapping between the angular acceleration $\dot{\boldsymbol{\omega}}$ and the body torques $\boldsymbol{\tau}$. In other words, $\boldsymbol{\tau}$ can be computed from $\dot{\boldsymbol{\omega}}$.
However, since a non-centralized MPC structure will be designed to reduce the computation time, then the MPC controller will not account for all the equations of motion in (2.16) and (2.17). In fact, they will be distributed as follows:

$$\text{Treated by the MPC: } \begin{cases} \dot{\boldsymbol{p}}_i &= \boldsymbol{v}_i \\ \dot{\boldsymbol{v}}_i &= \frac{T}{m} \begin{bmatrix} 2(q_w q_y + q_x q_z) \\ 2(q_y q_z - q_w q_x) \\ 1 - 2(q_x^2 + q_y^2) \end{bmatrix} + \boldsymbol{g} \\ \dot{\boldsymbol{q}}_i &= \frac{1}{2} \begin{bmatrix} 0 \\ \boldsymbol{\omega}_i \end{bmatrix} \otimes \boldsymbol{q}_i \end{cases}$$

$$\text{Treated by the Low-Level Controller: } \left\{ \dot{\boldsymbol{\omega}}_i = \boldsymbol{I}_i^{-1} \boldsymbol{\tau}_i - \boldsymbol{I}_i^{-1} (\boldsymbol{\omega}_i \times \boldsymbol{I}_i \boldsymbol{\omega}_i) \right.$$

Moreover, the control inputs that will be outputted by the MPC controller are:

- T: The total thrust force.
- $\omega_x, \omega_y, \omega_z$: The angular rates.

The Software-in-the-Loop simulations that make use of the Low-Level Controller will be presented in section 4.1.

2.2 Extended Kalman Filter for the Planar Quadrotor

In order to perform simple simulations with acados in the planar case under the presence of disturbance, it is useful to design an Extended Kalman Filter (EKF) for the planar quadrotor in order to reduce the effects of the noise on the system and increase the tracking accuracy of a desired trajectory. The Extended Kalman Filter was chosen instead of the Kalman Filter (KF) because the Kalman Filter works well for linear systems. However, the quadrotor dynamics are highly nonlinear as seen in 2.1.4, which the Extended Kalman Filter can handle well. The EKF works by linearizing the nonlinear equations of motion at each time step. This is done by using the current values of the state vector of the planar quadrotor, in addition to the Jacobian matrices of the state vector. However, it should be noted that the EKF will not provide proper measurements if the initial condition of the state vector is not properly defined. The equations used are from the following reference [69].

2.2.1 Evolution Model

The state vector used for the planar quadrotor is:

$$X = \begin{bmatrix} y & z & \phi & v_y & v_z & \dot{\phi} \end{bmatrix}^\intercal \tag{2.18}$$

So, the evolution model can be expressed as follows:

$$\begin{bmatrix} y_{k+1} \\ z_{k+1} \\ \phi_{k+1} \\ v_{y_{k+1}} \\ v_{z_{k+1}} \\ \dot{\phi}_{k+1} \end{bmatrix} = \begin{bmatrix} y_k \\ z_k \\ \phi_k \\ v_{y_k} \\ v_{z_k} \\ \dot{\phi}_k \end{bmatrix} + \Delta t \begin{bmatrix} v_{y_k} \\ v_{z_k} \\ \dot{\phi}_k \\ -\frac{u_{1_k}}{m} \sin \phi_k \\ -g + \frac{u_{1_k}}{m} \cos \phi_k \\ \frac{u_2}{I_{xx}} \end{bmatrix} \tag{2.19}$$

With Δt being the sample time.

2.2.2 Measurement Model

Generally, quadrotors make use of an embedded Inertial Measurement Unit (IMU) sensor, which typically measures angular rates and body-frame accelerations. In addition, an embedded air pressure sensor can also be used to measure the altitude of the quadrotor. However, since simple simulations will be conducted in acados, it is assumed the all the states can be measured. This will result in the following measurement model:

$$Y = CX = I_{6\times 6}X = [y, z, \phi, v_y, v_z, \dot{\phi}]^\intercal \tag{2.20}$$

So, in this very simplified case, the measurement vector is identical to the state vector. Moreover, for the case of the 3D drone, the quadrotor that will be used already has an EKF implemented that is able to take inputs from different sensors and fuse them in order to output filtered measurements. A figure showing a representation of the EKF implemented on the real quadrotor with all the available inputs and outputs is shown below:

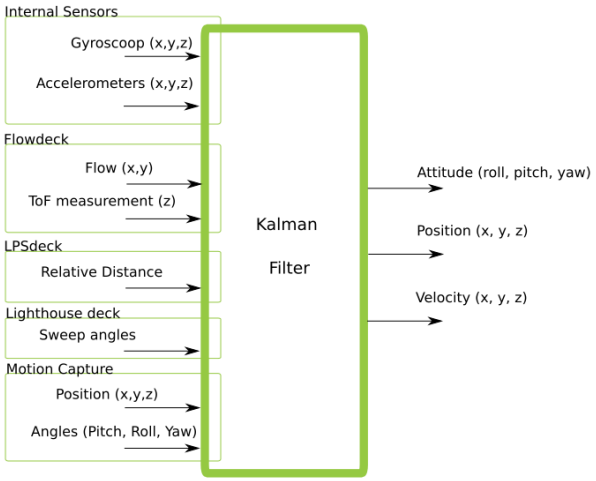


Figure 2.4: Diagram of the EKF embedded in the crazyflie 2.1 [14].

2.2.3 Prediction Phase

In the prediction phase, no measurement is taken. The state is predicted using the evolution model and Jacobian matrices of the state vector.
Using the state vector and the control inputs of the previous time instant, the current state vector can predicted using the evolution in model in 2.19. So,

$$X_k = evolution(X_{k-1}, U_{k-1}, \Delta t) \tag{2.21}$$

And, in order to show how accurate the prediction is, a covariance matrix for the prediction is computed as follows:

$$P_k = A_{k-1} P_{k-1} A_{k-1}^{\intercal} + B_{k-1} Q_\beta B_{k-1}^{\intercal} + Q_\alpha \tag{2.22}$$

Where:

- P_{k-1}: the covariance matrix of the previous time instant.
- A_{k-1}: the Jacobian of the state vector with respect to each state at the previous time instant.
- B_{k-1}: the Jacobian of the state vector with respect to the control inputs at the previous time instant.
- Q_α: the covariance matrix of the state noise.
- Q_β: the covariance matrix of the input noise.

And, the expressions of A_{k-1} and B_{k-1} are:

$$A_{k-1} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & -\frac{u1_{k-1}}{m} \cos\phi_{k-1} \Delta t & 1 & 0 & 0 \\ 0 & 0 & -\frac{u1_{k-1}}{m} \sin\phi_{k-1} \Delta t & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

$$B_{k-1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -\frac{\sin\phi_{k-1}}{m}\Delta t & 0 \\ -\frac{\cos\phi_{k-1}}{m}\Delta t & 0 \\ 0 & \frac{\Delta t}{I_{xx}} \end{bmatrix} \tag{2.24}$$

In addition, the expression of Q_α was found using trial and error based on the amount of noise added on the state and the inputs:

$$Q_\alpha = \begin{bmatrix} 5\times10^{-7} & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^{-5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-7} & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^{-4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^{-5} \end{bmatrix} \tag{2.25}$$

With each value representing the variance of the state noise on each element of the state vector. It is important not to have $Q_\alpha = 0_{6\times6}$ because in the case of the planar quadrotor, the previous states have a great influence on the evolution model as can be seen in (2.19).
As for the covariance matrix of the input noise Q_β:

$$Q_\beta = \begin{bmatrix} (10^{-3})^2 & 0 \\ 0 & (10^{-4})^2 \end{bmatrix} \tag{2.26}$$

Where the first diagonal term represents the variance of the noise on the thrust input u_1. This variance represents an input noise between $-0.001N$ and $0.001N$. As for the second diagonal term, it represent the variance of the noise on the torque input u_2. This variance represents an input noise between $-0.0001N.m$ and $0.0001N.m$. The variance values were chosen to be small because the quadrotor used will have a maximum thrust of $0.45N$, as will be shown in future sections, which is already very small.
In addition, the covariance matrix of the initial state is chosen to be:

$$P_0 = \begin{bmatrix} 10^{-2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^{-2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^{-2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^{-2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^{-2} \end{bmatrix} \tag{2.27}$$

P_0 represents the accuracy of the initial condition of the state vector.

2.2.4 Correction Phase

The correction phase is where the measurement from the sensors are taken and fused with the outputs of the predicted state to provide a proper estimation of the state.
First, the measurement prediction is computed:

$$\hat{Y}_k = CX_k \tag{2.28}$$

Where $C \equiv I_{6\times6}$, similarly to (2.20).
Then, the Kalman gain is computed as follows:

$$K = P_k C^\intercal (CP_k C^\intercal + Q_\gamma)^{-1} \tag{2.29}$$

Where, Q_γ is the covariance matrix of the measurement noise. The square root of each diagonal term in Q_γ was used as the magnitude for added white Gaussian noise of the corresponding state. So, Q_γ was chosen to be:

$$Q_\gamma = \begin{bmatrix} (10^{-2})^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & (10^{-2})^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & (\frac{\pi}{180})^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & (10^{-3})^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & (10^{-3})^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & (\frac{\pi}{180})^2 \end{bmatrix} \tag{2.30}$$

The diagonal terms in Q_γ mean the following:

- The added noise on y is between $-1cm$ and $1cm$.
- The added noise on z is between $-1cm$ and $1cm$.
- The added noise on the roll angle ϕ is between -1° and 1°
- The added noise on v_y is between $-0.01m/s$ and $0.01m/s$.
- The added noise on v_z is between $-0.01m/s$ and $0.01m/s$.
- The added noise on $\dot{\phi}$ is between $-1^\circ/s$ and $1^\circ/s$.

In addition, the state vector is updated as follows:

$$X_k = X_k + K(Y_k - \hat{Y}_k) \tag{2.31}$$

Finally, the covariance matrix of the state vector is updated. There exist many formulas to update the covariance matrix of the state after the state vector has been corrected. However, the Joseph form was used since it is better conditioned when compared to the other available formulas. So, the covariance matrix of the state vector is updated using the following:

$$P_k = (I - KC)P_k(I - KC)^\intercal + KQ_\gamma K^\intercal \tag{2.32}$$

2.3 Design of the MPC controller

2.3.1 Parameters of the Crazyflie 2.1

The parameters of the crazyflie were identified in [70]. However, to increase the accuracy of the simulations that will be performed, the drone was weighed using a small scale to determine its actual mass.

Figure 2.5: Mass of the crazyflie 2.1 equipped with the battery and properllers.

As can be seen in figure 2.5, the mass of the crazyflie is $29.5g$. However, it should be noted that the crazyflie is marketed to have a mass of $27g$ as shown in the crazyflie 2.1 datasheet [71]. Then, in order to find the maximum thrust, several experiments were conducted with different added masses to the quadrotor. And, the maximum thrust was found when the quadrotor was barely able to support its weight and hover.

Figure 2.6: Mass of the crazyflie 2.1 with the maximum added mass that it can support.

As can be seen in figure 2.6, the maximum thrust was found when the crazyflie had a total mass of $46g$. So, the maximum thrust of the drone is:

$$T_{max}^{actual} = 46 \times 10^{-3} kg \times 9.81 \frac{m}{s^2} = 0.45126N \tag{2.33}$$

So, the parameters of the quadrotor that will be used in all the simulations are:

- $m = 0.0295kg$
- $I_{xx} = 1.657171 \times 10^{-5} N.m$
- $I_{yy} = 1.657171 \times 10^{-5} N.m$
- $I_{zz} = 2.9261652 \times 10^{-5} N.m$
- $l = 0.046m$

It should be noted that in the case of the planar quadrotor, the mass and the total thrust will be divided by 2 since only 2 rotors will be present instead of 4 in the planar case.
In addition, the maximum thrust is never used. Because, when the maximum thrust is applied by the quadrotor, the quadrotor will not be able to do move freely in space. In other words, the quadrotor will only be able to move vertically and hover at a desired position. As a result, a general rule of thumb is to set the maximum thrust to be 90% of the actual maximum. This way, the remaining 10% of the thrust can be used to control the torque in order to perform maneuvers and move in the 3D space. In addition, the planar drone will also be restricted to keeping only a 10% margin for steering torque. This is done because the maximum torque τ_{max} is the maximum torque that can be given around any one axis. However, the torque must be limited greatly in order to stabilize and have more control of the quadrotor. These constraints are set in the MPC controller when it is being designed.

2.3.2 Planar Quadrotor MPC

The MPC optimization problem is formulated as follows:

$$\begin{aligned} \min_{x,u} \quad & \frac{1}{2}\|\boldsymbol{V}_x\boldsymbol{x} + \boldsymbol{V}_u\boldsymbol{u} + \boldsymbol{V}_z\boldsymbol{z} - \boldsymbol{y}_{ref}\|^2_{\boldsymbol{W}} + \frac{1}{2}\|\boldsymbol{V}^e_x\boldsymbol{x} - \boldsymbol{y}^e_{ref}\|^2_{\boldsymbol{W}_e} \\ \text{s.t.} \quad & f(\boldsymbol{x},\boldsymbol{u}) : \text{dynamics} \\ & 0 \leq T \leq T_{max} \\ & -\tau_{max} \leq \tau \leq \tau_{max} \end{aligned} \tag{2.34}$$

with:

- $\boldsymbol{V}_x \in \mathbb{R}^{n_y \times n_x}$
- $\boldsymbol{V}_u \in \mathbb{R}^{n_y \times n_u}$
- $\boldsymbol{V}_z \in \mathbb{R}^{n_y \times n_z}$
- $\boldsymbol{V}^e_x \in \mathbb{R}^{n_{ye} \times n_x}$

- $\boldsymbol{y}_{ref} \in \mathbb{R}^{n_y}$
- $\boldsymbol{y}_{ref_e} \in \mathbb{R}^{n_{y_e}}$
- $\boldsymbol{W} \in \mathbb{R}^{n_y \times n_y}$
- $\boldsymbol{W}_e \in \mathbb{R}^{n_{ye} \times n_{ye}}$

- $\boldsymbol{Q}, \boldsymbol{Q}_e \in \mathbb{R}^{n_x \times n_x}$
- $\boldsymbol{R} \in \mathbb{R}^{n_u \times n_u}$
- $\boldsymbol{W} = diag(\boldsymbol{Q}, \boldsymbol{R})$
- $\boldsymbol{W}_e = \boldsymbol{Q}_e$

And, by taking the state vector as shown in (2.18) and by using the equations of motion of the planar quadrotor as shown in (2.10), it is evident that the states evolve according to:

$$\dot{y} = v_y \tag{2.35}$$
$$\dot{z} = v_z \tag{2.36}$$
$$\dot{\phi} = \dot{\phi} \equiv r\dot{o}ll \tag{2.37}$$
$$\dot{v}_y = -\frac{u_1}{m}\sin(\phi) \tag{2.38}$$
$$\dot{v}_z = -g + \frac{u_1}{m}\cos(\phi) \tag{2.39}$$
$$\ddot{\phi} = \frac{u_2}{I_{xx}} \tag{2.40}$$

As shown in 2.1.4, the control inputs vector in this case is :

$$\boldsymbol{u} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^\intercal \tag{2.41}$$

Then, the initial condition of the states of the quadrotor is set:

$$\boldsymbol{x}_0 = \begin{bmatrix} x_0 & y_0 & \phi_0 & v_{y_0} & v_{z_0} & \dot{\phi}_0 \end{bmatrix}^\intercal \tag{2.42}$$

And, the desired trajectory will be fed to the MPC controller as a reference by using the future N waypoints of the trajectory, which contain the desired states at each time instant:

$$\boldsymbol{x}_d = \begin{bmatrix} x_d & y_d & \phi_d & v_{y_d} & v_{z_d} & \dot{\phi}_d \end{bmatrix}^\intercal \tag{2.43}$$

Furthermore, the desired control inputs $\boldsymbol{u}_d$ can also be fed to the MPC controller:

$$\boldsymbol{u}_d = \begin{bmatrix} u_{1_d} & u_{2_d} \end{bmatrix}^\intercal \tag{2.44}$$

The reference vector $\boldsymbol{y}_{ref}$ in the running cost can be constructed using the desired waypoints and the desired control inputs for the next $(N-1)$ time steps:

$$\boldsymbol{y}_{ref} = \begin{bmatrix} \boldsymbol{x}_d^\intercal & \boldsymbol{u}_d^\intercal \end{bmatrix}^\intercal \tag{2.45}$$

The reference vector $\boldsymbol{y}^e_{ref}$ in the terminal cost contains the desired waypoint at time step N:

$$\boldsymbol{y}^e_{ref} = \boldsymbol{x}_d \tag{2.46}$$

Moreover, the MPC parameters are chosen to be:

- $N = 100$: number of discretization steps in the prediction horizon.
- $T_f = 1s$: prediction horizon.

It should be noted that the number of discretization steps that was used is rather large, and will surely be reduced during the experimentation phase. However, it was still used in the simple acados simulations since the total computation time of the MPC while tracking a $22s$ trajectory as around $8.5s$ on an Inter Core i5-6300u 2.4GHz with 4 cores, which can be considered as a mid-range processor. An example of a MPC computation time for a planar quadrotor which is tracking a $20s$ circular trajectory with $2s$ of initial hovering time is shown in figure 2.8 below.

```
Total computation time: 8.444775819778442
Average computation time: 0.0038385344635356 56
Maximum computation time: 0.013800382614135742
```

Figure 2.7: Computation of the MPC for a planar quadrotor tracking a 22s trajectory.

Finally, the maximum thrust and torque are computed as explained in 2.3.1:

$$T_{max} = 0.9(\frac{T_{max}^{actual}}{2}) = 0.203067N \tag{2.47}$$

$$\tau_{max} = 0.1(\frac{1}{2}T_{max}l) = 0.0004670541N.m \tag{2.48}$$

And, as for the components of the linear least-squared cost function:

- $\boldsymbol{V}_x = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
- $\boldsymbol{V}_u = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$
- $\boldsymbol{V}_{x_e} = diag(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix})$
- $\boldsymbol{Q} = diag(\begin{bmatrix} 30 & 30 & 1 & 5 & 5 & 1 \end{bmatrix})$
- $\boldsymbol{Q}_e = diag(\begin{bmatrix} 30 & 30 & 1 & 5 & 5 & 1 \end{bmatrix})$
- $\boldsymbol{R} = diag(\begin{bmatrix} 1 & 1 \end{bmatrix})$

The matrices $\boldsymbol{V}_x$ and $\boldsymbol{V}_u$ are used to refer to each element of the state in the state vector $\boldsymbol{x}$ and to each element of the inputs in the input vector $\boldsymbol{u}$ in the running cost, respectively. And, it should be noted that there are no algebraic variables $\boldsymbol{z}$ in the problem at hand. However, it is included in the cost function of equation (2.34) for completeness. Furthermore, the matrix $\boldsymbol{V}_{x_e}$ is used to refer to each element of the state vector $\boldsymbol{x}$ in the terminal cost. And, it can be inferred that there are no control inputs at the terminal cost since the terminal cost represents the final position that the system is desired to reach.

Moreover, the diagonal terms of the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ represent the weights applied on each element of the state vector and the control inputs vector in the running cost, respectively. Finally, the diagonal terms in the matrix $\boldsymbol{Q}_e$ represent the weights applied on each element of the state vector in the terminal cost. Usually, the weights on the terminal cost are higher than the weights on the running cost, because, it is usually more important that the final desired position is reached. So, this will push the MPC controller to generate control inputs that will allow the system to reach the final desired position. However, in the case of tracking a desired trajectory, all the waypoints are equally important. For this reason, the weights on the elements of the state vector in both the running cost and the terminal cost, are equal ($\boldsymbol{Q} = \boldsymbol{Q}_e$).

The weights provided above were used to follow a circular trajectory with references on y, z, v_y and v_z. More weights were set on the y and z elements of the state vector since it is more important to follow the trajectory coordinates instead of perfectly following the desired velocities. And, even though no references were given for ϕ and $\dot{\phi}$ (both had a reference of 0 at each time instant), it is generally a good idea to have small weights on these states (rather than them having a weight of 0) so that the MPC controller will try to minimize ϕ and $\dot{\phi}$ as much as possible as long as the goals set on the other states (y, z, v_y and v_z) are satisfied. If a weight of 0 was set on ϕ and $\dot{\phi}$ instead, this means that that ϕ and $\dot{\phi}$ are free to have any value, so the MPC controller would not be controlling these states in that case.

In addition a weight of 1 was used on each of the 2 control inputs u_1 and u_2. And, since no reference was given to the torque control input u_2, the MPC will try to drive the torque u_2 to 0 as long as the other goals with the higher weights are satisfied. As for the thrust u_1, a reference of $\frac{m.g}{2}$ (the mass is divided by 2 in the planar case) was given at each time instant. Of course, the thrust will be larger than $\frac{m.g}{2}$ when the quadrotor is following a trajectory, but having this reference will inform the MPC of what is the minimum thrust required to allow the quadrotor to hover, which will result in smoother control inputs.

After the optimization problem is solved for the current time instant, the MPC controller will have the next N control input solutions for the next N time instants. However, the MPC will only output the first solution (which is the solution of the current time instant) for the control inputs and discard the other $N - 1$ solutions. Then, the system will take 1 step forward using the control inputs that were provided by the MPC controller. Finally, the optimization problem is repeated until the final waypoint of the trajectory is reached.

2.3.3 3D Quadrotor MPC

In the 3D quadrotor case, the MPC optimization problem has the same formulation as in (2.34) but with different constraints on the dynamics.

$$\begin{aligned} \min_{x,u} \quad & \frac{1}{2}\|\boldsymbol{V}_x\boldsymbol{x} + \boldsymbol{V}_u\boldsymbol{u} + \boldsymbol{V}_z\boldsymbol{z} - \boldsymbol{y}_{ref}\|^2_{\boldsymbol{W}} + \frac{1}{2}\|\boldsymbol{V}^e_x\boldsymbol{x} - \boldsymbol{y}^e_{ref}\|^2_{\boldsymbol{W}_e} \\ \text{s.t.} \quad & f(\boldsymbol{x}, \boldsymbol{u}) : \text{dynamics} \\ & 0 \leq T \leq T_{max} \end{aligned} \tag{2.49}$$

with:

- $\boldsymbol{V}_x \in \mathbb{R}^{n_y \times n_x}$
- $\boldsymbol{V}_u \in \mathbb{R}^{n_y \times n_u}$
- $\boldsymbol{V}_z \in \mathbb{R}^{n_y \times n_z}$
- $\boldsymbol{V}^e_x \in \mathbb{R}^{n_{ye} \times n_x}$

- $\boldsymbol{y}_{ref} \in \mathbb{R}^{n_y}$
- $\boldsymbol{y}_{ref_e} \in \mathbb{R}^{n_{ye}}$
- $\boldsymbol{W} \in \mathbb{R}^{n_y \times n_y}$
- $\boldsymbol{W}_e \in \mathbb{R}^{n_{ye} \times n_{ye}}$

- $\boldsymbol{Q}, \boldsymbol{Q}_e \in \mathbb{R}^{n_x \times n_x}$
- $\boldsymbol{R} \in \mathbb{R}^{n_u \times n_u}$
- $\boldsymbol{W} = diag(\boldsymbol{Q}, \boldsymbol{R})$
- $\boldsymbol{W}_e = \boldsymbol{Q}_e$

And, by taking the state vector as:

$$\boldsymbol{x} = \begin{bmatrix} x & y & z & q_w & q_x & q_y & q_z & v_x & v_y & v_z \end{bmatrix}^\intercal \tag{2.50}$$

And, by using the equations of motion of the 3D quadrotor in quaternion form as shown in (2.16) and (2.17), it is evident that the states evolve according to:

$$\dot{x} = v_x \tag{2.51}$$
$$\dot{y} = v_y \tag{2.52}$$
$$\dot{z} = v_z \tag{2.53}$$
$$\dot{q}_w = \frac{1}{2}(-\omega_x q_x - \omega_y q_y - \omega_z q_z) \tag{2.54}$$
$$\dot{q}_x = \frac{1}{2}(\omega_x q_w + \omega_z q_y - \omega_y q_z) \tag{2.55}$$
$$\dot{q}_y = \frac{1}{2}(\omega_y q_w - \omega_z q_x + \omega_x q_z) \tag{2.56}$$
$$\dot{q}_z = \frac{1}{2}(\omega_z q_w + \omega_y q_x - \omega_x q_y) \tag{2.57}$$
$$\dot{v}_x = 2(q_w q_y + q_x q_z)\frac{T}{m} \tag{2.58}$$
$$\dot{v}_y = 2(q_y q_z - q_w q_x)\frac{T}{m} \tag{2.59}$$
$$\dot{v}_z = (1 - 2q_x^2 - 2q_y^2)\frac{T}{m} - g \tag{2.60}$$

As shown in 2.1.5.2, the control inputs vector in this case is:

$$\boldsymbol{u} = \begin{bmatrix} T & \omega_x & \omega_y & \omega_z \end{bmatrix}^\intercal \tag{2.61}$$

Then, the initial condition of the states of the quadrotor is set:

$$\boldsymbol{x}_0 = \begin{bmatrix} x_0 & y_0 & z_0 & q_{w_0} & q_{x_0} & q_{y_0} & q_{z_0} & v_{x_0} & v_{y_0} & v_{z_0} \end{bmatrix}^\intercal \tag{2.62}$$

And, similarly to the planar quadrotor case, the desired trajectory will be fed to the MPC as a reference by using the future N waypoints of the trajectory, which contain the desired states at each time instant:

$$\boldsymbol{x}_d = \begin{bmatrix} x_d & y_d & z_d & q_{w_d} & q_{x_d} & q_{y_d} & q_{z_d} & v_{x_d} & v_{y_d} & v_{z_d} \end{bmatrix}^\intercal \tag{2.63}$$

Furthermore, the desired control inputs $\boldsymbol{u}_d$ can also be fed to the MPC controller:

$$\boldsymbol{u}_d = \begin{bmatrix} T_d & \omega_{x_d} & \omega_{y_d} & \omega_{z_d} \end{bmatrix}^\intercal \tag{2.64}$$

Similarly to the planar quadrotor case, the reference vector $\boldsymbol{y}_{ref}$ in the running cost can be constructed using the desired waypoints and the desired control inputs for the next $(N-1)$ time steps:

$$\boldsymbol{y}_{ref} = \begin{bmatrix} \boldsymbol{x}_d^\intercal & \boldsymbol{u}_d^\intercal \end{bmatrix}^\intercal \tag{2.65}$$

The reference vector $\boldsymbol{y}_{ref}^e$ in the terminal cost contains the desired waypoint at time step N:

$$\boldsymbol{y}_{ref}^e = \boldsymbol{x}_d \tag{2.66}$$

Moreover, the same MPC parameters as the planar quadrotor case were chosen:

- $N = 100$: number of discretization steps in the prediction horizon.
- $T_f = 1s$: prediction horizon.

In the 3D quadrotor case, the total number of ordinary differential equations (equations of motion) is larger than the planar quadrotor case. This will result in more computations implemented by the MPC controller at each time instant, which will increase the computation time of the MPC controller. An example of a MPC computation time for a 3D quadrotor which is tracking a $20s$ circular trajectory with $2s$ of initial hovering time is shown in figure 2.8 below.

```
Total computation time: 14.130622386932373
Average computation time: 0.006423010175878352
Maximum computation time: 0.016766548156738 28
```

Figure 2.8: Computation of the MPC for a 3D quadrotor tracking a $22s$ trajectory.

The maximum thrust of the 3D quadrotor is computed as explained in 2.3.1:

$$T_{max} = 0.9(T_{max}^{actual}) = 0.406134N \tag{2.67}$$

In order to properly control the angular rates, they also must be limited similar to the torque in the planar case. A general rule of thumb is that a quadrotor is able to perform aggressive maneuvers with a maximum angular rate of $720°/s$. So, the limits that are set on the angular rates inputs in the MPC controller are expressed as follows:

$$-4\pi\ rad/s \leq \omega_x \leq 4\pi\ rad/s \tag{2.68}$$
$$-4\pi\ rad/s \leq \omega_y \leq 4\pi\ rad/s \tag{2.69}$$
$$-4\pi\ rad/s \leq \omega_z \leq 4\pi\ rad/s \tag{2.70}$$

And, as for the components of the linear least-squared cost function:

- $\boldsymbol{V}_x = \left[\begin{array}{cccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$ • $\boldsymbol{V}_u = \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]$
- $\boldsymbol{V}_{x_e} = diag(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix})$
- $\boldsymbol{Q} = diag(\begin{bmatrix} 50 & 50 & 50 & 5 & 5 & 5 & 5 & 10 & 10 & 10 \end{bmatrix})$
- $\boldsymbol{Q}_e = diag(\begin{bmatrix} 50 & 50 & 50 & 5 & 5 & 5 & 5 & 10 & 10 & 10 \end{bmatrix})$
- $\boldsymbol{R} = diag(\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix})$

In this case, references were provided for x, y, z, v_x, v_y and v_z. Similar to the planar case, following x, y and z is more important than following v_x, v_y and v_z at each time instant. This is why x, y and z have larger weights. In addition, no control inputs were given for the quaternions, but setting weights of 5's on each element of the quaternion will drive the roll, pitch and yaw angles to 0 throughout the trajectory. And, if the desired trajectory is a flip trajectry, then having weights of 5's on each element of the quaternion will ensure that the flip trajectory is being followed properly. As for the inputs, no references were given for the angular rates. So, having weights of 1's will drive the angular rates to 0, which will stabilize the quadrotor throughout the trajectory. Finally, a reference of $m.g$ was given for the thrust. This will give the MPC an idea of where the thrust control input should be when the quadrotor is at a hovering position, which will result in smoother control inputs throughout the trajectory. However, when the desired trajectory is a flip trajectory, it should be noted that the weight on the ω_x input will be set to a value close to 0 (such as 10^{-3}). This way, the MPC controller will be allowed to compute larger values for ω_x that will allow to track the flip trajectory.
Thus, in order to track a flip trajectory in the acados simulator, the weight matrices will have the following form:

- $\boldsymbol{Q} = diag(\begin{bmatrix} 50 & 50 & 50 & 5 & 5 & 5 & 5 & 10 & 10 & 10 \end{bmatrix})$
- $\boldsymbol{Q}_e = diag(\begin{bmatrix} 50 & 50 & 50 & 5 & 5 & 5 & 5 & 10 & 10 & 10 \end{bmatrix})$
- $\boldsymbol{R} = diag(\begin{bmatrix} 1 & 10^{-3} & 1 & 1 \end{bmatrix})$

2.3.4 Design of the Circular Trajectory

The following parameters of the circular trajectory are the same in the case of the planar quadrotor and the 3D quadrotor:

$$r = 1m : \text{radius}$$
$$f = \frac{6\pi}{10} : \text{frequency (3 revolutions every 10 seconds)}$$

2.3.4.1 For the Planar Quadrotor case

The circular trajectory was designed using the following equations:

$$\begin{aligned} c_y &= 4 : \text{center coordinate on y} \\ c_z &= 5 : \text{center coordinate on z} \\ y_d &= r.\cos(f.t) + c_y \\ z_d &= r.\sin(f.t) + c_z \\ v_{y_d} &= -r.f.\sin(f.t) \\ v_{z_d} &= r.f.\cos(f.t) \end{aligned}$$

with t being the time.

2.3.4.2 For the 3D Quadrotor case

The circular trajectory was designed using the following equations:

$$\begin{aligned} c_x &= 0 : \text{center coordinate on x} \\ c_y &= 0 : \text{center coordinate on y} \\ x_d &= r.\cos(f.t) + c_y \\ y_d &= r.\sin(f.t) + c_z \\ z_d &= 1 \\ v_{x_d} &= -r.f.\sin(f.t) \\ v_{y_d} &= r.f.\cos(f.t) \\ v_{z_d} &= 0 \end{aligned}$$

with t being the time.

2.4 Simulations using acados

In this section, simulations will be performed to follow a circular trajectory in the planar quadrotor case and the 3D quadrotor case. However, contrary to the closed-loop control structure shown in figure 4.1, a low-level controller is not present in these simulations.

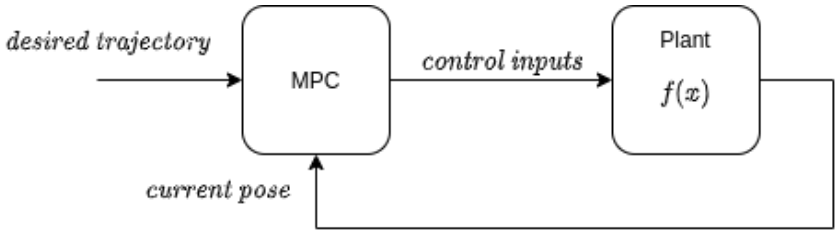


Figure 2.9: Diagram of the MPC closed-loop control structure used in the acados simulations.

So, even though these simulations will not provide an accurate representation of how the real system will behave, they will still provide a rough estimation of the behavior of the real system.

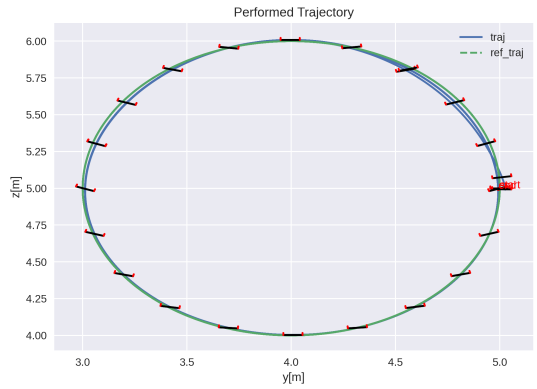
2.4.1 Planar Quadrotor Simulations

Noiseless measurements

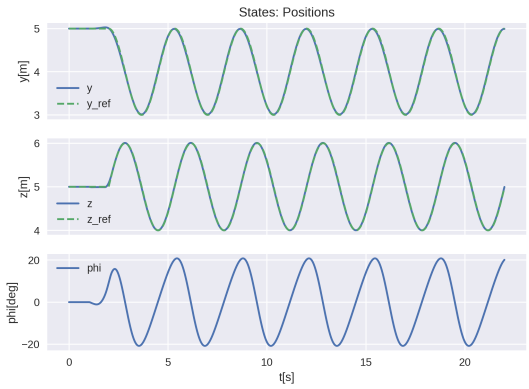
As can be seen in figure 2.10, the MPC controller was able to output control inputs that allowed the quadrotor to follow a circular trajectory. By looking at the control inputs in subfigure 2.10d, the thrust control input dips for a time instant just before the circular trajectory starts, then the thrust control input saturates until the desired velocities of the trajectory is reached, and then the thrust oscillates throughout the circular trajectory. Furthermore, by looking at subfigure 2.10b, it is evident that the circular trajectory along y and z is being followed perfectly. And, by looking at the error plots in subfigure 2.10e, it is clear that the errors along y and z are very small. Finally, by looking at the velocity states and their errors in subfigures 2.10c and 2.10f, respectively, it is clear that the largest errors occur at the beginning of the trajectory when the thrust input is saturated. Then, the errors decrease significantly when the desired velocities are reached and the planar quadrotor starts to follow the circular trajectory.

Noisy Measurements and Control Inputs

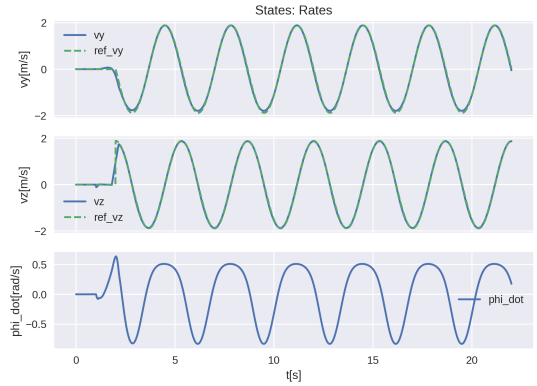
During this simulation, the MPC controller was working under the presence of white Gaussian noise on the state measurement and on the control inputs at each time instant. By looking at figure 2.11, the extended Kalman filter was able to filter out most of noise. By looking at the error plots on y and z in subfigure 2.11e, it is evident that the extended Kalman filter helped reduce the error greatly in the y position, and it also managed to reduce the error slightly in the z position.



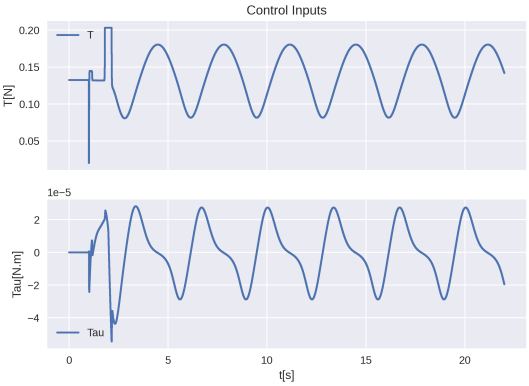
(a) Trajectory followed by the planar quadrotor



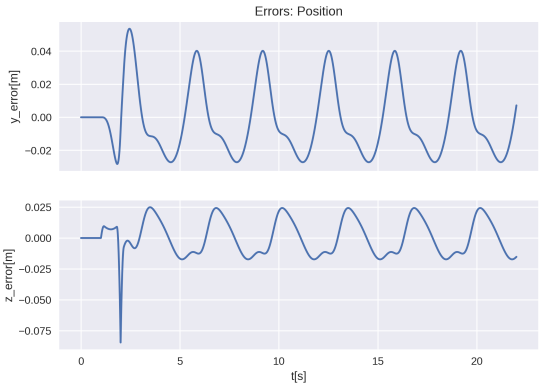
(b) Position states of the planar quadrotor.



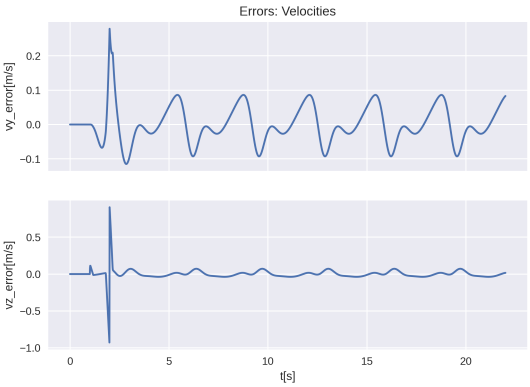
(c) Velocity states of the planar quadrotor.



(d) Control inputs of the planar quadrotor

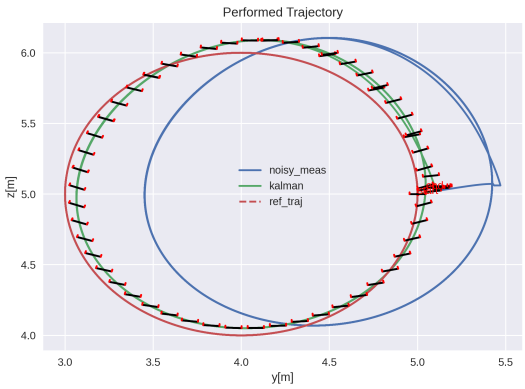


(e) Errors on y and z.

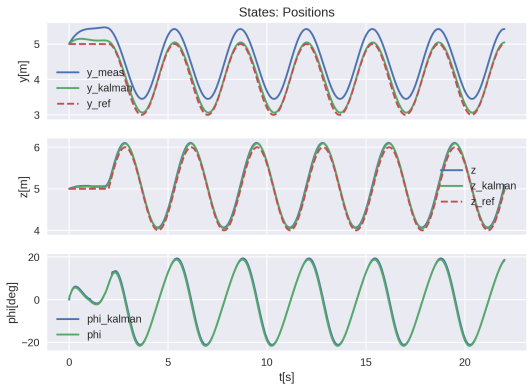


(f) Errors on v_y and v_z.

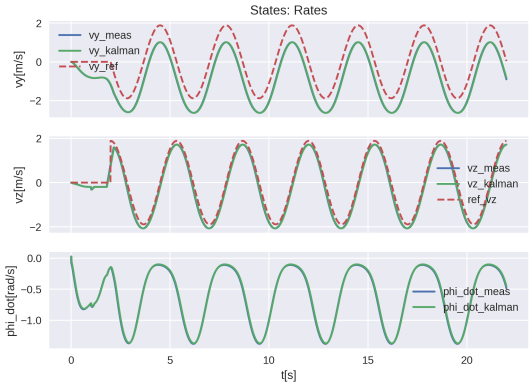
Figure 2.10: Simulation of the planar quadrotor following a circular trajectory without noise.



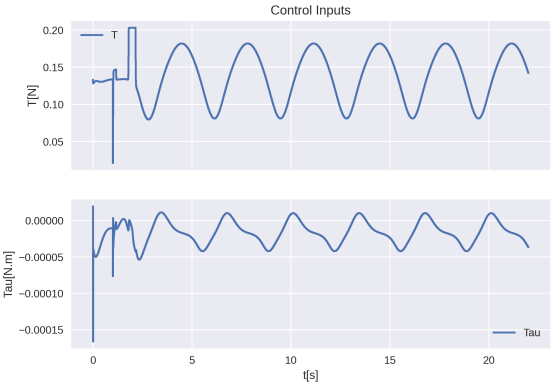
(a) Trajectory followed by the planar quadrotor



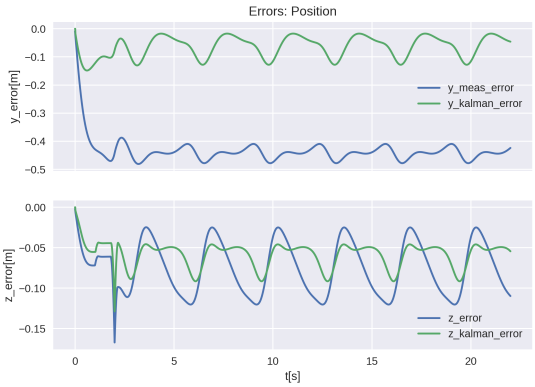
(b) Position states of the planar quadrotor.



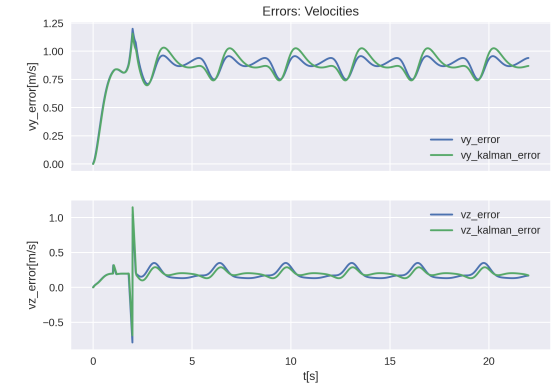
(c) Velocity states of the planar quadrotor.



(d) Control inputs of the planar quadrotor



(e) Errors on y and z.

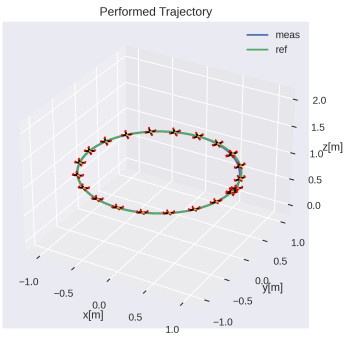


(f) Errors on v_y and v_z.

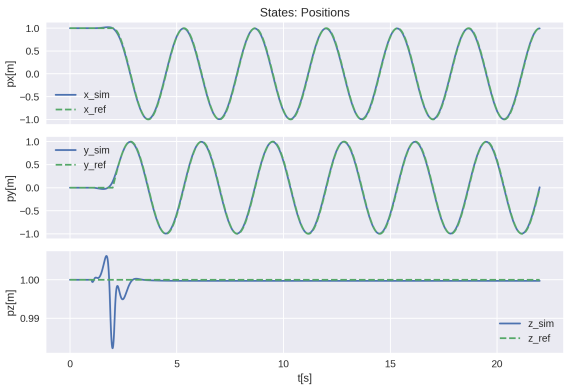
Figure 2.11: Simulation of the planar quadrotor following a circular trajectory under the presence of disturbance on the state measurement and on the control inputs.

2.4.2 3D Quadrotor Simulation

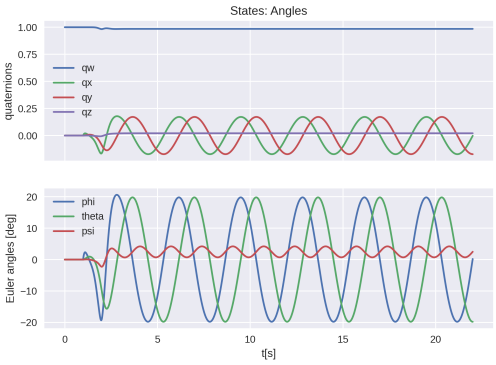
In the case of the 3D quadrotor, simulations will only be done without noise since the quadrotor that will be used for experimentation already has an extended Kalman filter implementation.



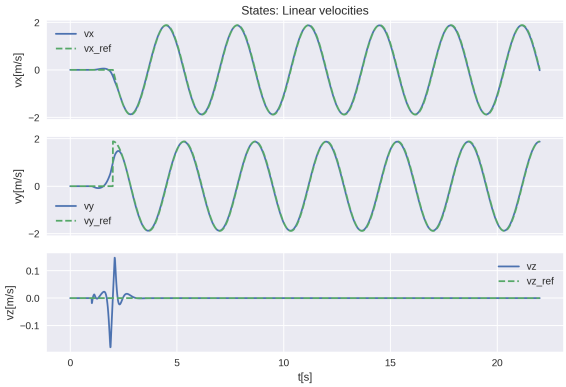
(a) Trajectory followed by the 3D quadrotor



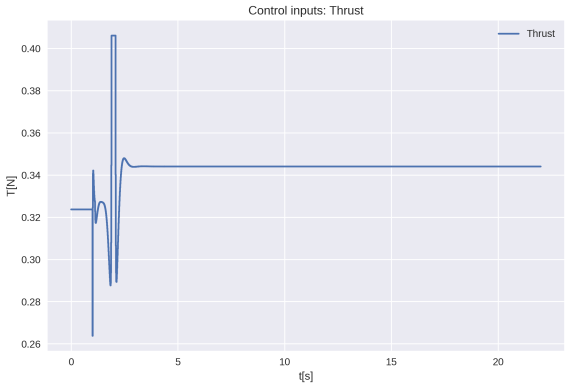
(b) Position states.



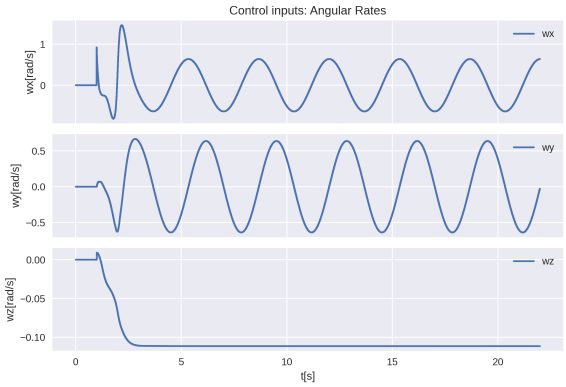
(c) Angular states.



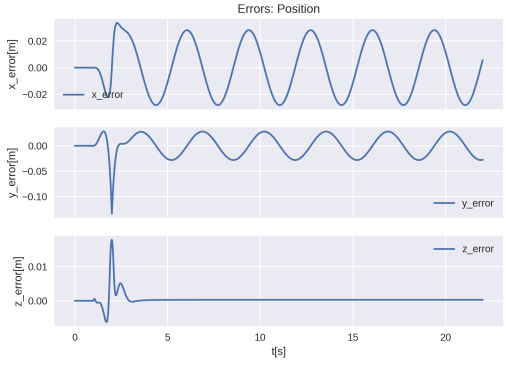
(d) Velocity states.



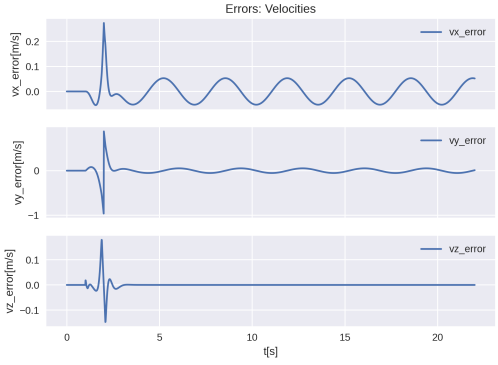
(e) Thrust input.



(f) Angular rates inputs.



(g) Errors on x, y and z.



(h) Errors on v_x, v_y and v_z.

Figure 2.12: Simulation of the 3D quadrotor following a circular trajectory without noise.

As can be seen in figure 2.12, the 3D quadrotor is able to follow the 3D trajectory. By looking at subfigure 2.12e, similarly to the planar quadrotor case, the thrust dips at the beginning of the circular trajectory for a time instant, then it saturates until the desired velocity is reached, then it settles at around $0.345N$ for the remaining of the circular trajectory. And, by looking at subfigure 2.12b, it is clear that the quadrotor is able to follow the circular trajectory perfectly with very minimal errors on x, y and z. Finally, by looking at the errors plots in subfigures 2.12g and 2.12h, it is evident that the largest amount of errors occur just when the 3D quadrotor switches from the hovering phase to tracking the circular trajectory. Then, the errors oscillate around zero after the thrust input saturates and settles at $0.345N$ throughout the trajectory.

Chapter 3

Flip Trajectory Generation and Simulations with acados

In this chapter, the design of the flip trajectory will be presented. And, simulations with acados will be made in order to try to follow the flip trajectory.

3.1 Design of the Flip Trajectory

3.1.1 Dynamic criterion for feasible trajectories

It has been demonstrated in [55] that a trajectory is dynamically feasible for a planar quadrotor if and only if it satisfies the following:

$$\ddot{y}\cos\phi + (\ddot{z} + g)\sin\phi = 0 \tag{3.1}$$

So, equation 3.1 represents the free dynamics of a planar quadrotor. In other words, it represents the dynamic constraints on the motion of the planar quadrotor, which must be satisfied in order to track a trajectory. This approach is similar to the case of parallel robots. Because, when a parallel robot that is tracking a predefined trajectory reaches a singularity configuration, it becomes locally under-actuated, which will result in free dynamics that must be satisfied in order for the motion of the parallel robot to be dynamically feasible. However, in the case of a quadrotor, the situation is more involved, because the free dynamics must be respected at each time instant. This is because a quadrotor is under-actuated by design, and not just locally.

3.1.2 Trajectory planning

For the sake of simplicity, a trajectory will be designed for the planar quadrotor case, and the same trajectory will be tracked by the 3D quadrotor. Using the same approach as demonstrated in [55], the trajectories along z and ϕ will be designed. Then, the trajectory along y will be the consequence of the other 2 trajectories. In other words, the trajectories along z and ϕ will be planned independently, while the trajectory along y will be computed from the dynamic criterion shown in equation (3.1) as follows:

$$\begin{cases} \ddot{y}(t) & = -(\ddot{z} + g)\sin\phi \\ \dot{y}(t) & = \dot{y}(0) + \int_0^t \ddot{y}(t)dt \\ y(t) & = y(0) + \int_0^t \dot{y}(t) \end{cases} \tag{3.2}$$

The main problem with the strategy is that $y(t)$ is smooth and bounded if and only if $\phi \in (-\frac{\phi}{2}, \frac{\phi}{2})$. Indeed, this by most of the trajectories that are tracked by a quadrotor, but not a flip trajectories, since they require a full rotation of 2π. In order satisfy the dynamic criterion during a flip trajectory, a simple way of doing so is to turn off the rotors, which will result in $u_1 = u_2 = 0$. And, the dynamic equations of the planar quadotor in this case become:

$$\begin{cases} \ddot{y} = 0 \\ \ddot{z} = -g \\ \ddot{\phi} = 0 \end{cases} \tag{3.3}$$

By substituting each term of equation (3.3) in equation (3.1), it is evident that the dynamic criterion is satisfied. However, it should be noted that this represents an ideal trajectory that cannot be implemented on the real quadrotor. Because, the motors of the quadrotor should never turn off in mid-flight so that the quadrotor can be controlled at all times. And, it will be shown in the coming simulations that the MPC controller is able to overcome this issue and track the flip trajectory due to the robustness of the MPC controller. Another positive note for having $u_1 = u_2 = 0$ while designing the trajectory, is that this will allow the trajectory along $y(t)$ to be much smaller, which will result in a flip trajectory that does not require a large area to be performed. Moreover, the trajectories for each coordinate during a free flight phase can be obtained by integrating equation (3.3) twice as follows:

$$\begin{cases} y(t) = y(0) + \dot{y}(0)t \\ z(t) = z(0) + \dot{z}(0)t - \frac{g}{2}t^2 \\ \phi(t) = \phi(0) + \dot{\phi}(0)t \end{cases} \tag{3.4}$$

So, an ideal flip trajectory will be designed in which the flip phase will completely rely on equation (3.4). So, the main idea is to divide the flip trajectory into three different phases:

Reaching phase: The quadrotor starts from a given hovering position with $\phi = 0$ and reaches a sufficient velocity along all the three coordinates (y, z and ϕ) in order to initialize correctly the flip phase, which makes use of equation (3.4).

Flip phase: The motors are turned off (ideal case) and the quadrotor performs the flip by using the inertia that was provided from the final velocities reached at the end of the reaching phase.

Recovery phase: Once the flip is almost completed, the control of the quadrotor is taken back again in order to prevent it from crashing. And, the final desired waypoint is a hovering position.

3.1.3 Trajectory generation using Polynomials of Order 9

One of the most used methods for generating trajectories is to use polynomials. The main idea is that the coefficients of a n^{th}-order polynomial can always be recovered if $n + 1$ constraints are provided. In the case of trajectory planning for robotics, these constraints are the position and the successive derivatives of a given coordinate at the beginning and at the end of the considered time span of the trajectory. Moreover, polynomials of order 9 ($n = 9$) are normally used for aggressive maneuvers in robotics, since they provide smoothness up to the jerk (the derivative of the acceleration) and continuity up to the snap (the derivative of the jerk).

Furthermore, the corresponding polynomial trajectory fora general coordinate $h(t)$ is expressed as follows:

$$h(t) = \sum_{i=0}^{9} a_i t^i = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 + a_6 t^6 + a_7 t^7 + a_8 t^8 + a_9 t^9 \tag{3.5}$$

In order to compute to coefficients a_i of the polynomial in equation (3.5), $n+1 = 10$ constraints need to be provided at at certain time instants throughout the trajectory. A common approach is to have half of the constraints at the initial time of the trajectory, and the other half of the constraints at the final time of the trajectory. Let t_0 be the initial time of the trajectory and t_f the final time of the trajectory. In the case of polynomials of order 9, $/frac(n+1)2 = 5$ constraints will be provided at the start of the trajectory, and $/frac(n+1)2 = 5$ constraints will be provided at the end of the trajectory. As an example, if a position constraint is to be imposed at the initial time of the trajectory t_0, the polynomial expression can be expressed as follows:

$$h(t_0) = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 + a_6 t_0^6 + a_7 t_0^7 + a_8 t_0^8 + a_9 t_0^9 \tag{3.6}$$

with $h(t_0)$ being the position of the coordinate at t_0. In addition, equation (3.6) is linear with respect to the set of coefficients. As a result, it can be rewritten as follows:

$$\boldsymbol{a}^\intercal \boldsymbol{x} = b \tag{3.7}$$

where:

$$\begin{cases} \boldsymbol{a}^\intercal = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 & t_0^6 & t_0^7 & t_0^8 & t_0^9 \end{bmatrix} \\ \boldsymbol{x} = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 \end{bmatrix}^\intercal \\ b = h(t_0) \end{cases} \tag{3.8}$$

If this procedure is repeated $n+1$ times and the equations are stacked, then the resulting linear system can be written as follows:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{3.9}$$

Assuming that $t_0 = 0$, matrix $\boldsymbol{A} \in \mathbb{R}^{(n+1)\times(n+1)}$ in equation (3.9) can be expressed as follows:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 24 & 0 & 0 & 0 & 0 & 0 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 & t_f^6 & t_f^7 & t_f^8 & t_f^9 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 & 6t_f^5 & 7t_f^6 & 8t_f^7 & 9t_f^8 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 & 30t_f^4 & 42t_f^5 & 56t_f^6 & 72t_f^7 \\ 0 & 0 & 0 & 6 & 24t_f & 60t_f^2 & 120t_f^3 & 210t_f^4 & 336t_f^5 & 504t_f^6 \\ 0 & 0 & 0 & 0 & 24 & 120t_f & 360t_f^2 & 840t_f^3 & 1680t_f^4 & 3024t_f^5 \end{bmatrix} \tag{3.10}$$

The vector $\boldsymbol{x} \in \mathbb{R}^{n+1}$ which contains the unknown polynomial coefficients can be computed by inverting the matrix $\boldsymbol{A}$:

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} \tag{3.11}$$

The same procedure can be implemented for retrieving the coefficients of any arbitrary polynomial, given that a sufficient number of constraints is provided.

3.1.4 Designing the Waypoints of the Flip Trajectory

As described earlier, the trajectories along the z and ϕ coordinates will be designed first. Then, the trajectory along the y coordinate will be a consequence of the other two trajectories, and will be computed from equation (3.2). The reason for choosing to design the trajectories along z and ϕ is:

- For z: Designing the trajectory along the z coordinate will ensure that the quadrotor will remain at a safe distance above the ground throughout the flip trajectory in order to prevent it from crashing.
- For ϕ: Designing the trajectory along the ϕ coordinate will ensure that the flip maneuver will be properly defined where the initial condition of the trajectory will be $\phi = 0$, and the final condition of the trajectory will be $\phi = 2\pi$.

As for the trajectory along the y coordinate, as mentioned earlier, having $u_1 = u_2 = 0$ throughout the flip phase of the trajectory will result in a trajectory with smaller horizontal displacement while remaining dynamically feasible, even though the motors will never be turned off throughout the trajectory. And, the robustness of the MPC controller will be relied on to handle this problem and still track this trajectory.

Flip Phase

The most important part of the flip trajectory is the flip phase, which consists (ideally) of free flight, during which most of the flip will be carried out. As shown in 3.4, the trajectories during the flip phase are low-order polynomials with a small number degrees of freedom, and the coefficients of the trajectories can be computed in closed form as follows.

- For the z coordinate:

$$Z_4(t) = a_{z_4,0} + a_{z_4,1}t + a_{z_4,2}t^2 = z_4(0) + \dot{z}_4(0)t - \frac{g}{2}t^2 \tag{3.12}$$

The last coefficient must be $-\frac{g}{2}$ since the acceleration must be constant and equal $-g$ throughout the (ideal) flip phase of the trajectory. Moreover, the other coefficients can be computed given an initial position z_4 at $t = 0$ and a final position z_5 when $t = t_4$:

$$\begin{cases} Z_4(0) = a_{z_4,0} = z_4 \\ Z_4(t_4) = z_4 + a_{z_4,1}t_4 - \frac{g}{2}t_2^2 = z_5 \Rightarrow a_{z_4,1} = \frac{z_5 - z_4}{t_4} + \frac{gt_4}{2} \end{cases} \tag{3.13}$$

Thus, the resulting trajectory on the z coordinate ($z(t)$) for going from z_4 to z_5 in the time span of t_4, with an ideal acceleration of $-g$ throughout the trajectory, can be expressed as:

$$Z_4(t) = z_4 + \left(\frac{z_5 - z_4}{t_4} + \frac{gt_4}{2}\right)t - \frac{g}{2}t^2 \tag{3.14}$$

- For the ϕ coordinate:

  A similar approach can be implemented for $\phi(t)$, which is easier to obtain since only two coefficients are required to be computed in this case:

$$\Phi_4(t) = a_{\phi_4,0} + a_{\phi_4,1}t = \phi_4 t(0) + \dot{\phi}_2(0)t \tag{3.15}$$

  Assuming that the desired initial angle ϕ_4 and desired final angle ϕ_5 are known for the flip phase, the coefficients of equation (3.15) can be computed as follows:

$$\begin{cases} \Phi_4(0) = a_{\phi_4,0} = \phi_4 \\ \Phi_4(t_4) = \phi_4 + a_{\phi_4,1}t_4 = \phi_5 \Rightarrow a_{\phi_4,1} = \frac{\phi_5 - \phi_4}{t_4} \end{cases} \tag{3.16}$$

  So, the trajectory of the ϕ coordinate throughout the flip phase can be expressed as follows:

$$\Phi_4(t) = \phi_4 + \left(\frac{\phi_5 - \phi_4}{t_4}\right)t \tag{3.17}$$

  Finally, for $Y_4(t)$, the resulting trajectory can be computed using equations (3.2) and (3.4).

Reaching Phase

The first phase of the overall trajectory is called the reaching phase, since the goal of the reaching phase is to reach a set of final conditions, which will be the initial conditions for the $Z(t)$ and $\Phi(t)$ trajectories of the flip phase. Moreover, the structure of the trajectories chosen for $Z(t)$ and $\Phi(t)$ for the reaching phase and the recovery phase, is polynomials of order 9, since they provide continuity up to the snap and smoothness for the lower derivatives such as the velocity and the acceleration. The general way for computing the coefficients for polynomials of order n, given $n+1$ boundary conditions has been presented in subsection 3.1.3. For this reason, the focus now will be on choosing the initial and final conditions of the polynomials. In addition, the trajectories in the reaching phase and the recovery phase were separated into three different parts in order to give more freedom to the solver. This will make it easier to find a flip trajectory that satisfies the physical constraints of the used quadrotor. As a result, the overall flip trajectory will consist of 7 parts.

- For the z coordinate:

  The goal is to start from a hovering position and reach the initial conditions for the flip phase on the given time span $t_{reachingphase} = t_1 + t_2 + t_3$. When $t = 0$, the boundary conditions are given by a hovering configuration at a certain height z_1, with all the derivatives set to zero:

$$Z_1 = \begin{bmatrix} z_1(0) & \dot{z}_1(0) & \ddot{z}_1(0) & z_1^{(3)}(0) & z_1^{(4)}(0) \end{bmatrix} = \begin{bmatrix} z_1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.18}$$

  Then, the two intermediate waypoints in the reaching phase will be free to take on any value so that they can be properly computed by the solver:

$$Z_2 = \begin{bmatrix} z_2(0) & \dot{z}_2(0) & \ddot{z}_2(0) & z_2^{(3)}(0) & z_2^{(4)}(0) \end{bmatrix} = \begin{bmatrix} z_2 & \dot{z}_2 & \ddot{z}_2 & 0 & 0 \end{bmatrix} \tag{3.19}$$

$$Z_3 = \begin{bmatrix} z_3(0) & \dot{z}_3(0) & \ddot{z}_3(0) & z_3^{(3)}(0) & z_3^{(4)}(0) \end{bmatrix} = \begin{bmatrix} z_3 & \dot{z}_3 & \ddot{z}_3 & 0 & 0 \end{bmatrix} \tag{3.20}$$

So, for now, the first trajectory in the reaching phase can be generated by using equation (3.18) as the initial condition, and equation (3.19) as the final condition. Moreover, the second trajectory in the reaching phase can be generated by using equation (3.19) as the initial condition, and equation (3.20) as the final condition.

Finally, the final configuration that has to be attained at the end of the reaching phase must match the initial conditions of the flip phase, specifically:

$$Z_4 = \begin{bmatrix} z_4(0) & \dot{z}_4(0) & \ddot{z}_4(0) & z_4^{(3)}(0) & z_4^{(4)}(0) \end{bmatrix} = \begin{bmatrix} z_4 & \left(\frac{z_5 - z_4}{t_4} + \frac{g t_4}{2}\right) & -g & 0 & 0 \end{bmatrix} \tag{3.21}$$

Thus, the third and final trajectory in the reaching phase can be generated using equation (3.20) as the initial condition, and equation (3.21) as the final condition.

- For the ϕ coordinate:

  Similarly, at the start of the reaching phase, the roll angle and all its successive derivatives must be zero. This is because the quadrotor is in a hovering position initially:

$$\Phi_1 = \begin{bmatrix} \phi_1(0) & \dot{\phi}_1(0) & \ddot{\phi}_1(0) & \phi_1^{(3)}(0) & \phi_1^{(4)}(0) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.22}$$

  Then, the two intermediate waypoints in the reaching phase will be free to take on any value so that they can be properly computed by the solver:

$$\Phi_2 = \begin{bmatrix} \phi_2(0) & \dot{\phi}_2(0) & \ddot{\phi}_2(0) & \phi_2^{(3)}(0) & \phi_2^{(4)}(0) \end{bmatrix} = \begin{bmatrix} \phi_2 & \dot{\phi}_2 & \ddot{\phi}_2 & 0 & 0 \end{bmatrix} \tag{3.23}$$

$$\Phi_3 = \begin{bmatrix} \phi_3(0) & \dot{\phi}_3(0) & \ddot{\phi}_3(0) & \phi_3^{(3)}(0) & \phi_3^{(4)}(0) \end{bmatrix} = \begin{bmatrix} \phi_3 & \dot{\phi}_3 & \ddot{\phi}_3 & 0 & 0 \end{bmatrix} \tag{3.24}$$

  Similarly, the first trajectory in the reaching phase can be generated by using equation (3.22) as the initial condition, and equation (3.23) as the final condition. Moreover, the second trajectory in the reaching phase can be generated by using equation (3.23) as the initial condition, and equation (3.24) as the final condition.

  Furthermore, the final configuration that has to be attained at the end of the reaching phase must match the initial conditions of the flip phase, specifically:

$$\Phi_4 = \begin{bmatrix} \phi_4(0) & \dot{\phi}_4(0) & \ddot{\phi}_4(0) & \phi_4^{(3)}(0) & \phi_4^{(4)}(0) \end{bmatrix} = \begin{bmatrix} \phi_4 & \left(\frac{\phi_5 - \phi_4}{t_4}\right) & 0 & 0 & 0 \end{bmatrix} \tag{3.25}$$

  Thus, the third and final trajectory in the reaching phase can be generated using equation (3.24) as the initial condition, and equation (3.25) as the final condition.

  Finally, the trajectories along the y coordinate are obtained from equation (3.2) in each case.

Recovery Phase

The objective of the final phase of the flip trajectory is to allow the quadrotor to recover from the flip phase and avoid it from crashing. The method used to compute this part of the trajectory is completely identical to the method used in the reaching phase.

- For the z coordinate:

  The goal is to bring back the quadrotor from the configuration at the end of the flip phase to a safe hovering position. As a result, the initial condition for the recovery phase is just the final condition of the flip phase:

$$Z_5 = \begin{bmatrix} z_5(0) & \dot{z}_5(0) & \ddot{z}_5(0) & z_5^{(3)}(0) & z_5^{(4)}(0) \end{bmatrix} = \begin{bmatrix} z_5 & \left(\frac{z_5 - z_4}{t_4} - \frac{g t_4}{2}\right) & -g & 0 & 0 \end{bmatrix} \tag{3.26}$$

  Then, the two intermediate waypoints in the reaching phase will be free to take on any value so that they can be properly computed by the solver:

$$Z_6 = \begin{bmatrix} z_6(0) & \dot{z}_6(0) & \ddot{z}_6(0) & z_6^{(3)}(0) & z_6^{(4)}(0) \end{bmatrix} = \begin{bmatrix} z_6 & \dot{z}_6 & \ddot{z}_6 & 0 & 0 \end{bmatrix} \tag{3.27}$$

$$Z_7 = \begin{bmatrix} z_7(0) & \dot{z}_7(0) & \ddot{z}_7(0) & z_7^{(3)}(0) & z_7^{(4)}(0) \end{bmatrix} = \begin{bmatrix} z_7 & \dot{z}_7 & \ddot{z}_7 & 0 & 0 \end{bmatrix} \tag{3.28}$$

  So, for now, the first trajectory in the recovery phase can be generated by using equation (3.26) as the initial condition, and equation (3.27) as the final condition. Moreover, the second trajectory in the recovery phase can be generated by using equation (3.27) as the initial condition, and equation (3.28) as the final condition.

  Finally, the final configuration that has to be attained at the end of the recovery phase is just a hovering position:

$$Z_8 = \begin{bmatrix} z_8(0) & \dot{z}_8(0) & \ddot{z}_8(0) & z_8^{(3)}(0) & z_8^{(4)}(0) \end{bmatrix} = \begin{bmatrix} z_8 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.29}$$

  Thus, the third and final trajectory in the recovery phase can be generated using equation (3.28) as the initial condition, and equation (3.29) as the final condition.

- For the ϕ coordinate:

  The same approach can be carried out for the roll angle coordinate ϕ along the recovery phase. At the end of the flip phase, the quadrotor should have the following configuration:

$$\Phi_5 = \begin{bmatrix} \phi_5(0) & \dot{\phi}_5(0) & \ddot{\phi}_5(0) & \phi_5^{(3)}(0) & \phi_5^{(4)}(0) \end{bmatrix} = \begin{bmatrix} \phi_5 & \left(\frac{\phi_5 - \phi_4}{t_4}\right) & 0 & 0 & 0 \end{bmatrix} \tag{3.30}$$

  Then, the two intermediate waypoints in the reaching phase will be free to take on any value so that they can be properly computed by the solver:

$$\Phi_6 = \begin{bmatrix} \phi_6(0) & \dot{\phi}_6(0) & \ddot{\phi}_6(0) & \phi_6^{(3)}(0) & \phi_6^{(4)}(0) \end{bmatrix} = \begin{bmatrix} \phi_6 & \dot{\phi}_6 & \ddot{\phi}_6 & 0 & 0 \end{bmatrix} \tag{3.31}$$

$$\Phi_7 = \begin{bmatrix} \phi_7(0) & \dot{\phi}_7(0) & \ddot{\phi}_7(0) & \phi_7^{(3)}(0) & \phi_7^{(4)}(0) \end{bmatrix} = \begin{bmatrix} \phi_7 & \dot{\phi}_7 & \ddot{\phi}_7 & 0 & 0 \end{bmatrix} \tag{3.32}$$

So, for now, the first trajectory in the recovery phase can be generated by using equation (3.30) as the initial condition, and equation (3.31) as the final condition. Moreover, the second trajectory in the recovery phase can be generated by using equation (3.31) as the initial condition, and equation (3.32) as the final condition.

At the end of the overall flip trajectory, the roll angle ϕ must be equal to 2π at the final hovering position:

$$\Phi_8 = \begin{bmatrix} \phi_8(0) & \dot{\phi}_8(0) & \ddot{\phi}_8(0) & \phi_8^{(3)}(0) & \phi_8^{(4)}(0) \end{bmatrix} = \begin{bmatrix} 2\pi & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.33}$$

Thus, the third and final trajectory in the recovery phase can be generated using equation (3.32) as the initial condition, and equation (3.33) as the final condition.

Theoretically, multi-flip trajectories can be generated using this methodology by simply replacing 2π by $2n\phi$, with n being the number of flips to be performed during the trajectory. However, for the sake of simplicity, a feasible single flip trajectory will be generated.

3.1.5 Parameter Optimization and Trajectory Generation

The main difficulty of the trajectory planning method described above is to properly define the set of parameters which will allow to completely determine the flip trajectory. In fact, there exist 37 parameters in total that must be chosen in order to generate the flip trajectory. And, they are:

- $z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8$
- $\dot{z}_2, \dot{z}_3, \dot{z}_6, \dot{z}_7$
- $\ddot{z}_2, \ddot{z}_3, \ddot{z}_6, \ddot{z}_7$
- $\phi_2, \phi_3, \phi_4, \phi_5, \phi_6, \phi_7$
- $\dot{\phi}_2, \dot{\phi}_3, \dot{\phi}_6, \dot{\phi}_7$
- $\ddot{\phi}_2, \ddot{\phi}_3, \ddot{\phi}_6, \ddot{\phi}_7$
- $t_1, t_2, t_3, t_4, t_5, t_6, t_7$

So, an optimization problem has to be defined in order to compute the proper value for each of the parameters above and generate a dynamically feasible flip trajectory. First, a parameter vector $\boldsymbol{p}$ containing all the parameters above must be created as follows:

$$\boldsymbol{p} = \begin{bmatrix} z_1 & \dots & t_7 \end{bmatrix} \tag{3.34}$$

So, the variables of the optimization problem is the elements of vector $\boldsymbol{p}$. In addition, a widely used objective function that can be used is the integral of motor torques squared:

$$J(\boldsymbol{p}) = \int_0^t u_1^2 \tag{3.35}$$

with $t = t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7$.
The objective function in equation (3.35) will try to minimize the applied thrust of the quadrotor throughout the trajectory.

Naturally, most of the parameters will have lower and upper bound constraints on them. And, they will be taken into consideration in the optimization problem. For example, the z_i parameters will have a minimum bound of $0.8m$ and an upper bound of $3m$. Having these bounds will allow the solver to generate trajectories with bounded heights that are relatively small. In general, the constraints on the vector of parameters $\boldsymbol{p}$ can be expressed as follows:

$$\boldsymbol{p}_{min} \leq \boldsymbol{p} \leq \boldsymbol{p}_{max} \tag{3.36}$$

In addition, there exist a set of nonlinear inequality constraints in order to specify bounds on the actuators. Inequality constraints can be expressed as follows:

$$\boldsymbol{c}(\boldsymbol{p}) \leq \boldsymbol{0} \tag{3.37}$$

It should be noted that trajectories will be generated at each iteration during the optimization problem. Then the set of $\ddot{\boldsymbol{z}}$ and $\boldsymbol{\phi}$ values will be extracted from the trajectory and used to compute the total thrust vector $\boldsymbol{u}_1$, which contains the value of the total thrust throughout the trajectory using the expression of $\ddot{z}$ in equation (2.10), which can be rewritten as follows:

$$\boldsymbol{u}_1 = \frac{m(\ddot{\boldsymbol{z}} + \boldsymbol{g})}{\cos \boldsymbol{\phi}} \tag{3.38}$$

However, it should be noted that the torque u_2 was not taken into consideration in the optimization problem, since no solution could be found when it was considered. As a result, a very small weight will be assigned to the torque reference when performing the flip trajectory in the planar case. As for the 3D quadrotor case, the torque is not a control input, since the angular rates are used as control inputs instead. So, the torque reference will not be used at all. And, it will be demonstrated that better results will be obtained in the 3D quadrotor case. Furthermore, the following set of inequality constraints were added to help the solver generate trajectories that are concave:

$$z_1 < z_2 \tag{3.39}$$
$$z_2 < z_3 \tag{3.40}$$
$$z_3 < z_4 \tag{3.41}$$
$$z_5 < z_4 \tag{3.42}$$
$$z_6 < z_5 \tag{3.43}$$
$$z_7 < z_6 \tag{3.44}$$
$$z_8 < z_7 \tag{3.45}$$

In conclusion, the overall parameter optimization problem can be expressed as follows:

$$\begin{aligned} \min_{\boldsymbol{p}} \quad & J(\boldsymbol{p}) = \int_0^t u_1^2 \\ \text{s.t.} \quad & \boldsymbol{p}_{min} \leq \boldsymbol{p} \leq \boldsymbol{p}_{max} \\ & \boldsymbol{c}(\boldsymbol{p}) \leq \boldsymbol{0} \end{aligned} \tag{3.46}$$

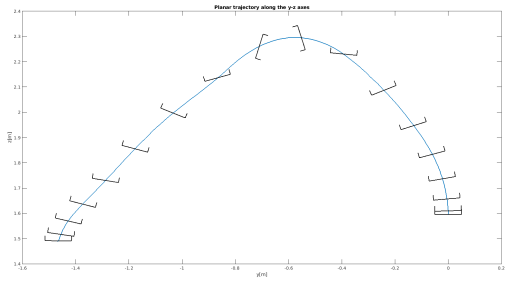
Different algorithms that are implemented in `MATLAB` were tested to find the best solution for this optimization problem, such as `fmincon`, `ga` (genetic algorithm) and `patternsearch`. However, the most promising solution was found using `fmincon`.

At the the of the optimization process, `fmincon` outputs the following message:

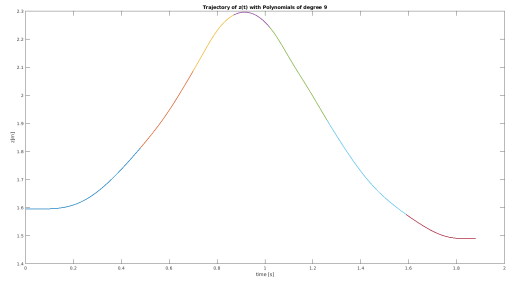
```
                                                First-order      Norm of
 Iter F-count            f(x)  Feasibility   optimality         step
  211     8785    7.646190e-02    0.000e+00    2.288e-02    2.199e-11
  212     8826    7.646190e-02    0.000e+00    2.288e-02    1.605e-13

Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.
```

Figure 3.1: Output of fmincon at the end of the optimization process

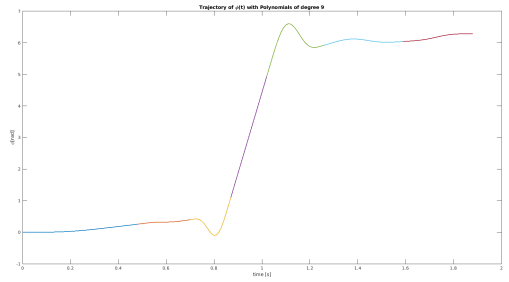
As can be seen from figure 3.1, `fmincon` has stopped because it has found a local minimum that satisfies the constraints to within the default constraint tolerance. Moreover, the solution found for the parameters in vector $\boldsymbol{p}$ result in the following trajectory:



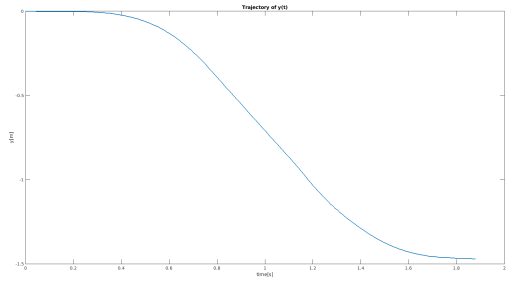
(a) Trajectory along the $y-z$ plane



(b) Trajectory along z.



(c) Trajectory along ϕ.



(d) Trajectory along y.

Figure 3.2: Resulting trajectory from the solution of the optimization problem using `fmincon`.

3.2 Simulations using acados

In this section, simulations in acados will be performed on the planar quadrotor and the 3D quadrotor in order to track the generated flip trajectory.

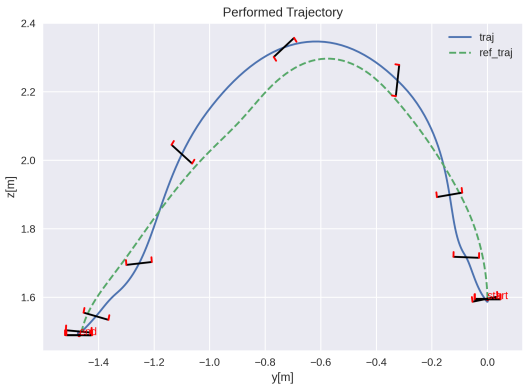
3.2.1 Planar Quadrotor Simulations

Noiseless Measurements

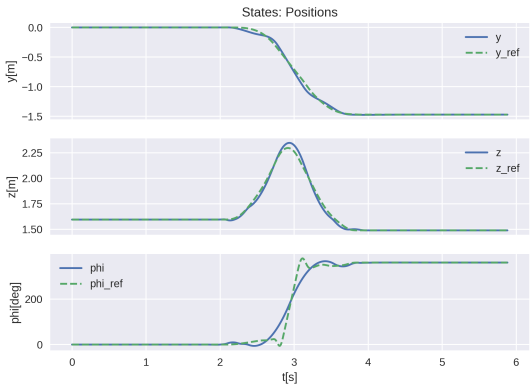
As can be seen in figure 3.3, the MPC controller was able to output control inputs that allowed the quadrotor to track the flip trajectory with minimal errors. By looking at the control inputs in subfigure 3.3d, there are some dips in the thrust throughout the trajectory. However, it is clear that the thrust never reaches the value of 0. Whereas for the the torque, the desired torque is much larger than the torque that can be applied by the quadrotor, and it is not able to be tracked. Despite this issue, the MPC controller was still able to generate control inputs that allowed the quadrotor to track the flip trajectory with minimal errors, which is evident by looking at the plots of the position states in subfigure 3.3b. Finally, by looking at the position error plots in subfigure 3.3e, the largest errors appear to be on the ϕ coordinate. However, this error indicates the the roll angle reference was changing at a very fast rate during the flip phase. But, this error directly converges to 0 after the quadrotor performs the flip. In addition, the errors on the y and z coordinates are relatively small.

Noiseless Measurements and Control Inputs

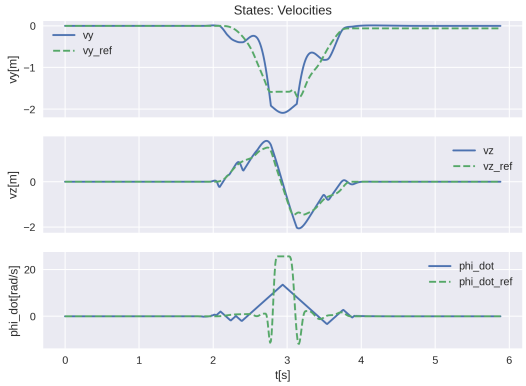
As can be seen in figure 3.4, it is evident that despite the noise on the measurements and the control inputs, the MPC controller was still able to output control inputs that allowed the quadrotor to track the flip trajectory at the cost of larger errors, which were minimized by the use of the extended Kalman filter. By looking at the control inputs in subfigure 3.4d, the thrust input oscillates throughout the trajectory, even at the initial hovering phase of 2 seconds. This is the result of the added noise on both the state measurement vector and the control inputs. Finally, by looking at the position error plots in subfigure 3.4e, it is clear that there are more errors on y and z when compared to the noiseless simulation. However, the extended Kalman filter was able to reduce this error along the y coordinate of the trajectory.



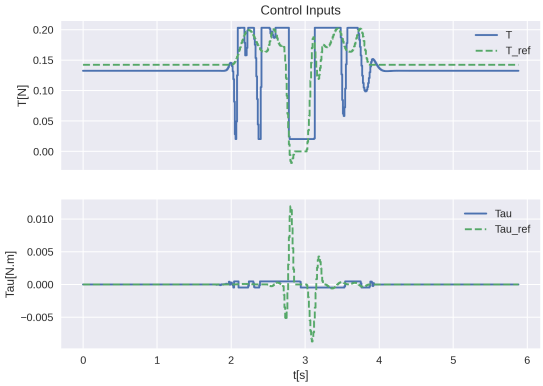
(a) Trajectory followed by the planar quadrotor



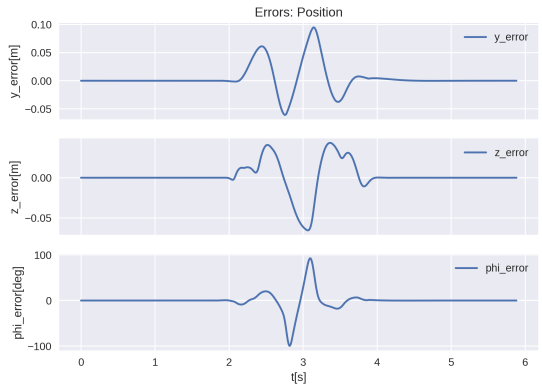
(b) Position states of the planar quadrotor.



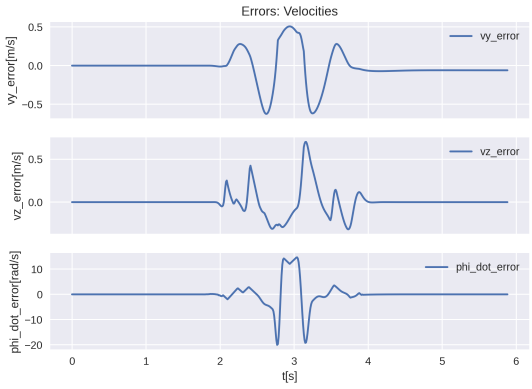
(c) Velocity states of the planar quadrotor.



(d) Control inputs of the planar quadrotor

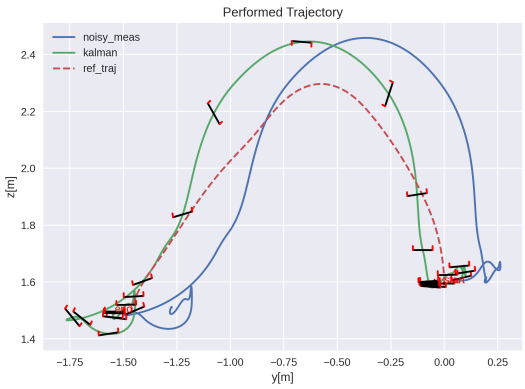


(e) Errors on y, z and ϕ.

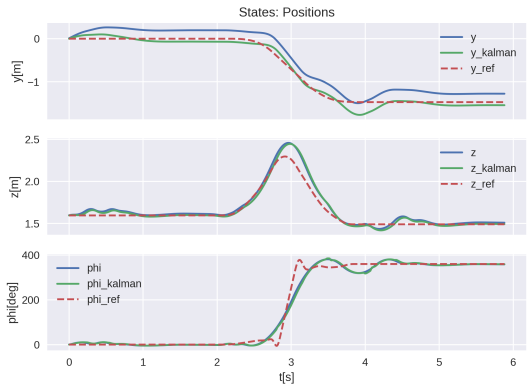


(f) Errors on v_y, v_z and $\dot{\phi}$.

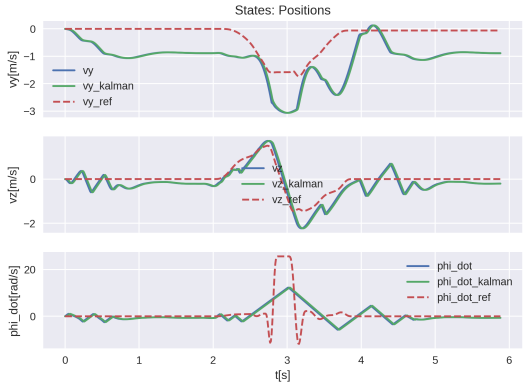
Figure 3.3: Simulation of the planar quadrotor tracking a flip trajectory without noise.



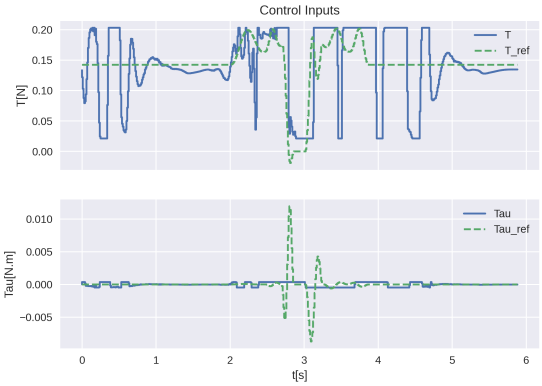
(a) Trajectory followed by the planar quadrotor



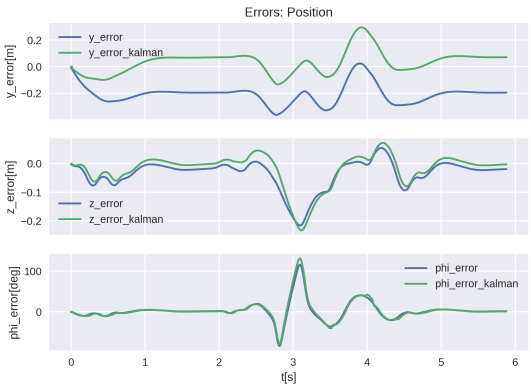
(b) Position states of the planar quadrotor.



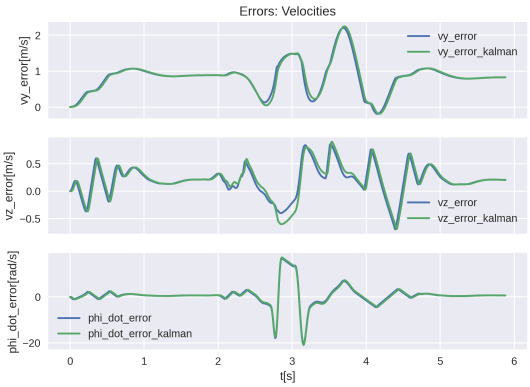
(c) Velocity states of the planar quadrotor.



(d) Control inputs of the planar quadrotor



(e) Errors on y, z and ϕ.

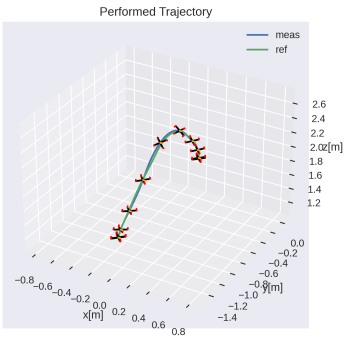


(f) Errors on v_y, v_z and $\dot{\phi}$.

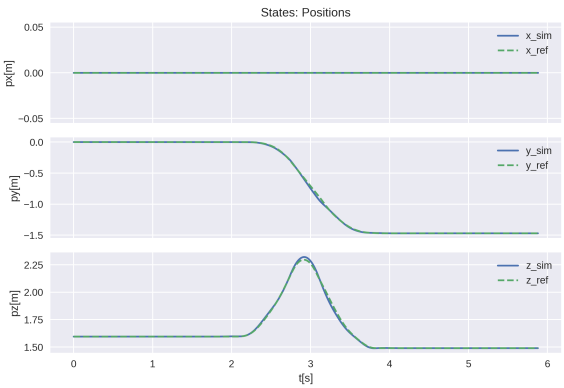
Figure 3.4: Simulation of the planar quadrotor tracking a flip trajectory with noise.

3.2.2 3D Quadrotor Simulation

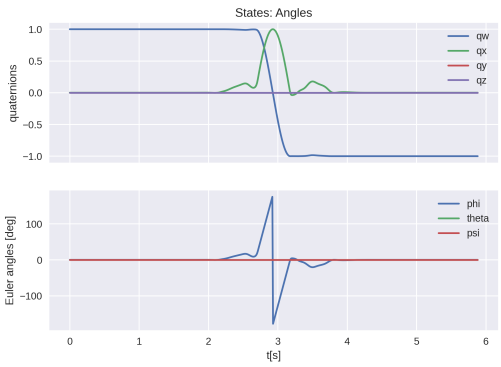
In the case of the 3D quadrotor, simulations will only be performed without noise since the quadrotor that will be used for the experimentation already has an extended Kalman filter implementation.



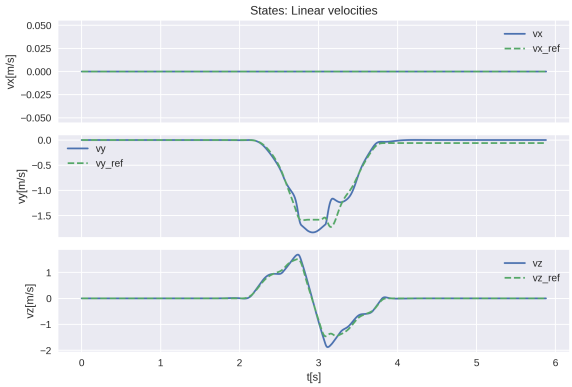
(a) Trajectory followed by the 3D quadrotor



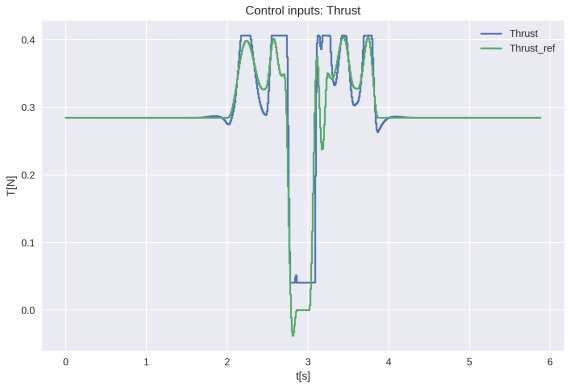
(b) Position states.



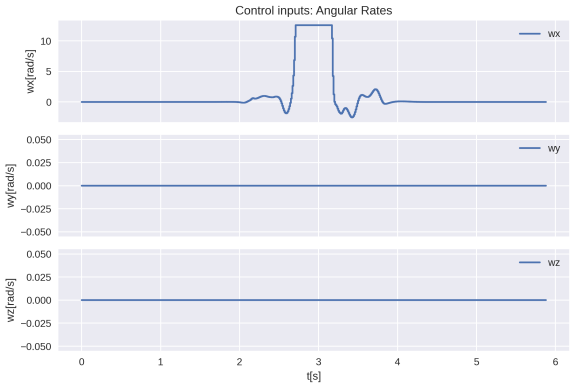
(c) Angular states.



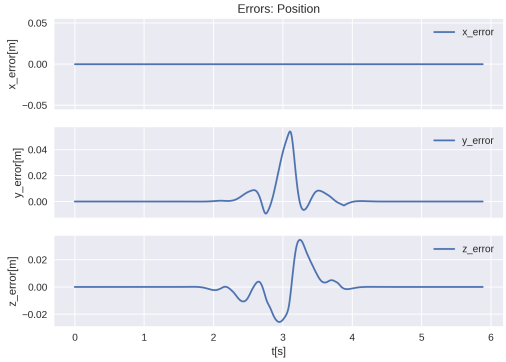
(d) Velocity states.



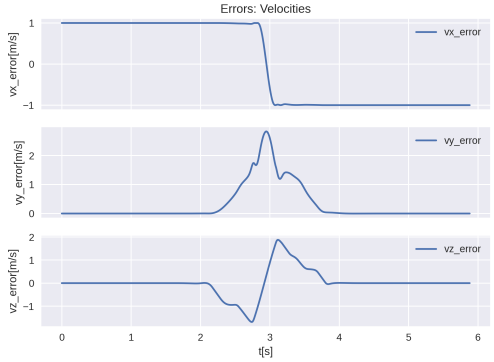
(e) Thrust input.



(f) Angular rates inputs.



(g) Errors on x, y and z.



(h) Errors on v_x, v_y and v_z.

Figure 3.5: Simulation of the 3D quadrotor tracking a flip trajectory without noise.

As can be seen from figure 3.5, the 3D quadrotor was able to follow the flip trajectory almost perfectly with very minimal errors on the states. By looking at subfigure 3.5e, the MPC controller tried to follow the open-loop control input reference of the thrust as much as possible, and made the required adjustments when it was needed. In addition, the thrust never reached the value of 0 which was the thrust reference during the flip phase. However, the quadrotor was still able to track the flip trajectory with very minimal errors due to the robustness of the MPC controller. Furthermore, by looking at subfigure 3.5f, the MPC controller was not provided references for the angular rates. And, in order to perform the flip, the angular rate along x (ω_x) was assigned a weight of 10^{-3}, which made the goal of minimizing ω_x much less important than the goal of following the flip trajectory for the MPC controller. As a result, the MPC controller was free to assign ω_x any value within the limits of $[-4\pi, 4\pi]$, which allowed the quadrotor to follow the flip trajectory with very minimal errors. Finally, by looking at subfigure 3.5g, it is evident that the errors along the y and z coordinates are very small. In conclusion, the flip trajectory was much easier to follow in the 3D quadrotor case when compared to the planar quadrotor case.

Chapter 4

Software-in-the-Loop Simulations in ROS2/Gazebo and Experimentation

In this chapter, the Software-in-the-Loop control structure will be presented along with the ROS2 MPC controller node and the simulation results. Finally, the ROS2 MPC controller node that is used in the experimentation phase will be presented and the experimentation results will be shown.

4.1 SIL Simulations using ROS2/Gazebo

The Software-in-the-Loop simulator package `ls2n_drone_ros2`[1]was provided by the LS2N lab engineer Dr. Damien Six[2]. Moreover, the MPC controller package was created on a separate branch. Interested readers are referred to the `ls2n_crazyflie_nmpc` package[3]on the `elie-dev` branch in order to have a look at the implemention of the MPC controller in the ROS2 node.

4.1.1 Closed-Loop Control Structure of the Software-in-the-Loop

In order to have a better representation of how the system will behave, a Software-in-the-Loop simulation must be performed. Software-in-the-Loop (SIL) represents the incorporation of compiled production source code into a mathematical model simulation. As a result, it will provide a practical, virtual simulation environment for the development and evaluation of detailed control strategies for large and complex systems [72]. So, the closed-loop control structure in this case becomes:

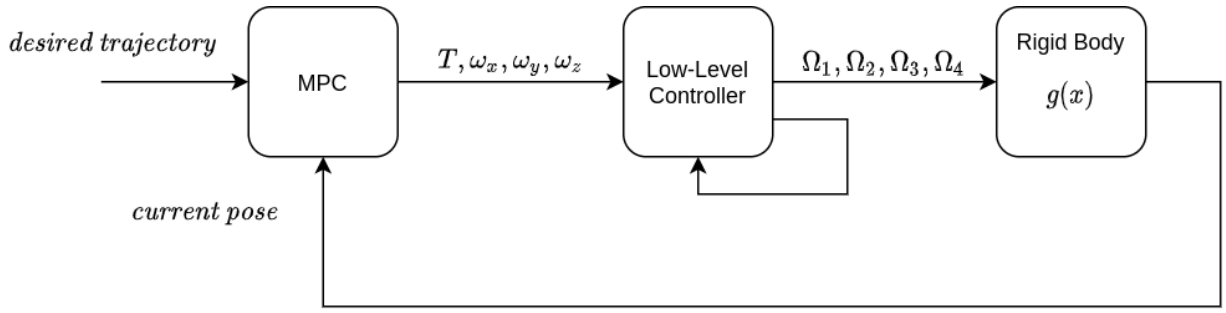


Figure 4.1: Diagram of the resulting non-centralized MPC closed-loop control structure.

[1] `https://gitlab.univ-nantes.fr/ls2n-drones/ls2n_drone_ros2`, accessed on 23/08/2021.

[2] `https://scholar.google.fr/citations?user=3-YYgFAAAAAJ&hl=en`, accessed on 23/08/2021.

[3] `https://gitlab.univ-nantes.fr/ls2n-drones/ls2n_drone_ros2/-/tree/elie_dev/ls2n_crazyflie_nmpc`, accessed on 23/08/2021.

As shown in figure 4.1, the MPC controller will solve an optimization problem at each iteration, and will output the total thrust T, and the angular rates ω_x, ω_y and ω_z, which will be fed to the Low-Level Controller. Then, the Low-Level Controller will map the angular rates to the body torques. Afterwards, the total thrust T and the body torques $\boldsymbol{\tau}$ will then be converted to motor speeds, which will be outputted to the quadrotor in order to track the desired trajectory. However, it should be noted that the Low-Level Controller that is used in the Software-in-the-Loop simulations is the PX4 Autopilot Low-Level Controller. The PX4 Autopilot [73] is an open-source system which is aimed towards inexpensive autonomous aircraft. Subsequently, the results will still not be totally accurate since the Low-Level Controller that is used in the crazyflie 2.1 is different than the PX4 Low-Level Controller.

4.1.2 Design of the ROS2 MPC Controller Node

In order to send the MPC control inputs to the Low-Level Controller and control the quadrotor, a ROS2 node has been created.

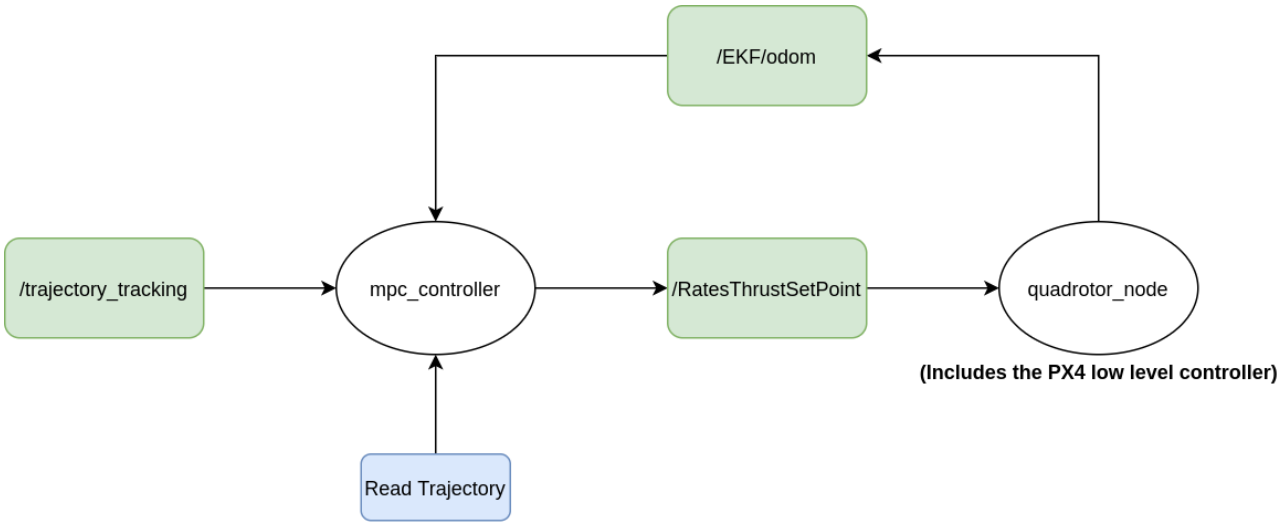


Figure 4.2: Diagram showing how the MPC node and the quadrotor node communicate with each other.

As can be seen from figure 4.2, the `mpc_controller` node is subscribed to the `/trajectory_tracking` topic which transfers Boolean messages to the `mpc_controller`. And, initially the Boolean variable inside the `mpc_controller` node is set to `False`. The `mpc_controller` node is also subscribed to the `/EKF/odom` topic which receives the odometry messages published by the `quadrotor_node` at a rate of 100Hz. In addition, the `mpc_controller` publishes its control inputs messages to the `/RatesThrustSetPoint` topic, for which the `quadrotor_node` is subscribed to. So, `quadrotor_node` receives the control inputs message from the `/RatesThrustSetPoint` topic and the PX4 Low-Level Controller will output the desired motor speeds to control the behavior of the quadrotor.

The `mpc_controller` node will first read and store the desired trajectory. Then, it will take the first waypoint of the desired trajectory and it will use it as a reference position. Afterwards, it will wait until an odometry message is received from the `/EKF/odom` topic. As soon as an odometry message is received, it will take the current pose of the quadrotor from the odometry message, and it will use it to compute the desired control inputs to reach the desired initial waypoint of the trajectory. Subsequently, it will publish the control inputs via a custom message to the `/RatesThrustSetPoint` topic. Finally, in order to track the desired trajectory, a Boolean

message with a value set to `True` will be manually published to the `/trajectory_tracking` topic via the terminal.This will allow the MPC controller to take the next N waypoints of the stored desired trajectory and calculate the control inputs at each iteration in order to properly track the trajectory.

Morever, the MPC controller is the same as the one designed in subsection 2.3.3. However, in order to ensure that the computations of the MPC controller can keep up with the odometry messages which are published at 100Hz, the MPC parameters were changed to the following:

- $N = 50$: number of discretization steps in the prediction horizon.
- $T_f = 0.5s$: prediction horizon.

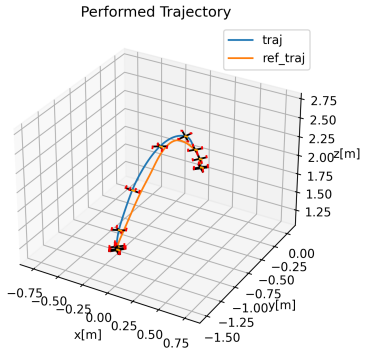
This is to ensure that the MPC controller will still be publishing at a rate of 100Hz. Furthermore, it should be noted that the desired trajectory must be synchronized with the MPC controller. In other words it must also be discretized at a rate of 100Hz (the time instant between each waypoint in the trajectory should be $0.01s$).

Furthermore, the running, terminal and input weights were retuned:

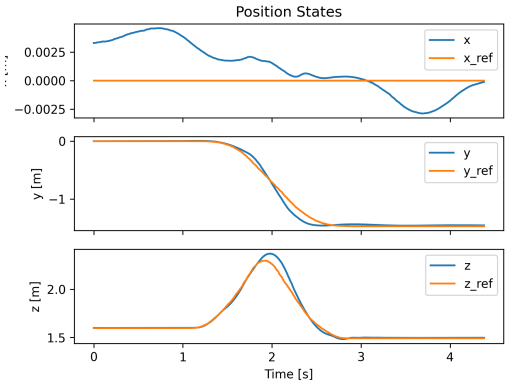
- $\boldsymbol{Q} = diag(\begin{bmatrix}50 & 50 & 50 & 10 & 10 & 10 & 10 & 1 & 1 & 1\end{bmatrix})$
- $\boldsymbol{Q}_e = diag(\begin{bmatrix}50 & 50 & 50 & 10 & 10 & 10 & 10 & 1 & 1 & 1\end{bmatrix})$
- $\boldsymbol{R} = diag(\begin{bmatrix}1 & 10^{-1} & 1 & 1\end{bmatrix})$

4.1.3 SIL Simulation Results

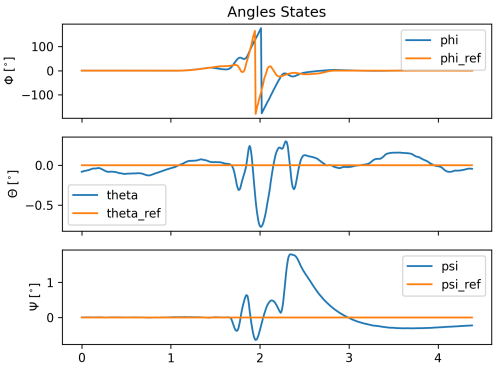
In order to analyze the data of the Software-in-the-Loop simulation, ROS2 bags were used to record the data during the simulation. Then, the relevant parts of the data were extracted and plotted. The results obtained from the SIL simulation are presented in figure 4.3 in which it is evident that the quadrotor was able to track the flip trajectory with small errors along the y and z coordinates. Moreover, by looking at subfigure 4.3e, similarly to the acados simulations, the MPC controller tried to track the open-loop control input reference of the thrust as much as possible, and made the required adjustments when it was needed. Furthermore, the thrust never reached the value of 0 which was the thrust reference during the flip phase. However, the quadrotor was still able to track the flip trajectory with small errors due to the robustness of the MPC controller. In addition, by looking at subfigure 3.5f, the MPC controller was not provided references for the angular rates. And, in order to perform the flip, the angular rate along x (ω_x) was assigned a weight of 10^{-1}, which made the goal of minimizing ω_x less important than the goal of following the flip trajectory for the MPC controller. However, it should be noted that the weight assigned to ω_x was slightly higher than the weight assigned to it during the acados simulation in subsection 3.2.2. This because during the SIL simulation, having a very small weight on ω_x resulted in a very unstable flight even while just hovering. This is the case because minimizing ω_x was not important for the MPC controller, which resulted in higher values of ω_x even at a hovering position, which destabilized the quadrotor. For this reason, the weight on ω_x was slightly increased in order to provide stability during flight, while the MPC is still free to give it high values in order to properly track the flip trajectory. Finally, by looking at subfigure 4.3g, it is evident that the largest errors along the y and z coordinates occur during the flip phase. Then, the errors are driven back to zero during the recovery phase.



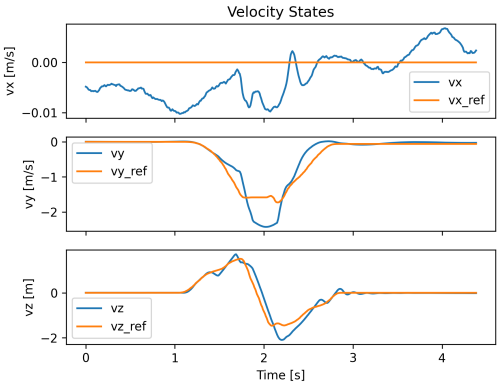
(a) Trajectory followed by the 3D quadrotor



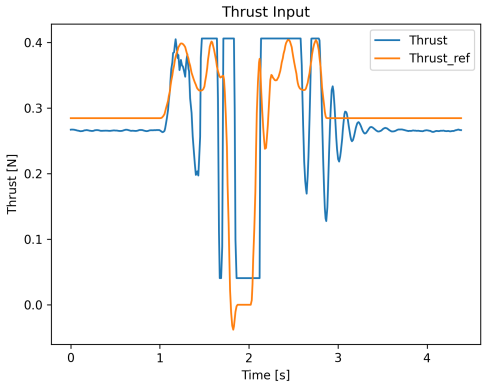
(b) Position states.



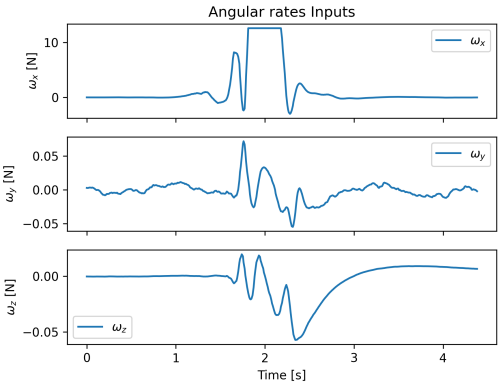
(c) Angular states.



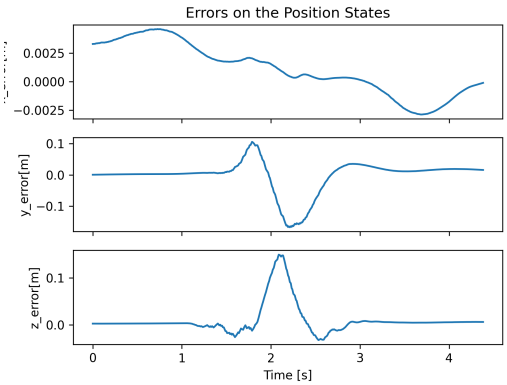
(d) Velocity states.



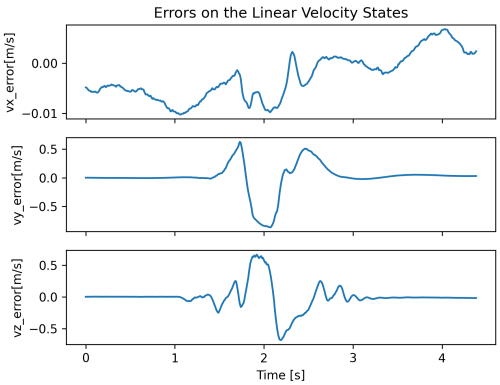
(e) Thrust input.



(f) Angular rates inputs.



(g) Errors on x, y and z.



(h) Errors on v_x, v_y and v_z.

Figure 4.3: Software-in-the-Loop simulation of a 3D quadrotor tracking a flip trajectory.

4.2 Experimentation

Conclusion

Appendix A

Transformation of the MPC Problem to a Quadratic Program

Proof. The problem statement in (1.9) can be converted into a more standard optimization problem as follows:

$$
\begin{aligned}
\mathbf{z}(k|k) &= C\mathbf{x}(k|k) \\
\mathbf{z}(k+1|k) &= C\mathbf{x}(k+1|k) = C(A\mathbf{x}(k|k) + B\mathbf{u}(k|k)) \\
&= CA\mathbf{x}(k|k) + CB\mathbf{u}(k|k) \\
\mathbf{z}(k+2|k) &= C\mathbf{x}(k+2|k) \\
&= C(A\mathbf{x}(k+1|k) + B\mathbf{u}(k+1|k)) \\
&= CA(A\mathbf{x}(k|k) + B\mathbf{u}(k|k)) + CB\mathbf{u}(k+1|k) \\
&= CA^2\mathbf{x}(k|k) + CAB\mathbf{u}(k|k) + CB\mathbf{u}(k+1|k) \\
&\vdots \\
\mathbf{z}(k+N|k) &= CA^N\mathbf{x}(k|k) + CA^{N-1}B\mathbf{u}(k|k) + \dots \\
&\quad + CB\mathbf{u}(k+(N-1)|k)
\end{aligned}
$$

The equations above can then be grouped as follows:

$$
\begin{bmatrix} \mathbf{z}(k|k) \\ \mathbf{z}(k+1|k) \\ \mathbf{z}(k+2|k) \\ \vdots \\ \mathbf{z}(k+N|k) \end{bmatrix} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^N \end{bmatrix} \mathbf{x}(k|k)
$$

$$
+ \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ CB & 0 & 0 & & 0 \\ CAB & CB & 0 & & 0 \\ \vdots & & & & \\ CA^{N-1}B & CA^{N-2}B & CA^{N-3}B & \dots & CB \end{bmatrix} \begin{bmatrix} \mathbf{u}(k|k) \\ \mathbf{u}(k+1|k) \\ \vdots \\ \mathbf{u}(k+N-1|k) \end{bmatrix}
$$

Now $Z(k)$ and $U(k)$ can be defined as follows:

$$Z(k) \equiv \begin{bmatrix} \mathbf{z}(k|k) \\ \vdots \\ \mathbf{z}(k+N|k) \end{bmatrix} \quad , \quad U(k) \equiv \begin{bmatrix} \mathbf{u}(k|k) \\ \vdots \\ \mathbf{u}(k+N-1|k) \end{bmatrix}$$

It should be noted that:

$$\sum_{j=0}^{N} \mathbf{z}(k+j|k)^\intercal R_{zz} \mathbf{z}(k+j|k) = Z(k)^\intercal W_1 Z(k)$$

with W_1 denoting the weighting matrix.
Thus, the elements of the cost function in (1.9) can now be expressed as follows:

$$\begin{aligned} Z(k)^\intercal W_1 Z(k) + U(k)^\intercal W_2 U(k) &= (G\mathbf{x}(k) + HU(k))^\intercal W_1 (G\mathbf{x}(k) + HU(k)) + U(k)^\intercal W_2 U(k) \\ &= \mathbf{x}(k)^\intercal H_1 \mathbf{x}(k) + H_2^\intercal U(k) + \frac{1}{2} U(k)^\intercal H_3 U(k) \end{aligned} \tag{A.1}$$

With

$$H_1 = G^\intercal W_1 G, \quad H_2 = 2(\mathbf{x}(k)^\intercal G^\intercal W_1 H), \quad H_3 = 2(H^\intercal W_1 H + W_2)$$

Since the term $\mathbf{x}(k)^\intercal H_1 \mathbf{x}(k)$ in (A.1) does not contain the component to be minimized $U(k)$, this means that it will be constant throughout the optimization process. As a result, it can be omitted. Finally, the MPC problem can be re-written as:

$$\begin{aligned} \min_{\mathrm{U}(k)} \quad & \tilde{J} = H_2^\intercal U(k) + \frac{1}{2} U(k) H_3 U(k) \\ \text{s.t.} \quad & \begin{bmatrix} I_N \\ -I_N \end{bmatrix} U(k) \leq u_m \end{aligned} \tag{A.2}$$

Thus, the MPC problem has been transformed to the form of a quadratic program for which there exists various efficient tools to solve the problem. □

Appendix B

Sequential Quadratic Programming and Real-Time Iterations in acados

B.1 Components of an Embedded SQP Algorithm

An embedded SQP algorithm must feature:

- Numerical integration of the continuous-time dynamics.
- Generation of first-order and possibly second-order sensitivities of objective and constraints.
- A procedure for approximating the Hessian matrix.
- An efficient QP solver.
- Initialization strategies for warm-starting the algorithm for the next iteration.

Moreover, the SQP algorithm in acados has the following form:

$$w^{[i+1]} = w^{[i]} + \Delta w^{QP} \tag{B.1}$$
$$\pi^{[i+1]} = \pi^{QP} \tag{B.2}$$
$$\lambda^{[i+1]} = \lambda^{QP} \tag{B.3}$$

Where:

- $w^{[i]} = [x_0^{[i]\intercal}, \ldots, u_0^{[i]\intercal}, \ldots, x_N^{[i]\intercal}]^\intercal$ is the primal iterate at SQP iteration i.
- π^i and λ^i are the dual iterates, readily available from the QP solution.

It should be noted that the algebraic variables are eliminated from the OCP, but numerical approximations of these values are accessible from the numerical integration routine.

Furthermore, the step Δw^{QP} is computed by solving the following QP:

$$
\begin{aligned}
\min_{\substack{\Delta x_0,\ldots,\Delta x_N \\ \Delta u_0,\ldots,\Delta u_{N-1} \\ \Delta s_o,\ldots,\Delta s_N}} \quad & \sum_{k=0}^{N-1} \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix}^\intercal \overbrace{\begin{bmatrix} Q_k & S_k^\intercal \\ S_k & R_k \end{bmatrix}}^{H_k} \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix} + \begin{bmatrix} q_k \\ r_k \end{bmatrix}^\intercal \begin{bmatrix} \Delta x_k \\ \Delta u_k \end{bmatrix} \\
& + \Delta x_N^\intercal Q_N \Delta x_N + q_N^\intercal \Delta x_n \\
& + \sum_{k=0}^{N} s_k^\intercal P_k s_k + p_k^\intercal s_k \qquad \text{(B.4)} \\
\text{s.t.} \quad & \Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k + \phi_k^x - x_{k+1}, k = 0, 1, \ldots, N-1, \\
& \Delta x_0 = \bar{x}_0 - x_0 \\
& - g_k \geq G_k^x \Delta x_k + G_k^u \Delta u_k + s_k, \quad k = 0, 1, \ldots, N-1, \\
& - g_N \geq g_N^x \Delta x_N + s_N, \\
& 0 \leq s_k, \quad k = 0, 1, \ldots, N.
\end{aligned}
$$

B.2 Numerical Integration and Sensitivities

In the family of single step integration methods, there are:

- **Explicit** methods: Easy to implement, they rely on a direct combination of explicit evaluations of the right-hand side of the system dynamics. Example: Runge-Kutta method of order 4.
- **Implicit** methods: More complex, it results in a nonlinear system which generally needs to be solved numerically using an iterative procedure such as a Newton-type method. And, when compared to explicit methods, implicit methods have improved numerical stability properties and higher order accuracy.

When using numerical integration methods within direct multiple shooting, the first (and possibly second) order derivatives of the simulation results with respect to the state and control input values also need to be computed:

$$
\frac{\partial \phi_k^x(x_k, u_k)}{\partial x_k}, \frac{\partial \phi_k^x(x_k, u_k)}{\partial u_k}, \sum_{i=1}^{n_x} \pi_{k,i} \frac{\partial^2 \phi_{k,i}^x(x_k, u_k)}{\partial^2 (x_k, u_k)}
$$

where $\pi_k \in \mathbb{R}^{n_x}$ is called the *seed vector* [74], for which the Lagrange multipliers are used to compute the exact Hessian of the Lagrangian.

B.3 Convex Hessian Approximation Methods

Gauss-Newton Hessian approximation In the case of a least-squares objective in (1.11):

- $l(x_k, u_k) = \|r(x_k, u_k)\|_2^2, \quad k = 0, \ldots, N-1$
- $M(x_N) = \|r_N(x_N)\|_2^2,$

with r: $\mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \Rightarrow \mathbb{R}^{n_r}$, r_N: $\mathbb{R}^{n_x} \Rightarrow \mathbb{R}^{n_{r_N}}$
This kind of residual function is a common case in embedded optimization.
The Gauss-Newton Hessian approximation amounts to [74]:

$$H^{GN} = blkdiag(H_0^{GN}, \ldots, H_N^{GN}) \tag{B.5}$$

at the linearization point $w^{[i]\intercal} = [x_0^{[i]\intercal}, \ldots, u_0^{[i]\intercal}, \ldots, x_N^{[i]\intercal}]^\intercal$, where the Hessian blocks are defined as:

$$H_k^{GN} = \left(\frac{\partial r}{\partial(x,u)}(x_k^i, u_k^i)\right)^\intercal \left(\frac{\partial r}{\partial(x,u)}(x_k^i, u_k^i)\right), \quad k = 0, \ldots, N-1 \tag{B.6}$$

$$H_N^{GN} = \left(\frac{\partial r_N}{\partial x}(x_N^i)\right)^\intercal \left(\frac{\partial r_N}{\partial x}(x_N^i)\right) \tag{B.7}$$

Moreover, the Gauss-Newton Hessian approximation offers a competitive alternative to SQP with exact Hessians, but it only converges linearly. And, for a quadratic objective function in (1.11), the same quadratic objective arises in (B.4), and no additional computations are needed. There exist other Hessian approximation methods in `acados` and interested readers are referred to [74] for the full description of the other methods.

B.4 Structure-exploiting embedded QP solvers

There exist different strategies to solve for QP (B.4) in `acados` [74]:

- **Sparse approach**: OCP (B.4) can be solved directly by using a general-purpose sparse QP solver.
  - Example: OOQP(primal-dual interior point solver), OSQP(first order method).
- **Structured approach**: OCP (B.4) can be solved by exploiting its multi-stage structure, but dense linear algebra is used.
  - Example: HPMPC, HPIPM (interior-point solvers).
- **Condensing approach**: It uses the dynamics to eliminate the states from the decision variables.
  - The state variables are expressed as an explicit function of the current state and future control inputs.
  - A smaller QP is then obtained with only the control inputs and (possibly) the initial state as optimization variables.
- **Partial condensing approach**: It is a mix between structured and condensing, but only per blocks of N/N_2 (with N an integer multiple of N_2), and N_2 the new horizon length of the partially condensed problem.
  - This approach allows finding a better trade-off between horizon length and number of optimization variables for a given problem.

B.5 Real-time iterations

A downside of `acados` is that the length of control horizon m cannot be set. In other words, after setting the length the prediction horizon N, the control horizon m will directly take the same length as N and it cannot be set to have a different length than N. As mention earlier in subsection (1.4.2), this could result in significant computational loads.
However, in a real-time control setting, the environment is continuously changing, and it is often sufficient to solve NLP (1.11) approximately. In other words, it is useless to have a solution with high accuracy if the computation time deadline has passed.
An existing online method for this approach is the *real-time iteration* (RTI) scheme [75]. It allows to solve an inequality-constrained QP in each iteration. The resulting generalized predictor is better suited for predictions across active-set changes, than for example a tangential predictor acquired from an interior-point method [74].
In each RTI, one full iteration of an SQP-type scheme is performed, including generation of the sensitivities with respect to all variables.
This approach made it possible to implement `acados` in real-time experiments as shown in [76] and [77].

Bibliography

[1] "Controllers in the crazyflie." [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/controllers/

[2] K. R. Klemens Fritzsche, Yuhang Guo, "Some remarks concerning differential flatness and tangent systems." *23rd International Conference on System Theory, Control and Computing (ICSTCC),10.1109/ICSTCC.2019.8886157*, pp. 173–179, 2019.

[3] Jonathan How, *Principles of Optimal Control*. MIT, 2008.

[4] R. W. Erickson and D. Maksimović, *AC Equivalent Circuit Modeling*. Cham: Springer International Publishing, 2020, pp. 215–275. [Online]. Available: https://doi.org/10.1007/978-3-030-43881-4_7

[5] MathWorks, "How to run mpc faster," https://www.mathworks.com/videos/understanding-model-predictive-control-part-5-how-to-run-mpc-faster-1533108818950.html, 2018, from the Model Predictive Control Toolbox.

[6] R. S. Alvarez-Valle and P. S. Rivadeneira, "Design of predictive control structures to track trajectories for multi-rotor unmanned aerial vehicle," in *2019 IEEE 4th Colombian Conference on Automatic Control (CCAC)*, 2019, pp. 1–6.

[7] R. Verschueren, G. Frison, D. Kouzoupis, N. van Duijkeren, A. Zanelli, R. Quirynen, and M. Diehl, "Towards a modular software package for embedded optimization," in *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC)*, 2018.

[8] F. Oliva-Palomo, A. Sanchez-Orta, P. Castillo, and H. Alazki, "Nonlinear ellipsoid based attitude control for aggressive trajectories in a quadrotor: closed-loop multiflips implementation," *Control Engineering Practice*, pp. 150–161, 2018.

[9] J. H. Gillula, H. Huang, M. P. Vitus, and C. J. Tomlin, "Design and analysis of hybrid systems, with applications to robotic aerial vehicles," *Robotics Research*, pp. 139–149, 2011.

[10] S. Lupashin and R. D'Andrea, "Adaptive fast open-loop maneuvers for quadrocopters," *Autonomous Robots*, pp. 89–102, 2012.

[11] Y. Chen and N. O. P´erez-Arancibia, "Generation and real-time implementation of high-speed controlled maneuvers using an autonomous 19-gram quadrotor," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3204–3211, 2016.

[12] I. Fantoni, "Lecture notes in modelling of terrestrial and aerial vehicles," December 2016.

[13] "Aerial robotics lecture 3a - 1 2-d quadrotor control." [Online]. Available: https://www.scribd.com/doc/297378844/Aerial-Robotics-Lecture-3A-1-2-D-Quadrotor-Control

[14] "State estimation." [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.06/functional-areas/state_estimators/

[15] K. Kozak, "State-of-the-art in control engineering," *Journal of Electrical Systems and Information Technology*, vol. 1, p. 1–9, 05 2014.

[16] S. Bouabdallah, "Design and control of quadrotors with application to autonomous flying," Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, 2007.

[17] J. Gallardo-Alvarado, *An Overview of Parallel Manipulators.* Manhattan, NY: Springer, 2016.

[18] S. Lupashin and R. D'Andrea, *Adaptive fast open-loop maneuvers for quadrocopters.* Manhattan, NY: Springer, 2012.

[19] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories." *IEEE Robot. Autom. Lett*, pp. 620–626, 2018.

[20] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[21] F. Sabatino, "Quadrotor control: modeling, nonlinear control design, and simulation." Master's thesis, 2015.

[22] S. Bouabdallah, A. Noth and R. Siegwart., "Pid vs lq control techniques applied to an indoor micro quadrotor." *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 2451–2456, 2004.

[23] Ian D. Cowling, Oleg A. Yakimenko, James F. Whidborne, and Alastair K. Cooke., "A prototype of an autonomous controller for a quadrotor uav." *2007 European Control Conference (ECC)*, pp. 4001–4008, 2007.

[24] Ly Dat Minh and Cheolkeun Ha., "Modeling and control of quadrotor mav using visionbased measurement." *2010 International Forum on Strategic Technology (IFOST)*, pp. 70–75, 2010.

[25] Keun Uk Lee, Han Sol Kim, Jin Bae Park, and Yoon Ho Choi., "Hovering control of a quadrotor." *2012 12th International Conference on Control, Automation and Systems (ICCAS)*, pp. 162–167, 2012.

[26] Bora Erginer and Erdinc Altug., "Modeling and pd control of a quadrotor vtol vehicle." *2007 IEEE Intelligent Vehicles Symposium*, pp. 894–899, 2007.

[27] A. Zulu and SamuelJohn, "A review of control algorithms for autonomous quadrotors," 2016.

[28] V. Utkin, "Variable structure systems with sliding modes," *IEEE Transaction on Automatic Control*, pp. 212–222, 1977.

[29] R. A. DeCarlo, S. H. Zak, and G. Mathews, "Variable structure control of nonlinear multivariable systems: A tutorial," *Proceedings of the IEEE, vol. 76, No. 3*, 1988.

[30] J. Y. Hung, W. Gao, and J. Hung, "Variable structure control: A survey," *IEEE Trans. on Industrial Electronics, vol. 40, No. 1*, 1993.

[31] K. Young, V. Utkin, and . Özgüner, "A control engineer's guide to sliding mode control," *IEEE Transactions on Control Systems Technology 7*, pp. 328–342, 1999.

[32] G. Bartolini, L. Fridman, A. Pisano, and E. Usai, *Modern Sliding Mode Control Theory. New Perspectives and Applications.* Springer Lecture Notes in Control and Information Sciences, 2008.

[33] T. Madani and A. Benallegue, "Backstepping control for a quadrotor helicopter." *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3255–3260, 2006.

[34] Z. Fang and W. Gao, "Adaptive integral backstepping control of a micro-quadrotor." *2nd International Conference on Intelligent Control and Information Processing (ICICIP)*, pp. 910–915, 2011.

[35] C. Diao, B. Xian, Q. Yin, W. Zeng, H. Li, and Y. Yang, "A nonlinear adaptive control approach for quadrotor uavs," *2011 8th Asian Control Conference (ASCC)*, pp. 223–228, 2011.

[36] G. Antonelli, E. Cataldi, P. Robuffo Giordano, S. Chiaverini, and A. Franchi, "Experimental validation of a new adaptive control scheme for quadrotors mavs," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2439–2444, 2013.

[37] R. Gabasov, F. Kirillova, and N. Balashevich, "Open-loop and closed-loop optimization of linear control systems," *Asian Journal of Control*, vol. 2, pp. 155 – 168, 09 2000.

[38] MathWorks, "Choose sample time and horizons," https://www.mathworks.com/help/releases/R2018a/mpc/ug/choosing-sample-time-and-horizons.html?s_eid=PSM_15028, 2018, from the Model Predictive Control Toolbox.

[39] ——, "Specify constraints," https://www.mathworks.com/help/releases/R2018a/mpc/ug/specifying-constraints.html?s_eid=PSM_15028, 2018, from the Model Predictive Control Toolbox.

[40] ——, "Tune weights," https://www.mathworks.com/help/mpc/ug/tuning-weights.html?s_eid=PSM_15028, 2018, from the Model Predictive Control Toolbox.

[41] S. Kostova, I. Ivanov, L. Imsland, and N. Georgieva, "Infinite horizon lqr problem of linear discrete time positive systems," *Proceeding of the Bulgarian Academy of Sciences*, vol. 66, 01 2013.

[42] J. Zeman and B. Rohá I'-Ilkiv, "Robust model predictive control of linear time invariant system with disturbances," *IFAC Proceedings Volumes*, vol. 36, no. 18, pp. 325 – 332, 2003, 2nd IFAC Conference on Control Systems Design (CSD '03), Bratislava, Slovak Republic, 7-10 September 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S147466701734689X

[43] M. Bujarbaruah, X. Zhang, U. Rosolia, and F. Borrelli, "Adaptive mpc for iterative tasks," in *2018 IEEE Conference on Decision and Control (CDC)*, 2018, pp. 6322–6327.

[44] L. Cui, L. Chen, and D. Duan, "Gain-scheduling model predictive control for unmanned airship with lpv system description," *Journal of Systems Engineering and Electronics*, vol. 26, no. 5, pp. 1043–1051, 2015.

[45] Naoyuki Hara and Akira Kojima, "Reduced order model predictive control - an approach based on system decomposition -," in *SICE Annual Conference 2007*, 2007, pp. 2226–2229.

[46] P. Bemporad, *Explicit Model Predictive Control*. London: Springer London, 2013, pp. 1–9.

[47] S. Hovland and J. Gravdahl, "Complexity reduction in explicit mpc through model reduction," vol. 17, 07 2008.

[48] T. Mumcu and K. Gulez, "Suboptimal solutions for time varying time delayed mpc controllers," *Electronics and Electrical Engineering*, vol. 20, 02 2014.

[49] K. Ling, W. Bingfang, H. Minghua, and Z. Yu, "A model predictive controller for multirate cascade systems," in *Proceedings of the 2004 American Control Conference*, vol. 2, 2004, pp. 1575–1579 vol.2.

[50] M. Herceg, M. Kvasnica, C. Jones, and M. Morari, "Multi-Parametric Toolbox 3.0," in *Proc. of the European Control Conference*, Zürich, Switzerland, July 17–19 2013, pp. 502–510, http://control.ee.ethz.ch/~mpt.

[51] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpoases: a parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, p. 327–363, 2014.

[52] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones, "Efficient interior point methods for multistage problems arising in receding horizon control," *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, 2012.

[53] ——, "Efficient interior point methods for multistage problems arising in receding horizon control," *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, 2012.

[54] "Ipopt documentation." [Online]. Available: https://coin-or.github.io/Ipopt/

[55] M. Orsingher, "Making aggressive maneuvers with drones thanks to parallel singularity crossing approaches." Master's thesis, 2019.

[56] A. Marino, "Robust and feasible trajectory generation of aggressive maneuvers for quadrotors," Master's thesis, 2020.

[57] J. H. Gillula, H. Huang, M. P. Vitus, and C. J. Tomlin, "Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice," *2010 IEEE international conference on Robotics and Automation (ICRA)*, pp. 1649–1654, 2010.

[58] S. Lupashin, A. Sch¨ollig, M. Sherback, and R. D'Andrea, "A simple learning strategy for high-speed quadrocopter multi-flips," *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1642–1648, 2010.

[59] S. Lupashin and R. D'Andrea, "Adaptive open-loop aerobatic maneuvers for quadrocopters," *IFAC Proceedings Volumes*, pp. 2600–2606, 2011.

[60] T. Lee, M. Leoky, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," *49th IEEE Conference on Decision and Control (CDC)*, pp. 5420–5425, 2010.

[61] A. A. El-Badawy and M. A. Bakr, "Quadcopter aggressive maneuvers along singular configurations: An energy-quaternion based approach," *Journal of Control Science and Engineering*, 2016.

[62] L. Wang and J. Su, "Switching control of attitude tracking on a quadrotor uav for large-angle rotational maneuvers," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2907–2912, 2014.

[63] Y. Chen and N. O. P´erez-Arancibia, "Lyapunov-based controller synthesis and stability analysis for the execution of high-speed multi-flip quadrotor maneuvers," *2017 American Control Conference (ACC)*, pp. 3599–3606, 2017.

[64] Y. Chen and N. O. Perez-Arancibia, "Nonlinear adaptive control of quadrotor multiflipping maneuvers in the presence of time-varying torque latency," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9, 2018.

[65] J. Erskine, R. Balderas-Hill, I. Fantoni, and A. Chriette, "Model Predictive Control for Dynamic Quadrotor Bearing Formations," in *IEEE International Conference on Robotics and Automation*, Xi'an (virtual), China, May 2021. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03177635

[66] P. C. Goldstein, H., and J. Safko, *Classical Mechanics*. London: Adisson Wesly Series in Physics, Adison-Wesley,U.S.A. World Scientific, 1980.

[67] R. M. et al., "A mathematical introduction to robotic manipulation," *CRC*, 1994.

[68] Y.-B. Jia, "Quaternions and rotations," Sep 2013. [Online]. Available: http://graphics.stanford.edu/courses/cs348a-17-winter/Papers/quaternion.pdf

[69] E. Le-Carpentier, *Statistical Signal Processing and Estimation Theory*, Nantes, FR, 2020.

[70] J. Förster, "System identification of the crazyflie 2.0 nano quadrocopter," 2015.

[71] Bitcraze, "Crazyflie 2.1 datasheet." [Online]. Available: https://www.bitcraze.io/documentation/hardware/crazyflie_2_1/crazyflie_2_1-datasheet.pdf

[72] "Smart grid simulation archives." [Online]. Available: https://hivepower.tech/tag/smart-grid-simulation/

[73] "Open source autopilot for drones," Jul 2021. [Online]. Available: https://px4.io/

[74] R. Verschueren, G. Frison, D. Kouzoupis, N. van Duijkeren, A. Zanelli, B. Novoselnik, J. Frey, T. Albin, R. Quirynen, and M. Diehl, "acados: a modular open-source framework for fast embedded optimal control," 10 2019.

[75] M. Diehl, H. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959152401000233

[76] B. B. Carlos, T. Sartor, A. Zanelli, G. Frison, W. Burgard, M. Diehl, and G. Oriolo, "An efficient real-time nmpc for quadrotor position control under communication time-delay," *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2020.

[77] D. Kloeser, T. Schoels, T. Sartor, A. Zanelli, G. Prison, and M. Diehl, "Nmpc for racing using a singularity-free path-parametric model with obstacle avoidance," *IFAC-PapersOnLine*, vol. 53, no. 2, p. 14324–14329, 2020.