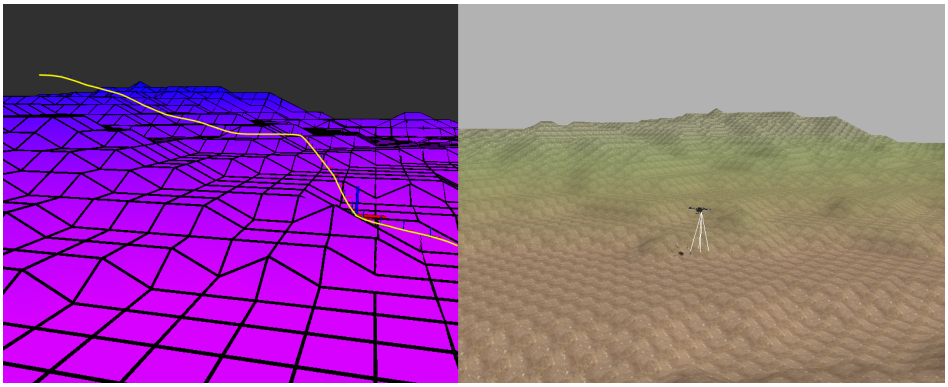
TÉCNICO LISBOA



Vertical navigation and trajectory tracking for UAV

Pedro Alexandre Pais Pereira

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Dr. Alberto Manuel Martinho Vale
Prof. Rodrigo Martins de Matos Ventura

Examination Committee

Chairperson: Prof. Paulo Jorge Coelho Ramalho Oliveira
Supervisor: Dr. Alberto Manuel Martinho Vale
Member of the Committee: Prof. Bruno João Nogueira Guerreiro

January 2021

Acknowledgments

I would like to start by thanking my thesis supervisors, Doctor Alberto Manuel Martinho Vale and Professor Rodrigo Martins de Matos Ventura for the support, knowledge and contributions that allowed me to successfully develop and complete my dissertation. Without their help and recommendations this work would have never been possible. I would also like to recognize the full team behind FRIENDS for allowing me to participate and contribute in the completion of this project.

Moreover, I would also like to thank all my family and in particular my mother, father and sister for standing beside me in the hard times and during this strange and rough year. Without their support I could have never finished this work before the initial deadline.

Finally, I would like to thank all my friends that supported and encourage me throughout my University years making them as enjoyable as they could ever be. Their support, suggestions and corrections were also crucial for the completion of this dissertation and for this I will be forever grateful.

Resumo

O recente avanço tecnológico de sensores e unidades de processamento permitiu a integração em diversas aplicações civis e militares de sistemas autônomos, como Veículos Aéreos Não Tripulados (*UAV*), capazes de executar com eficiência missões de vigilância e monitorização em locais anteriormente considerados perigosos ou inacessíveis.

Algumas dessas missões requerem a obtenção de dados de sensores em proximidade a objetos de interesse localizados no solo. Nesta tese, é necessário determinar o caminho que permitirá ao drone voar no local da missão em segurança evitando facilmente colisões com o terreno e mantendo uma altitude baixa e aproximadamente constante acima do solo. Esta tese propõe um algoritmo de geração de trajetória ótima baseado em métodos iterativos para gerar o caminho ideal para o UAV e um controlador preditivo capaz de seguir esta trajetória com precisão. A solução proposta considera a dinâmica não linear do modelo do sistema autônomo e determina uma trajetória discreta que satisfaz as restrições de seguimento de terreno e de passagem em pontos predefinidos enquanto minimiza uma função de custo que visa reduzir o tempo e a aceleração da missão. O otimizador de trajetória desenvolvido efectua uma otimização offline tendo em conta a informação de altitude obtida em missiês anteriores. O desempenho do planeador de trajetória é analisado e discutido para vários cenários com diferentes perfis de elevação.

Para seguir com precisão a trajetória otimizada na presença de perturbações, este trabalho também propõe a formulação, implementação e validação de um Controlo Preditivo baseado em modelos dinâmicos. Esta estratégia de planeamento e controlo, à semelhança do problema de geração de trajetória, trata-se de um problema de otimização que tem como objetivo minimizar uma função de custo quadrática que penaliza erros de seguimento enquanto satisfaz as restrições dinâmicas impostas pelo modelo. Este Controlador foi testado num ambiente de simulação (Gazebo), onde o modelo do drone do projeto Frota de drones para inspeção radiológica, comunicação e salvamento [1] e um terreno realista permitiram validar a estabilidade e robustez do controlador projetado, bem como a integração do mesmo com o optimizador de trajetória.

Palavras-chave: *UAV*, Trajetória Ótima, Seguimento de Terreno, Passagem em Pontos Predefinidos, Controlo Preditivo baseado em Modelos Dinâmicos, Seguimento Preciso de Trajetória

[1] https://www.ipfn.tecnico.ulisboa.pt/FRIENDS/index.html

Abstract

The recent technological achievements in sensors and embedded systems have prompted the integration in several civil and military applications of autonomous systems such as Unmanned Aerial Vehicles (UAV) capable of efficiently executing surveying and monitoring missions in locations otherwise considered dangerous or inaccessible.

Some of those missions require the acquisition of sensor data close to objects and areas of interest located on the ground. In this thesis, it is necessary to determine the path that will allow the UAV to roam the mission site safely and dynamically by easily avoiding terrain obstacles and while maintaining a constant and close proximity to the ground. This thesis proposes an optimal trajectory generation algorithm based on gradient search methods to generate the flight path for the UAV and an optimization controller capable of precisely track those trajectories. The proposed solution considers a non-linear model dynamics approximation of the autonomous system and determines a discrete trajectory that satisfies Terrain-following and waypoint tracking constraints and minimizes a cost function that aims to reduce the mission's time and acceleration. The trajectory optimizer performs an offline preliminary optimization based on previously acquired data of the terrain elevation in the mission's site. The path planner's performance is tested and discussed for several scenarios with different elevation profiles.

In order to precisely perform Trajectory tracking and path following of the determined optimum trajectory in the presence of disturbances, this paper also presents the design, implementation and testing of a Model Predictive Control (MPC). This online planning and control strategy is also an optimization problem that aims to minimize a quadratic function designed to penalize state errors while satisfying the imposed dynamic constraints. The MPC was tested in a physics simulation environment (Gazebo) where a realistic terrain and a model of the Fleet of dRones for radIological inspEction, commuNication anD reScue project [2] UAV was integrated to further validate the stability and robustness of the designed controller.

Keywords: UAV, Optimal trajectory, Terrain-following, Waypoint tracking, Model Predictive Control, Trajectory tracking and path following

[2] https://www.ipfn.tecnico.ulisboa.pt/FRIENDS/index.html

Contents

Acknowledgments . . . iii
Resumo . . . v
Abstract . . . vii
List of Tables . . . xiii
List of Figures . . . xv
Nomenclature . . . xix
Glossary . . . xxiii

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Objectives . . . 3
1.3 State of the Art . . . 4
1.3.1 Trajectory Generation . . . 4
1.3.2 UAV Control . . . 6
1.4 Thesis Contributions . . . 7
1.5 Thesis Outline . . . 7

2 Theoretical Overview **9**
2.1 Optimization Overview . . . 9
2.2 Trajectory Optimization . . . 14
2.3 Model Predictive Controller . . . 20
2.3.1 Stability . . . 22

3 Hexarotor Dynamics and Kinematics **25**
3.1 Reference systems for the hexacopter . . . 25
3.2 Applied forces and torques . . . 27

4 Trajectory Generation Formulation **31**
4.1 System Dynamics . . . 31
4.2 Waypoint Objective . . . 33
4.3 Terrain Following and Terrain Avoidance Objective . . . 34
4.4 Time-Optimal Objective . . . 35

4.5 Terrain Modelling . . . 37
4.6 Solver Implementation . . . 38
4.6.1 Problem Discretization . . . 38
4.6.2 Optimization Function . . . 39
4.6.3 Constraints . . . 40
4.6.4 Experimental Results with Validation Data . . . 41

5 Model Predictive Control Formulation **49**
5.1 System Dynamics . . . 49
5.2 Cascaded Control Strategy . . . 51
5.3 Attitude System Identification . . . 52
5.4 External Disturbance Estimation . . . 53
5.5 Solver Implementation . . . 55
5.5.1 Optimization Function . . . 57
5.5.2 Constraints . . . 57

6 Simulation **59**
6.1 Gazebo Model . . . 59
6.2 Height map Model . . . 61
6.3 Software Architecture . . . 62
6.3.1 PX4 Autopilot . . . 62
6.3.2 Implemented Control Architecture . . . 63
6.4 MPC Simulation Results . . . 64
6.4.1 Hovering Performance . . . 65
6.4.2 Step Reference . . . 66
6.4.3 Trajectory Tracking without Wind . . . 67
6.4.4 Trajectory Tracking with Wind . . . 69
6.5 Full Control Architecture Simulation . . . 71

7 Conclusions **73**
7.1 Future Work . . . 74

Bibliography **75**

A Polynomial Trajectory Planning **83**
A.1 Problem Formulation . . . 83
A.2 Objective function . . . 84
A.3 Constraints . . . 84
A.4 Unconstrained QP Reformulation . . . 85
A.5 Time Optimization . . . 86
A.6 State Inequality Constraints . . . 86

A.7 Limitations and Results 87

List of Tables

4.1 Waypoint discrete allocation 33
4.2 Trajectory optimization parameters and coefficients. 41
4.3 Time steps of the optimum trajectory on Scenario 1. 42
4.4 Time steps of the optimum trajectory on Scenario 1. 42

6.1 FRIENDS hexacopter parameters and control input constraints. 65
6.2 MPC weight constants. 65

List of Figures

1.1 Real Hexacopter Model of Project FRIENDS. 2

2.1 Schematic representation of the various categories of optimization methods. 11
2.2 Example of SQP iteration Path for a problem with two optimization variables. 14
2.3 Example of IPM iteration Path for a problem with two optimization variables. 14
2.4 Example of a trajectory optimization problem that employs single shooting discritization. . 17
2.5 Example of a multiple shooting optimization where the defect of the equality constraints is non-zero meaning the problem is infeasible. 18
2.6 Hermite-Simpson Collocation example. 20
2.7 Overview of a Model Predictive Controller. 21
2.8 Practical Asymptotic Stability illustration. 24

3.1 Representation of Inertial and Body frames. 26
3.2 Hexarotor with X configuration. 28

4.1 Representation of maximum and minimum altitude constraints imposed by the terrain profile. 35
4.2 Illustration of Time Elastic Band Formulation for multiple waypoints. 36
4.3 Illustration of a scenario where the initial guess of the optimization is outside the feasible region. 37
4.4 Example of discrete data interpolation in 1D. 38
4.5 Graphical representation of the trajectory in the scenario with smooth terrain data and large fixed time steps. 42
4.6 Graphical representation of the trajectory in the scenario with smooth terrain data and small fixed time steps. 43
4.7 Graphical representation of the trajectory in the scenario with smooth terrain data and with variable time intervals determined by the optimizer. 44
4.8 Graphical representation of the optimized states and inputs in the scenario with smooth terrain data and with variable time intervals determined by the optimizer. 45
4.9 Graphical representation of the trajectory in the scenario with high terrain gradients. . . . 46
4.10 Graphical representation of the optimized states and inputs in the scenario with high terrain gradients. 46

4.11 Graphical representation of the trajectory in the scenario with a big terrain discontinuity and low terrain-following cost K_{TF}. 47
4.12 Graphical representation of the optimized states and inputs in the scenario with a big terrain discontinuity and low terrain-following cost K_{TF}. 47
4.13 Graphical representation of the trajectory in the scenario with a big terrain discontinuity and high terrain-following cost K_{TF}. 48
4.14 Graphical representation of the optimized states and inputs in the scenario with a big terrain discontinuity and high terrain-following cost K_{TF}. 48

5.1 PX4 Multicopter Attitude Controller. 51
5.2 PX4 Multicopter Angular Rate Controller. 52
5.3 ACADO Toolkit Architecture. 56
5.4 Schematic representation of the MPC's software Architecture. 56

6.1 A draft of a multirotor helicopter with four non-symmetrical aligned rotors. 60
6.2 Gazebo model of the Project FRIENDS' Hexacopter. 60
6.3 Gazebo model of the Project FRIENDS' Sensor Box. 61
6.4 Gazebo Model of a Realsense D435 Camera. 61
6.5 Gazebo Model of a VLP-16 Lidar. 61
6.6 Gazebo world with terrain elevation data as floor. 62
6.7 PX4 Multicopter Control Architecture. 63
6.8 Scheme of the Control architecture of this Thesis. 64
6.9 Graphical representation of the 3D performance of the MPC while hovering around a fix position, applied to the simulated FRIENDS' hexacopter. 66
6.10 Graphical representation of the MPC output commands during the hovering mission. . . . 66
6.11 Graphical representation of the 3D performance of the MPC when a step on the y position reference is introduced. 67
6.12 Graphical representation of the MPC output commands when a step on the y position reference is introduced. 67
6.13 Graphical representation of the 3D trajectory executed without wind. 68
6.14 Position tracking error of the MPC during the mission without wind. 68
6.15 Graphical representation of the MPC output commands during the polynomial tracking mission without wind. 68
6.16 Graphical representation of the 3D trajectory executed with wind and with EKF. 69
6.17 Position tracking error of the MPC during the mission with wind and EKF active. 69
6.18 Graphical representation of the MPC output commands during the polynomial tracking mission with wind and EKF active. 70
6.19 Graphical representation of the 3D trajectory executed with wind and without EKF. 70
6.20 Position tracking error of the MPC during the mission with wind and EKF inactive. 71

6.21 Graphical representation of the MPC output commands during the polynomial tracking mission with wind and EKF inactive. 71
6.22 Graphical representation of the 3D trajectory executed in the Terrain-Following Mission. . 72
6.23 Graphical representation of the vertical trajectory executed in the Terrain-Following Mission. 72

A.1 3D polynomial trajectory and optimization limits. 88
A.2 Optimized Polynomial trajectory. 88

.

Nomenclature

Greek symbols

α	Optimization search step.
χ	Optimization Variable vector.
ΔT	Time Interval.
Δt	Discretization Time.
η	Body frame orientation vector.
λ	Adjoint variables.
ω	Natural frequency vector.
ω_i	Rotational velocity of rotor i.
ϕ, θ, ψ	Roll, pitch and yaw Euler angles.
τ	Time constant vector.
ξ	Damping constant vector.
ζ	Constraint Defect.

Mathematical Notation

$\nabla^2_\chi f$	Hessian function of f.
$\nabla_\chi f$	Jacobian function of f.
$\times$	Cross Product.

Roman symbols

$\mathbf{e}$	Unit vector.
$\mathcal{F}$	Optimization Function.
$\mathcal{F}_B$	Terminal optimization cost.
$\mathcal{F}_P$	Path optimization cost.

f	Dynamic system behaviour function.
F_g	Force of gravity.
F_i	Individual force in rotor i.
F_{ext}	External Forces vector.
g	Gravity constant.
h	Absolute altitude.
$\mathcal{J}$	Momentum of Inertia.
K	Number of waypoints.
M	Torque vector.
m	Mass.
N	Number of discretization steps.
$\mathbf{p}$	Inertial Frame position vector.
P	Input cost matrix.
p	Optimization search direction.
Q	State cost matrix.
$\mathcal{R}^{ab}$	Rotation matrix of a described in frame b.
R	Final state cost matrix.
$\tilde{T}$	Normalized Thrust.
T_h	Time Horizon.
U	Inputs vector.
v	Velocity vector.
w	Angular Velocity vector.
X	States vector.

Subscripts

0	Initial Condition.
x,y,z	Cartesian components.
cmd	Command Input.
f	Final Condition.

int Integrator state.

Superscripts

$*$ Optimum solution.

$+$ Upper bound.

$-$ Lower bound.

$\mathfrak{B}$ Body Frame.

$\mathfrak{I}$ Earth Frame.

des Desired.

ref Reference.

w Waypoint.

T Transpose.

Glossary

2D	Two-Dimensional
3D	Three-Dimensional
BVP	Boundary Value Problem
CAD	Computer-Aided Design
DEM	Digital Elevation Map
EKF	Extended Kalman Filter
ENU	East, North, Up
EoM	Equations of Motion
GB	Gradient Based
IMU	Inertial Measuring Unit
IPOPT	Interior-Point Optimizer
LQR	Linear-Quadratic Regulator
MPC	Model Predictive Control
NLP	Non-Linear Programming
NMPC	Non-linear Model Predictive Control
ODE	Ordinary Differential Equations
PID	Proportional, Integral and Derivative
RHC	Receding Horizon Control
ROS	Robot Operating System
SDF	Simulation Description Format
SITL	Software-In-The-Loop
SQP	Sequential Quadratic Programming
TA	Terrain-Following
TF	Terrain-Avoidance
UAV	Unmanned Aerial Vehicle

Chapter 1

Introduction

This chapter starts by providing the motivation that prompted the development of this dissertation. Then I introduce the state of the art that will act as a foundation for the study and implementation of the proposed solution. Subsequently, the objectives of this dissertation are defined and finally the structure of this document is detailed.

1.1 Motivation

The modern concept of Unmanned Aerial Vehicle (UAV) first appeared in 1849 when the Austrians attacked the Italian city of Venice with 200 unmanned balloons loaded with bombs. During the first World War, occurs the earliest attempt to use an unmanned aerial vehicle controlled from the ground known as the ”flying bomb” [1]. Early efforts continued during the Inter-war Period and World War II and were prompt by the military goal of reducing the loss of human life in the battleground [2].

However, in more recent years, this type of systems have gained significant popularity among aircraft hobbyists, academic researchers and industries which has promoted a reduction in costs of several sensors, actuators and batteries, making them more practical and accessible, technically enabling their integration in a wide range of civil applications.

In fact, this increase in accessibility of sensors coupled with the fast improvement in efficiency and processing power of small computers has allowed the integration of more computationally intensive processes onboard the UAVs, such as 3D mapping and reconstruction, localization, planning and optimization-based control techniques. All those advancements combined with ease of deployment, low maintenance cost, remarkable agility and robustness [3] prompted the integration of these vehicles in several areas such as infrastructure inspection [4], exploration tasks in unknown environment [5], search and rescue operations as presented in Silvagni et al. [6], forest resources monitoring such as Berie and Burud [7], police and military surveillance as shown in Rangel and Terra [8], mapping [9], cinematography [10], food delivery[11], etc..

Project SEAGULL [12] is an example of the application of UAVs for surveillance and monitoring missions. It is developed by the Portuguese Air Force in collaboration with several research institutes

and aims to support maritime situational awareness by developing intelligent systems based on UAVs equipped with video and thermal cameras.

In some surveillance and inspection operations, measurements have to be performed close to the objects of interest that are placed on the ground. This is the case for project FRIENDS[1] (Fleet of dRones for radIological inspEction, commuNication anD reScue), where this thesis is integrated on. Therefore, the motivation for this paper arises from the requirement that the project's UAV must navigate in close proximity to the ground, where the expected low intensity sources of radiation are located, in order to estimate the radioactivity level of the scenario using reduced sensitivity sensors.

The use of UAVs for monitoring and mapping of radioactive scenarios has gained popularity in recent years with the appearance of several projects and commercial solutions.

In [13] a remotely controlled drone that can combine locational and radiological data performs radiation detection and mapping in order to safely identify irradiated areas in the event of a nuclear emergency. Similarly, in project MOBISIC [14] the goal was to construct a UAV capable of detecting, identifying and localizing radionuclides, which can be components of a bomb.

The main interest of this thesis is to design a strategy capable of realizing autonomous flight by optimizing and following trajectories in space, converting the multirotor platform, driven initially by a human operator, into a fully Unmanned Autonomous Vehicle. In the tackled case of low-altitude flights in irregular terrain such as mountainous regions, it is highly desirable to find a trajectory that not only follows the terrain profile but also guarantees the safety of flight and takes the UAV to the destination by optimizing certain factors including time, vertical position, acceleration, etc..

In the FRIENDS' project, the Unmanned aerial system selected to perform the required mission is a standard Hexarotor (see fig. 1.1) due to its agility and ability to perform tasks that humans or ground vehicles are unable to do such as exploration missions in steep terrain or densely vegetated areas. It consists of six electrical rotors with propellers mounted at the ends of a cross-shaped structure. The dynamical configuration permits the take-off and land manoeuvres in reduced spaces, hover above targets and the omnidirectional motion in the space making it the best option for low altitude missions.

Figure 1.1: Real Hexacopter Model of Project FRIENDS.

The following approach addresses the design of an onboard real-time predictive controller for the

[1] https://www.ipfn.tecnico.ulisboa.pt/FRIENDS/index.html

project's UAV that is able to accurately and safely follow a pre-determined trajectory. The optimization of this trajectory is based on the objectives and constraints that follow from the requirements of the project's surveillance and inspection missions.

1.2 Objectives

This thesis focuses on the design, implementation and validation of trajectory optimization and control techniques for a hexacopter so that the vehicle can perform an inspection mission by safely following waypoints while keeping its vertical distance to the ground approximately constant. In outdoor inspection missions, maintaining the altitude above ground is fundamental to guarantee the safety of the UAV and allow the measurements of onboard devices such as depth cameras and reduced sensitivity radioactive sensors to be as accurate and useful as possible reducing errors and post-processing work.

However, it is also important to consider the limited range of the hexacopter dictated by its battery capacity. This means the trajectory optimization process should also attempt to minimize the flight time required to fulfil the waypoint tracking mission while satisfying maximum velocity and acceleration constraints.

In this thesis is assumed that waypoints are only defined in the horizontal plane and are determined during flight based on an intelligent algorithm developed by Brouwer [15] that uses the real-time sensor data and terrain knowledge to determine the best curse of action. Therefore, and taking into account all mentioned objectives, the overall problem of optimization is simultaneously a waypoint tracking problem, a terrain-following problem and a minimum-time optimization.

Two different methods for trajectory optimization will be implemented and compared, in order to determine the best trajectory that fulfils all mission goals.

The main objective of this thesis is the design of a complete control and navigation strategy capable of executing the terrain following mission proposed by project FRIENDS. In order to achieve this goal, I defined the following sub-objectives:

- Generate optimal trajectories that are able to perform terrain following and waypoint tracking while minimizing flight time and fulfilling constraints imposed not only by the physical limitations of the vehicle but also the characteristics of the mission and environment such as maximum velocity restrictions imposed by the radioactive sensors;

- Implement a stable and robust controller capable of precisely track such trajectories even if subjected to external disturbances.

Both objectives are implemented as separate optimization problems that will ultimately integrate the high-level onboard navigation system of the UAV in a cascade architecture. The low-level control is performed onboard by a separate computing unit called Pixhawk that runs the PX4 open-source autopilot software. In this way, the vehicles will be able to follow optimal trajectories and perform an efficient inspection mission in real-time without the need for a human operator.

1.3 State of the Art

The problem at hand can be divided into two main problems that have been intensively studied in the last few years with the formulation, verification and validation of multiple strategies and algorithms. The first problem is the generation of an optimal trajectory capable of maintaining a constant distance to the ground while following a set of user defined horizontal waypoints. The second problem is the design of a robust UAV controller that performs precise trajectory tracking.

1.3.1 Trajectory Generation

Although both problems could by simultaneously solved by one Model Predictive Controller, the introduction of complex constraints and objectives such as terrain following and avoidance or minimum time optimization can make the problem impossible to tackle in the short time tolerances allowed for a MPC. Therefore, the optimization of the trajectory must be implemented as a separate optimization problem.

In the literature, there are two main ways of optimizing a UAV trajectory in space. The first is to determine an optimum continuous path based on the optimization of the derivatives and coefficients of a polynomial or the control points of a B-spline. The second set of methods is based on the discretization of the model dynamics of the UAV by considering a vector of states and inputs that are evaluated at each discretization point.

Terrain Following

Several studies have been conducted with the goal of designing a terrain-following trajectory for all types of UAVs in two-dimensional flight in mountainous regions. This is the case of [16] where the trajectory is optimized for minimum vertical acceleration and minimum flying time while satisfying the imposed terrain following/avoidance constraints. The resulting optimization control problem is discretized by employing a direct collocation method and then solved as a non-linear programming problem. Lu and Pierson [17] and Menon et al. [18] formulated a similar problem but instead used an inverse dynamics approach to solve achieve a numerical solution for the problem. Following this work, Feng et al. [19] proposes a regeneration method capable of avoiding collision with previously unknown obstacles by using the Linear Gauss Pseudo-Spectral method that locally re-optimizes the trajectory, evading unknown obstacles.

A similar approach presented by Khademi et al. [20] tackles the three-dimensional flight by also implementing an optimal control problem formulation. In this case, the full dynamic system of the UAV is considered in the optimization problem resulting in a 3D optimum terrain following-trajectory.

Another studied approach is to use a grid approximation scheme to convert the continuous constrained optimization problem into a discrete search problem, [21]. A variant of the Minimum Cost Network Flow is applied to the problem and a solution is determined by minimizing a cost function based on Digital Terrain Elevation Data and discrete dynamic equations of motion.

A different approach addressed by Rahim and Malaek [22] is the use of fuzzy logic methods to model the flight altitude and gradient of the terrain. In that work, the implementation was done in two phases.

The first phase includes offline navigation in which path planning is done through cost function, flight vehicle dynamics, and terrain data with the fuzzy method. The second phase includes online navigation in a way that the flying vehicle follows the designed path while is capable of altering the parameters of membership functions through fuzzy learning method to achieve the best solution.

As mentioned before, apart from discrete optimization models, it is also possible to optimize a continuous spline to perform terrain avoidance flight. This is the case of [23] where the tree search structure is extended using a spline-RRT* method to generate smooth paths without any post-processing. A cost function is used to ensure that the resulting paths are sufficiently far from several hazardous positions and close to the surface of the local terrain.

Finally, the authors of [24] address the terrain-following problem in previously unknown environments, where global trajectory planning methods cannot be applied directly. In this paper, a self-learning terrain following method is proposed to realize the automatic generation of a terrain-following controller.

Waypoint Tracking

Another goal of this dissertation is to obtain a trajectory capable of flying through previously computed waypoints. Since the previous objectives of the trajectory generation are mainly tackled by optimization problems, it is fundamental to understand how this final objective can be incorporated in an equivalent formulation.

A framework for autonomous trajectory generation through waypoints for UAVs is proposed in Eslamiat et al. [25] where the optimization framework consists of an optimal and smooth trajectory generation algorithm capable of minimizing the trajectory derivatives by approximating the UAV dynamics using a discrete linear model.

Falanga et al. [26] presented a fast optimization control algorithm for quadrotors with a cable-suspended payload based on an Euler-Lagrange dynamic model of the full system. This method is capable of flying through waypoints by assigning a certain number of discretization steps between each waypoint and then optimizing the time for each of the segments as an independent optimization variable, resulting in a path with variable discretization time steps.

In order to maintain a constant discretization time step while flying through waypoints and still being able to optimize the total trajectory time, Foehn and Scaramuzza [27] introduces a formulation where the progress through the waypoints is bounded by complementarity constraints. Those progress variables change via complementarity constraints if the UAV passes in close proximity to a waypoint, enabling the simultaneous optimization of the trajectory and the time-allocation of the waypoints since they are not allocated to a specific discrete node. The main setback with this method is the high non-convexity of the problem that substantially increases the computation times.

Finally, in [28] the trajectory of the UAV is optimized based on a polynomial generation by minimizing path derivatives and total segment time. This method uses jointly optimized polynomial path segments in an unconstrained quadratic program. This approach generates high-quality trajectories much faster than purely sampling-based optimal kinodynamic planning methods but sacrifices the guarantee of asymptotic convergence to the global optimum. Due to their continuous nature, it is also more complex to

introduce path constraints such as terrain-following in this formulation.

1.3.2 UAV Control

Designing a control system for aerial robots capable of performing precise trajectory tracking is fundamental to successfully perform exploration and inspection tasks in realistic outdoor environments where external factors have to be taken into account. This leads to the formulation and study of a stable and safe control technique capable of flying the UAV in real-time while avoiding obstacles and reducing the effects of unpredictable external disturbances.

The simplest and most common strategy to control UAVs is the standard PID controller [29, 30]. This technique disregards the dynamic model of the UAV and relies on a feedback mechanism capable of following references and rejecting disturbances by running the state errors through a closed-loop based on proportional, integral, and derivative terms.

A more complex approach is presented in [31] where precise trajectory-tracking control is developed based on a Lyapunov back-stepping technique that approximates the non-linear dynamics of the UAV to a linear state space. This controller was shown to be capable to follow reference trajectories by keeping the attitude states bounded. Xingling et al. [32] also implemented a back-stepping controller subject to parametric uncertainties and external disturbances. In this formulation, an extended state observer is constructed to online estimate the unmeasurable velocity states and lumped disturbances allowing for the implementation of a robust dynamic surface flight controller that guarantees asymptotic stability. However, the previously mentioned back-stepping techniques have the limitation that the linearised systems are only valid for conditions where the derivatives of the states are relatively low.

Along with back-stepping techniques, many different optimization control strategies have been presented to tackle the problem of trajectory tracking. In [33] a Non-Linear Guidance Logic is coupled with Model Predictive Control to achieve real-time reference tracking. The trajectory problem is implemented by representing the 3D path using only the multirotor's jerk and then solving a convex optimization problem on each decoupled axis allowing for efficient control.

Another formulation is presented by Sferrazza et al. [34] where state and input trajectories are represented as linear combinations of Laguerre functions in order to approximate the infinite-horizon optimization control problem. It accounts for disturbances by applying an iterative learning scheme resulting in a controller with a tracking performance that improves over time. The use of the full dynamic system of a UAV can be highly computational demanding, which resulted in approaches like the one presented by Raffo et al. [35] where the dynamic motion equations used in the MPC are obtained by the Lagrange-Euler formalism.

In this work, I propose the use of a non-linear Model Predictive Control (NMPC) scheme to solve the problem of tracking a reference trajectory, in real-time and in a real-world scenario where disturbances are present. My approach is based on the system simplification presented in [36] where the low-level control is assumed to be controlled by a separate module. In this work the performance of a linear Model Predictive Control applied to an approximation of the UAV dynamics is compared to a non-linear

implementation. The results of this work allow me to conclude that the non-linear MPC is not only more computationally efficient but also displays a better performance when it comes to precisely follow trajectories. This optimization control does not, however, include any specific restrictions regarding terrain avoidance and obstacles due to the complexity of such constraints making it impossible to perform real-time control. Therefore, it requires the development of a trajectory optimizer capable of determining a safe and efficient terrain following trajectory.

1.4 Thesis Contributions

In order to fulfil the objectives and requirements mentioned this thesis tackled the development of a offline optimum trajectory generator that compiles several different objectives and requirements usually tackled separately to achieve the ideal path for the proposes of this project. This optimization problem includes the tracking of 2D waypoints, the terrain-following objectives and constraints and the integration of minimum time and acceleration objectives.

Regarding the control of the UAV it was implemented a MPC that considers external disturbances and UAV dynamics and can be integrated as a high-level controller in several commercial autopilots. This controller includes an External Kalman Filter capable of estimating the external forces acting on the UAV reducing the steady state error of the MPC.

Finally, this thesis implemented a Simulation in Gazebo+SITL where a model of the project's UAV was created and tested allowing for the validation of the mentioned control and planning algorithms.

1.5 Thesis Outline

The outline of the remainder of the thesis is as follows:

Chapter 2 starts by presenting a theoretical overview of generic optimization problems by classifying them and discussing the computational algorithms used to compute the desired solution. Then, it studies the special cases of optimization control trajectories applied to offline and real-time problems.

Chapter 3 presents the deduction of the full mathematical model of a generic Hexacopter. This model will later be used as a basis to derive a simplified dynamic system employed in the design and implementation of the control strategies.

In Chapter 4 the strategy implemented for the trajectory optimization problem is described. The Chapter starts by defining the objectives and proposed solutions for the optimum UAV trajectory. In the end, the strategy is analysed and validated using an off-the-shelf optimization framework.

The problem of real-time control is addressed in Chapter 5. In this chapter, the dynamics of the UAV model are simplified and a method of system identification is used to determine the new parameters. An Extended Kalman filter is also designed and coupled with the final Model predictive control to reduce the effect of external disturbances. This MPC is then constructed using a highly efficient optimization control tool to be tested and implemented in a real drone.

Chapter 6 presents the setup of the simulation environment required to validate and verify the implemented control strategy as well as the interface with the PX4 software used in the low-level control of the real UAV.

Finally, Chapter 7 evaluates the thesis as a whole, summarising the goals achieved and discussing what could be improved as future work.

Chapter 2

Theoretical Overview

In this chapter, an overview of the formulation and solving techniques of linear and non-linear optimization problems is presented. Then, I discuss the special cases of optimal control trajectories that try to find the best path for a dynamic model given an objective function and a set of constraints. In this type of optimization problem, a discretization algorithm is required to convert the continuous-time dynamics of the robot into a set of equality and inequality constraints. Finally, I describe some basic concepts and properties of a Model Predictive Control (MPC), also called Receding Horizon Control (RHC), that allows the implementation of a real-time, robust and stable control strategy for the FRIENDS' project Hexacopter.

2.1 Optimization Overview

Optimization is the task of finding the best solutions to particular problems. These solutions are found by adjusting the problem's parameters to give either a maximum or a minimum value for the optimization function. The optimization function is usually called the cost function since in the majority of optimization problems it represents the variables' cost.

The general statement of an optimization problem where the goal is to minimize the objective function is:

$$\begin{aligned} \text{Minimize} \quad & \mathcal{F}(\chi) \\ \text{w.r.t.} \quad & \chi\,, \\ \text{subject to} \quad & G_i(\chi) = 0,\ i \in \mathcal{E} \\ & H_i(\chi) \leq 0,\ i \in \mathcal{I} \end{aligned} \tag{2.1}$$

where $\mathcal{F}$ is the cost function, χ is the vector of optimization variables, $\mathcal{E}$ and $\mathcal{I}$ are the sets of equality and inequality constraints respectively [37].

Based on the definition of the objective function $\mathcal{F}$ and the constraints G_i and H_i with respect to the

optimization variables, optimization problems can usefully be divided into linear and non-linear, which dictates the complexity of the methods required to compute the optimum solution.

Due to their complexity, non-linear problems may have many local optimum solutions, which are minimum in a specific sub-region of the solution space. Unfortunately, for general non-linear, and in particular non-convex problems global minimizers are hard to find in practice since there is only local knowledge of the function $\mathcal{F}$ and its derivative with respect to the optimization variables, $\nabla_{\chi}\mathcal{F}$. Due to this local knowledge limitation, the only way to guarantee that the global minimizer is found is to evaluate the objective function $\mathcal{F}$ in a big portion of the feasible state space using Heuristic Methods. However, if the problem has a big number of optimization variables or the feasible state space is too large these methods can require considerable computation times. In order to make these optimizations practical for real-life applications, it is often necessary to use more efficient methods that only evaluate a small number of vectors χ. As a consequence, if we construct an algorithm under these restrictions we can not be sure if we reached a global minimizer. Nevertheless, we are often able to identify a so-called local minimizer which is considered acceptable in some cases [38].

Optimization problems can be further divided as discrete or continuous based on which values the optimization variables can be assigned with. In some problems, the parameters χ can take any real values to give an optimal solution, but in others, this parameters are constrained to have the form $\chi_i \in \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers, or binary constraints (integer programming) or even a mixture of real and integer or binary variables (mixed-integer programming). In these cases, the values of the objective functions and constraints may change significantly when moving between feasible points making the optimization problem highly non-convex, and therefore far more difficult to solve. By contrast, in continuous problems, since χ can take on any real numbers the optimization problems become easier to solve as due to its smoothness it is possible to deduce the values of the objective function and constraints around a specific point in χ [37].

After defining the optimization objective as a mathematical function, it might be thought that calculus could be employed effectively to find the optimum solution. This is true, but the number of differential coefficients required, and the complexity of the resulting algebra, for anything other than simple problems with a very small number of variables in vector χ, rule out this approach for most practical applications.

Therefore, it is necessary to select an alternative numerical method suited for the defined problem. There are two main categories of iterative algorithms that take substantially different approaches to find the solution: deterministic and heuristics. Heuristic methods try to search for a global optimal solution by exploring the feasible subset in some structured manner based on a heuristic function. This category includes methods such as Genetic Algorithms [39], Ant Colonies [40] and Particle Swarm [41], etc. Deterministic methods take advantage of the analytical properties of the problem to generate a sequence of points that converge to a local optimal solution. These can be Gradient Free (GF) or Gradient-Based (GB).

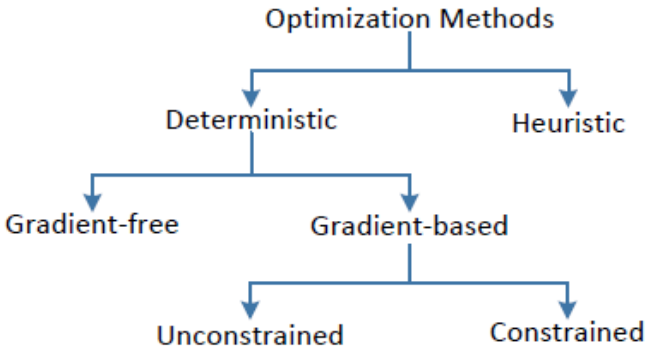


Figure 2.1: Schematic representation of the various categories of optimization methods [42].

Brute Force Methods

The most basic Heuristic optimization algorithm is the Brute Force method, where the optimal value of a function is determined by systematically calculating all possible solutions and decide afterwards which one is the best. This approach in optimization requires considerable computation power making it only feasible for small problems (in terms of the dimensionality of the state space) since the number of possible states of the system increases exponentially with the number of dimensions. In the case of continuous predictor variables, the number of states is infinite and this method cannot be used directly.

Despite these drawbacks, brute force methods do have the benefit of being rather simple to implement since no gradients or intelligent heuristics are required. As a consequence, brute force methods are often seen as reference algorithms for calculating the number of states, or the number of calculations necessary to find the global optimum solution. Hence, it can be used for the estimation of the effort to solve a problem.

Gradient Based Methods

The algorithms described here, generally called gradient-based methods, have proved effective in finding optimal values of functions, but not necessarily global optimum. If the problem is non-convex and the optimization function has many local minima then the likelihood that the algorithm will find the global optimum diminishes and depends heavily on the initial guess. One method used to avoid being stuck on a local minimum is the use of multiple starting points distributed in the design space, however, depending on the number of optimization variables, this can substantially increase the computation time required to determine the solution.

These numerical methods utilize the information of the first and sometimes second-derivatives of the objective function (and/or constraints) to choose the best direction in the design space that leads to the optimal solution. This leads to much fewer function evaluations required to achieve the closest minimum when compared to Heuristic methods. If the objective function is known to be smooth and the design variables are continuous, the GB algorithms are typically the best choice.

In GB it is also important to consider the methods employed to compute the derivative information

of the objective function known as sensitivity analysis methods, required to drive the iteration process towards the optimum solution. The ideal method to compute the gradient of a function is the through analytic computations which directly differentiates said function. However, this method can only be used if the function is defined analytically with respect to the optimization variables, which does not happen for the majority of the cases. An alternative method is the Finite-Difference (FD) approximation that uses local approximation algorithms to derive the gradient of the cost function every iteration with respect to all the optimization variables.

Depending on the characteristics of the problem, gradient-based methods can be further divided into unconstrained (Steepest Descent [43], quasi-Newton [44], etc.) and constrained optimization (Reduced Gradient [45], Sequential Convex Programming [46], etc.), depending on the existence or not of constraints.

Unconstrained Optimization

Although most optimization problems are constrained, it is very useful to understand how unconstrained optimization problems are solved since the basic principles are similar [38].

In order to solve an unconstrained optimization problem with a twice continuously differentiable objective function an initial guess for the optimization variables, χ_0, must be provided. A good initial guess, i.e., a vector close to a minimizer, can usually only be obtained by utilizing knowledge on the process. The algorithm starts with an initialization step, α, and a new estimate of the solution χ^* is computed according to $\overline{\chi} = \chi + \alpha p$, where p is the search direction. This process is repeated iteratively until the convergence conditions are verified and a solution is computed. If the conditions are not met, then it is necessary to keep searching, so a new search direction and step length are computed, ensuring that the condition $\mathcal{F}(\overline{\chi}) < \mathcal{F}(\chi)$ is respected [38].

The most common method used to compute the search direction and step length using a line-search strategy is the Newton method [38]. First the local behaviour of the objective function $\mathcal{F}$ at the current iterate χ is approximated by second order Taylor series expansion as follows,

$$\mathcal{F}(\chi + p) \approx \mathcal{F}(\chi) + p^T \nabla_\chi \mathcal{F} + \frac{1}{2} p^T \nabla_\chi^2 \mathcal{F} p \,. \tag{2.2}$$

where $\nabla_\chi \mathcal{F}$ and $\nabla_\chi^2 \mathcal{F}$ are, respectively, the gradient and Hessian of the objective function. This approximation is minimized in order to compute the search direction p. The corresponding step length can then be determined by minimizing $F(\overline{\chi})$ with respect to α. The search direction and step length are, finally, used to determine the new value of the design variables $\overline{\chi}$ and the loop restarts [38].

Constrained Optimization

The following definition of Constrained Optimization is based on the work presented in [38, 47].

As previously stated, most optimization problems, and in particular optimal control problems, have to satisfy sets of equality and inequality constraints.

Following the formulation presented in equation (2.2), the equality and inequality constraints, G_i and H_i respectively, will change the feasible set making it not include the global optimum of the unconstrained problem. Therefore it is necessary to modify the search method by altering either the magnitude of the step, α, via a line-search or the direction itself, p.

First, all constraint functions are replaced by suitable approximations. The simplest is the linear approximation.

After defining the constraints approximations and determine the feasible set of solutions and their boundaries it is necessary to compute the search direction for the next iteration of the optimization problem. The feasible moving directions are now restricted by all the equality constraints G_i and by some inequality constraints. If $H_i(\chi) > 0$ holds, then since H_i is continuous I get $H_i(\chi + \alpha p) > 0$ for all $p \in \mathbb{R}^n$ provided $\alpha > 0$ is sufficiently small. If, however, $H_i(\chi) = 0$ holds, then an arbitrarily small change of χ in the "wrong" direction may lead to $H_i(\chi + \alpha p) < 0$. In the first case, the inequality constraints are called inactive and in the second case they are called active since they restrict the feasible search directions [38].

In order to deal with all active constraints, $C(\chi)$, including equality constraints, it is necessary to introduce a auxiliary function called *Lagrangian*:

$$L(\chi, \lambda) = \mathcal{F}(\chi) - \lambda^T C(\chi) \tag{2.3}$$

which is a scalar function of the χ and the Lagrange multipliers λ. The idea behind this definition is that the additional term $-\lambda^T C(\chi)$ penalizes violations of the state constraints.

Based on the *Lagrangian* it is possible to define a linear system referred to as the Kuhn-Tucker (KT) or Karush-Kuhn-Tucker (KKT) system which connects the gradient of the cost function to active constraints by serving as a guideline to find local minimizers in the constrained optimization.

Finally, the optimization algorithm can proceed as in the unconstrained case and look for the minimizers of the cost function $\mathcal{F}$ amongst the feasible points of χ.

There are two common approaches to solve non-linear constrained optimization problems. These are the so-called Sequential Quadratic Programming (SQP) and the Interior-Point Method (IPM).

In SQP the solution converges to the optimum by simultaneously improving the objective and tightening feasibility of the constraints. However, only the optimal and final iteration is guaranteed to be feasible. This results from the fact that SQP works computing the search direction p by solving the KKT equations, which due to their cost nature do impose that the intermediate iterations are inside the feasible set. This is a major advantage of this method since the computation of feasible points in the case of non-linear constraint functions may be a difficult task [48].

The class of interior-point methods generate a iteration sequence which always lies in the interior of the feasible set. For generating this sequence, in each iterate the entire set of inequality constraints H_i, presented in equation (2.2), is used by transforming them into equality constraints using slack variables as follows,

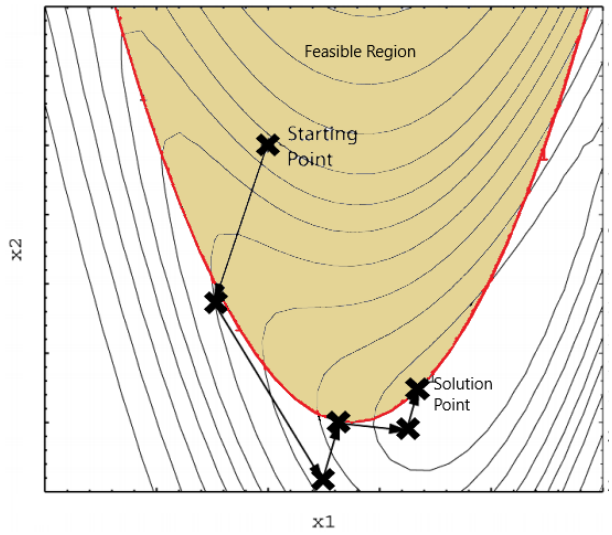


Figure 2.2: Example of SQP iteration Path for a problem with two optimization variables. The Feasible Region is represented in yellow. The intermediate iterations can be outside the feasible region.

$$\min_{s_i \geq 0} H_i(\chi) - s_i, i \in \mathcal{I}\,. \tag{2.4}$$

This way the number of constraints to be considered in each iteration may become considerably larger and thus the computational effort in each iteration grows. On the other hand, one avoids the potentially time-consuming identification of the working set [38].

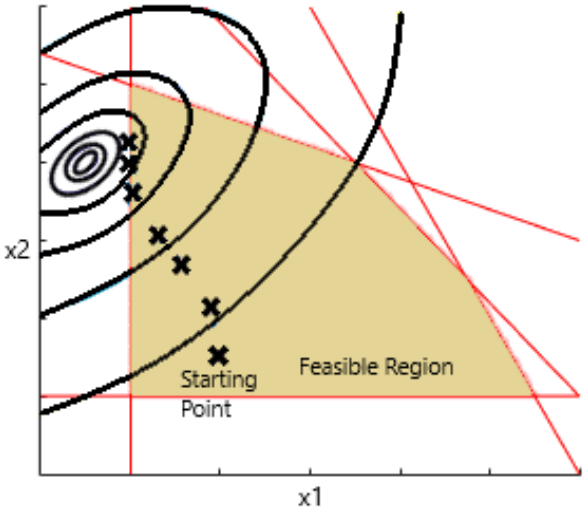


Figure 2.3: Example of IPM iteration Path for a problem with two optimization variables. The Feasible Region is represented in yellow. The intermediate iterations allows stay inside the feasible region.

2.2 Trajectory Optimization

Numerical trajectory optimization algorithms find the optimal path by representing mathematically a robot with motion equations. This optimum solution is a sequence of controls that moves the dynamic system

between an initial and final point in state space. The trajectory will minimize some cost function, which is typically an integral along the trajectory while satisfying a set of user-defined constraints.

Numerical trajectory optimization algorithms solve variations of the following problem,

$$
\begin{aligned}
\text{Minimize} \quad & \mathcal{F}_B\big(t_0, t_f, X(t_0), X(t_f)\big) + \int_{t_0}^{t_f} \mathcal{F}_P\big(\tau, X(\tau), U(\tau)\big) d\tau \\
\text{w.r.t.} \quad & t_0,\ t_f,\ X(t),\ U(t)\,, \\
\text{subject to} \quad & \dot{X}(t) = f\big(t, X(t), U(t)\big) && \textbf{system dynamics} \\
& X^- \leq X(t) \leq X^+ && \textbf{path state bounds} \\
& U^- \leq u(t) \leq U^+ && \textbf{path control bounds} \\
& t^- < t_f \leq t^+ && \textbf{limits on final time} \\
& X(t_0) = X_0 && \textbf{initial state boundary} \\
& X_f^- \leq X(t_f) \leq X_f^+ && \textbf{limits on final state}
\end{aligned}
$$

where $X(t)$ represents the vector of states, $U(t)$ is the set of control variables at time t and $\mathcal{F}_B$, $\mathcal{F}_P$ are the terminal and path costs respectively. A state variable is a variable that is differentiated in the dynamics equation, whereas a control variable only appears algebraically [49].

This optimization problem is also called a two-point boundary value problem (BVP) since it is defined by a set of ordinary differential equations (ODE), that describe the system dynamics, with the specification of state values in the boundaries of the problem, $X(t_0)$ and $X(t_f)$.

The dynamic constraints of the system, $f\big(t, X(t), U(t)\big)$, are continuous functions of the optimization variables and must be converted to discrete-time models in order to be introduced in the optimization problem as equality and inequality constraints.

The following discretization methods are based on the work of Betts [47].

Single Shooting

Single shooting is probably the simplest method for transcribing an optimal control problem. This method solves boundary value problems by first considering an initial value problem where only the initial state is bounded. The system dynamics are then iteratively simulated until the final boundary value problem is satisfied.

As shown in figure 2.4 the optimization interval can further be divided into N sub-steps to reduce the transcription errors. However, due to the formulation of this method, in every iteration of the optimization, all sub-steps of the trajectory are re-optimized but only the final state X_f must satisfy the final boundary value problem, which can result in large changes in the path from iteration to iteration [47].

This method can be divided into direct and indirect formulations. Direct shooting methods integrate the state equations directly between the discretization nodes. Therefore, the variables for a direct shooting application are chosen as a subset of the initial and final conditions and the parameters. Thus for each phase the vector of optimization variables χ^k is defined as:

$$\chi^k = \{X(t_0),\ t_0,\ X(t_f),\ t_f,\ p\} \tag{2.5}$$

The total set of NLP variables to optimize is then

$$\chi \subset \{\chi^1,\ \chi^2,\ \ldots,\ \chi^N\} \tag{2.6}$$

Indirect shooting methods iteratively solve the initial value problem and then evaluate constraints to adjust initial conditions. Therefore, for indirect shooting, the variables are chosen as a subset of the boundary values for the optimal control necessary conditions. In this case, the Hamiltonian is defined as:

$$H = \mathcal{F}_P(t,\ X,\ U) + \lambda^T f \tag{2.7}$$

where λ adjoins the equations of motion constraints to the path objective function. The conditions for optimality require that λ satisfy

$$\dot{\lambda} = -\left(\frac{\partial H}{\partial X}\right)^T \tag{2.8}$$

$$\frac{\partial H}{\partial U} = 0 \tag{2.9}$$

To obtain the adjoint vector, equation (2.8) is integrated back in time starting from the terminal condition.

$$\dot{\lambda} = -\left(\frac{\partial f}{\partial X}\right)^T \lambda - \left(\frac{\partial \mathcal{F}_P}{\partial X}\right)^T \tag{2.10}$$

$$\lambda(t_f) = \frac{\partial J_P(X_f, t_f)}{\partial X} \tag{2.11}$$

Now, the optimal control $U()$ must minimize the Hamiltonian function and can be computed over the time period by solving for U.

$$\left(\frac{\partial J_P}{\partial U}\right) + \left(\frac{\partial f}{\partial U}\right)^T \lambda = 0 \tag{2.12}$$

Therefore, for the indirect shooting case the NLP variables are as follows,

$$\chi = \{\lambda(t_0), t_f\} \tag{2.13}$$

A major difference between direct and indirect shooting occurs in the definition of the control functions $U(t)$. For indirect shooting, the control is defined at each point in time by the maximum principle (2.9). Thus in some sense, the values $\lambda(t_0)$ become the ”parameters” that define the optimal control function. For direct shooting, the control must be defined implicitly or explicitly at each discretization node by a

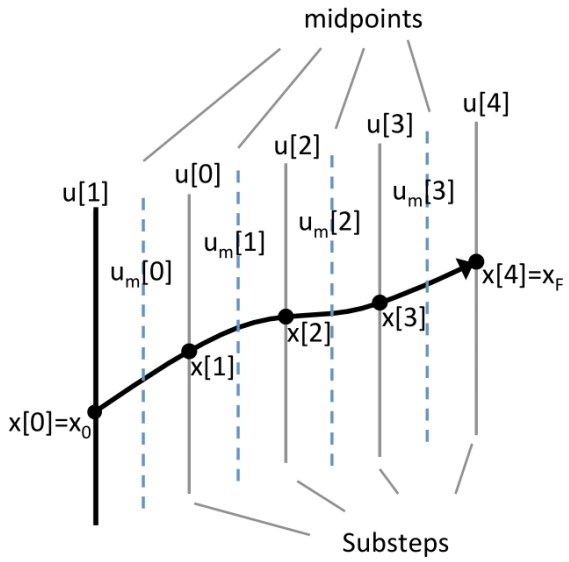


Figure 2.4: Example of a trajectory optimization problem that employs single shooting discritization (Image from [50]).

finite set of parameters p that are part of the set of optimization variables [47].

However, these methods suffer from a big setback when solving non-linear problems. The essential shortcoming of these methods is that small changes introduced early in the trajectory can propagate into very non-linear changes at the end of the trajectory. While this effect can be catastrophic for an indirect method, it also represents a substantial limitation on the utility of a direct formulation [47].

Multiple Shooting

In a general form, the multiple shooting method can be stated as follows: compute the unknown initial values $\mathbf{y}(\mathbf{t_0}) = \mathbf{y_0}$ for each shooting segment such that the boundary condition

$$0 = \phi[\mathbf{y}(\mathbf{t_f}),\ \mathbf{t_f}] \tag{2.14}$$

holds for some value of $t_0 < t_f$ which satisfies $\dot{\mathbf{y}} = f(\mathbf{y}(\mathbf{t}), \mathbf{t})$, where $\mathbf{y}$ is the set of dynamic variables in the optimization problem.

Multiple shooting works by breaking up a trajectory into some number of shorter segments and using single shooting to solve for each segment. Thus I break the time domain into smaller intervals of the form

$$t_0 < t_1 < \cdots < t_N = t_f \tag{2.15}$$

As the segments get shorter, the relationship between the optimization variables and the objective function and constraints becomes more linear, reducing the discretization errors.

Let me denote $\mathbf{y_i}$ for $i = 0,\ \ldots,\ (N-1)$, as the initial value for the dynamic variable at the beginning of each segment. For segment i I can integrate the differential equations from t_i to the end of the segment at t_{i+1}. Denote the result of this integration by $\mathbf{y_i'}$. Collecting the results for all segments let me

define a set of NLP variables

$$\chi = \{\mathbf{y_0}, \mathbf{y_1}, \ldots, \mathbf{y_{N-1}}\} \tag{2.16}$$

In multiple shooting, the end of one segment will not necessarily match up with the start of the next if the problem is infeasible, as shown in figure 2.5. This difference is known as a defect, and it is added to the constraint vector in order to be minimized [47].

$$c(\chi) = \begin{bmatrix} \mathbf{y_1} - \mathbf{y_0'} \\ \mathbf{y_2} - \mathbf{y_1'} \\ \vdots \\ \phi[\mathbf{y_N},\ \mathbf{t_f}] \end{bmatrix} = 0 \tag{2.17}$$

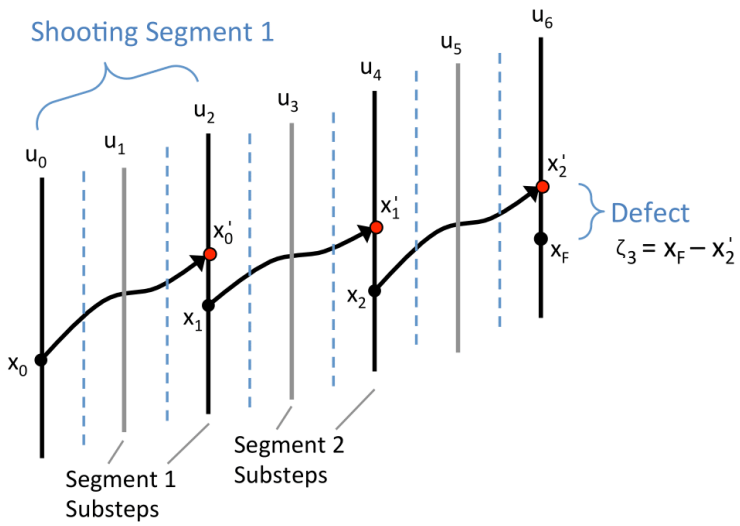


Figure 2.5: Example of a multiple shooting optimization where the defect of the equality constraints is non-zero meaning the problem is infeasible.(Image from [50]).

One obvious result of the multiple shooting approach is an increase in the size of the problem that the Newton iteration must solve since additional variables (the start of each segment) and constraints (defects) are introduced for each shooting segment. Although it might seem that this would make the low-level optimization problem harder, it actually turns out to make it easier because the Hessian matrix $\nabla_\chi \mathcal{F}$ which appears in the calculation of the Newton search direction is sparse since variables early in the trajectory do not change constraints at the end of it.[47]

The multiple shooting concept can be incorporated into either a direct or indirect method. The distinction between the two occurs in the definition of the dynamic variables $\mathbf{y}$, the dynamic system, and the boundary conditions. For a direct multiple shooting method, I can identify the dynamic variables $\mathbf{y}$ with the state and control (X, U). By analogy, the dynamics are given by the original state equation and path constraints. For an indirect multiple shooting algorithm, the dynamic variables $\mathbf{y}$ must include the state, control, and adjoint variables $(X,\ U,\ \lambda)$. The dynamics are given by the original state equations and the appropriate necessary conditions for the adjoint variables [47].

One difficulty with direct shooting methods is that it is difficult to implement path constraints since the intermediate state variables are not decision variables in the non-linear program. Another difficulty with shooting methods, particularly with direct shooting, is that the relationship between the decision variables and constraints is often highly non-linear, which can cause poor convergence in some cases [49].

Some of the most commonly used numerical techniques for Direct Single and Multiple shooting discretization are the Runge-Kutta methods, which are a family of implicit and explicit iterative methods. The most widely known member of the Runge-Kutta family is the 4th order Runge-Kutta method where the system dynamics is approximated by the present value (X_i) plus the weighted average of four increments, where each increment is the product of the size of the interval, Δt_i, and an estimated slope specified by the dynamics f.

$$X_i' = X_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.18}$$

$$t_{i+1} = t_i + \Delta t_i \tag{2.19}$$

where the coefficients k_j are defined by the system dynamics in four midpoints as

$$k_1 = f(t_i, X_i, U_i) \tag{2.20}$$

$$k_2 = f(t_i + \frac{\Delta t_i}{2}, X_i + \Delta t_i \frac{k_1}{2}, U_i) \tag{2.21}$$

$$k_3 = f(t_i + \frac{\Delta t_i}{2}, X_i + \Delta t_i \frac{k_2}{2}, U_i) \tag{2.22}$$

$$k_4 = f(t_i + \Delta t, X_i + \Delta t_i k_3, U_i) \tag{2.23}$$

Collocation

Just as in multiple shooting, in collocation or transcription methods I break the time domain into smaller intervals of the form (2.15). Let me consider taking a single step with an explicit method such as Euler's. Following the multiple shooting methodology, I then must impose constraints of the form

$$0 = X_{i+1} - X_i' = X_{i+1} - (X_i + \Delta t_i \ f_i) \tag{2.24}$$

where $\Delta t_i = t_{i+1} - t_i$ for all of the segments $i = 0, \ldots, (N-1)$.

The NLP variables then become the values of the state and control at the grid points, namely,

$$\chi = \{X_0, \ U_0, \ X_1, \ U_1, \ \ldots, \ X_{N-1}, \ U_{N-1}\} \tag{2.25}$$

The set of NLP variables may be augmented to include the times t_0 and t_f and, for some discretizations, the values of the state and control at collocation points between the grid points. Similarly to multiple shooting, the key notion of the collocation methods is to replace the original set of ODE with a

set of defect constraints $\zeta_i = 0$ which are imposed on each interval in the discretization.

Of course there is no reason to restrict the approximation to an Euler method. One of the most popular and effective choices for the defect constraint in collocation methods is the Hermite-Simpson method:

$$0 = X_{i+1} - X_i - \frac{\Delta t_{i+1}}{6}[f_{i+1} + 4f_{i+\frac{1}{2}} + f_i] = \zeta_i \tag{2.26}$$

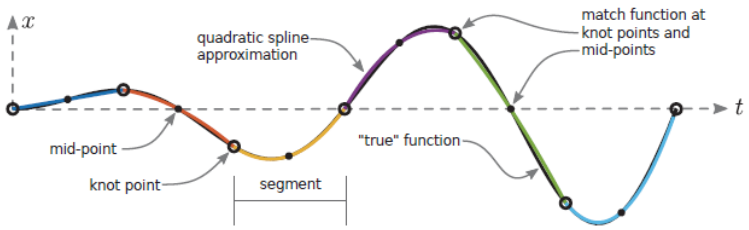


Figure 2.6: Hermite-Simpson Collocation example from Kelly [49].

This method provides a higher-order approximation when compared to the Euler method, resulting in a smaller transcription error at the cost of higher computation times. The Hermite-Simpson method approximates the objective function and system dynamics as piecewise quadratic functions by intruding an additional collocation point, $f_{i+\frac{1}{2}}$, in the middle of the original transcription segment. An additional benefit of the Hermite-Simpson collocation method is that the state trajectory is a cubic Hermite spline, which has a continuous first derivative [49].

Schemes of this type are referred to as collocation methods because the solution is a piecewise continuous polynomial that collocates (i.e. satisfies) the ODE's at the so-called collocation points in the subinterval $t_i \leq t \leq t_{i+1}$. The points t_i are also called grid points, mesh points, or nodes.

The non-linear programming problem which results from this formulation is large, similar to what happens in the multiple shooting method. Fortunately, the pertinent matrices for this NLP problem, namely the Jacobian and the Hessian are also sparse. Consequently exploiting sparsity to reduce both storage and computation time is a critical aspect of a successful implementation when using a direct transcription method [47].

2.3 Model Predictive Controller

In this section, I will discuss the concept and capabilities of Model Predictive Control. MPC is based on iterative, finite-horizon optimization where an explicitly formulated process model is used to predict future behaviour. Similar to the trajectory optimization described previously, the general design objective of MPC is to determine the optimum values for the control variables $U(t)$ in the prediction horizon that minimize a cost function based on the plant output $X(t)$. Its primary applications are stabilization and tracking problems.

The optimization performed by a Model Predictive Controller is illustrated in figure 2.7 and can be

described as follows: at time t the current plant state is sampled and an optimization problem is numerically solved for a relatively short time horizon in the future: $[t,\ t+T_h]$. The control problem is solved using the methods described in the trajectory optimization problems until time $t+T_h$ by discretization the dynamics in N steps. The optimum solution is a cost-minimizing control strategy of which only the first step is implemented in the real robot model. Then the plant state is sampled again and the calculations are repeated starting from the new current state, yielding a new control and new predicted state path. The prediction horizon keeps being shifted forward and for this reason, MPC is also called Receding Horizon Control (RHC) [51].

The time horizon T_h and the sampling steps N of the control problem are critical and must be carefully determined in order to find a balance between a better control performance achieved with large T_h and a reasonable and low computation time achieved with a lower T_h.

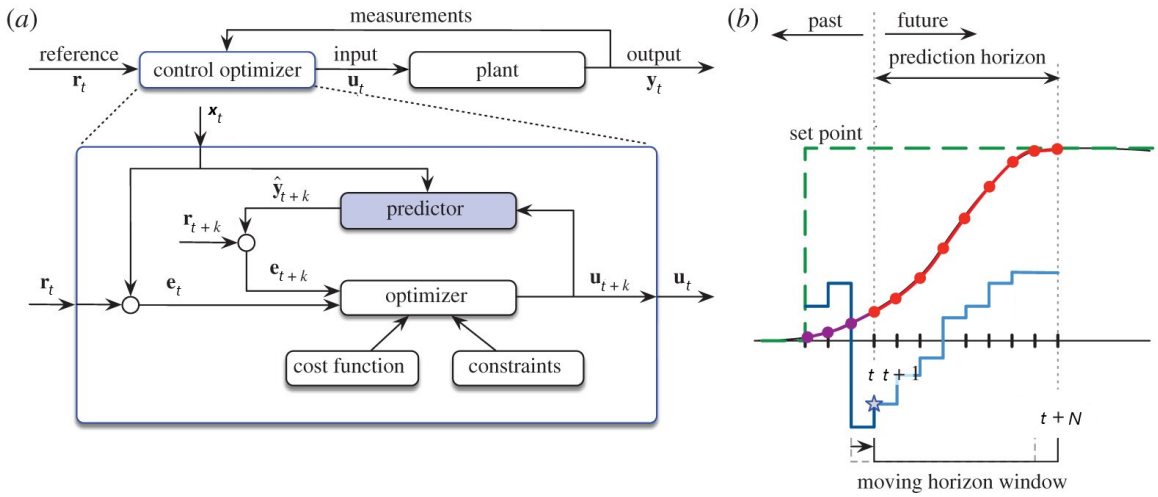


Figure 2.7: Overview of a Model Predictive Controller taken from [52].

Similar to the Trajectory optimization problem, most MPC controllers are able to account for constraints both in manipulated variables and states/controlled variables through the formulation of the optimization problem.

MPC strategies are also in general applicable to non-linear dynamics of the form:

$$\dot{X} = f(X, U) \tag{2.27}$$

When formulating the optimization problem in MPC, it is important to ensure that it can be solved in the short time available. For that reason, the optimization problem is typically cast into a Quadratic programming (QP) formulation where the objective function used is the common and popular Linear Quadratic Regulator (LQR). Therefore, the state feedback-based MPC optimization problem takes the following form [53]:

$$
\begin{aligned}
\text{Minimize} \quad & \|X_{t+N} - X_{t+N}^{ref}\|_R^2 + \sum_{i=0}^{N-1} \left(\|X_{t+i} - X_{t+i}^{ref}\|_Q^2 + \|U_{t+i} - U_{t+i}^{ref}\|_P^2 \right) \\
\text{w.r.t.} \quad & X, U \\
\text{subject to} \quad & X_{t+i+1} = f(X_{t+i}, U_{t+i}) \\
& X_t = X(t) \\
& X_{t+i} \in \mathfrak{X}_{\mathfrak{C}} \\
& U_{t+i} \in \mathfrak{U}_{\mathfrak{C}}
\end{aligned}
\tag{2.28}
$$

where i is the index along the prediction horizon, N is the length of the prediction horizon, Q, R, P are the state error, goal state error and control action error weight matrices respectively, X_{t+i} is the predicted system state vector, X_{t+i}^{ref} is the state reference signal, U_{t+i} is the predicted approximately optimal control action, U_{t+i}^{ref} is the input reference, the subscript $t+i$ is used to denote the sample of a signal at i steps ahead of the current time t, while $t+i+1$ indicates the next evolution of that step, $\mathfrak{U}_{\mathfrak{C}}$, $\mathfrak{X}_{\mathfrak{C}}$ represent the set of input and state constraints respectively and $X(t)$ is the value of the state vector at the beginning of the current MPC iteration.

In order for a solution in this problem to exist and be unique, Q, R and P should be real and symmetric, and in particular, Q should be positive semi-definite and P and R positive definite. These matrices can be tuned for each situation in order to obtain the desired optimal solution.

The solution of this optimization problem leads again to an approximately optimal control sequence

$$
\{U_t^*,\ U_{t+1}^*,\ \ldots,\ U_{t+N-1}^*\}
$$

where only U_t^* is applied while the whole process is then repeated.

2.3.1 Stability

The stability formulation present in this subsection is derived from the work introduced in [38, 54, 55].

In order to verify that my non-linear MPC controller achieves asymptotic stability, I will utilize the concept of Lyapunov functions and Lyapunov Stability Theory.

Consider the system in eq. (2.29), let $\mathbb{X}$ be a set that contains a point X' and $f(X_i) = Y \subseteq \mathbb{X}$ be a subset of the state space.

$$
X_{i+1} = f(X_i), \quad X(0) = X_0 \tag{2.29}
$$

where $X_i \in \mathbb{X}$ with $i \geq 0$ is the time step sequence that satisfies the model, and X' is an equilibrium point of the system, i.e. $f(X') = X'$.

For this formulation, we say that X' is locally asymptotically stable if there exists $\eta > 0$ and a continuous function β such that the inequality

$$|X_i|_{X'} \leq \beta(|X_0|_{X'}) \tag{2.30}$$

holds for all $|X_0|_{X'} \leq \eta$.

We say that X' is asymptotically stable on a forward invariant set Y with $X' \in Y$ if there exists β such that (2.30) holds for all $X_0 \in Y$ and we say that X' is globally asymptotically stable if X' is asymptotically stable on $Y = \mathbb{X}$.

A function $V : Y \to R_0^+$ is a Lyapunov function on Y if there exists functions α_1 and α_2 that are continuous, strictly increasing and bounded with $\alpha(0) = 0$ and a function α_v that is continuous and strictly increasing such that:

$$\alpha_1(\|X - X'\|) \leq V(X) \leq \alpha_2(\|X - X'\|) \tag{2.31}$$

$$V(X_{k+1}) \leq V(X) - \alpha_v(\|X - X'\|) \tag{2.32}$$

holds for all $X \in Y$.

Then, if the system (2.29) admits a Lyapunov function V on a forward invariant set Y, I can conclude that X' is an asymptotically stable equilibrium on Y.

Therefore, the goal is to find an optimal value function, (2.33), that is also a Lyapunov function, proving stability for a general MPC formulation with an infinite prediction horizon.

$$V_N(X_0) := \inf_{\text{U admissible}} \mathcal{F}_N(X_0, U) \tag{2.33}$$

where $\mathcal{F}_N(\cdot)$ is the optimization function (2.28) with $N \to \inf$.

However, this formulation can only be applied directly to infinite horizon problems, requiring the addition of a terminal feedback law that can include either a terminal constraint or a regional terminal state cost in the case of finite-horizon problems. Nevertheless, general constrained optimization problems can be extremely difficult to solve, and simply adding terminal constraints may not be feasible without reformulating the problem.

The MPC formulation presented in (2.28) can be written as $\mathcal{F}_N(X_0, U) = \mathcal{F}_f(X_{t+N}) + \sum_{N-1}^{i=0} \mathcal{F}_f(X_{t+i}, U_{t+i})$. For this formulation I impose the following requirements:

- The horizon cost $\mathcal{F}_f(X_{t+i}, U_{t+i})$ satisfies $\mathcal{F}_f(0, 0) = 0$ and $\mathcal{F}_f(X_{t+i}, U_{t+i}) \geq \alpha_v$ for all $X_{t+i} \in Y_N$, $U_{t+i} \in \mathbb{U}$, where $Y_N \subset \mathbb{X}$ is the set of initial feasible states;
- The terminal cost $\mathcal{F}_f(X_{t+N})$ satisfies $\mathcal{F}_f(0) = 0$, $\mathcal{F}_f(X_{t+N}) \geq 0$ for all $X_{t+N} \in \mathbb{X}_f$, where $\mathbb{X}_f$ is a terminal set that is invariant under the terminal control law, and there exists a control law $K_f : \mathbb{X}_f \Rightarrow \mathbb{U}$ such that $\mathbb{X}_f(f(X_{t+i}, K_f(X_{t+i}))) - \mathbb{X}_f(X_{t+i}) \leq \mathbb{X}_f(X_{t+i}, K_f(X_{t+i}))$ for all $X_{t+i} \in \mathbb{X}_f$;
- The set $\mathbb{X}_f$ is positive invariant under $K_f(X_{t+i})$;

If all those requirements are fulfilled than the set of feasible positive initial states, Y_N, is positive invariant for the closed loop system and if $X' \in Y_N$ and V_N is continuous on some neighbourhood of X',

then X' is asymptomatically stable in S_N.

For the formulation presented in (2.28), the term final cost term $\|X_{t+N} - X^{ref}_{t+N}\|^2_R$ has a critical role for the closed-loop stability, as it accounts for the impact of events that may lie beyond the finite horizon by imposing a terminal safe region $\mathbb{X}_0$ around X'.

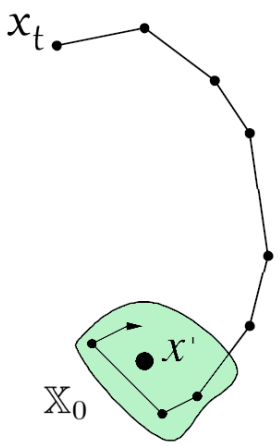


Figure 2.8: Practical Asymptotic Stability illustration.

Chapter 3

Hexarotor Dynamics and Kinematics

As mentioned before, the goals of this dissertation are to implement a trajectory optimizer and controller that are capable of generating and following optimal trajectories for an Hexarotor. To determine this trajectory while fulfilling all mission requirements and satisfying dynamical constraints of the UAV is fundamental to understand the kinematic model that describes the multicopter.

The chapter starts by defining two reference systems of coordinates that are fundamental to derive the mathematical model. Then, I analyse the full dynamics of a generic Hexarotor by defining the forces and moments acting on it during a flight mission and finally I derive the equations of motion that fully describe the multirotor.

3.1 Reference systems for the hexacopter

In order to derive the equations of motion (EoM) for a multirotor I consider the preliminary assumption that the curvature of the Earth and its rotation can be ignored in the mission's area since the duration and scale of the operation is considerably small when compared to the Earth.

Writing the equations that portray the complex dynamics of an aircraft implies first defining the system of coordinates to use. Therefore, it is useful to define two coordinate systems:

- A Earth fixed frame $\{\mathfrak{I}\}$ ($\mathbf{e}_\mathbf{x}^\mathfrak{I}$; $\mathbf{e}_\mathbf{y}^\mathfrak{I}$; $\mathbf{e}_\mathbf{z}^\mathfrak{I}$) tangent to the earth surface. Such frame is considered Inertial based on the previous assumption and uses the East, North, Up (ENU) system of coordinates.
- A body fixed frame $\{\mathfrak{B}\}$ ($\mathbf{e}_\mathbf{x}^\mathfrak{B}$; $\mathbf{e}_\mathbf{y}^\mathfrak{B}$; $\mathbf{e}_\mathbf{z}^\mathfrak{B}$) whose center coincides with the center of mass of the vehicle and such that $\mathbf{e}_\mathbf{z}^\mathfrak{B}$ is in the same direction of thrust generation.

In the $\{\mathfrak{I}\}$ frame the origin of the referential is free, meaning it can be placed based on the problem characteristics. In this case, it is set as the initial position of the UAV's center of gravity. This location was chosen to match the local origin of the state estimator incorporated in the flight controller that estimates position based on the integration of IMU data. The positive direction of the $\mathbf{e}_\mathbf{z}^\mathfrak{I}$ axis is in the direction normal to the earth ground level and pointing up. The remaining $\mathbf{e}_\mathbf{x}^\mathfrak{I}$ and $\mathbf{e}_\mathbf{y}^\mathfrak{I}$ axis point East and North respectively as shown in fig. 3.1.

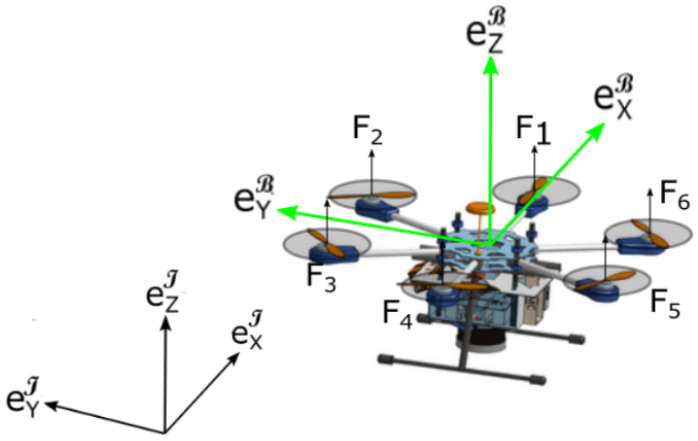


Figure 3.1: Representation of Inertial and Body frames. The green frame is the Body frame that is fixed to the UAV. The black frame is the Inertial Frame that is fixed to an arbitrary point.

In control theory, knowledge about the dynamic behaviour of a given system can be acquired through its states. The vehicle configuration is described by the position of center of mass in the inertial frame $\{\mathfrak{I}\}$ and the orientation of the $\{\mathfrak{B}\}$ frame with respect to the inertial frame [56]. The last is usually defined by means Euler angles: roll ϕ, pitch θ and yaw ψ. For sake of simplicity, let me denote the vectors for inertial position and body frame orientation by means of $\mathbf{p} = [\mathbf{x},\ \mathbf{y},\ \mathbf{z}]^{\mathbf{T}}$ and $\eta = [\phi,\ \theta,\ \psi]^T$, respectively, where $\phi \in\ ]-\pi/2,\ \pi/2[$, $\theta \in\ ]-\pi,\ \pi[$, and $\psi \in\ ]-\pi,\ \pi[$.

Notice that Siciliano et al. [57] states that the attitude representation using Euler angles suffers from the so-called gimbal-lock or loss of one degree of freedom when $\theta = \pm k\frac{\pi}{2}$, with $k = 1,\ 3,\ 5,\ \ldots$. However, in this project those singularities do not pose a significant problem since the designed controlled does not allow for those pitch angles.

Now I must deduce the equations describing the orientation of the mobile frame relative to the fixed one, which can be achieved through successive rotations about the three axes. I will use a $\mathbf{e_z} - \mathbf{e_y} - \mathbf{e_x}$ rotation where I first rotate about $\mathbf{e}_\mathbf{z}^\mathfrak{I}$ by the the yaw angle. This is followed by a rotation about the $\mathbf{e_y}$ axis in the rotated frame through the roll angle, followed by a third pitch rotation about the new $\mathbf{e_x}$ axis through the pitch angle that results in the following rotation matrix [58]:

$$\mathcal{R}^{\mathfrak{B}\mathfrak{I}}(\eta) = \begin{bmatrix} c_\theta c_\psi & c_\psi s_\theta s_\phi - c_\phi s_\psi & c_\phi c_\psi s_\theta + s_\phi s_\psi \\ c_\theta s_\psi & c_\phi c_\psi + s_\theta s_\phi s_\psi & c_\phi s_\theta s_\psi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \tag{3.1}$$

where c and s are shorthand forms for cosine and sine, respectively.

Then the translational transformation law for the multirotor is defined by:

$$\dot{\mathbf{p}} = \mathcal{R}^{\mathfrak{B}\mathfrak{I}}(\eta)\mathsf{v}_\mathfrak{B} \tag{3.2}$$

where the absolute linear velocity of the aerial vehicle is defined by the vector $\mathsf{v}_\mathfrak{B}$.

As for the rate of change of the Euler angles, in order to represent the angular velocity of the Body Frame $\{\mathfrak{B}\}$ with respect to the inertial frame I consider a new rotation matrix that derives from the measurement of the angular rates in different rotated frames. The roll angle undergoes two rotations, the pitch angle one rotation, and the final Euler angle, the yaw, is measured in the Inertial Frame $\{\mathfrak{I}\}$ with no rotations. Therefore, the rotational transformation law can be defined by:

$$\mathsf{w}_{\mathfrak{B}} = \mathcal{L}^{\mathfrak{I}\mathfrak{B}}(\eta)\dot{\eta} = \begin{bmatrix} 1 & 0 & -s_\phi \\ 0 & c_\phi & s_\phi c_\theta \\ 0 & -s_\phi & c_\theta c_\phi \end{bmatrix} \dot{\eta} \tag{3.3}$$

where the angular velocity of the aerial vehicle around each axis of the $\{\mathfrak{B}\}$ frame is defined by the vector $\mathsf{w}_{\mathfrak{B}}$ and $\mathcal{L}^{\mathfrak{I}\mathfrak{B}}$ is the Euler rate rotation matrix from frame $\mathfrak{I}$ to frame $\mathfrak{B}$.

3.2 Applied forces and torques

In order to derive equations of motion for the full system, the following assumptions taken from Alexis et al. [59] have to be made:

- The hexacopter is a rigid body;
- The hexacopter has a symmetrical structure;
- The body frame $\mathfrak{B}$ has its origin in the Center of Mass (CoM) of the hexacopter ;
- All external forces are applied on CoM and therefore do not cause torques;

To mathematically write the movement of an aircraft I must employ Newton's second law of motion:

$$\begin{cases} m\dot{\mathsf{v}}_{\mathfrak{I}} = \sum_i F_i \\ \mathcal{J}\dot{\mathsf{w}}_{\mathfrak{B}} = -\mathsf{w}_{\mathfrak{B}} \times (\mathcal{J}\mathsf{w}_{\mathfrak{B}}) + \sum_i M_i \end{cases} \tag{3.4}$$

where m is the mass, F_i and M_i are the vectors of forces and torques applied in the rigid-body, respectively and $\mathcal{J} \in \mathbb{R}_{3\times3}$ is the mass moment of inertia, where $\mathcal{J}$ is aligned with the body axes, given by

$$\mathcal{J} = \begin{bmatrix} \mathcal{J}_{xx} & \mathcal{J}_{xy} & \mathcal{J}_{xz} \\ \mathcal{J}_{yx} & \mathcal{J}_{yy} & \mathcal{J}_{yz} \\ \mathcal{J}_{zx} & \mathcal{J}_{zy} & \mathcal{J}_{zz} \end{bmatrix} \tag{3.5}$$

The axis of each propeller is parallel to the third axis of the $\{\mathfrak{B}\}$. Each propeller generates thrust along its axis that is proportional to the square of the propeller rotation speed. Furthermore, as the motor dynamics are considerably faster compared to those of the vehicle body and can be neglected.

The FRIENDS' project drone is a generic Hexarotor X geometry, with six rotors, mounted symmetrically along two orthogonal axes, as depicted in fig. 3.2. The axis of each propeller is parallel to the third

axis of the $\{\mathfrak{B}\}$. Each propeller generates thrust along its axis that is proportional to the square of the propeller rotation speed. Furthermore, as the motor dynamics are considerably faster compared to those of the vehicle body and can be neglected [56]. The individual thrusts are denoted by $F_i \in \mathbb{R},\ i = 1, ..., 6$. Note that if a propeller is spinning "clockwise", then the two adjacent ones will be spinning "counter-clockwise", so that torques are balanced if all propellers are spinning at the same rate.

During flight, the main forces applied in the hexarotor are:

- Force of gravity represented on frame $\mathfrak{I}$: $F_g = -mg$
- Rotor's Thrust represented on frame $\mathfrak{B}$: $F_i = k_n \omega_i^2$

where k_n is the Thrust constant and $\omega_i \geq 0$ is the rotational velocity of i-th rotor.

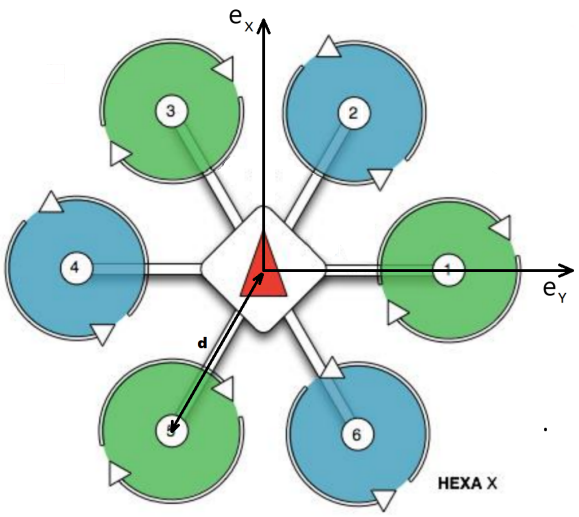


Figure 3.2: Hexarotor with X configuration adapted from [60].

As mentioned in [36, 61], I can additionally consider the influence of two important aerodynamic effects that appear in case of dynamic manoeuvres. These effects are the blade flapping and induced drag, which induce forces in the x-y rotor plane that contribute to the horizontal stability UAV as shown in Mahony et al. [58]. Although these two effects are treated as separate in [58], Omari et al. [62] showed that it is possible to combine these effects into a lumped coefficient $A_d = diag(c_d^x,\ c_d^y,\ 0)$. If the UAV is symmetric in the x-y axis the two coefficients can be further considered as equal, $c_d^x = c_d^y = c_d$.

This leads to the aerodynamic force $F_{a,i}$:

$$F_{a,i} = F_i A_d \mathsf{v}_{\mathfrak{B}} \tag{3.6}$$

The roll and pitch torques derive from the propellers' thrust and can be determined by $M_i = \sum r_i \times F_i$, where r_i is the momentum arm of each propeller's thrust in the $\{\mathfrak{B}\}$ referential.

Finally, the yaw torque is produced by varying the angular velocity of propellers rotating "clockwise" and the ones rotating counter "clockwise" creating unbalance in the sum of moments with respect to $\mathbf{e}_\mathbf{z}^{\mathfrak{B}}$ due to differences in rotor's drag, causing rotation.

All together, I find that the torques in the body frame are:

$$M_{\mathfrak{B}} = \begin{bmatrix} d\ k_n(-\omega_1^2 + \omega_4^2 + \frac{1}{2}(-\omega_2^2 + \omega_3^2 + \omega_5^2 - \omega_6^2)) \\ d\ k_n \frac{\sqrt{3}}{2}(\omega_2^2 + \omega_3^2 - \omega_5^2 - \omega_5^2) \\ b(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2 + \omega_5^2 - \omega_6^2) \end{bmatrix} \tag{3.7}$$

where b is the rotor's drag coefficient that relates the yawing moment about the body z-axis to the thrust of the six motors and d is the momentum arm for the symmetrical hexacopter.

In order to simplify the implementation of the dynamic equations in the optimization problems designed in the next chapters, the translational dynamics of the UAV are fully defined in the Inertial frame $\{\mathfrak{I}\}$. Therefore, in the following total dynamics of the vehicle, the state $\mathsf{v}_{\mathfrak{I}} = [\dot{x},\ \dot{y},\ \dot{z}]$ is used to represent the linear velocity in the I frame:

$$\begin{aligned} \dot{\mathbf{p}} &= \mathsf{v}_{\mathfrak{I}} \\ \mathsf{w}_{\mathfrak{B}} &= \mathcal{L}^{\mathfrak{I}\mathfrak{B}}(\eta)\dot{\eta} \\ \dot{\mathsf{v}}_{\mathfrak{I}} &= \frac{1}{m}\left(\mathcal{R}^{\mathfrak{B}\mathfrak{I}}(\eta)\sum_{i=1}^{6} F_i + \mathcal{R}^{\mathfrak{B}\mathfrak{I}}(\eta)\sum_{i=1}^{6} F_{a,i} + F_{ext} + F_g\right) \\ \dot{\mathsf{w}}_{\mathfrak{B}} &= \mathcal{J}^{-1}(-\mathsf{w}_{\mathfrak{B}} \times \mathcal{J}\mathsf{w}_{\mathfrak{B}} + M_{\mathfrak{B}}) \end{aligned} \tag{3.8}$$

where F_{ext} is the vector of the external forces acting on the vehicle (i.e wind).

Chapter 4

Trajectory Generation Formulation

This chapter addresses the problem stated in Section 2.2 applied to the objectives defined for this project. Therefore, I design an optimization problem capable of generating optimal and feasible discrete trajectories for a multicopter UAV in such a way that certain non-linear dynamic constraints and linear/non-linear non-dynamic constraints are satisfied and a cost function is optimized.

The algorithm was formulated using Python's symbolic framework CasADi [63] and was solved with the open-source and highly efficient IPOPT (Interior Point OPTimizer) [1] solver that is designed to find (local) optimum solutions of large-scale non-linear optimization problems using an interior-point method.

Alternative strategies were also investigated and tested in simulation. A different approach, described in Appendix A, was studied, where instead of optimizing the trajectory based on a discretization of the non-linear UAV dynamic system, the waypoint and minimum time objectives are converted into an optimization problem that treats the multirotor trajectory as a piecewise continuous polynomial function where the optimization variables are the polynomial coefficients.

In the end, although the polynomial optimization is considerably less computationally demanding and the final UAV trajectory is smoother due to the continuous formulation, I concluded that the alternative approach did not fulfil the terrain-following and avoidance objectives in the way they are imposed by the FRIENDS project, as it is not possible to define constraints directly on the dynamic states and inputs of the UAV and there is no freedom to optimize the altitude of the overall trajectory since each waypoint must be fully defined in 3D space to be converted into a polynomial. In other words, the altitude of each waypoint must be enforced by the user and therefore is not an optimization variable as desired.

4.1 System Dynamics

In this section, I describe the system dynamics applied in the trajectory optimization problem. As previously mentioned, the goal of the trajectory generation is to determine the optimal control path to be executed by the multirotor. This path is directly used in the optimization control described in the next section by setting the result of the trajectory problem as the reference for the Model Predictive Controller.

[1] https://coin-or.github.io/Ipopt/

In the next chapter, I explain why and how the system dynamics of the multirotor can be simplified in order to be efficiently used in the control problem. This simplification approximates the rotational dynamics of the UAV by a first-order system and instead of using the usual rotor's rotational velocity as inputs, I use normalized thrust and attitude references.

In this optimization problem, using the same approximated model as the MPC is important, not only, to reduce the computation time, but also to facilitate the integration of the optimization result in the MPC. However, the model used in the MPC does not allow for constraints in angular velocity or linear acceleration that are required to obtain a smooth trajectory. In order to include such constraints, the attitude dynamics applied in this optimization problem are approximated by a second-order system with respect to η of the form present in [64], that allows the control inputs to be the desired angles like in the MPC but gives a good estimation of the angular velocity. In fact, since I am approximating the attitude dynamics of the system and not defining the real dynamics of the PID controllers, this second-order approximation was found, through system identification tests in several simulated flights, to represent more accurately the attitude response of the system when compared to the first order approximation used in the MPC. In order to minimize and set a maximum limit for linear acceleration, I added an integrator to the system dynamics, since the translational dynamics that describe a multirotor do not include acceleration as a state.

The full UAV dynamics used in the trajectory generation is described by the following equations:

$$\dot{\mathbf{p}} = \mathsf{v}_{\mathfrak{I}}\,, \tag{4.1a}$$

$$\dot{\mathsf{v}}_{\mathfrak{I}} = \mathcal{R}^{\mathfrak{B}\mathfrak{I}}(\eta)\begin{bmatrix}0\\0\\\tilde{T}\end{bmatrix} + \begin{bmatrix}0\\0\\-g\end{bmatrix}\,, \tag{4.1b}$$

$$\dot{\eta} = \mathsf{w}_{\mathfrak{I}}\,, \tag{4.1c}$$

$$\dot{\mathsf{w}}_{\mathfrak{I}} = -2\xi\omega\mathsf{w}_{\mathfrak{I}} + \omega^2(K\eta_{cmd} - \eta)\,. \tag{4.1d}$$

where $\mathsf{w}_{\mathfrak{I}} = [\dot{\phi},\ \dot{\theta},\ \dot{\psi}]^T$ is the vector of the Inertial Frame angular rates, $\tilde{T}$ is the mass normalized Thrust command, $\xi = [\xi_\phi,\ \xi_\theta,\ \xi_\psi]^T$, $\omega = [\omega_\phi,\ \omega_\theta,\ \omega_\psi]^T$ and $K = [K_\phi,\ K_\theta,\ K_\psi]^T$ are the vectors of damping factor, natural frequency and gains for the second order approximation of the attitude dynamics respectively and $\eta_{cmd} = [\phi_{cmd},\ \theta_{cmd},\ \psi_{cmd}]^T$ is the vector of attitude reference inputs. All the dynamic constants are obtained through a system identification curve fitting that outputs the vectors ξ, ω and K.

The dynamics of the added velocity integrator is as follows:

$$\dot{\mathsf{v}}_{int} = a\,. \tag{4.2}$$

where v_{int} is a state of the integrator that represents the velocity of the UAV and a is the acceleration in the inertial frame $\mathfrak{I}$ defined as an input in the integrator's dynamics. In order to link the integrator dynamics with the overall multirotor dynamic model, the velocity v_{int} must be equal to the velocity of the UAV model, $\mathsf{v}_{\mathfrak{I}}$. This is guaranteed by adding the following path constraint:

$$\mathsf{v}_{int} = \mathsf{v}_{\mathfrak{I}} \,. \tag{4.3}$$

The described system dynamics represent the following state vector

$$X = \begin{bmatrix} \mathbf{p}^T & \mathsf{v}_{\mathfrak{I}}^T & \eta^T & \mathsf{w}_{\mathfrak{I}}^T & \mathsf{v}_{int}^T \end{bmatrix}^T \tag{4.4}$$

And control input vector, U, consisting of four inputs,

$$U = \begin{bmatrix} \tilde{T} & \phi_{cmd} & \theta_{cmd} & \psi_{cmd} & a^T \end{bmatrix}^T \tag{4.5}$$

4.2 Waypoint Objective

As previously mentioned, one of the objectives of this optimization problem is to generate a trajectory capable of passing through pre-defined waypoints. However, and unlike most previously studied cases, those waypoints are only defined in the 2D horizontal plane. This formulation is chosen because another goal of the trajectory optimizer is to follow and avoid the terrain which means that the vertical coordinate z should be fully determined by the optimization process. Therefore, in the following formulation each waypoint is represented by the vector $p_k^w = [x_k^w,\ y_k^w]$.

Table 4.1: Representation of waypoint discrete allocation.

time	t_0	$\cdots$	T_1	$\cdots$	$T_K = t_f$
x	x_0	free	x_1^w	free	x_K^w
y	y_0	free	y_1^w	free	y_K^w
z	z_0	free			

Let the set of K waypoints be given by tuples $(p_1^w, T_1), (p_2^w, T_2), \cdots (p_K^w, T_K)$, where the time instants corresponding to these waypoints are denoted by the subscript T_k, with $k = 1, \ldots, K$. These time instants to which the waypoints are allocated too are initially considered fixed and user defined based on the distance between waypoints and the maximum horizontal velocity allowed. To generate a trajectory passing through this sequence of waypoints one would typically define a distance cost or constraint and construct a discrete optimization control problem such that the output state vector X_{T_k} passes through the waypoints at specified time instants, i.e. $p'_{T_k} = p_k^w$, for $k = 1, \ldots, K$.

For cost-based formulations, quadratic distance costs are robust in terms of convergence and implemented as

$$\mathcal{F}_{dist,k} = (p'_{T_k} - p_k^w)^T (p'_{T_k} - p_k^w) \tag{4.6}$$

where p'_{T_k} is the part of the state vector X that represents the horizontal position state at a user-defined time T_k. However, such a cost-based formulation is only a soft requirement and if summed with other cost terms does not imply that the waypoint is actually passed within a certain tolerance [27]. To guarantee that the trajectory passes all waypoints within a tolerance, a constraint-based formulations can be used,

such as

$$(p'_{T_k} - p_k^w)^T (p'_{T_k} - p_k^w) \leq \sigma_w^2 \tag{4.7}$$

which in the general problem is part of the set of inequality path constraints, and ensures the optimized trajectory passes by p_k^w at instant T_k within tolerance σ_w.

4.3 Terrain Following and Terrain Avoidance Objective

The safe flyable motion in the vicinity of complex terrain in the vertical plane is named Terrain-Following (TF), whereas the motion in the horizontal plane is called Terrain Avoidance (TA) [16]. The high complexity of the terrain when incorporated in the problem with the vehicle flying dynamics can result in optimization not only highly convoluted but also non-convex. However, in most cases of this type of problem, a local minimum is considered acceptable if the appropriate constraints are fulfilled since it would be impractical to analyse the entire state space, with many minimums and maximums in the terrain model, to compute the global minima.

In my formulation, in order to be used by the optimizer, it is assumed that the terrain being followed is preliminarily available from a known DEM file. Terrain following necessarily entails terrain collision avoidance because the optimal aircraft flight path must be free of terrain obstacles while maintaining the aircraft as close to the desired altitude above ground as possible. A variety of different techniques have been used to plan such trajectories. Typically, a weighted norm of the aircraft distance from the terrain is used as a performance index. In addition, since the cost is an integral over the path, this choice of cost together with the dynamic constraints and time optimal cost does not keep the aircraft as close to the desired altitude as possible, but rather minimizes the distance only in an ”average” sense [65]. Consider the difference between the drone's height above the terrain and the desired altitude,

$$h(t) \equiv z(t) - h^{des} - h_{terrain}\big(x(t), y(t)\big) \tag{4.8}$$

where $h_{terrain}\big(x(t), y(t)\big)$ is the height of the terrain at the multirotor's position, h^{des} is the desired altitude above ground and $\Big[x(t), y(t), z(t)\Big]$ is the 3D position of the UAV in the inertial ENU referential.

The terrain-following problem involves minimizing the altitude distance to the desired altitude above ground.

$$\mathcal{F}_{TF} = \int_{t_0}^{t_F} k_{TF}[h(t)]^2 \, dt \tag{4.9}$$

where $h(t)$ is squared to guarantee differentiability of the cost function with respect to the states and k_{TF} is a constant coefficient that adjusts the weight of this cost function in the total cost of the optimization problem.

Finally, the Terrain Following and Avoidance problem require the definition of additional path constraints (see fig. 4.1) to guarantee that the UAV does not fly above a maximum height, defined in this

project by the sensors operational range, and below a minimum absolute height defined by the altitude of the ground [16, 65].

$$h^- \leq h'(t) \leq h^+ \tag{4.10}$$

where $h'(t)$ is the drone's height above the terrain defined as:

$$h'(t) \equiv z(t) - h_{terrain}\big(x(t), y(t)\big) \tag{4.11}$$

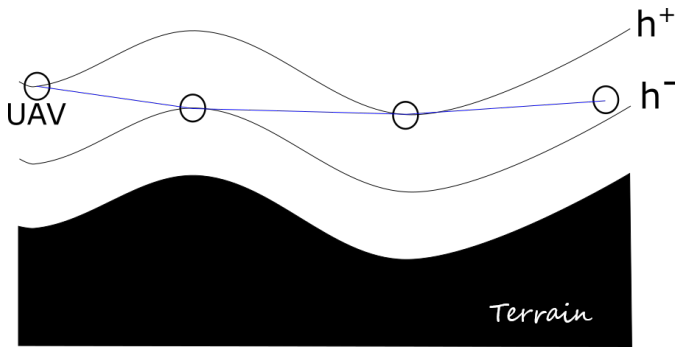


Figure 4.1: Representation of maximum and minimum altitude constraints imposed by the terrain profile. The circles represent discretization nodes of the optimization problem and the blue line illustrates the UAV path.

In practice, the minimum altitude above ground, h^-, must be equal or higher than a set clearance height, used so that the aircraft maintains a safe tolerance above the terrain to compensate for flight control errors or other disturbances.

4.4 Time-Optimal Objective

One of the biggest limitations when designing a UAV trajectory is the battery capacity and thus its autonomy on air. Therefore, to perform a long-range mission it is fundamental to minimize as much as possible the flight time. In trajectory optimization problems, it is common to include the total trajectory time, t_f, as an optimization variable and defining the number of collocation points, N, for the discretization of the dynamics as the independent variable.

Optimizing for a time-optimal trajectory means that the problem's cost function must contain a term penalizing the total trajectory time, $\mathcal{F}_t = t_f$. Therefore, the optimization variable t_f must be positive $t_f > 0$ which allows the integration step to be safely determined as $\Delta t = \frac{t_f}{N}$.

The main problem of a time-optimal trajectory arises when this objective is coupled with the waypoint tracking problem.

As mentioned before, adding waypoints to a trajectory requires that a cost or constraint is allocated to a specific node at a time that depends on the total trajectory time, $t_k = k\Delta t$. This means that even if I optimize the total time t_f the time t_k is still a fraction of t_f. In most of the optimization problems, it is not possible to know a priori how much time or what fraction of the total time is optimal to spend between

two waypoints, making it impossible to determine the optimum solution with such formulation [27].

In order to solve this limitation, Neunert et al. [66] implemented an intermediate waypoint cost spread along the trajectory using an exponential weight formulation. However, the waypoint objective is defined as a soft constraint where parameters such as the width and mean of the time spread must be user-defined leading to suboptimal solutions in most cases. Another approach is presented by [27] which introduces complementary constraints that change a set of progress variables every time the trajectory passes in the proximity of each waypoint. This approach does not allocate costs or constraints to a specific node allowing for almost optimum time solutions. However, due to the use of complementary constraints, the optimization problem becomes considerably more complex which results in very long computation times.

The approach described in this paper to tackle this problem is similar to what Falanga et al. [26] implemented and is based on a time elastic band, where now we no longer assume that the time allocated to each waypoint node T_k is fixed. Therefore, I formulated that the time to go from one waypoint to the next, ΔT_k, is now an optimization variable. This implies different discretization steps Δt for each waypoint segment. Unfortunately, allowing varying time steps Δt_k can negatively impact the discretization consistency over the trajectory mainly if the resulting Δt_k are very different from each other. In order to solve this limitation, if Δt_k is higher than a certain threshold, based on the dynamic constraints of the UAV and errors of discretization, the k section of the trajectory is re-optimized with a higher number of collocation points.

In this formulation, the node in which the UAV must go through each of the waypoints is fixed, however, that node's time is an optimization variable, allowing for an independent time optimization for each waypoint. Therefore, the independent and fixed variable in this formulation is the number of discretization intervals between each waypoint, N_k^w. In this way, the integration time for each node is different for every waypoint path segment as illustrated in fig. 4.2 and is defined by

$$\Delta t_k = \frac{\Delta T_k}{N_k^w} \,. \tag{4.12}$$

where $\Delta T_k = T_k - T_{k-1}$ is an optimization variable that represents the time necessary to go from waypoint $p^w_{T_{k-1}}$ to $p^w_{T_k}$.

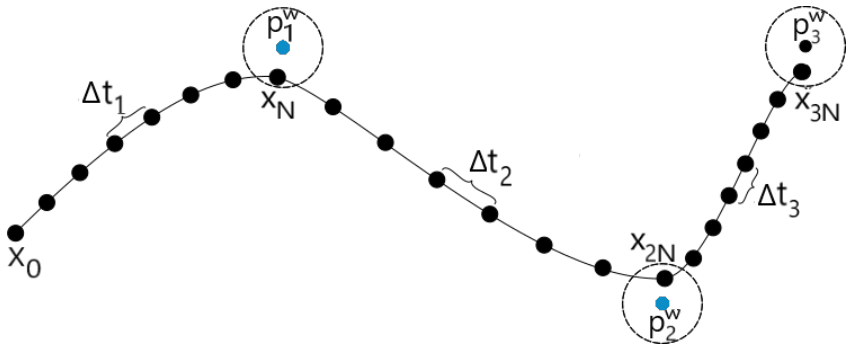


Figure 4.2: Illustration of Time Elastic Band Formulation for multiple waypoints. The black dots represent the discretization nodes. The waypoints are represented by the blue dots and the tolerance.

Finally, in order to optimize the total trajectory time a cost function penalizing the terminal time is added:

$$\mathcal{F}_t = \int_{t_0}^{t_f} k_{time} d\tau = k_{time}(t_f - t_0) \tag{4.13}$$

where k_{time} is the terminal cost factor and t_f is the total trajectory time defined in this formulation by the sum of the times of every waypoint.

$$t_f = \sum_{i=1}^{K} \Delta T_k \,. \tag{4.14}$$

Another advantage of the implemented time elastic band is the possibility of stretching the time interval between waypoints if there are any obstacles in the path. In the particular case of Terrain Following and Avoidance, since the waypoints are only defined in the horizontal plane, the initial time guess might be too small if the UAV has to climb or descent steep terrain, as demonstrated in fig. 4.3.

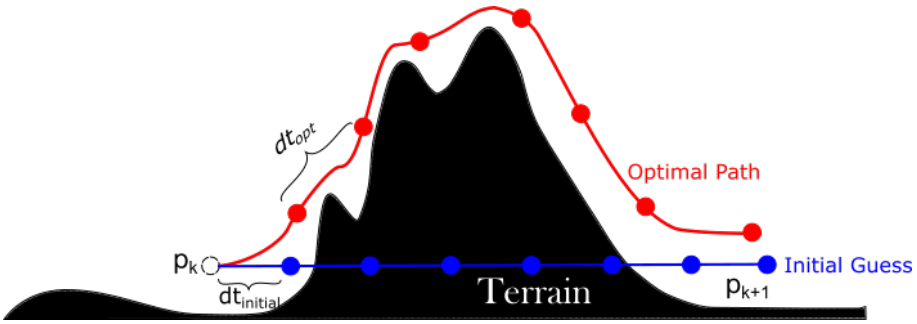


Figure 4.3: Illustration of a scenario where the initial guess of the optimization, represented in blue, is outside the feasible region and after the optimization, the optimal path, represented in red, requires a bigger discretization step to have the nodes outside the terrain obstacle.

4.5 Terrain Modelling

The terrain used in the optimisation problem is modelled using a matrix of elevation data provided by a DEM (Digital Elevation Map) file. This data was collected in previous manual commanded missions using a Lidar sensor. The point cloud obtained is then processed and saved as a set of x and y coordinates, and a matrix of z coordinates representing the elevation. In order for the solution algorithm to be effective, smooth derivatives of the terrain data are required [65]. In CasADi this can be achieved with the use of the 2D lookup-tables feature that provides C^2 continuity by approximating the data matrix with a tensor product cubic B-spline (Fig. 4.4) of the form

$$h_{terrain}(x, y) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} c_{i,j} B_i(x) B_j(y) \,, \tag{4.15}$$

where $c_{i,j}$ are a set of coefficients and $B_i(x)$ and $B_j(y)$ form the basis for cubic B-splines. In addition, CasADi calculates gradients of the lookup-tables by way of analytic derivatives, reducing the numerical errors of the approximation.

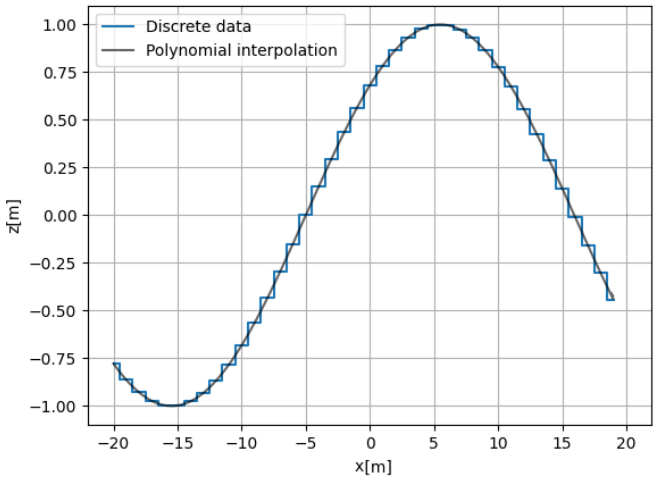


Figure 4.4: Example of discrete data interpolation in 1D. The discrete data is represented in blue by discontinuous steps that are approximated by a polynomial that allays passes the middle point of the original discrete step data.

4.6 Solver Implementation

The Trajectory optimization problem described in this chapter is implemented and tested using the CasADi toolbox [2] that is flexible and can be used to define almost any optimization problem without the restrictions of the black box architecture of most tools designed to solve very specific problems like Altro [67], OpEn [68] or GuSTO [46].

CasADi is an open-source software tool for generic numerical optimization problems. It is available for C++, Python and MATLAB/Octave with almost no difference in performance.

As stated in [63], CasADi can generate derivative information efficiently using algorithmic differentiation, to set up, solve and perform forward and adjoint sensitivity analysis for systems of ordinary differential equations (ODE) as well as to formulate and solve non-linear program problems. CasADi is a general-purpose tool for gradient-based numerical optimization that tries to provide the user with a set of ”building blocks” that can be used to implement general-purpose or specific-purpose OCP solvers efficiently with a modest programming effort.

There are several NLP solvers interfaced with CasADi. The most popular one is IPOPT. Others, that require the installation of third-party software, include SNOPT[69] and KNITRO [70]. Whatever the NLP solver used, the CasADi generates automatically the information that it needs to solve the NLP. Typically an NLP solver will need a function that gives the Jacobian of the constraint function and a Hessian of the Lagrangian function with respect to optimization variables [71].

4.6.1 Problem Discretization

The trajectory of a flying robot can be accurately modelled as a collection of some discrete points in a continuous coordinate system from the initial time to the final time. More precisely, if the number

[2] https://web.casadi.org

of these identified points is sufficiently high, a suitable continuous trajectory can be recovered as a passable approximation of the main path [16].

As mentioned in section 2.2, there are several discretization methods that could be applied in this trajectory optimization problem. I decided to use collocation methods since they have several numerical advantages over other techniques, such as shooting methods. Furthermore, although they result in larger optimization problems, these are sparse and can be solved very fast [47, 72].

I incorporate the system dynamics as equality constraints between time steps X_i and X_{i+1} using a first-order backward Euler approximation. Such an equality constraint represents the minimization of the defect variable ζ_i defined as:

$$\zeta_i = X_{i+1} - X_i - f(X_{i+1}, U_{i+1})\Delta t_k \,. \tag{4.16}$$

where $f(X_{i+1}, U_{i+1})$ is the derivative of the robot states in the collocation point $i+1$ derived from the solution of equations (4.1).

This first-order approximation is the simplest possible integration scheme and therefore leads to efficient computation. Furthermore, it results in a well-defined approximation of the non-linear problem which can be provided to CasADi that easily computes its derivatives (Jacobian and Hessian) using numerical approximations such as the finite differences method.

4.6.2 Optimization Function

As previously described, the trajectory optimization problem implemented has more than one objective that must be optimized and fulfilled. However, the optimization function is defined as a scalar at each collocation point which means that the objectives are not minimized individually and a compromise between them needs to be found to obtain the best solution.

In order to satisfy all the necessary objectives, the final cost function implemented in the problem is defined as the sum of independent objective functions. The compromise in the solution is determined by the relative weighting applied to each cost which can substantially change the optimum trajectory.

However, defining the cost function as a sum of all the objectives of this project can lead to unwanted results. For example, if the cost defined in (4.6) is added together with all the other costs it is possible that the optimal solution does not go through the waypoints if the relative weights are not defined correctly. Therefore, the implemented cost function only includes some of the objective functions while the other objectives will be fulfilled by a set of path constraints.

The final implemented optimization function is a weighted sum of the minimum time cost defined in (4.13), the terrain following cost (4.9), a minimum acceleration cost function defined in (4.17) and a yaw cost (4.18) that aims to keep ψ equal to the angle between the previous and next waypoints.

The acceleration cost is as follows:

$$\mathcal{F}_a = \int_{t0}^{t_f} a^T k_a a dt \tag{4.17}$$

where a is the acceleration vector of the UAV defined in the added integrator by equation (4.2) and k_a is the weight coefficients for the acceleration.

The yaw cost is defined as:

$$\mathcal{F}_\psi = \int_{t0}^{t_f} k_\psi [\psi - \psi_k^{des}]^2 dt \tag{4.18}$$

where k_ψ is the weight of this cost function and ψ_k^{des} is the desired yaw for the waypoint segment $k = 1, \ldots, K$ defined as:

$$\psi_k^{des} = atan2(\frac{y_k^w - y_{k-1}^w}{x_k^w - x_{k-1}^w}) \tag{4.19}$$

where x_k^w and y_k^w are the x and y coordinates of the k-th waypoint.

4.6.3 Constraints

Equality Constraints

The constraints of an optimization problem can be defined as path and boundary constraints.The path constraints are evaluated during the optimization whilst the boundary constraints are only applied to the initial and final states. In this case, all the equality constraints are defined in the boundaries of the optimization and are the following:

$$\begin{aligned} &X(t_0) = X_0 && \text{bound on initial state} \\ &\big[x(t_f),\ y(t_f)\big] = (x_f,\ y_f) && \text{bound on final waypoint} \\ &\big[\eta(t_f),\ \mathsf{v}_\mathfrak{I}(t_f),\ \mathsf{w}_\mathfrak{I}(t_f)\big] = (0^{3\times1},\ 0^{3\times1},\ 0^{3\times1}) && \text{bound on final state} \end{aligned} \tag{4.20}$$

Inequality Constraints

In contrast to the equally constrained case, now the number of inequality constraints may be greater than the number of variables to optimize. For this problem, the set of path inequality constraints are the waypoint tracking constraint defined in (4.7) and the limits for the minimum and maximum height allowed above the terrain presented in equation (4.10). Additionally, a constraint for maximum horizontal velocity is added to ensure the sensor data collected by the UAV can be correctly used.

$$\mathsf{v}_x^2 + \mathsf{v}_y^2 \leq (V_{xy}^+)^2 \qquad \text{Maximum horizontal velocity constraints} \tag{4.21}$$

Other than the path constraints some state and input inequality constraints were applied to prevent over actuation leading to undesired high pitch and roll.

$$
\begin{aligned}
|\mathbf{v}_z| &\leq V_z^+ && \text{Maximum vertical velocity constraints} \\
|\dot{\phi}| &\leq \dot{\phi}^+ && \text{Maximum roll rate constraints} \\
|\dot{\theta}| &\leq \dot{\theta}^+ && \text{Maximum pitch rate constraints} \\
|\dot{\psi}| &\leq \dot{\psi}^+ && \text{Maximum yaw rate constraints}
\end{aligned} \tag{4.22}
$$

$$
\begin{aligned}
&\tilde{T}^- \leq \tilde{T} \leq \tilde{T}^+ \\
&|\phi_{ref}| \leq \phi^+ && \text{Actuation constraints} \\
&|\theta_{ref}| \leq \theta^+ \\
&|a| \leq a^+
\end{aligned} \tag{4.23}
$$

4.6.4 Experimental Results with Validation Data

Several different tests were made, varying the terrain profile, waypoint arrangement and even discretization steps. In the end, I chose to present the results of three scenarios to assess the capabilities and difficulties of the selected approach. The tests were conducted on a Hp Laptop running Ubuntu 18.04 and equipped with an Intel Core i7-9750H CPU @2.60GHz and 16,00GB of RAM.

The model constants and coefficients common to all the studied scenarios are presented in the following tables. On the left table, I show the second-order attitude dynamic parameters from equation (4.1d) which are constants obtained from the system identification procedure described in the next chapter applied to the simulated Hexacopter model. The right table contains the state and path limits of the problem's inequality constraints defined in the previous section.

Table 4.2: Trajectory optimization parameters and coefficients.

Attitude Constants	
K_ϕ	0.9757
K_θ	0.9862
K_ψ	0.9762
ω_ϕ	6.2179
ω_θ	6.0429
ω_ψ	3.8762
ξ_ϕ	0.9353
ξ_θ	0.9216
ξ_ψ	0.8653

Optimization Limits		
V_{xy}^+	1.0	$[m/s]$
V_z^+	1.0	$[m/s]$
$\dot{\phi}^+$	180	$[^\circ/s]$
$\dot{\theta}^+$	180	$[^\circ/s]$
$\dot{\psi}^+$	25.0	$[^\circ/s]$
$\tilde{T}^-$	7.0	$[m/s^2]$
$\tilde{T}^+$	15.0	$[m/s^2]$
ϕ^+	25.0	$[^\circ]$
θ^+	25.0	$[^\circ]$
a^+	1.0	$[m/s^2]$

The first scenario is a simple example that uses terrain elevation data obtained from United States Geologic Survey National Elevation Dataset [3]. In this scenario I included three waypoints bounded to a specific node and with a tolerance $\sigma_w = 0.5[m]$ as described in section 4.2. The number of discretization steps was chosen to be $N = 80$ based on the horizontal distance and maximum velocity between waypoints. For the height constraints I chose to set the minimum and maximum altitude allowed above

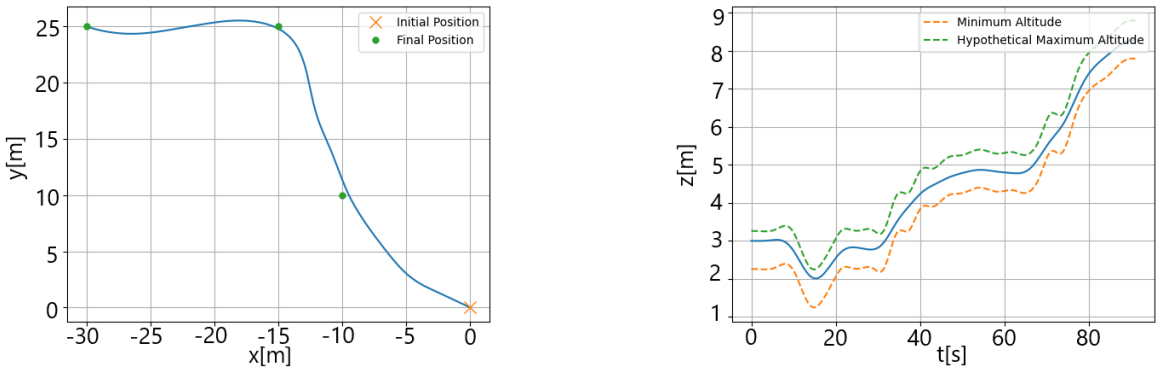
[3] https://viewer.nationalmap.gov

the terrain to $h^- = 2.5m$ and $h^+ = 3.5m$ respectively.

This scenario was tested with three different time formulations to show the advantages and disadvantages of the time-optimal method. Initially, I tested the scenario with large fixed time intervals between waypoints ΔT_k. The fixed discretization time steps for this case are presented in table 4.3. The optimization results of this scenario are presented in figures 4.5.

Table 4.3: Time steps of the optimum trajectory on Scenario 1.

Δt_1	Δt_2	Δt_3
0.363 $[s]$	0.400 $[s]$	0.375 $[s]$



(a) Top-down view of the trajectory.

(b) Trajectory in Altitude.

Figure 4.5: Graphical representation of the trajectory in the scenario with smooth terrain data and large fixed time steps.

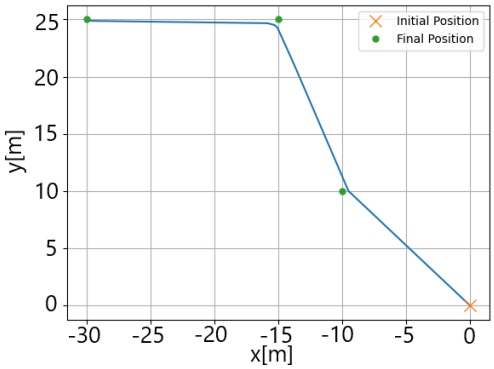
For the second test of this initial scenario, the fixed time step was reduced to small values that resulted in an infeasible problem. For this optimization, IPOPT converged to a point of local infeasibility. Some of the results of this test are represented in figure 4.6 where it is shown that the maximum horizontal velocity constraint and the maximum and minimum altitude constraints were not fulfilled.

In the third test, the time-optimal formulation introduced in section 4.4 is employed where the time intervals between waypoints are adjusted by the optimizer. The final discretization time steps for each waypoint segment are presented in table 4.4. The optimized trajectory is shown in figure 4.7 where it is possible to confirm that the waypoint tracking and terrain-following objectives of the trajectory were fulfilled with the altitude of the UAV remaining inside the allowable interval. The remaining system states and inputs can be seen in Figure 4.8 that show the that the maximum horizontal and vertical restrictions are fulfilled. The yaw results is also optimized according to the cost function that aims to keep the UAV pointing towards the next waypoint, while the maximum angular rate for yaw is respected. Finally, the acceleration constraints are also fulfilled contributing for a relatively smooth trajectory.

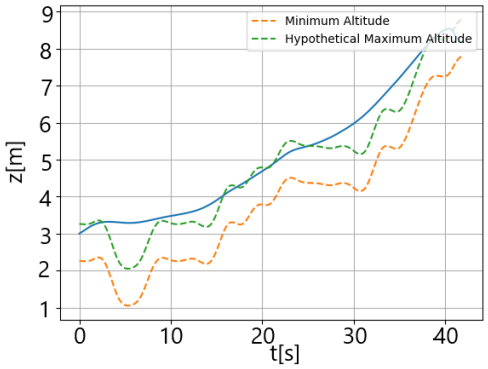
Table 4.4: Time steps of the optimum trajectory on Scenario 1.

Δt_1	Δt_2	Δt_3
0.178 $[s]$	0.198 $[s]$	0.197 $[s]$

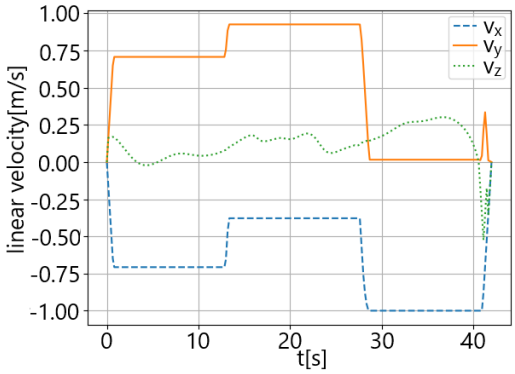
This test was executed a total of 50 times in order to obtain a good estimate of the computation time



(a) Top-down view of the trajectory.



(b) Trajectory in Altitude.



(c) Optimized Velocity.

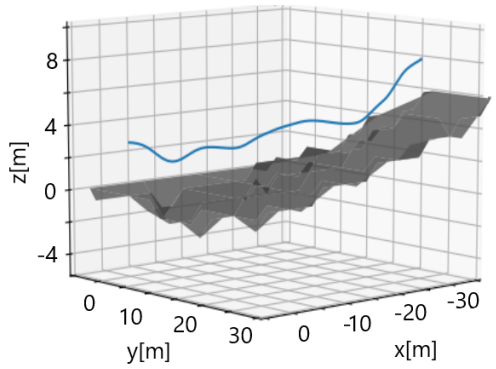
Figure 4.6: Graphical representation of the trajectory in the scenario with smooth terrain data and small fixed time steps.

necessary to solve the problem with IPOPT. The mean computation time was then determined to be 0.842 [s] for this scenario.

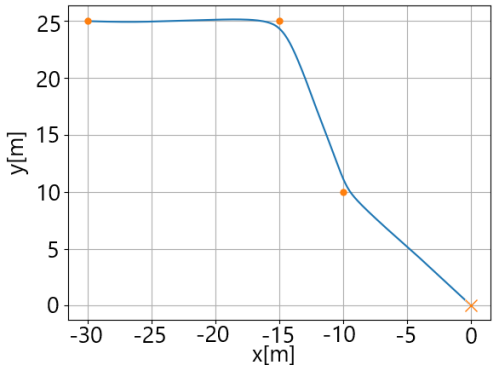
The three tests performed in the first scenario allowed me to conclude that the time-optimal formulation shows better results when compared to using arbitrarily large fixed time steps since the resulting trajectory for the time-optimal test is not only shorter in terms of time but also in terms of traveled space. The second test illustrates the infeasible case that occurs when the time step used is smaller than the minimum time imposed by the input and path constraints of the mission. The final test used the time-optimal formulation which solves the problems of the previous tests by determining the time interval as part of the optimization process.

However, one of the drawbacks of this time optimal formulation is the possibility that it becomes very large if it is required by the problem scenario. Those large Δt increase substantially the discretization error, mainly in the simple Euler method employed and can result in trajectories that do not satisfy the problem constraints for large segments of the path since the constraints are only imposed in the discretization nodes.

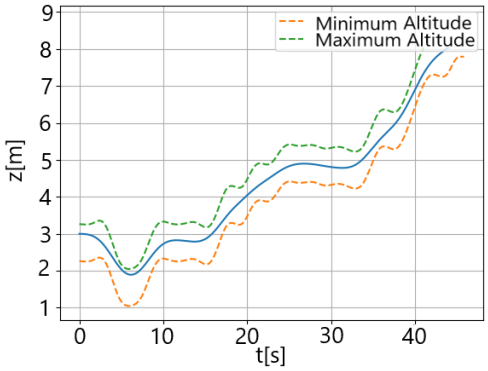
In the second scenario, an artificial terrain profile was created with trigonometric functions in order to obtain steep ground inclinations with relative smoothness. This scenario is similar to the example illustrated in figure 4.3 where the optimization's initial guess is outside the feasible region. For this test,



(a) Trajectory in 3D.



(b) Top-down view of the trajectory.



(c) Trajectory in Altitude.

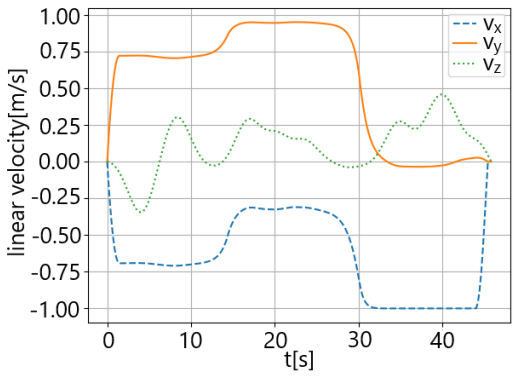
Figure 4.7: Graphical representation of the trajectory in the scenario with smooth terrain data and with variable time intervals determined by the optimizer.

I used a higher number of discretization steps, $N = 200$, since the distance between the two waypoints is considerably higher than in scenario 1 and it is important to have a small discretization step to reduce model errors and make sure the constraints are always fulfilled. Similarly to the first scenario, in this case I also set the minimum and maximum altitude allowed to $h^- = 2.5m$ and $h^+ = 3.5m$ respectively.

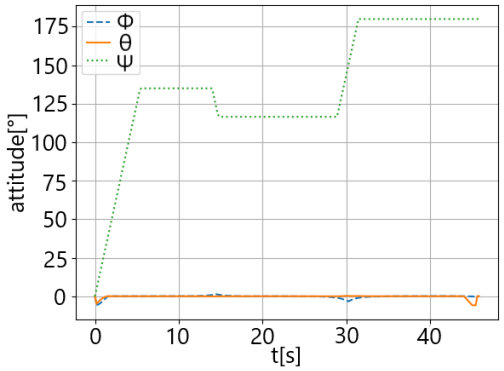
The optimized trajectory, displayed in figure 4.9, confirms that the safety altitude constraints are fulfilled for the entire path despite the high terrain inclinations that result in a small feasible space for the z coordinate. The final trajectory time is $t_f = 37.966[s]$ which corresponds to a discretization step of $\Delta t = 0.190[s]$. In this scenario, the minimum trajectory time is also limited by the vertical velocity which, as seen in figure 4.10, results in two moments where the horizontal speed is reduced to make sure the UAV stays inside the feasible region. The attitude peaks present in the beginning and end of the trajectory are caused by the higher cost attributed to the total trajectory time when compared to the minimum acceleration cost. Even thought, this formulation causes the UAV to accelerate sharply in the beginning and end of the path the dynamic constraints are always fulfilled.

Once again the optimization process for this scenario was repeated 50 times and a mean computation time for the iterative process performed by IPOPT was determined to be 1.066 [s].

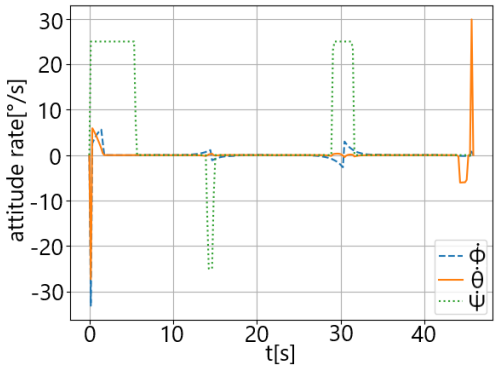
Finally, the third scenario is formulated to assess the results of the proposed approach when the terrain data contains a significant discontinuity that could represent a cliff or wall. In this case, I introduced



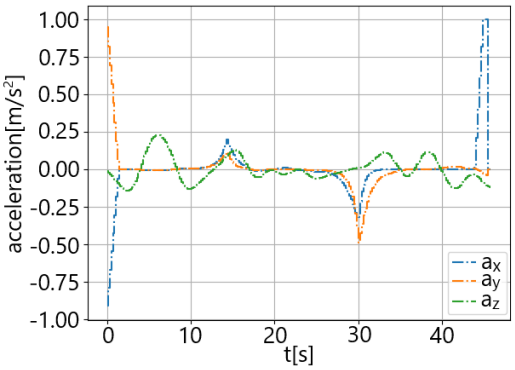
(a) Optimized Velocity.



(b) Optimized Attitude.



(c) Optimized Attitude Rate.



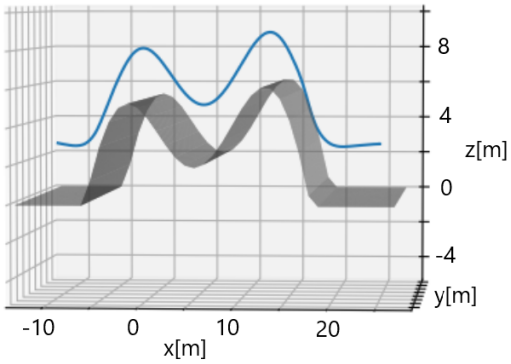
(d) Acceleration Commands from the Integrator.

Figure 4.8: Graphical representation of the optimized states and inputs in the scenario with smooth terrain data and with variable time intervals determined by the optimizer.

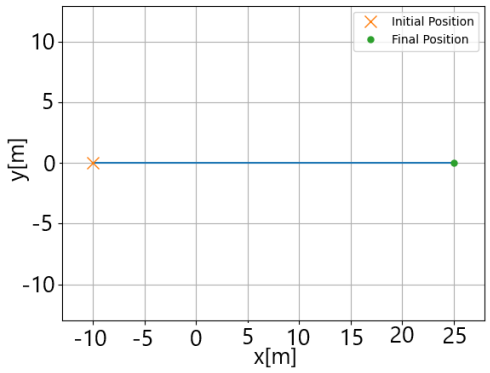
a 7 m wall in the altitude matrix data with 0.9 m of width. From the previous scenario, I can conclude that as the terrain becomes steeper and more discontinuous the feasible tunnel of the z coordinate becomes less wide and for a vertical wall it would result in an impossible problem. Therefore, for the following scenario I did not implement the maximum altitude constraint but instead relied on the terrain following cost (Eq. (4.6)) to maintain the UAV close to the desired altitude. If the weight k_{TF} is higher the resulting trajectory will climb the wall almost vertically to maintain the vertical distance to the ground as constant as possible, if this weight is smaller the minimum time objective takes over and the trajectory transposes the wall with an arc-like trajectory.

To take this into account this third scenario was tested with two different cost weights k_{TF}. In the results present in figures 4.11 and 4.12 I used $k_{TF} = 0.05$ and in figures 4.13 and 4.14 the weight was set to $k_{TF} = 0.5$. In both tests the number of discretization steps was $N = 200$ and the minimum altitude allowed was $h^- = 2.5m$.

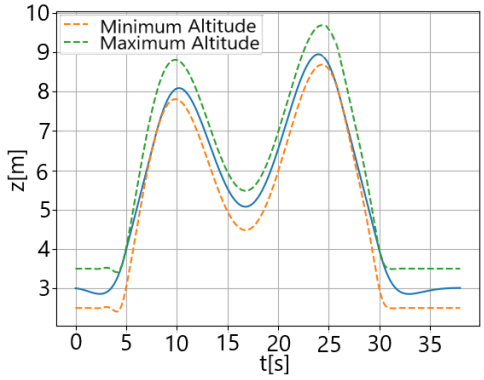
The optimized trajectory shown in figure 4.11, confirms that if the maximum altitude constraint was implemented the optimization problem would become infeasible due to the small space between the two bounds. The altitude plot also shows that the cubic spline used to represent a continuous terrain profile introduces considerable oscillations when approximating discontinuities, meaning it is not ideal to represent such terrain profiles.



(a) Trajectory in 3D.

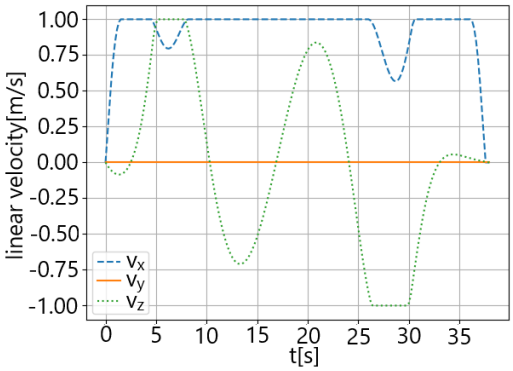


(b) Top-down view of the trajectory.

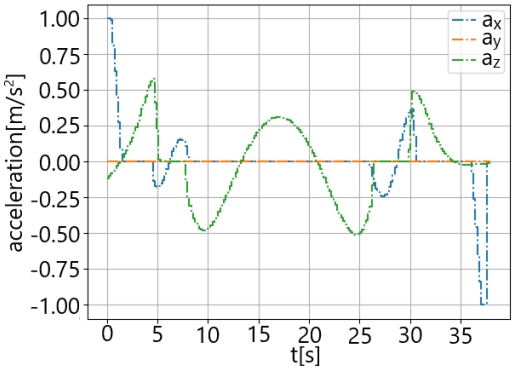


(c) Trajectory in Altitude.

Figure 4.9: Graphical representation of the trajectory in the scenario with high terrain gradients.



(a) Optimized Velocity.

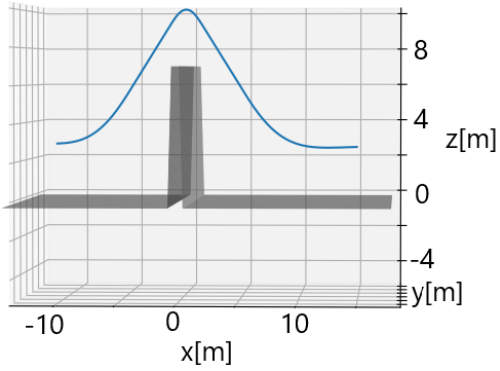


(b) Acceleration Commands from the Integrator.

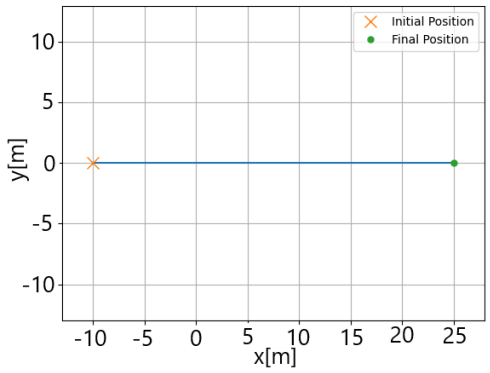
Figure 4.10: Graphical representation of the optimized states and inputs in the scenario with high terrain gradients.

The final trajectory time is $t_f = 37.966[s]$ which corresponds to a discretization step of $\Delta t = 0.190[s]$. The mean computation time for the iterative optimization process in this case is 5.719 [s].

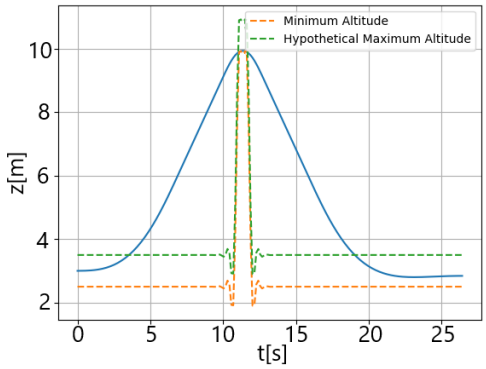
The trajectory generated in the second test, shown in figure 4.13, stays below the hypothetical maximum altitude for almost the entire path which results in a close to vertical climb when in the proximity of to the wall. Therefore, the optimized trajectory takes more time to complete when compared to the first



(a) Trajectory in 3D.

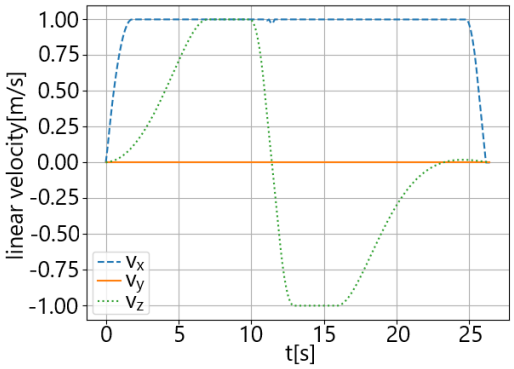


(b) Top-down view of the trajectory.

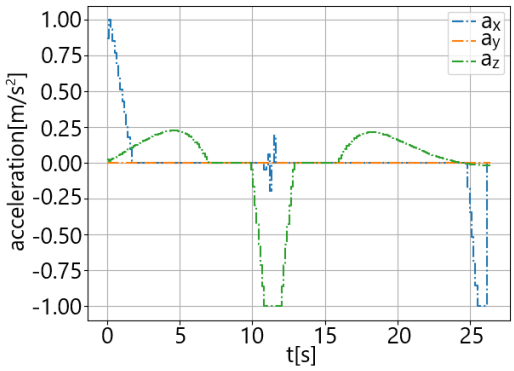


(c) Trajectory in Altitude.

Figure 4.11: Graphical representation of the trajectory in the scenario with a big terrain discontinuity and low terrain-following cost K_{TF}.



(a) Optimized Velocity.



(b) Acceleration Commands from the Integrator.

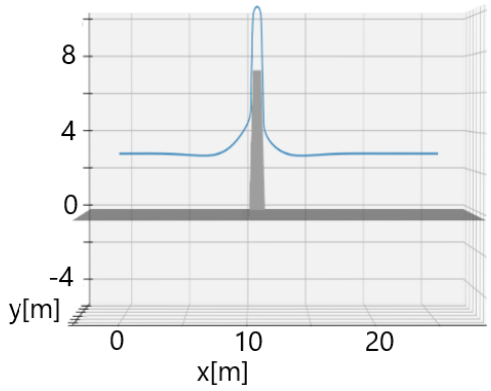
Figure 4.12: Graphical representation of the optimized states and inputs in the scenario with a big terrain discontinuity and low terrain-following cost K_{TF}.

test resulting in $t_f = 38.363[s]$ which corresponds to a discretization step of $\Delta t = 0.192[s]$.

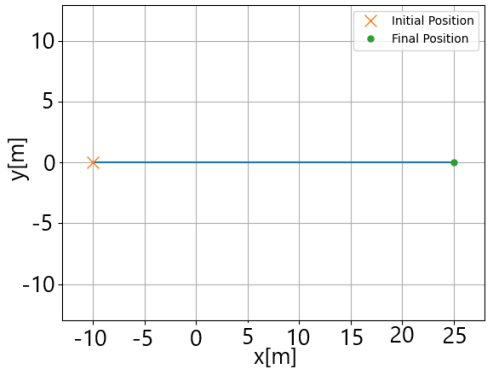
Finally, due to the high gradients of the altitude constraints and the closer proximity to the wall this test requires a higher computation time with a mean of 10.526 [s] over 50 optimizations.

Although the minimum altitude constraint is always fulfilled, the UAV is considered as a point meaning that in the second test a 3D UAV would collide with the wall in the horizontal plane. To solve this problem

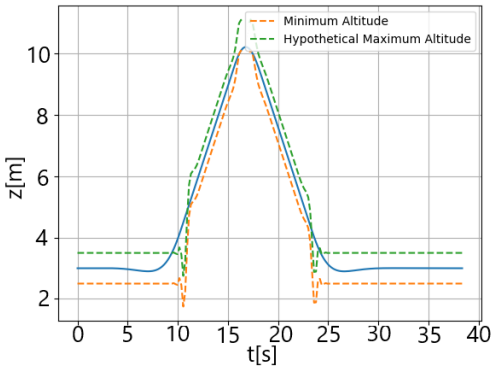
further improvement in the UAV representation or the minimum altitude function should be performed in future work.



(a) Trajectory in 3D.

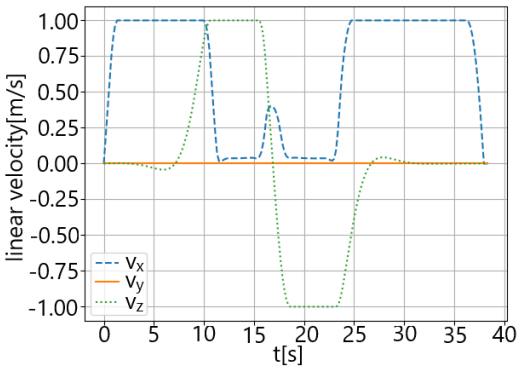


(b) Top-down view of the trajectory.

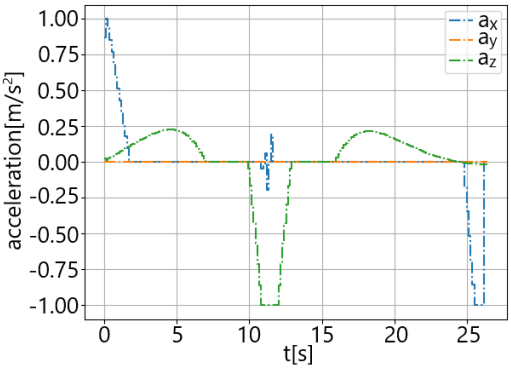


(c) Trajectory in Altitude.

Figure 4.13: Graphical representation of the trajectory in the scenario with a big terrain discontinuity and high terrain-following cost K_{TF}.



(a) Optimized Velocity.



(b) Acceleration Commands from the Integrator2.

Figure 4.14: Graphical representation of the optimized states and inputs in the scenario with a big terrain discontinuity and high terrain-following cost K_{TF}.

Chapter 5

Model Predictive Control Formulation

This chapter is dedicated to the design and implementation of a continuous-time Model Predictive Controller, which can follow in real-time the paths computed by the trajectory optimizer designed in the previous chapter. The method implemented was based on the work presented in [36, 54], adapted to the problem at hand and software used.

First, I define the dynamic equations that efficiently and accurately represent the FRIENDS' project UAV. The optimization function is then defined along with the constraints and bounds of the optimization problem. The predictive control is finally implemented as explained in Section 2.3, where it is obtained a sequence of inputs and state predictions by solving an optimization problem each sampling time for a limited time horizon. The first input given by the sequence can then be applied to the simulated system. After a certain time, the state of the UAV is re-sampled and the MPC is updated with a new initial condition repeating the process.

5.1 System Dynamics

In order for the control optimization algorithm to be able to run in real-time the computation time at each time step must be considerably smaller than the sampling time of the controller, since big delays between state estimation and the application of the control commands would invalidate the MPC dynamic model, which considers no computation delays in its implementation. This can be achieved by reducing the number of state variables and simplifying the system dynamics of the robot.

In this project, I followed a cascaded approach studied in Blösch et al. [73] and assume that the vehicle attitude is controlled by a low-level attitude Proportional controller that can track desired roll, ϕ^{des}, pitch, θ^{des}, and yaw, ψ^{des} angles [36]. To achieve accurate trajectory tracking, the high-level optimization controller must consider the inner loop system dynamics. Therefore, it is necessary to consider a simple model of the attitude closed-loop response. These dynamics can either be calculated by simplifying the closed-loop dynamic equations (if the full controller implementation is known) or by a simple system identification procedure in case of an unknown attitude controller used on commercial platforms such as

PX4 [1] or Ardupilot [2].

In the case of this thesis and as stated in [36, 54], the attitude dynamics can be accurately represented by a first-order inner-loop approximation provided that the states and inputs of the system are correctly bounded, in other words, this approximation would correctly represent the system if acrobatic flights are performed. Therefore, for the mission performed in this project, this approximation provides sufficient information to the MPC to take into account the low-level controller behaviour. The first-order approximation was confirmed to be accurate after several system identification tests where simulated UAV attitude data was compared to a first order system and the curve fit percentage was allays above 70%.

In order to simplify the full hexacopter dynamics presented in (3.9), I first decouple the rotational dynamics from the translational dynamics.

Translational dynamic

In the simplified model, since the torques produced by the rotors are not individually considered in the simplified rotational dynamics, I represent the UAV model solely in the Inertial frame and, instead of using each rotor's rate of rotation as inputs, I consider, as control variables, the reference commands for the UAV's roll, ϕ_{cmd}, and pitch, θ_{cmd}, on the inertial frame and the mass-normalized thrust $\tilde{T}$.

$$\tilde{T} = \frac{\sum_{i=1}^{6} F_i}{m}$$

where F_i is the thrust generated by the i-th motor and m is the mass of the Hexarotor.

After implementing this simplifications I can represent the new translational dynamics as follows:

$$\begin{cases} \dot{\mathbf{p}} = \mathsf{v}_{\mathfrak{I}} \\ \dot{\mathsf{v}}_{\mathfrak{I}} = \mathcal{R}^{\mathfrak{B}\mathfrak{I}}(\eta) \begin{bmatrix} 0 \\ 0 \\ \tilde{T} \end{bmatrix} + F_a + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + F_{ext} \end{cases} \tag{5.1}$$

where $F_{ext} = [F_{xext},\ F_{yext},\ F_{zext}]$ is the vector of external disturbances determined outside the optimization loop, $F_a = \tilde{T}\mathcal{R}^{\mathfrak{B}\mathfrak{I}}(\eta)A_d\mathcal{R}^{\mathfrak{I}\mathfrak{B}}(\eta)^T\mathsf{v}_{\mathfrak{I}}$ is the term representing the aerodynamic effects from equation (3.6) and $\mathcal{R}^{\mathfrak{B}\mathfrak{I}}$ is the rotation matrix from the Body $\mathfrak{B}$ frame to the Inertial frame $\mathfrak{I}$.

Simplified Rotational dynamic

As already mentioned, with respect to the rotational dynamics, the simplified model attitude is described as a first order system that is considered independent for each of the Euler angles. This dynamics can be represented as follows:

[1] https://px4.io
[2] https://ardupilot.org

$$\dot{\eta} = \frac{1}{\tau}(K\eta_{cmd} - \eta) \tag{5.2}$$

where $\eta = [\phi,\ \theta,\ \psi]$ is the attitude vector of the UAV, $\eta_{cmd} = [\phi_{cmd},\ \theta_{cmd},\ \psi_{cmd}]$ is the vector containing attitude control references and $\tau = [\tau_\phi,\ \tau_\theta,\ \tau_\psi]$, $K = [K_\phi,\ K_\theta,\ K_\psi]$ are the vectors of time constants and gains of the inner-loop first-order approximation behaviour respectively. The yaw reference for the internal attitude controller is computed outside the MPC since yaw does not influence the translational dynamics of the UAV. Therefore, inside the MPC the yaw dynamics is simply simulated by a first order system in order to reduce the non-linear effect of a varying yaw trajectory.

5.2 Cascaded Control Strategy

As previously mentioned, the designed control strategy follows a cascade approach where the control of the transitional and rotational dynamics is performed independently in different modules and with different control methods. The main focus of this work is the control of the transitional dynamics, which is performed using a Model Predictive Controller off-board the hexacopter's autopilot. The attitude and normalized thrust outputs of the translational controller are then used as set-points for the attitude PX4 controller running in-loop inside the Pixhawk.

The low-level attitude controller that controls the UAV model is part of the PX4 Firmware and is based on unit quaternions. The formulation, implementation and testing of this non-linear PID controller is described in paper [74]. The gains and weights of these controllers are adjusted until minimum overshoot and reduced oscillatory behaviour is obtained from the UAV. After tuning all the internal control loops of the PX4 autopilot, test flights are performed and the behaviour of the attitude loops is analysed. With the help of system identification techniques like curve fitting, the gains and time constants of the first-order approximation as well as the gains, natural frequency and damping factor of the second-order approximation for the Rotational dynamics are determined and the system dynamics applied in the MPC and the Trajectory optimizer are formulated.

For the sake of completeness, the architecture of internal attitude loops of the PX4 Firmware is shown in figures 5.1 and 5.2.

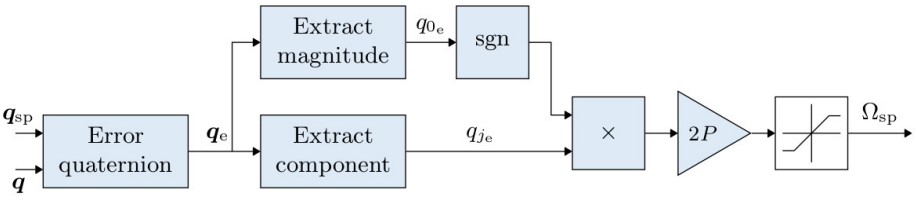


Figure 5.1: PX4 Multicopter Attitude Controller taken from [60].

Since the angular rate controller loop runs at a frequency of 500Hz the time constant of the system response is much lower than the control rate of the MPC. Therefore, it is correct to consider the tracking

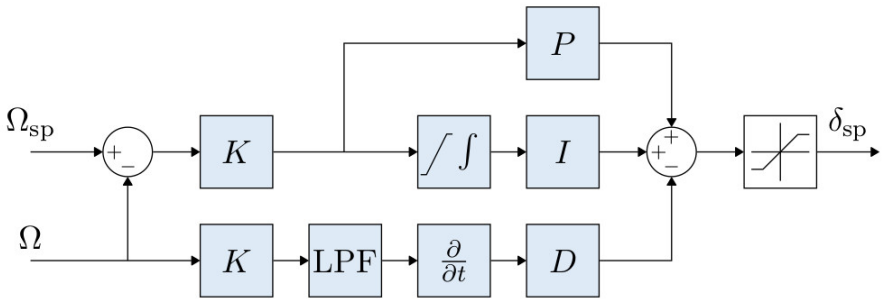


Figure 5.2: PX4 Multicopter Angular Rate Controller taken from [60].

of attitude rate set-points almost instantaneous which further validates the first-order approximation for the attitude controller.

Regarding the translational control, the MPC is formulated with the continuous dynamics presented in (5.1) and (5.2), which can be represented as a non-linear state-space model of form $\dot{X} = f(X, U)$.

The state-space vector of the optimization control problem is:

$$X = \begin{bmatrix} \mathbf{p^T} & \mathsf{v}_\mathfrak{I}^T & \eta^T \end{bmatrix}^T \tag{5.3}$$

And the control input vector, U, consists of tree inputs,

$$U = \begin{bmatrix} \tilde{T} & \phi_{cmd} & \theta_{cmd} \end{bmatrix}^T \tag{5.4}$$

To solve this optimization problem, it is necessary to discretize the system dynamics with a sampling time Δt and to consider a time horizon T_h, so that at each iteration, I obtain a sequence of N inputs U and sequence of $N + 1$ state predictions X, with $N = \frac{T_h}{\Delta t}$.

5.3 Attitude System Identification

As mentioned, when the knowledge about the attitude controller used onboard of the vehicle is limited or not easily modelled mathematically, a loop identification process is recommended to find the parameters of the previously described approximation.

To perform this system identification, typically a test flight is performed where the vehicle's axes are excited in free flight. During this flight, attitude references of the onboard control loop along with the estimated vehicle attitude are logged with an accurate time stamp.

As mentioned in [54], typically two datasets are collected, one is used for parameters estimation and the other is used for validation purposes. In this project the parameter identification is performed in the MATLAB environment using the System Identification Toolbox [75]. A simulated flight is performed in Gazebo where the drone is subject to aggressive step inputs in the position control loop of the PX4 autopilot. The flight log is fully saved by the PX4 Firmware as a .ulog file. This log file includes not only the vehicle attitude estimated onboard by the EKF, which fuses data from the IMU and magnetometer to

estimate orientation but also the set-points of the attitude control loop, all correctly and consistently time stamped. The .ulog file is then uploaded to MATLAB and the data is properly scaled and interpolated before the system identification step is executed. The controller parameters will be finally calculated along with a curve fitting percentage to confirm the validity of the identification procedure. The estimated response should closely match the actual and, as a rule of thumb, if the given fit percentage isn't at least 70% something has gone wrong or the system cannot be represented by the desired model.

5.4 External Disturbance Estimation

In this section, I discuss the external disturbance estimator employed to achieve offset-free trajectory tracking. In the dynamic model (5.1) used in the Optimization Controller, I added independent external forces that need to be computed outside the MPC optimization problem. The external disturbances F_{ext} are estimated by an Extended Kalman Filter (EKF) that employs the same translational model used in the MPC (5.1), but with a second-order attitude dynamic approximation that was also used in the trajectory optimization design, (4.1d), augmented with a vector of external Torques M_{ext} also estimated by the EKF.

A Kalman filter is an iterative mathematical process that uses a set of dynamic equations and consecutive data inputs to efficiently estimate the true value of all the states of the system being measured. An extended Kalman filter extends this estimation ability to non-linear systems such as the one used. Consider the non-linear dynamics:

$$X_{k+1} = f(X_k, U_k) + w_k \tag{5.5}$$

$$Y_k = h(X_k, U_k) + v_k \tag{5.6}$$

where w_k and v_k are the process or system and measurement noise respectively. They are white Gaussian with zero mean and covariance matrix defined as

$$E[v_k v_k^T] = R_k\,, \qquad\qquad E[w_k w_k^T] = Q_k\,. \tag{5.7}$$

Now, let $\mathbb{Y} = \{Y_1,\ Y_2,\ \ldots,\ Y_k\}$ be a set of system observations. The filter's goal is to obtain an estimate of the system's state based on these measurements.

To commence the Kalman filtering process, an initial state vector, X_0, and a covariance matrix that represents the error in the system or state estimates, P_0, are declared. The vector X_0 and matrix P_0 become the past state estimate, $\hat{X}_{(k-1|k-1)}$ and past predicted error covariance matrix, $P_{(k-1|k-1)}$ respectively [76].

The first step of each iteration is to linearise the system dynamics $f(X_k, X_k)$ around the previous estimate $\hat{X}_{(k-1|k-1)}$. The partial derivates of functions f and h, with respect to the state vector, X_k, and

noise vectors v_k and w_k, are derived by computing the Jacobian denoted by:

$$F_{k-1} = \frac{\partial f}{\partial X}|_{\hat{X}_{(k-1|k-1)}}, \tag{5.8a}$$

$$H_k = \frac{\partial h}{\partial X}|_{\hat{X}_{(k|k-1)}}. \tag{5.8b}$$

Then I predict the new state vector new error covariance matrix by applying the prediction step as follows:

$$\hat{x}_{(k|k-1)} = f(\hat{X}_{(k-1|k-1)}, U_k) \tag{5.9}$$

$$P_{(k|k-1)} = F_{k-1} P_{(k-1|k-1)} F_{k-1}^T + Q_k \tag{5.10}$$

Then I linearise the observation dynamics around $\hat{X}_{(k|k-1)}$ as shown in (5.8b).

Next the Kalman gain, K_k, is computed. The Kalman gain weighs the error between predictions and the measurements obtained from sensor readings. The Kalman gain is computed using the equation below,

$$K_k = (P_{(k|k-1)} H_k) \left(H_k P_{(k|k-1)} H_k^T + R_k\right)^{-1} \tag{5.11}$$

The next step is to compute the estimated state vector and error covariance matrix. These variables are computed using the following equations:

$$\hat{X}_{(k|k)} = \hat{X}_{(k|k-1)} + K_k[Y_k - H_k \hat{X}_{(k|k-1)}] \tag{5.12}$$

$$P_{(k|k)} = [I - K_k H_k] P_{(k|k-1)} \tag{5.13}$$

where Y_k is the measurement obtained from the sensor(s), or in the specific case the states estimated by the primary EKF of the PX4 autopilot.

The obtained estimates of the state vector and error covariance matrix become the past predicted state vector, $\hat{X}_{(k-1|k-1)}$ and past predicted error covariance matrix, $P_{(k-1|k-1)}$ for the next iteration of the Kalman filter [76]. This estimation procedure is carried out for each time step.

In my case, the implemented External Kalman Filter used to estimate external Forces and Torques is defined as follows:

$$X_k = \begin{bmatrix} \mathbf{p}^T & \mathbf{v}_\mathfrak{J}^T & \eta^T & \mathbf{w}_\mathfrak{J}^T & F_{ext}^T & M_{ext}^T \end{bmatrix}^T \tag{5.14}$$

$$U_k = \begin{bmatrix} \tilde{T} & \phi_{cmd} & \theta_{cmd} & \psi_{cmd} \end{bmatrix}^T \tag{5.15}$$

$$Y_k = \begin{bmatrix} \mathbf{p}^T & \mathbf{v}_\mathfrak{J}^T & \eta^T \end{bmatrix}^T \tag{5.16}$$

And the non-linear dynamics and measurement model are expressed as:

$$f(X_k, U_k) = \begin{bmatrix} \mathsf{v}_{\mathfrak{I}} \\ \mathcal{R}^{\mathfrak{B}\mathfrak{I}}(\eta) \begin{bmatrix} 0 \\ 0 \\ \tilde{T} \end{bmatrix} + F_a + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + F_{ext} \\ \mathsf{w}_{\mathfrak{I}}^T \\ -2\xi\omega\mathsf{w}_{\mathfrak{I}}^T + \omega^2(K\eta_{cmd} - \eta) + M_{ext} \\ 0 \\ 0 \end{bmatrix} \tag{5.17}$$

$$h(X_k, U_k) = \begin{bmatrix} \mathbf{p} \\ \mathsf{v}_{\mathfrak{I}} \\ \eta \end{bmatrix} \tag{5.18}$$

This estimator will reduce, not only, the effect of unpredictable external disturbances, like wind, but also capture modelling error that the model may have in the attitude and drag approximations applied, achieving zero steady-state tracking error Borrelli et al. [77], as well as reducing the MPC calculation time by calculating externally forces like ground effect, that if considered in the dynamic model, would make the state space significantly more complex.

5.5 Solver Implementation

The ACADO toolbox is an open-source and user-friendly algorithm collection that implements tools for automatic control and dynamic optimization(fig. 5.3), such as model predictive control and state and parameter estimation. It is implemented as a self-contained C/C++ library with an object-oriented design that allows for easy coupling and extending of existing optimization packages and user-written algorithms [78].

The optimization problem was implemented in a C++ interface for ACADO that can be easily compiled and executed, solving the optimization problem and allowing for a quick preliminary validation test. To formulate the problem in ACADO, a set of differential states, control inputs and online data variables must be provided. The states, X, and control inputs, U, are the ones defined in (5.3) and (5.4) respectively and represent the optimization variables. The online data are external parameters that can vary for each iteration and discretization step. In this case, the online data are the attitude first-order approximation constants (τ and K), the coefficients of the linear drag approximation (K_D), the yaw reference of the attitude loop (ϕ_{cmd}) and the estimated external Forces (F_{ext}). The full control loop is illustrated in fig. 5.4.

After the definition of the model variables, the differential equations of the dynamic model represented in (5.1) and (5.2) are added. Finally, the MPC parameters such as time horizon, sampling time, the objective function and constraints are set.

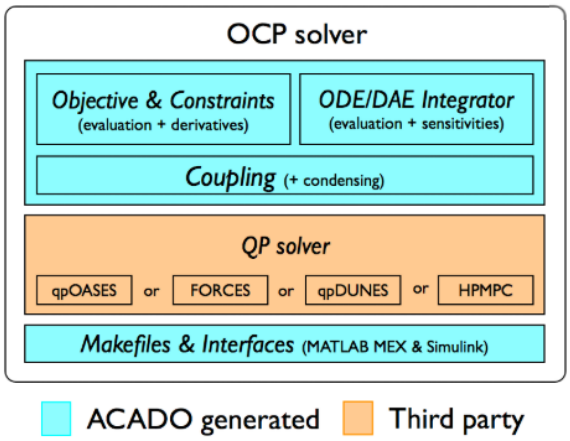


Figure 5.3: ACADO Toolkit Architecture from [79].

The ACADO Toolkit also includes the option to export optimized, highly efficient C-code to solve the non-linear model predictive control problem. The exported code is self-contained and can easily be integrated with external autopilot software. Before generating code there are several options related to discretization and integration algorithms that can be set based on the specifications of the problem to reduce the approximation errors and improve the computation time.

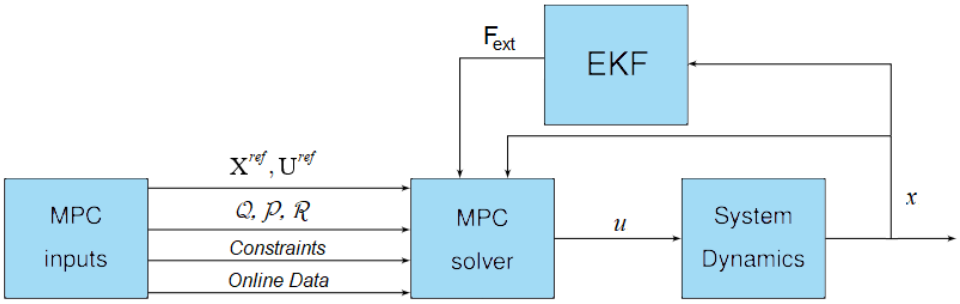


Figure 5.4: Schematic representation of the MPC's software Architecture.

In this implementation, a real-time iteration scheme based on Gauss-Newton is applied to approximate the non-linear optimization problem and iteratively improve the solution during the runtime. A Multiple shooting technique is employed to discretize the dynamics of the optimization problem. In particular, an implicit Range-Kutta integration scheme with a Gauss-Legendre integrator of order 4 is employed to forward simulate the system dynamics and constraints over a coarse discrete-time grid $t_0, \ldots, t_N$ within the time interval $[t_i, t_{i+1}]$. For each interval, a Boundary Value Problem (BVP) is solved, where additional continuity constraints are imposed. At this point, the OCP can be expressed as a Non-linear Program (NLP) that can be solved using Sequential Quadratic Programming (SQP) such as qpOASES solver.

Besides C code, ACADO also generates executable mexa64 files that can be used to test the algorithm in a Simulink diagram. To do so, one must define for each iteration the state, input references, the cost matrices Q, P and R, the bounds for the state and input constraints and online data for the next

N time steps. Then, I can solve the current NMPC problem by including the generated ACADO block in Simulink. Afterward, the first input U_0 can be applied to the simulated system dynamics of the Hexacopter, which can be implemented using the Aerospace Toolbox[3], from which I obtain a newly updated state that will be used in the next MPC iteration.

After validating the NMPC in Matlab the generated code can be integrated into a more complex simulation environment such as Gazebo before it is ultimately used in a real-world model.

5.5.1 Optimization Function

The optimization problem solved by the MPC is based on the previously introduced formulation, (2.28), where the goal is to minimize a Quadratic cost function composed of a sum of state and input path costs for the N steps of the optimization horizon and a final boundary cost for the step $N + 1$ that, as mentioned in section 2.3, is fundamental to the stability of the controller.

The weight matrices for states and inputs, Q and P, are tuned based on the analysis of the response of the Hexarotor in multiple simulations performed with PX4 SITL software integrated on Gazebo. These matrices can be time-varying by exponentially decaying throughout the prediction horizon or be constant inside each optimization step.

In the precise trajectory tracking problem, the state and input references, X_{t+i}^{ref} and U_{t+i}^{ref}, are calculated by the trajectory optimizer and represent the intended dynamics for the UAV, serving as a baseline for the trajectories to be generated by the MPC. However, this controller can also be used to optimize the path to reach a certain reference position or to hover above the current location. This is achieved by setting the reference X_{t+i}^{ref} constant for the all optimization horizon and equal to the desired position.

5.5.2 Constraints

The first set of constraints that must be respected by the optimization control problem are the system dynamics represented by equations (5.1) and (5.2). In order to be applied to the non-linear problem, these equations are discretized using one of the methods described in section 2.2 that convert the continuous UAV dynamics into a set of discrete constraints for each discretization step.

For this simple control problem, the only extra constraints considered are bounds imposed on the control inputs and dynamic states, in order to avoid saturation of the actuators and prevent the system from showing unwanted behaviour.

$$\tilde{T}^{-} \leq \tilde{T}_i \leq \tilde{T}^{+}\,, \tag{5.19a}$$

$$-\phi^{+} \leq \phi_{cmd_i} \leq \phi^{+}\,, \tag{5.19b}$$

$$-\theta^{+} \leq \theta_{cmd_i} \leq \theta^{+}\,. \tag{5.19c}$$

[3]https://www.mathworks.com/products/aerospace-toolbox.html

The limits imposed in some states to guarantee that the solution is feasible can be written as:

$$-V_{xy}^{+} \leq v_{x_i} \leq V_{xy}^{+}\,, \tag{5.20a}$$

$$-V_{xy}^{+} \leq v_{y_i} \leq V_{xy}^{+}\,, \tag{5.20b}$$

$$-V_{z}^{+} \leq v_{z_i} \leq V_{z}^{+}\,, \tag{5.20c}$$

In the next chapter, we discuss the integration of this MPC in a physics simulation environment (Gazebo) and show the results of several performance test that prove the robustness and stability of this controller. The simulation environment allows for the testing and verification of this algorithm in a simulated UAV model that closely resembles the real Hexacopter of the project FRIENDS.

Chapter 6

Simulation

In this chapter, I discuss a very important stage of the validation and verification of the previously described optimization problems, in order to guarantee that both the results from optimized trajectory and the MPC are feasible, safe and fulfil all the proposed objectives.

Initially, a custom 3D model of the Hexacopter used in the real world missions of the FRIENDS project was designed and integrated into the Gazebo environment. The goal is not only to have a simulation model similar to the real Hexacopter but also to create an environment where it is possible and practical to simulate and test different sensors.

In order to simulate and control the Multirotor, there are two main options that are well documented and easy to work with. The first option is RotorS [80], which simulates the UAV dynamics and sensors with the help of Gazebo, performs state estimation and UAV control.

The second option is PX4 SITL (Software-In-The-Loop) simulations that perform the same functions as RotorS but where the UAV control, state estimation and data communication are performed by modules that are part of the PX4 Autopilot. This Firmware is compatible with several versions of PixHawk which is an independent open-hardware project providing readily-available, low-cost, and high-end, autopilot hardware designs to the academic, hobby and industrial communities [81]. Therefore, PX4 SITL was found to be the best option to perform the simulation validation since the PixHawk was the chosen platform to perform the low-level control of the real drone making the transition from the simulation environment to the real model relatively simple.

Finally, the software developed in this research is meant to run off-board the PixHawk, in a NVidia jetson nano, in a ROS (Robot Operating System) environment that communicates with the PX4 modules through the MAVROS protocol[1].

6.1 Gazebo Model

As mentioned, the first step of the simulation was importing the CAD (Computer-aided design) model of the Hexacopter to Gazebo. Gazebo has its own format to describe robots, objects, and the environment,

[1] http://wiki.ros.org/mavros

called Simulation Description Format (SDF). In this file extension, the multi-body dynamics of the Robot are defined as links, that can be fixed or rigid bodies, which are connected with joints. For my model only fixed and revolute joints were used. The fixed joints are connecting two links rigidly, such that there is no movement possible, and the revolute joints are hinge joints with mechanical limits that allow the component to rotate, such as my rotors as represented in figure 6.1 [80]. Every link added to the SDF file requires the definition of position, mass, inertia and, if desired, can be connected to a mesh file with the .stl extension that contains the visual representation (CAD) of that link. The full UAV model can be seen in fig. 6.2.

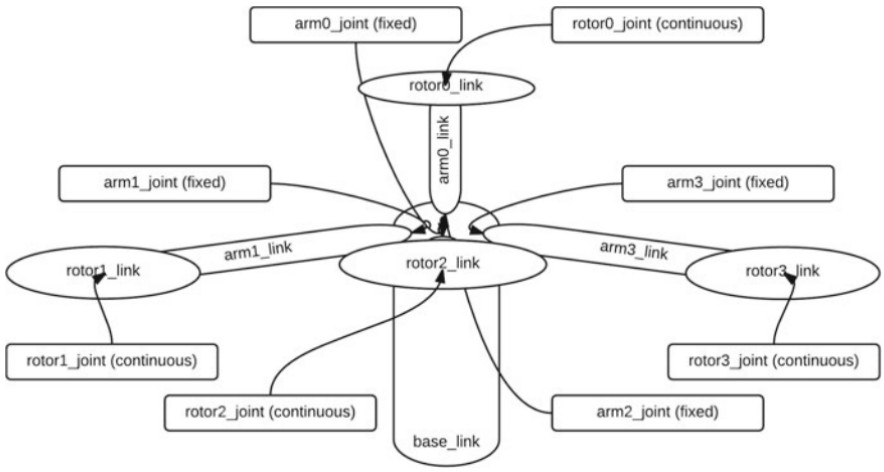


Figure 6.1: A draft of a multirotor helicopter with four non-symmetrical aligned rotors. Taken from [80].

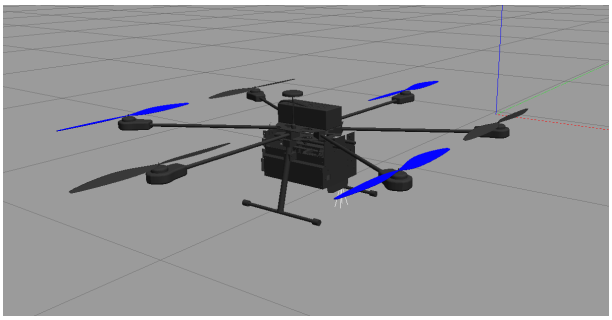


Figure 6.2: Gazebo model of the Project FRIENDS' Hexacopter.

For the FRIENDS Project, the UAV is comprised of two main components, the main frame and a removable sensor box. In order to facilitate the assembly and testing of the different drone configurations used in the three mission stages of this project, the sensor box is defined as a separate SDF model (see fig. 6.3) that can be easily and independently modifiable when it comes to adding and removing sensors.

Finally, the SDF file also includes all the Gazebo plugins necessary to simulate the rotor's physics and the incorporated sensors. In my model, the dynamic proprieties of the rotors are simulated by plugin libgazebo_motor_model[2] that, if provided with the correct constants and coefficients of the real rotors, can accurately simulate the real model behaviour.

[2]https://github.com/PX4/PX4-SITL_gazebo/tree/master/include

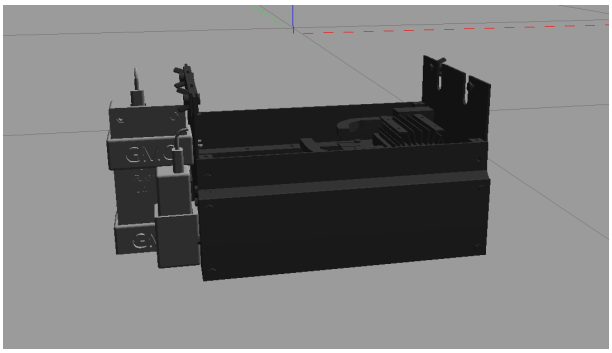


Figure 6.3: Gazebo model of the Project FRIENDS' Sensor Box.

The final SDF model also includes several simulated sensors like the IMU, Magnetometer, barometer and GPS that interface directly with the PX4 modules to estimate the states and errors using the autopilot's internal Extended Kalman Filter.

All the other sensors, like the Intel Realsense D435 (see fig. 6.4) and the Velodyne VLP-16 lidar (see fig. 6.5), require special plugins that can directly communicate with the ROS environment by publishing the sensor simulated information as topics that can be easily subscribed by any control nodes.

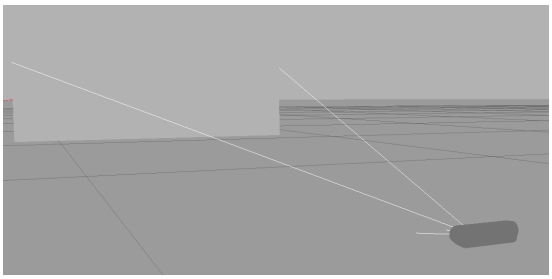


Figure 6.4: Gazebo Model of a Realsense D435 Camera.

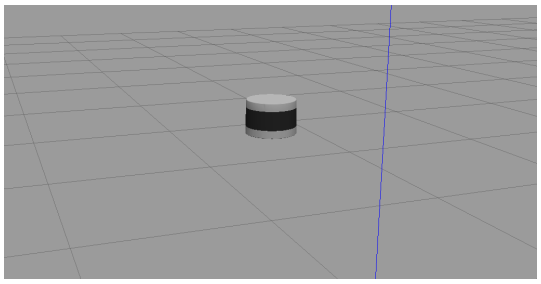


Figure 6.5: Gazebo Model of a VLP-16 Lidar.

6.2 Height map Model

One important step to fully test the implemented control algorithms is to simulate a realistic environment that validates the terrain following the trajectory objective generated by the optimization problem. Therefore, a DEM file was obtained from the United States Geologic Survey National Elevation Dataset[3] and imported as a gazebo world.

A Digital Elevation Model (DEM) is a 3D representation of a terrain's surface that does not include any objects like buildings or vegetation. DEMs are frequently created by using a combination of sensors, such as LIDAR, radar, or cameras. The terrain elevations for ground positions are sampled at regularly-spaced horizontal intervals and are then saved in a discrete matrix data format [82].

The term DEM is just a generic denomination, not a specific format. In fact, the DEMs can be represented as a grid of elevations (raster) or as a vector-based triangular irregular network (TIN). Currently, Gazebo only supports raster data in the supported formats available in Geospatial Data Abstraction

[3] https://viewer.nationalmap.gov

Library(GDAL)[82]. The format chosen to represent the elevation data is .asc (ascii) because it can be easily parsed without the need for any additional libraries.

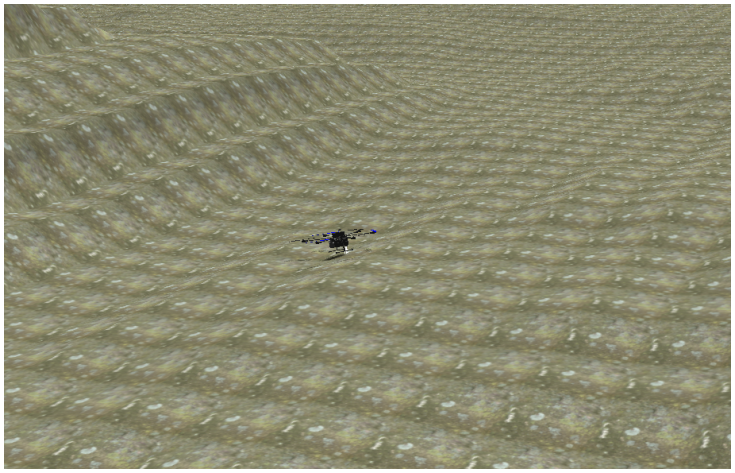


Figure 6.6: Gazebo world with terrain elevation data as floor.

This DEM file will not only be imported to Gazebo but will also be loaded and used directly as the terrain height data in the trajectory optimization algorithm described in chapter 4.

6.3 Software Architecture

6.3.1 PX4 Autopilot

PX4 is a Professional Autopilot developed by world-class developers from industry and academia and supported by an active world wide community. It powers all kinds of vehicles from racing and cargo drones through to ground vehicles and submersibles [83].

There are several options for flight control hardware that can be used to run the PX4 flight stack. As previously mentioned, the autopilot hardware used on this project's drone belongs to the Pixhawk series that can run PX4 on NuttX OS.

Pixhawk is an independent open-hardware project providing readily-available, low-cost, and high-end, autopilot hardware designs to the academic, hobby and industrial communities. This flight control board has some embedded sensors to provide useful data to a Ground Control Station or off-board API: a gyroscope, an accelerometer, a barometer and a magnetometer [81].

PX4 autopilot consists of two main layers: the flight stack which is an estimation and flight control system, and the middleware which is a general robotics layer that can support any type of autonomous robot, providing internal/external communications and hardware integration [60].

The communication to off-board APIs is performed employing the MAVLink protocol, which can be used on a Serial link or UDP network connection. For internal communications, the system uses uORB when the messages are not directly handled in the module. The uORB is the internal pub-sub messaging system, used for communication between modules [60].

The PX4 autopilot that runs onboard the Pixhawk includes three independent control modules based

on a PID architecture that perform a basic level control for standard waypoint missions. For multicopters, the control is performed by a cascade of PIDs as illustrated in the following figure.

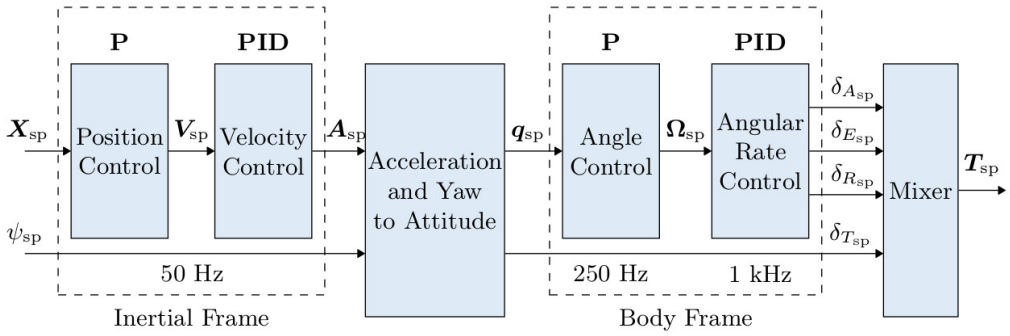


Figure 6.7: PX4 Multicopter Control Architecture taken from [60].

As previously mentioned, PX4 also offers a few simulation options in several platforms like Gazebo, jMAVSim and AirSim. For the Gazebo platform, it includes support not only for multirotors with several motor configurations but also for planes, VTOLs, Rovers and UUVs. PX4 supports both Software In the Loop (SITL) simulation, where the flight stack runs on a computer (either the same computer or another computer on the same network) and Hardware In the Loop (HITL) simulation using a simulation firmware on a real flight controller board.

Finally, some changes were performed to the default parameters of PX4 Firmware to fulfil the objectives of this project. The first change was adding the FRIENDS Hexacopter model to de SITL model's directory and adding the configuration file for the model that includes the PID gains and Mixer type used to control the multirotor.

The second change is related to the rate of messages published by the autopilot. Since the translational control is performed off-board it is necessary to increase the rate of the local position odometry messages from 30Hz to 50Hz in order to improve the results of the MPC and guarantee a good and efficient estimation of external disturbances. The rate of some non-essential messages was also reduced to prevent an overflow of the communication channel that could lead to the lost of messages.

6.3.2 Implemented Control Architecture

As already described, the Model Predictive Controller and trajectory optimizer run off-board in a ROS Melodic environment that communicates with the PX4 autopilot via MAVROS.

The proposed architecture of control and navigation focused by this thesis is represented in image 6.8. Initially, the mission trajectory is generated in a horizontal path planning module wielding latitude and longitude waypoints, that can adaptively change or be refined during the mission execution. The trajectory is then optimized by the trajectory optimizer described in chapter 4 that outputs a minimum time, terrain-following reference to be tracked by the position Modular Predictive Controller. In order to convert the output of the trajectory optimizer to the input of the MPC a quadratic interpolation of the first is obtained and sampled to the time step of the second.

The Optimization controller outputs attitude and normalized thrust references that are sent to the on-board PX4 autopilot control loops. The low-level attitude control is then performed by PX4 that ultimately converts the MPC references to rotor velocity commands.

Finally, the off-board controller receives the estimated states information from the autopilot and solves the next iteration of the MPC closing the control loop.

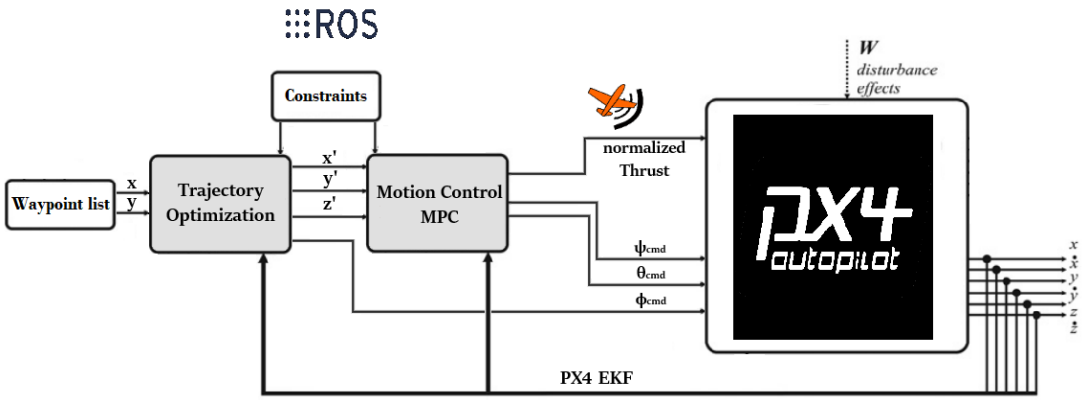


Figure 6.8: Scheme of the Control architecture of this Thesis. First I optimize the trajectory creating a 3D reference and then the MPC performs the high-level control by sending commands to the PX4 autopilot.

6.4 MPC Simulation Results

In order to properly tune the designed MPC, several simulation scenarios were tested and analysed, where the MPC parameters such as sampling time and weights of the cost matrices were varied until the desired behaviour was presented by the simulated UAV. The following tests were conducted using the PX4 SITL simulation environment, explained in the next section, integrated with ROS Melodic and Gazebo 9. This simulation setup is tested on the same Hp Laptop running Ubuntu 18.04 and equipped with an Intel Core i7-9750H CPU @2.60GHz and 16,00GB of RAM.

In my setup, the optimization controller is running at 50 Hz while internally the prediction time step is $\Delta t = 0.1s$, in this way I achieve a longer prediction horizon with less computational effort. By enforcing the iteration to run roughly 5x faster than the discretization time, I obtain small deviations of the predicted state vector between iterations which facilitates convergence. The prediction horizon is chosen to be $N = 20$ steps resulting into $T_h = 2$ seconds prediction horizon.

The MPC parameters used in the following simulations, presented in table 6.1, were determined by applying the system identification method described in section 5.3 and are characteristic of the UAV model and the attitude control loops and gains of the PX4 autopilot.

As for the cost matrices, I assigned weights to each component of the Least Quadratic objective function (Table 6.2) so that the higher weighted states in the cost function are the most important in a trajectory tracking control problem. Each cost matrix is defined as a diagonal matrix as demonstrated in (6.1) and (6.2).

Table 6.1: FRIENDS hexacopter parameters and control input constraints.

Parameter	Value
K_ϕ	0.995
K_θ	0.963
K_ψ	0.990
τ_ϕ	0.1430 $[s]$
τ_θ	0.1650 $[s]$
τ_ψ	0.4020 $[s]$
V_{xy}^+	2.0 $[m/s]$
V_z^+	3.0 $[m/s]$
$\tilde{T}^+$	4.0 $[m/s^2]$
$\tilde{T}^-$	15.0 $[m/s^2]$
ϕ^+	30 $[^\circ]$
θ^+	30 $[^\circ]$

Table 6.2: MPC weight constants.

x^q	y^q	z^q	v_x^q	v_y^q	v_z^q	ψ^q	θ^q	$\tilde{T}^q$	ψ_{cmd}^q	θ_{cmd}^q
90	90	120	80	80	90	10	10	1.5	55	55

$$Q = diag(x^q,\ y^q,\ z^q,\ \mathsf{v}_x^q,\ \mathsf{v}_y^q,\ \mathsf{v}_z^q,\ \psi^q,\ \theta^q,\ 0)\,, \qquad P = diag(\tilde{T}^q,\ \psi_{cmd}^q,\ \theta_{cmd}^q) \tag{6.1}$$

The final cost matrix does not include cost for the attitude states since the main goal of this optimization is to perform control of the translational dynamics. Regarding the remaining weights, the final cost matrix is scaled by 2 which means that the terminal cost is always 2^2 times more than the stage cost for x.

$$R = diag(2x^q,\ 2y^q,\ 2z^q,\ 2\mathsf{v}_x^q,\ 2\mathsf{v}_y^q,\ 2\mathsf{v}_z^q,\ 0,\ 0,\ 0) \tag{6.2}$$

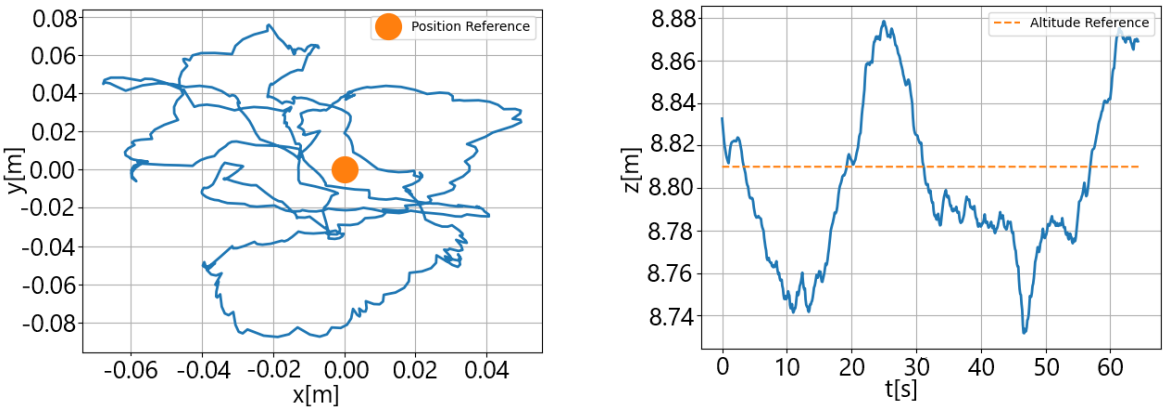
The implemented weights penalize mainly the position and velocity errors that are fundamental for accurate trajectory tracking. The input weights are considerably smaller and will allow the optimizer to correct the most penalized errors by adjusting the attitude and mainly the thrust references calculated by the trajectory optimizer.

6.4.1 Hovering Performance

An initial test was performed to validate the capabilities of the designed Model Predictive Controller to maintain the UAV hovering in the proximity of a given reference. This first test is essential to guarantee the stability of the MPC in a practical scenario where it is impossible to completely eliminate the state error.

From the data in Figure 6.9 I can confirm that the implemented MPC is stable even when subject to small disturbances and modelling errors since the position of the UAV stays close to the reference with

a maximum error of 8 cm in the horizontal plane and 9 cm in the vertical plane.



(a) Horizontal Position during Hover Mission.

(b) Vertical Position during Hover Mission.

Figure 6.9: Graphical representation of the 3D performance of the MPC while hovering around a fix position, applied to the simulated FRIENDS' hexacopter.

Figure 6.10 shows the command outputs of the MPC during the Hover experiment. Although it is possible to identify considerable oscillation in the following plots, it is important to state that the commanded angles only oscillate 2.0 deg around a reference point and therefore are unnoticeable and do not represent visible oscillations in the simulated model.

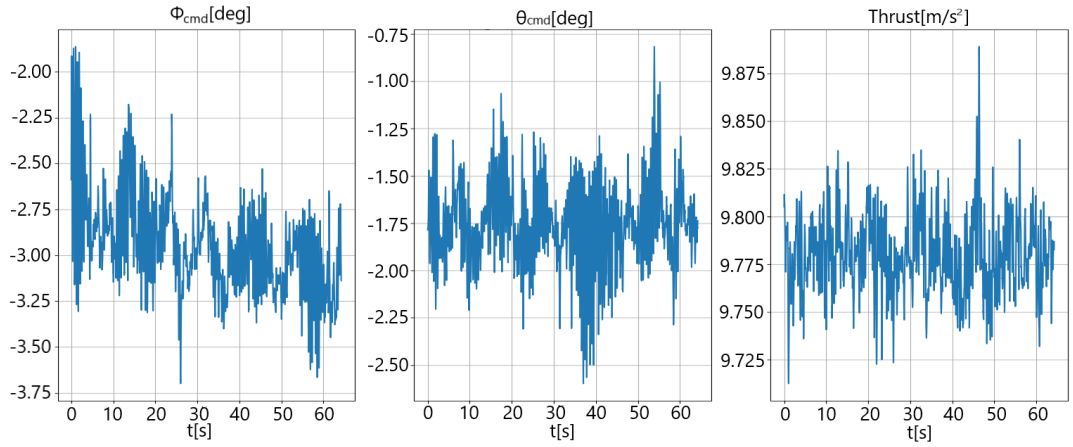


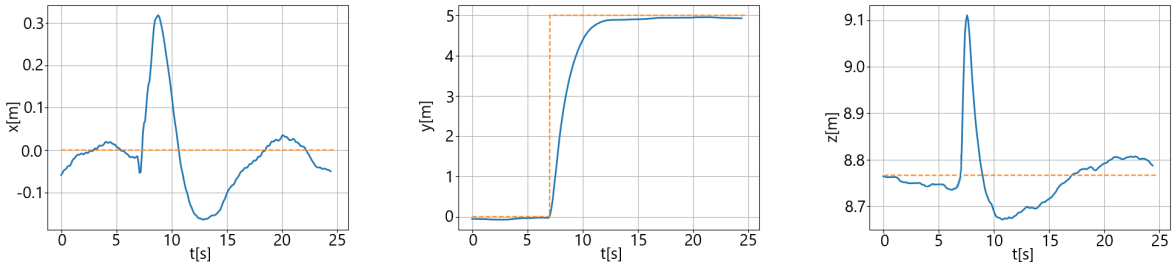
Figure 6.10: Graphical representation of the MPC output commands during the hovering mission.

6.4.2 Step Reference

As a second test, I introduced a step in the MPC's position reference to evaluate the performance of the optimization controller when the reference is not a smooth function. In this case, no prior optimization is performed meaning that the MPC optimizes the full path to reach the new position reference while accounting for the constraints imposed in the control (Table 6.1).

In Figure 6.11 I can see the response of the MPC to a step-change in the reference of the y coordinate while maintaining the references for the remaining states constant. These graphs show that in order to

reduce the error of the y position, caused by the step input, the UAV also deviated its x and z coordinates from the reference. A likely explanation for this behaviour is related to the input saturation of the thrust and pitch commands, demonstrated in Figure 6.12, resulting in an optimum trajectory different from the expected straight line from the previous position to the new reference.



(a) Position X (b) Position Y (c) Position Z

Figure 6.11: Graphical representation of the 3D performance of the MPC when a step on the y position reference is introduced.

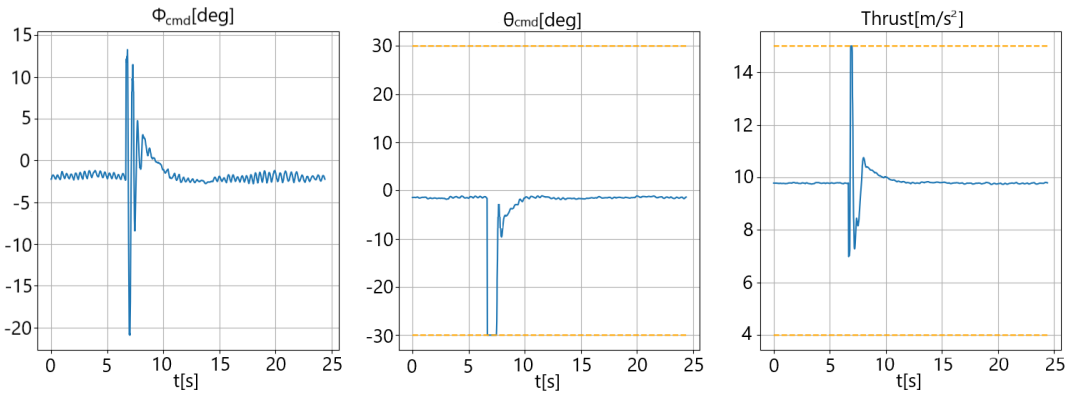


Figure 6.12: Graphical representation of the MPC output commands when a step on the y position reference is introduced.

6.4.3 Trajectory Tracking without Wind

Since the main goal of the designed Model Predictive Control is to precisely track previously computed trajectories, I tested the performance of the MPC in my UAV model by using a smooth polynomial path with aggressive turns and steep descents.

In Figure 6.13, I represent the reference smooth trajectory obtained from the polynomial optimization described in Appendix A with user defined waypoints in order to have relatively sharp turns. The position errors along the trajectory shown in Figure 6.14 confirm that the designed MPC can efficiently and precisely track smooth trajectories. In the tested flight, I have a maximum error of around 15 cm for each axis that occurs close to sharp corners.

The command oscillations present in the results from the simulation (Figure 6.15) do not translate to oscillations in the translational states, i.e. the position and velocity are smooth functions of time along the trajectory. However, those oscillations can be reduced by further improving the UAV model used

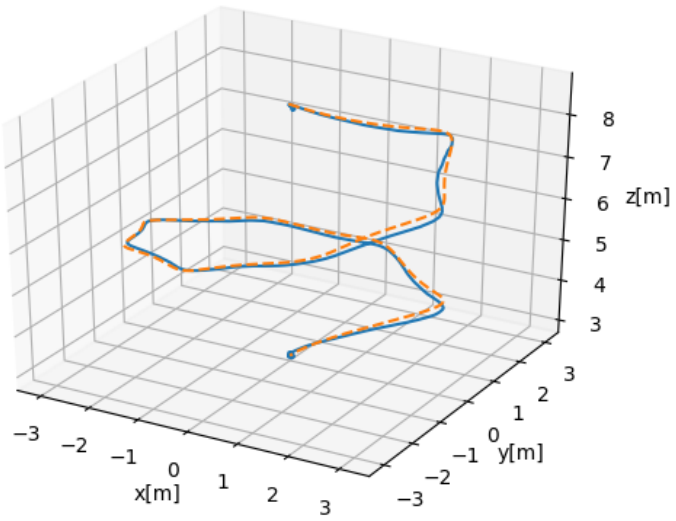
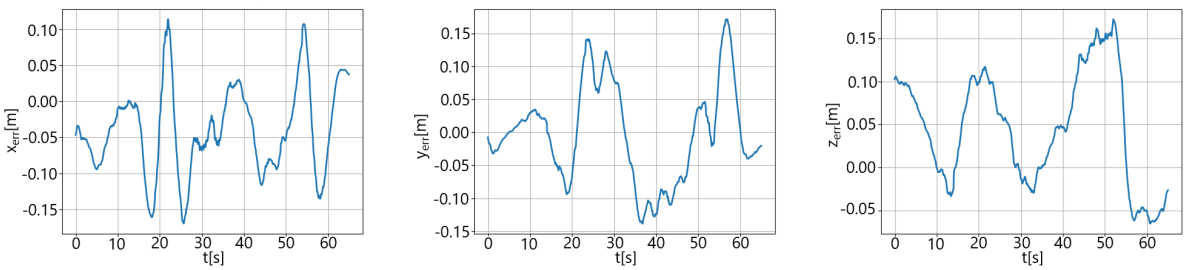


Figure 6.13: Graphical representation of the 3D trajectory executed without wind. In orange I represent the reference polynomial trajectory to be tracked and in blue is the trajectory performed by the simulated UAV.



(a) Error in x position. (b) Error in y position. (c) Error in z position.

Figure 6.14: Position tracking error of the MPC during the mission without wind.

in Gazebo, which appears to show some instability even when controlled by the PID loops of the PX4 autopilot and increasing the rate of control currently set to 50 Hz which will allow for a more precise and stable control, reducing unwanted oscillations that can affect the sensors data acquisition.

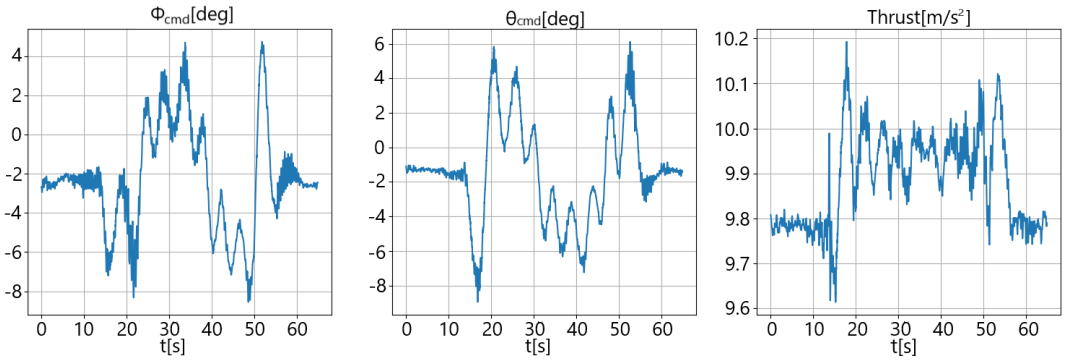


Figure 6.15: Graphical representation of the MPC output commands during the polynomial tracking mission without wind.

6.4.4 Trajectory Tracking with Wind

Finally, the previous trajectory was tracked once again but now with the presence of wind in order to test the capability of the MPC coupled with the EKF to eliminate the effect of unpredictable external disturbances. In order to perform this test, I used the wind plugin provided by the PX4 community that employs a white Gaussian noise to simulate external wind with a mean velocity $v_W = 4[m/s]$ and pointing on the positive direction of the y axis.

For this case, two tests were performed to demonstrate the advantages of using a EKF to estimate external disturbances. In the first experiment present in figure 6.16, the EKF is active and estimates the wind force which is approximately constant.

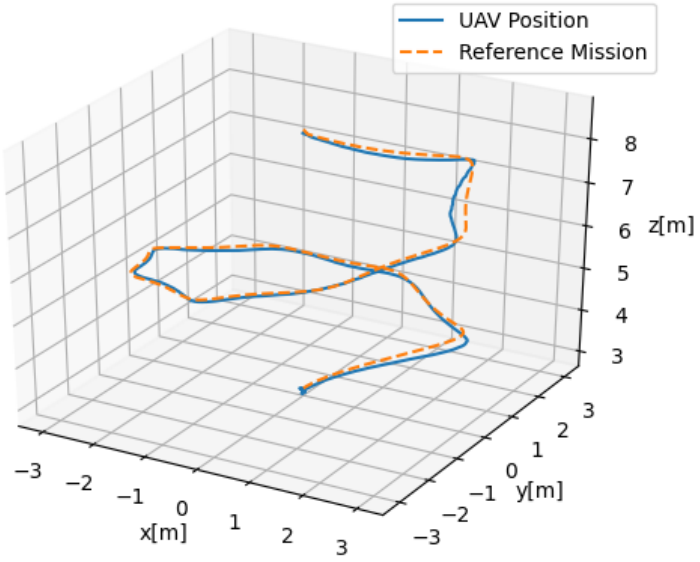
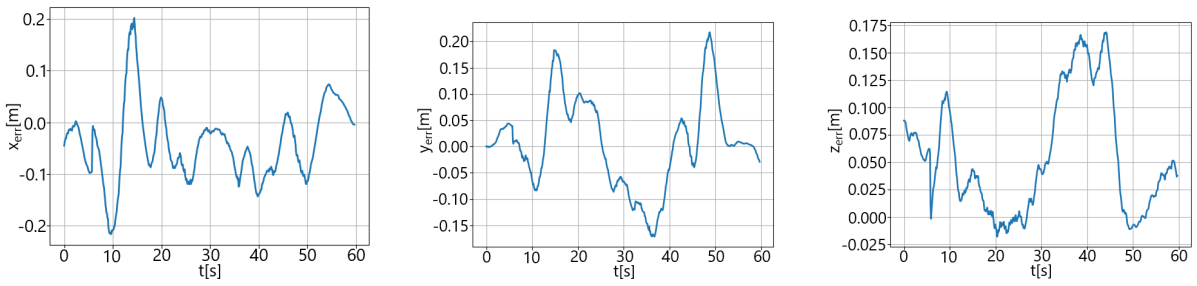


Figure 6.16: Graphical representation of the 3D trajectory executed with wind and with EKF. In orange I represent the reference polynomial trajectory to be tracked and in blue is the trajectory performed by the simulated UAV.



(a) Error in x position. (b) Error in y position. (c) Error in z position.

Figure 6.17: Position tracking error of the MPC during the mission with wind and EKF active.

In Figure 6.18 I can see that in order to eliminate the effects of the wind disturbance the commanded pitch angle variates around 11 degrees and the thrust force increased to around 10.2 $[m/s^2]$. The position error along the tracked trajectory also only shows a slight increase when compared to the case without any disturbances proving that the MPC coupled with an external disturbance estimator EKF effectively eliminates the wind effects.

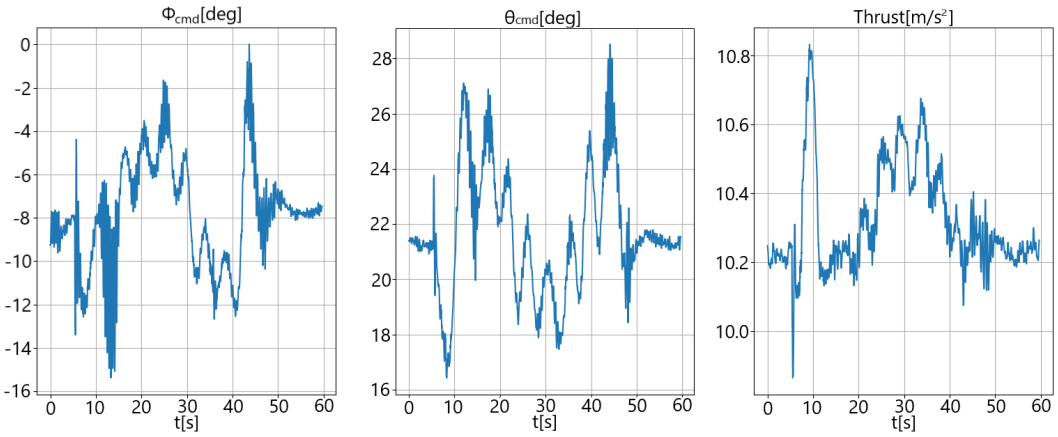


Figure 6.18: Graphical representation of the MPC output commands during the polynomial tracking mission with wind and EKF active.

In the second experiment shown in figure 6.19, the EKF was turned off and the external forces present in the MPC dynamic model were set to zero. From the position errors present in 6.20, it is visible that the external wind force in the y direction causes a steady state position error of about 0.9 m. This errors appears due to the inability that the MPC has to predict or understand this disturbance using only the dynamic model of the UAV.

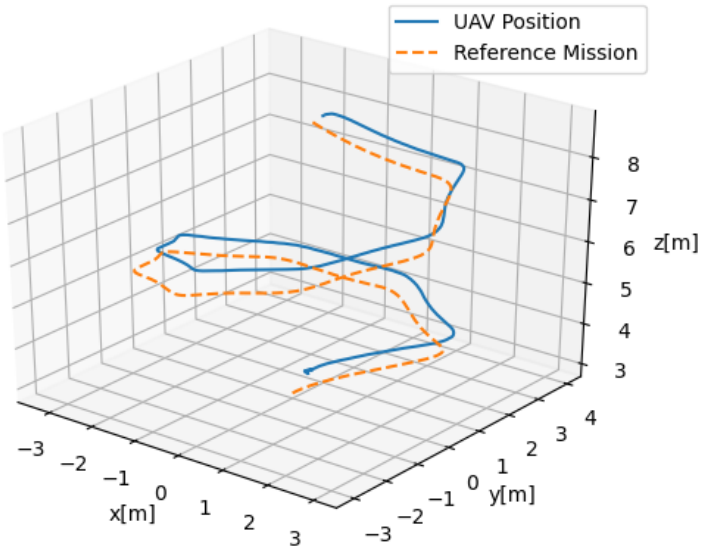
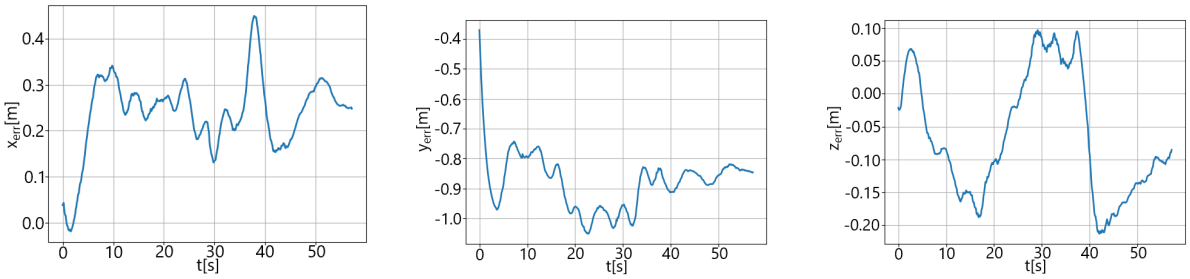


Figure 6.19: Graphical representation of the 3D trajectory executed with wind and without EKF. In orange I represent the reference polynomial trajectory to be tracked and in blue is the trajectory performed by the simulated UAV.

These results validate the importance of using an EKF to predict external disturbances. It not only to reduces the effect of modelling errors and small uncounted drag forces but also efficiently estimates big external disturbance allowing the MPC to achieve a null steady state error.



(a) Error in x position. (b) Error in y position. (c) Error in z position.

Figure 6.20: Position tracking error of the MPC during the mission with wind and EKF inactive.

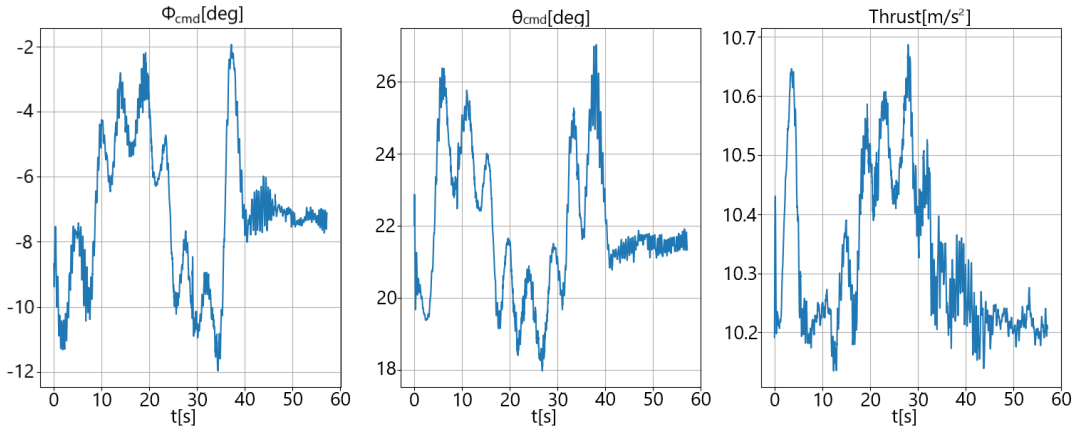


Figure 6.21: Graphical representation of the MPC output commands during the polynomial tracking mission with wind and EKF inactive.

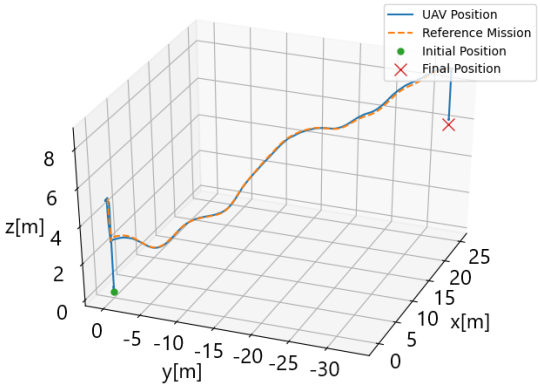
6.5 Full Control Architecture Simulation

In this section, a final simulation is performed where the path optimized in the first scenario of section 4.6 is used as a reference for the MPC. This experiment will allow me to validate the full cascade architecture presented in figure 6.8. The mission performed by the simulated UAV can be divided into 3 distinct stages. Initially, the UAV performs vertical take-off using only the PX4 autopilot controlling loops.

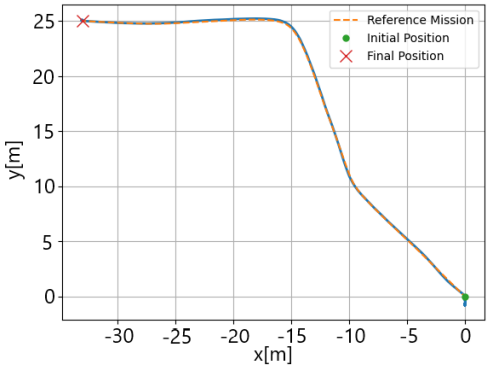
Then the second phase starts where the translational control is shifted from the autopilot to the off-board MPC. In this phase, the UAV starts by adjusting the height to the desired altitude above the terrain. After the terrain following trajectory is optimized the execution of the mission resumes.

Finally, after arriving at the last waypoint the UAV lands using once again the PX4 autopilot controller and the mission terminates.

The full mission can be seen in figure 6.22 and the three phases are illustrated in figure 6.23 where the altitude of the UAV is represented as a function of time. I can therefore conclude that the designed tracking controller is also capable of efficiently and precisely track the trajectory optimized by the terrain following algorithm which validates the final objective of coupling the controller and trajectory optimizer is possible.



(a) 3D representation of the terrain-following mission.



(b) Top-down view of the terrain-following mission.

Figure 6.22: Graphical representation of the 3D trajectory executed in the Terrain-Following Mission where both the MPC and the trajectory optimizer are working in cascade. In orange I represent the reference polynomial trajectory to be tracked and in blue is the trajectory performed by the simulated UAV.

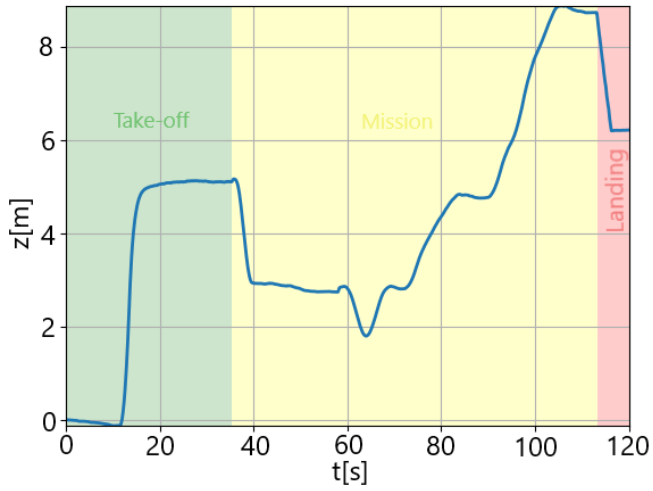


Figure 6.23: Graphical representation of the vertical trajectory executed in the Terrain-Following Mission where both the MPC and the trajectory optimizer are working in cascade. The trajectory is divided in 3 separate stages. In green is represented the Takeoff segment of the mission, in yellow is represented the terrain following stage and in red is represented de Landing phase.

Chapter 7

Conclusions

This thesis proposes a solution for terrain-following with an control trajectory optimization and the design and validation of an online controller capable of following the optimized path even in the presence of external disturbances. The first goal was achieved with an initial trajectory optimization that uses known terrain data and pre-determine waypoints and the second objective was tackled by an online control algorithm based on MPC.

Initially, I addressed the trajectory generation problem by designing a non-linear optimization problem that employed the UAV dynamics to determine an optimized control path that satisfies all mission requirements. The multirotor dynamics were discretized using a direct transcription method that converts continuous-time functions into a set of equality constraints. The trajectory optimization algorithm was able to determine a 3D path that passes through a pre-determined set of horizontal waypoints while keeping the vertical distance to the ground inside a safe and operational window. Another tackled goal was to obtain a trajectory that finds a balance between the minimum time a minimum acceleration to wield smooth trajectories that minimize the mission's duration. All these objectives were incorporated into the optimization problem by way of weighted cost functions and equality/inequality constraints for every discretization time step. The algorithm was tested and validated using CasADi and the interior-point solver IPOPT, where I concluded that the optimization was able to determine terrain-following trajectories even in the presence of elevation discontinuities.

Secondly, I tackled the design and implementation of a controller capable of precisely track the optimized path in the presence of unpredictable external disturbances. My approach is based on a Model Predictive Control, where an iterative optimization problem generates control inputs in real-time. The UAV dynamics implemented in the optimization problem were simplified based on a cascade approach where the rotational dynamics are controlled by a separate autopilot that uses high rate PID feedback loops. The MPC was developed in the C++ interface for ACADO, where I had to specify certain MPC parameters, such as sampling time and time horizon, as well as an objective function and a set of constraints. The goal of this control problem is to minimize the state error while fulfilling the multirotor dynamic constraints. This algorithm was ultimately used in a ROS environment allowing for test and validation of the controller in a SITL simulation where a model of the FRIENDS' Hexarotor was integrated

on. This model was developed using CAD models and approximated inertial data and was incorporated in the PX4 Simulation Firmware in order to use the same autopilot software as the real-life UAV.

Finally, both algorithms were coupled together by using the terrain-following optimized trajectory as a state and input reference for the NMPC. This integration allowed me to conclude that the online controller can reasonably track the optimum trajectory validating the initially planned cascade navigation architecture.

7.1 Future Work

In terms of future work, several lines of research could be pursued.

The next logical step is completing the control and navigation pipeline in the real Hexacopter model from Project FRIENDS. This will allow for further validation of the precise trajectory tracking capabilities of the MPC when executing missions in a real outdoor environment.

Regarding the trajectory generation and optimization, better integration between the trajectory optimizer and the MPC controller is still necessary. In other words, the trajectory optimization needs to be integrated into the ROS framework so that it can run autonomously onboard the UAV. The presented algorithm also requires further investigation in terms of efficiency and robustness in order to be ultimately used in a real-world application. Another interesting approach could be an augmentation or reformulation of the current trajectory optimization algorithm in order to include obstacle avoidance constraints and real-time terrain data. This obstacle avoidance problem requires a sense and detection algorithm capable of using data from several sensors to locate new obstacles in the mission environment and then representing them as convex constraints in the trajectory optimization algorithm.

Another important area of improvement is the simulation of the FRIENDS UAV. In order to accurately represent a real multirotor model in a simulation environment, further tests have to be carried out. It is necessary to accurately measure all moments of inertia and determine the correct parameters that describe the motors and propellers of the real model.

Finally, another requirement of project FRIENDS regarding vertical control and optimization is to locate a safe place to land close to the desired location. This can be seen as another optimization problem that is initially solved using the same Digital Elevation Data as the trajectory optimizer and then once above the landing site further real-time re-optimization can be performed using sensor data.

Bibliography

[1] V. Prisacariu. The history and the evolution of uavs from the beginning till the 70s. In *Journal of Defense Resources Management*, volume 8, page 181. Gale Academic OneFile, 2017.

[2] J. Keane and S. Carr. A brief history of early unmanned aircraft. *Johns Hopkins Apl Technical Digest*, 32:558–571, 2013.

[3] S. Hayat, E. Yanmaz, and R. Muzaffar. Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint. *IEEE Communications Surveys Tutorials*, 18(4):2624–2661, 2016.

[4] C. Deng, S. Wang, Z. Huang, Z. Tan, and J. Liu. Unmanned aerial vehicles for power line inspection: A cooperative way in platforms and communications. *Journal of Communications*, 9:687–692, 2014.

[5] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart. Receding horizon "next-best-view" planner for 3d exploration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1462–1468, 2016.

[6] M. Silvagni, A. Tonoli, E. Zenerino, and M. Chiaberge. Multipurpose uav for search and rescue operations in mountain avalanche events. *Geomatics, Natural Hazards and Risk*, 8(1):18–33, 2017.

[7] H. T. Berie and I. Burud. Application of unmanned aerial vehicles in earth resources monitoring: focus on evaluating potentials for forest monitoring in ethiopia. *European Journal of Remote Sensing*, 51(1):326–335, 2018.

[8] R. K. Rangel and A. C. Terra. Development of a surveillance tool using uav's. In *2018 IEEE Aerospace Conference*, pages 1–11, 2018.

[9] M. Nagai, T. Chen, R. Shibasaki, H. Kumagai, and A. Ahmed. Uav-borne 3-d mapping system by multisensor integration. *IEEE Transactions on Geoscience and Remote Sensing*, 47(3):701–708, 2009.

[10] G. Rousseau, C. STOICA MANIU, S. Tebbani, N. Martin, and M. Babel. Minimum-time b-spline trajectories with corridor constraints. application to cinematographic quadrotor flight plans. *Control Engineering Practice*, 89:190–203, 2019.

[11] E. National Academies of Sciences and Medicine. *Innovations in the Food System: Exploring the Future of Food: Proceedings of a Workshop-in Brief*. The National Academies Press, 2019.

[12] M. M. Marques, P. Dias, N. P. Santos, V. Lobo, R. Batista, D. Salgueiro, A. Aguiar, M. Costa, J. E. da Silva, A. Ferreira, J. Sousa, M. de Fatima Nunes, E. Pereira, J. Morgado, R. Ribeiro, J. Marques, A. Bernardino, M. Griné, and M. Taiana. Unmanned aircraft systems in maritime operations: Challenges addressed in the scope of the seagull project. *OCEANS 2015 - Genova*, pages 1–6, 2015.

[13] C. Cai, B. Carter, M. Srivastava, J. Tsung, J. Vahedi-Faridi, and C. Wiley. Designing a radiation sensing uav system. In *2016 IEEE Systems and Information Engineering Design Symposium (SIEDS)*, pages 165–169, 2016.

[14] K. Boudergui, F. Carrel, T. Domenech, N. Guenard, J. . Poli, A. Ravet, V. Schoepff, and R. Woo. Development of a drone equipped with optimized sensors for nuclear and radiological risk characterization. In *2011 2nd International Conference on Advancements in Nuclear Instrumentation, Measurement Methods and their Applications*, pages 1–9, 2011.

[15] Y. J. Brouwer. Radiological monitor for mobile robots operating in scenarios with nuclear threats. Master's thesis, Instituto Superior Tecnico, 2019.

[16] A. Kosari, H. Maghsoudi, and A. Lavaei. Path generation for flying robots in mountainous regions. *International Journal of Micro Air Vehicles*, 9(1):44–60, 2017.

[17] P. Lu and B. Pierson. Optimal aircraft terrain-following analysis and trajectory generation. *Journal of Guidance, Control, and Dynamics*, 18:555–560, 1995.

[18] P. Menon, V. Cheng, and E. Kim. Optimal trajectory synthesis for terrain-following flight. *Journal of Guidance Control and Dynamics*, 14:807–813, 1991.

[19] Y. Feng, W. Chen, L. Yang, and D. Wei. Optimal terrain following trajectory regeneration using linear gauss pseudo-spectral method. *2019 IEEE International Conference on Unmanned Systems (ICUS)*, pages 205–211, 2019.

[20] I. Khademi, B. Maleki, and A. Nasseri Mood. Optimal three dimensional terrain following/terrain avoidance for aircraft using direct transcription method. In *2011 19th Mediterranean Conference on Control Automation (MED)*, pages 254–258, 2011.

[21] R. Zardashti, A. Nikkhah, and M. Yazdanpanah. Constrained optimal terrain following/threat avoidance trajectory planning using network flow. *Aeronautical Journal*, 118:523–539, 2014.

[22] M. Rahim and S. Malaek. Aircraft terrain following flights based on fuzzy logic. *Aircraft Engineering and Aerospace Technology: An International Journal*, 83:94–104, 2011.

[23] D. Lee and D. H. Shim. Spline-rrt* based optimal path planning of terrain following flights for fixed-wing uavs. In *2014 11th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 257–261, 2014.

[24] S. Li, G. Liu, and J. Wu. A self-learning terrain-following method for aircrafts. *2017 36th Chinese Control Conference (CCC)*, pages 3437–3442, 2017.

[25] H. Eslamiat, Y. Li, N. Wang, A. K. Sanyal, and Q. Qiu. Autonomous waypoint planning, optimal trajectory generation and nonlinear tracking control for multi-rotor uavs. In *2019 18th European Control Conference (ECC)*, pages 2695–2700, 2019.

[26] D. Falanga, P. Foehn, D. Scaramuzza, N. Kuppuswamy, and R. Tedrake. Fast trajectory optimization for agile quadrotor maneuvers with a cable-suspended payload. 2017.

[27] P. Foehn and D. Scaramuzza. Cpc: Complementary progress constraints for time-optimal quadrotor trajectories, 2020.

[28] C. Richter, A. Bry, and N. Roy. *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*, pages 649–666. Springer International Publishing, 2016.

[29] B. Kada. Robust pid controller design for an uav flight control system. Newswood, 10 2011.

[30] P. Pounds, D. Bersak, and A. Dollar. Stability of small-scale uav helicopters and quadrotors with added payload mass under pid control. *Autonomous Robots*, 33, 2012.

[31] L. Sun, R. Beard, and D. Pack. Trajectory-tracking control law design for unmanned aerial vehicles with an autopilot in the loop. *Proceedings of the American Control Conference*, pages 1390–1395, 2014.

[32] S. Xingling, J. Liu, H. Cao, C. Shen, and W. Honglun. Robust dynamic surface trajectory tracking control for a quadrotor uav via extended state observer. *International Journal of Robust and Nonlinear Control*, 28, 2018.

[33] G. Vasan, A. Singh, and K. Krishna. Model predictive control for micro aerial vehicle systems (mav) systems. *ArXiv*, abs/1412.2356, 2014.

[34] C. Sferrazza, M. Muehlebach, and R. D'Andrea. Trajectory tracking and iterative learning on an unmanned aerial vehicle using parametrized model predictive control. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 5186–5192, 2017.

[35] G. V. Raffo, M. G. Ortega, and F. R. Rubio. Mpc with nonlinear h-infiny control for path tracking of a quad-rotor helicopter. *IFAC Proceedings Volumes*, 41(2):8564 – 8569, 2008.

[36] M. S. Kamel, M. Burri, and R. Siegwart. Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine*, 50, 2017.

[37] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[38] L. Grne and J. Pannek. *Nonlinear Model Predictive Control: Theory and Algorithms*. Springer Publishing Company, Incorporated, 2013.

[39] J. McCall. Genetic algorithms for modelling and optimisation. *Journal of Computational and Applied Mathematics*, 184(1):205 – 222, 2005.

[40] A. Jevtic, D. Andina, A. Jaimes, J. Gomez, and M. Jamshidi. Unmanned aerial vehicle route optimization using ant system algorithm. pages 1 – 6, 2010.

[41] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[42] S. S. Rodrigues. *Aero-thermal analysis and design of turbomachinery blades using multi-stage adjoint methods*. PhD thesis, Instituto Superior Tecnico, 2019.

[43] T. E. Abrudan, J. Eriksson, and V. Koivunen. Steepest descent algorithms for optimization under unitary matrix constraint. *IEEE Transactions on Signal Processing*, 56, 2008.

[44] J. E. Dennis, Jr. and J. J. Moré. Quasi-newton methods, motivation and theory. *SIAM Review*, 19 (1):46–89, 1977.

[45] G. A. Gabriele and K. M. Ragsdell. The Generalized Reduced Gradient Method: A Reliable Tool for Optimal Design. *Journal of Engineering for Industry*, 99(2):394–400, 1977.

[46] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone. Gusto: Guaranteed sequential trajectory optimization via sequential convex programming. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6741–6747, 2019.

[47] J. T. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, pages 193–207, 1998.

[48] B. A.R. and J. Hedengren. *Optimization Methods for Engineering Design*. Brigham Young University, second edition, 2018.

[49] M. Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59:849–904, 2017.

[50] M. P. Kelly. Optimtraj. `https://github.com/MatthewPeterKelly/OptimTraj/tree/master/docs/AnalyticGradients/Figures`. [Online; accessed 29-12-2020].

[51] B. Mehta and Y. Reddy. Chapter 19 - advanced process control systems. In *Industrial Process Automation Systems*, pages 547 – 557. Butterworth-Heinemann, 2015.

[52] E. Kaiser, J. Kutz, and S. Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 474, 2017.

[53] S. Skogestad. Model-based predictive control. `http://folk.ntnu.no/skoge/vgprosessregulering/lectures/part3_multivariable/MPC_Hovd_Pages-from-kurskompendium.pdf`. [Online; accessed 11-12-2020].

[54] M. S. Kamel, T. Stastny, K. Alexis, and R. Siegwart. *Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System*. 2017.

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

[56] M. Kamel, K. Alexis, M. Achtelik, and R. Siegwart. Fast Nonlinear Model Predictive Control for Multicopter Attitude Tracking on SO(3), 2015.

[57] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[58] R. Mahony, V. Kumar, and P. Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics Automation Magazine*, 19(3):20–32, 2012.

[59] K. Alexis, G. Nikolakopoulos, and A. Tzes. Model predictive quadrotor control: attitude, altitude and position experimental studies. *IET Control Theory Applications*, 6(12), 2012.

[60] PX4. Px4 user guide. `https://docs.px4.io/master/en/`. [Online; accessed 10-12-2020].

[61] J.-M. Kai, G. Allibert, M.-D. Hua, and T. Hamel. Nonlinear feedback control of quadrotors exploiting first-order drag effects. *IFAC-PapersOnLine*, 50:8189 – 8195, 2017.

[62] S. Omari, M. Hua, G. Ducard, and T. Hamel. Nonlinear control of vtol uavs incorporating flapping dynamics. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2419–2425, 2013.

[63] J. Andersson, J. Akesson, and M. Diehl. Casadi: A symbolic package for automatic differentiation and optimal control. volume 87, 2012.

[64] APMonitor. Second order systems. `https://apmonitor.com/pdc/index.php/Main/SecondOrderSystems`. [Online; accessed 30-12-2020].

[65] T. Sharma, P. Williams, C. Bil, and A. Eberhard. Optimal three dimensional aircraft terrain following and collision avoidance. *Anziam Journal*, 47:695–711, 2007.

[66] M. Neunert, C. de Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli. Fast nonlinear model predictive control for unified trajectory optimization and tracking. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1398–1404, 2016.

[67] T. A. Howell, B. E. Jackson, and Z. Manchester. Altro: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679, 2019.

[68] P. Sopasakis, E. Fresk, and P. Patrinos. Open: Code generation for embedded nonconvex optimization. *IFAC World Congress 2020, Berlin*, 2020.

[69] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Rev.*, 47:99–131, 2005.

[70] R. H. Byrd, J. Nocedal, and R. A. Waltz. *Knitro: An Integrated Package for Nonlinear Optimization*, pages 35–59. Springer US, 2006.

[71] CasADi. CasADi's documentation. https://web.casadi.org/docs/#using-lookup-tables. [Online; accessed 09-12-2020].

[72] M. Posa and R. Tedrake. *Direct Trajectory Optimization of Rigid Body Dynamical Systems through Contact*, pages 527–542. Springer Berlin Heidelberg, 2013.

[73] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart. Vision based mav navigation in unknown and unstructured environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 21–28, 2010.

[74] D. Brescianini, M. Hehn, and R. D'Andrea. Nonlinear quadrocopter attitude control. Technical report, Eidgenössische Technische Hochschule Zürich, Departement Maschinenbau und Verfahrenstechnik, 2013.

[75] MathWorks. System Identification Toolbox: User's Guide (R2020b). https://www.mathworks.com/help/pdf_doc/ident/ident_ug.pdf, 2020. [Online; accessed 09-12-2020].

[76] M. Ribeiro and I. Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties, 2004.

[77] F. Borrelli, A. Bemporad, and M. Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[78] B. Houska, J. Ferreau, and M. Diehl. Acado toolkit-an open source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32:298 – 312, 2011.

[79] ACADO. The acado code generation tool. https://acado.github.io/cgt_overview.html. [Online; accessed 29-12-2020].

[80] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. *RotorS - A Modular Gazebo MAV Simulator Framework*, volume 625, pages 595–625. 2016.

[81] Pixhawk. Pixhawk series. https://docs.px4.io/master/en/flight_controller/pixhawk_series.html. [Online; accessed 10-12-2020].

[82] Gazebo. Digital elevation models. http://gazebosim.org/tutorials/?tut=dem. [Online; accessed 10-12-2020].

[83] L. Meier, D. Honegger, and M. Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6235–6240, 2015.

[84] M. Burri, H. Oleynikova, , M. W. Achtelik, and R. Siegwart. Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments. In *Intelligent Robots and Systems (IROS 2015), 2015 IEEE/RSJ International Conference on*, 2015.

[85] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011.

[86] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran. Continuous-time trajectory optimization for online uav replanning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

Appendix A

Polynomial Trajectory Planning

In this chapter, we will present another planning method that was also implemented in order to solve the waypoint tracking and minimum time problem. This method is based on the works in [28, 84], in which the trajectory planning problem for a generic multicopter is implemented as the optimization of polynomial coefficients with a pre-defined degree, resulting in smooth and continuous trajectories.

Since the main goal of this optimization formulation is efficiency and integration with the ROS environment and PX4 autopilot, the following non-linear optimization problem was formulated and solved with the open-source tool nlopt that interfaces directly with C++.

A.1 Problem Formulation

The approximation of a UAV trajectory by a piecewise polynomial in space is based on the differential flatness property of its dynamics. In other words, the states and the inputs of the multirotor dynamics can be analytically mapped from a 3D path and its derivatives. This powerful property effectively guarantees the feasibility of any smooth trajectory, provided that its derivatives are reasonably bounded to avoid input saturation, thus eliminating the need for iterative simulation in the search for trajectories. The Differential flatness of the general multirotor model was demonstrated by Mellinger and Kumar [85].

Minimum-snap polynomials have proven very effective as multirotor trajectories since the motor commands and attitude accelerations of the vehicle are proportional to the snap, or forth derivative, of the path [85]. Minimizing the snap of a trajectory quantifies a notion of ”smoothness” that is desirable for maintaining the quality of onboard sensor measurements as well as avoiding paths that would require abrupt or excessive control inputs [28].

For multicopters, a single trajectory segment between two points in state space is composed of independent polynomial trajectories for the flat output variables x, y, z and yaw angle. Each polynomial segment of order N with $N + 1$ coefficients at time t can be expressed as:

$$P(t) = \sum_{n=0}^{N} p_n t^n \tag{A.1}$$

Considering now the problem of navigating through M waypoints at specified times. We define the overall trajectory as piecewise polynomial functions of the form (A.1) over M time intervals such that:

$$\boldsymbol{p}(t) = \begin{cases} P_1(t), & t_0 \leq t < t_1\,, \\ P_2(t), & t_1 \leq t < t_2\,, \\ \vdots & \\ P_M(t), & t_{M-1} \leq t < t_M\,. \end{cases} \tag{A.2}$$

This optimization problem can be easily formulated as a constraint QP where we optimize the coefficients p_n and connect the polynomials P_n by setting continuity constraints. However, this method presents several limitations related to numerical issues. Since the trajectories depend directly on the term t^n, the problem becomes unsolvable when using polynomials of high degrees and for trajectories extended for long periods of time, as it exceeds the precision of the computer. Therefore, the optimization problem presented in paper [28], is based on an unconstrained QP reformulation of the constraint problem, which is robust to numerical instability.

A.2 Objective function

The objective function consists on the integral of the derivatives of $\boldsymbol{p}(t)$, where each polynomial is valid from $t = 0$ to the segment duration $t = T_{s,i}$, $i = 1, \ldots, M$. In case of multiple dimensions, each segment M consists of D polynomials, such that the cost over the whole trajectory is:

$$J_{pol} = \sum_{i=1}^{M} \sum_{d=1}^{D} \int_0^{T_{s,i}} \sum_{j=0}^{N} \mu_j \left\| \frac{d^j P_{i,d}(t)}{dt^j} \right\| \tag{A.3}$$

The cost $J_{i,d}$ for a polynomial in a segment i in dimension d with its derivatives weighted by μ_j can be written as:

$$J_{i,d} = \boldsymbol{c}_{i,d}^T \cdot \boldsymbol{H}_i \cdot \boldsymbol{c}_{i,d} \tag{A.4}$$

In this expression, $\boldsymbol{c}_{i,d}$ is a vector of the N coefficients of a single polynomial. In our system, we generate trajectories that minimize the integral of the square of the snap. In order to minimize snap, all derivative penalties in the cost function except for μ_4 would be set to zero. The construction of the Hessian matrix $\boldsymbol{H}_i$ follows from the differentiation of the square of the polynomial with respect to each of its coefficients [28]. Since the cost of a given polynomial is a function of its duration $T_{s,i}$, we must fix all T prior to optimization.

A.3 Constraints

The designed optimization problem is subject to equality constraints in terms of derivatives at the start and end of each segment.

The equality constraints imposed for each polynomial allow the endpoints to be pinned to known locations in space, or assign specific values of velocity, acceleration, jerk or snap[28]. Such constraints are useful to enforce, for example, that the multirotor start from rest at the beginning of a trajectory. The equality constraints of the polynomial optimization problem can be formulated as:

$$A_i \boldsymbol{c}_{i,d} = \boldsymbol{d}_{i,d}\,, \qquad \boldsymbol{A}_i = \begin{bmatrix} \boldsymbol{A}(t=0) \\ \boldsymbol{A}(t=T_{s,i}) \end{bmatrix}_i\,, \qquad \boldsymbol{d}_{i,d} = \begin{bmatrix} \boldsymbol{d}_{i,d}(t=0) \\ \boldsymbol{d}_{i,d}(t=T_{s,i}) \end{bmatrix}_i \tag{A.5}$$

Where $\boldsymbol{A}_i$ is a mapping matrix between coefficients, $\boldsymbol{c}_{i,d}$, and a vector containing the derivative values for the beginning and end of the i-th segment in a polynomial trajectory $\boldsymbol{d}_{i,d}$.

Note that both the Hessian matrix $\boldsymbol{H}_i$ and mapping matrix $\boldsymbol{A}_i$ only depend on the segment time $T_{s,i}$ and thus are constant overall dimensions for the segment, which allows for computation-time savings in the case of multiple dimensions [84].

Continuity constraints must be imposed to ensure the trajectories are continuous in between intervals. To do so, we must enforce continuity of the position and yaw angle, as well as of their first four and two derivatives respectively, by imposing equality between the end of the i-th segment derivatives and he beginning of the (i+1)-th segment [28]:

$$\boldsymbol{A}_i(t=T_{s,i})\boldsymbol{c}_{i,d} = \boldsymbol{A}_{i+1}(t=0)\boldsymbol{c}_{i+1,d} \tag{A.6}$$

$d_{i,d}$, H_i and A_i can now compiled into a single set of linear equality constraints for the joint optimization problem over the whole trajectory:

$$\begin{bmatrix} \mathbf{A}_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{A}_M \end{bmatrix} \begin{bmatrix} \mathbf{c}_{1,d} \\ \vdots \\ \mathbf{c}_{M,d} \end{bmatrix} = \begin{bmatrix} \mathbf{d}_{1,d} \\ \vdots \\ \mathbf{d}_{M,d} \end{bmatrix} \tag{A.7}$$

A.4 Unconstrained QP Reformulation

For solving this problem, we refer to Richter et al. [28] that showed how to convert the problem into an unconstrained QP, and its superior numerical stability compared to a constrained QP, which becomes ill-conditioned for more than several segments, polynomials of a high order, and when widely varying segment times are involved.

In their method, the QP is solved directly for endpoint derivatives as decision variables, rather than solving for polynomial coefficients. Once the optimal waypoint derivatives are found, the minimum-order polynomial connecting each pair of waypoints can be obtained by inverting the appropriate constraint matrix.

By merging equations (A.3), (A.4) and (A.7), we obtain a new formulation for the cost function of each polynomial trajectory:

$$J_{pol} = \begin{bmatrix} \boldsymbol{d}_{1,d} \\ \vdots \\ \boldsymbol{d}_{M,d} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{A}_1 & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & \boldsymbol{A}_M \end{bmatrix}^{-T} \begin{bmatrix} \boldsymbol{H}_1 & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & \boldsymbol{H}_M \end{bmatrix} \begin{bmatrix} \boldsymbol{A}_1 & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & \boldsymbol{A}_M \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{d}_{1,d} \\ \vdots \\ \boldsymbol{d}_{M,d} \end{bmatrix} \tag{A.8}$$

In order to compute the inverse of each mapping matrix $\boldsymbol{A}_i$, this implantation uses the matrix manipulation described in [84] that exploits the structure of the mapping matrix to allow the use of the Schur-Complement.

Then the new optimization variables $d_{i,d}$ are re-order such that fixed derivatives and the free derivatives are separately grouped. As shown in paper Richter et al. [28], after re-ordering the endpoint derivatives the cost function (A.8) can be written as a sum of individual cost representing both sets of optimization variables. This optimization function can be differentiated and equated to zero resulting in an expression for the optimal endpoint free derivatives as a function of the fixed derivatives and the cost matrix. This means that the optimum solution for this problem can be obtained algebraically without the need for any iterative optimization process.

The polynomials can then be recovered from individual evaluations of the appropriate constraint equations mapping derivatives back into the space of coefficients.

A.5 Time Optimization

The proposed unconstrained QP problem requires the use of fix time intervals $T_{s,i}$ for each polynomial since these times are needed to determine the cost matrix (A.8). However, one of the trajectory optimization goals is to minimize the segment times between each waypoint while respecting maximum velocity and acceleration constraints. For this end, we follow the formulation proposed in [28, 84] where an iterative non-linear optimum problem is solved to determine the desired polynomial paths.

In this formulation, the segment times $T_{s,i}$ are added as optimization variables and a reformulation of the cost function will now perform a trade-off between minimizing snap and total trajectory time. The objective function of the new optimization problem is as follows:

$$J = J_{pol} + k_T \Big(\sum_{i=1}^{M} T_{s,i} \Big)^2 \tag{A.9}$$

where k_T is a user-specified constant that weights the mentioned trade-off. The total set of optimization variables consists now of the free end-point derivatives from the unconstrained reformulation and the segment times $T_{s,1}, \ldots, T_{s,M}$.

A.6 State Inequality Constraints

After defining the optimization function (A.9) it is necessary to incorporate state limitations for the newly formulated non-linear problem.

In order to constraint the derivates inside each polynomial segment we require the definition of inequality constraints in parameters that do not belong to the set of optimization variables of the problem. The method presented in [84], tackles the problem of state limitations as guidelines instead of hard constraints. Therefore, they implemented those constraints as soft constraints by adding an additional cost term, that heavily penalizes states outside the boundary limits imposed:

$$J_{soft\ constraint} = exp(\frac{x_{max,actual} - x_{max}}{x_{max}\epsilon}k_s) \tag{A.10}$$

where $x_{max,actual}$ is the maximum of the norm of state x in the given 3D trajectory, ϵ defines how much the deviation from the state maximum is tolerated and k_s is a constant that allows to set how much the violation of a constraint is weighted. The norm of the velocity can be written in terms of the polynomials of position as:

$$v_{norm}(t) = \sqrt{(\dot{p}(t)_x)^2 + (\dot{p}(t)_y)^2 + (\dot{p}(t)_z)^2} \tag{A.11}$$

To find the candidates for the state maximum, we need to compute the derivative with respect to time and equal it to zero. As shown in [84] the real roots can be calculated with the numerically stable Jenkins-Traub algorithm allowing $v_{max,actual}$ to be determined. The same methodology of derivative constraints can be applied for higher-order derivatives allowing for the optimization of the desired trajectory.

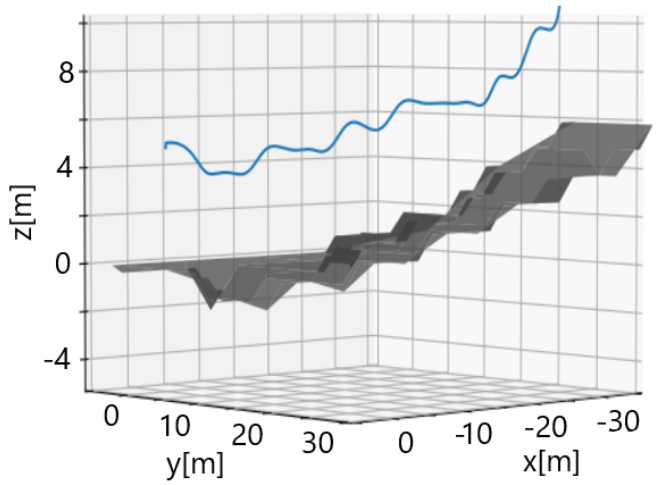
A.7 Limitations and Results

The presented formulation was only explored to solve the waypoint following and minimum time objective of this paper. The terrain-following goal was accounted for by adding extra waypoints every d_{max} meters in the straight line between the initial waypoints and setting the z coordinate of each point to be the desired altitude above ground. The trajectory is then sampled and if the altitude of any point is above or below certain limits an extra waypoint is added and that specific segment is re-optimized. This method is common in many commercial applications.

The following test was conducted on the same HP Laptop running Ubuntu 18.04 and equipped with an Intel Core i7-9750H CPU @2.60GHz and 16,00GB of RAM. These results illustrate a simple waypoint constrained trajectory with minimum time and snap objectives where four polynomials are optimized to wield a trajectory fully defined in x, y, z and yaw. The maximum limits for the path derivatives are represented in Figure A.1 and the final trajectory polynomials are shown in Figure A.2.

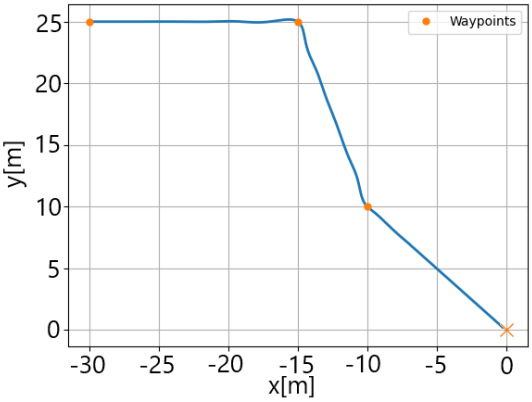
The utilized method does not properly include the altitude above the terrain as an optimization variable since it restricts the nodes' z coordinate to a specific value before the optimization is performed. However, the presented polynomial optimization can be further augmented in the problem formulation with new sets of inequality constraints or soft constraints to include the objective of terrain avoidance.

Some examples of those augmentations are described by Mellinger and Kumar [85] where corridor constraints are added to some trajectory segments by sampling the polynomial for a specific time interval and then constraining those sampling points to be inside the desired tunnel. However, this formulation

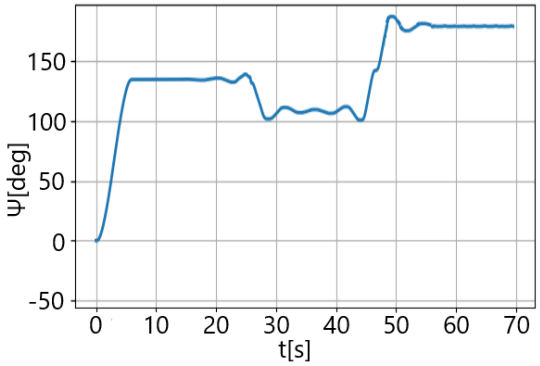


Polynomial Limits	
h_{min}	4.0 $[m]$
h_{max}	6.0 $[m]$
d_{max}	2.0 $[m]$
v_{max}	1.0 $[m/s]$
a_{max}	2.0 $[m/s^2]$
$\dot{\psi}_{max}$	$\frac{\pi}{4}$ $[rad/s]$
$\ddot{\psi}_{max}$	$\frac{\pi}{2}$ $[rad/s^2]$

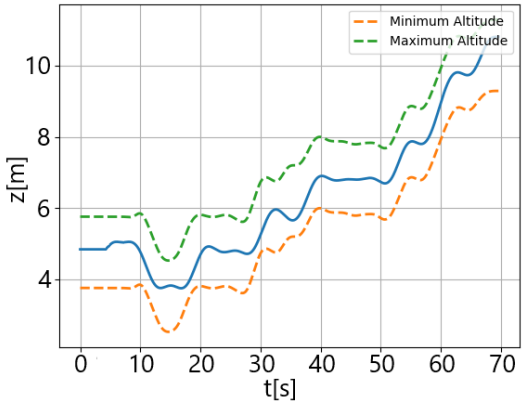
Figure A.1: 3D polynomial trajectory and optimization limits.



(a) Top-down view of the trajectory.



(b) Yaw polynomial.



(c) Altitude polynomial.

Figure A.2: Optimized Polynomial trajectory.

is not very efficient due to the non-convexity inequality constraints of the sampling points. Another formulation presented by Oleynikova et al. [86] considers a grid-like three-dimensional space where each cell can either be occupied or not. In this paper, the path is sampled based on the resolution of the

grid and an extra smooth collision cost is added to the optimization function to prevents the polynomial trajectory to be inside an occupied cell.

However, the characteristics of the terrain surface and the difficulty to represent it as sets of constraints or cost terms for the polynomial segments of the trajectory coupled with the inability to directly constraint UAV states and inputs such as pitch, roll or thrust makes this formulation, although considerably faster, not adequate for the problem of this dissertation.