UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

BACHELOR FINAL THESIS

Real-Time Optimal Trajectory Generation for Fixed-Wing UAVs in Firefighting Missions via Numerical Integration and Constrained Optimization

Document:

Report

Author:

Abimael Campillo Simón

Director/Co-director:

Prof. Dr. Alex Ferrer Ferre / Miguel Pareja Muñoz

Degree:

Bachelor in Aerospace Technology Engineering

Examination session:

Autumn 2025

"The airplane has unveiled for us the true face of the earth. No longer confined to roads that bend around obstacles and follow the will of cities and rivers, the pilot looks down on mountains, deserts, and oceans as a single, living map. To fly is to discover that the world is wider, harsher, and more beautiful than the paths laid out for us on the ground."

— ***Antoine de Saint-Exupéry***, *Wind, Sand and Stars.*

Abstract

Given the increase in the frequency and intensity of wildfires, the necessity for autonomous aerial firefighting solutions has become increasingly urgent. Within this context, this final bachelor's degree thesis aims to develop the foundations of an optimal control algorithm capable of generating trajectories for firefighting manoeuvres. The methodological approach combines Optimal Control Theory (OCT) with a Direct Collocation (DC) implementation in Non-linear Programming (NLP) using the CasADi framework. Results indicate that while the system is robust to instantaneous disturbances, a clear trade-off exists between manoeuvre duration and smoothness; specifically, end-time penalisation accelerates the mission but induces aggressive, spiky control actions. Optimal performance was achieved by easing constraint margins, thereby increasing the KKT optimality margin. Conversely, schemes lacking a balanced cost functional led to unstable bang-bang oscillations or non-optimal feasible solutions. While promising, the current solution requires further integration of real-world environmental factors, such as Data Elevation Models (DEM), No-Fly Zones (NFZ), and real-time atmospheric conditions.

Keywords: Optimal Control Theory, OCT, Direct Collocation, DCM, Non-linear Programming, NLP, CasADi, Firefighting Manoeuvres, Flight Trajectory Generation, Path Planning

Resum

Davant l'augment de la freqüència i la intensitat dels incendis forestals, la necessitat de solucions aèries autònomes per a l'extinció d'incendis s'ha tornat cada vegada més urgent. En aquest context, aquest treball de final de grau té com a objectiu desenvolupar les bases d'un algorisme de control òptim capaç de generar trajectòries per a maniobres d'extinció d'incendis. L'enfocament metodològic combina la teoria de control òptim OCT amb una implementació de col · locació directa DC en programació no lineal NLP utilitzant l'entorn CasADi. Els resultats indiquen que, tot i que el sistema és robust davant pertorbacions instantànies, existeix un compromís clar entre la durada de la maniobra i la seva suavitat; concretament, la penalització del temps final accelera la missió però indueix accions de control agressives i irregulars. L'optimitat màxima s'ha assolit relaxant els marges de les restriccions, augmentant així el marge d'optimitat KKT. Per contra, els esquemes sense un funcional de cost equilibrat han derivat en oscil · lacions inestables de tipus bang-bang o solucions factibles no òptimes. Tot i ser prometedora, la solució actual requereix una major integració de factors ambientals reals, com ara models digitals d'elevació DEM, zones de vol prohibit NFZ i condicions atmosfèriques en temps real.

Paraules clau: Control Òptim, OCT, Col · locació Directa, DCM, Programació No Lineal, NLP, CasADi, Extinció d'Incendis, Generació de Trajectòries de Vol, Planificació de Rutes

Acknoledgments

Firstly, I would like to thank my bachelor's degree thesis director, Alex Ferre, for his guidance, patience and dedication through all this project development. I would also thank him for introducing me such a passionate world as flight trajectory optimisation.

In second place, I would like to show gratitude to Singular Aircraft and all its participants for the opportunity to work on a project of this magnitude and to be able to contribute to creating a better world. Specially, I would to appreciate to Miguel Pareja for his invaluable comments and observations, as well as all the lessons and experiences I have had. To Pol Padilla, I would also thank his observations and deep knowledge in the matter. Guys, this is only the beginning.

A mis amigos, me gustaría agradaecerles su apoyo incondicional todos estos años, así como todos los momentos de alegría, las risas, las vivencias y todas las experiencias que hemos compartido. A mi familia, y en especial a ti mamá, gracias por estar siempre ahí y tener paciencia mientras cumplía uno de mis sueños. Os quiero.

Preface

This preface chapter includes a brief description of the scope of the project followed by the motivation that has led all the development. Then, the methodological approach is summarised, eventually followed by the report's structure.

Scope and Objectives

The scope of this project will include:

- Comprehensive review of trajectory optimisation problems and the approach solutions to the matter, identificating the limitations and research gaps.
- A theoretical comprehension and explanation of optimal control theory and restricted vertical flight trajectories modelisation.
- A mathematical formulation of firefighting manoeuvres under varying mass and heavy operational constraints.
- NLP implementation and solution of the formulated problem above.
- A specialised water-drop trajectory planner algorithm.

Therefore, this project will not include:

- Physical design or manufacturing of the Unmanned Aerial Vehicle (UAV) and its payloads.
- Real-world flight testing or hardware tests.
- Modification of low-level autopilot firmware or inner-loop control laws.
- High-fidelity fluid modelling of the water-drop print or fire plume behaviour.
- Formal certification -e.g. RTCA DO-178C- or cybersecurity regulatory assessment.
- Full migration of the Python codebase to C/C++ or embedded deployment.

Eventually, the main objective of this thesis is to develop an algorithm capable of computing the optimal flight trajectory for a UAV operating in firefighting missions. The

algorithm is designed to generate restricted trajectories within a vertical plane while considering multiple operational constraints. Specifically, the algorithm aims to determine the descent path during the water-drop manoeuvre and the subsequent ascent trajectory, ensuring feasability and safety. This thesis and project has been developed in collaboration with Singular Aircraft, as part of an extracurricular internship agreement along with other tasks.

Motivation and Justification

In recent years, the increasing occurrence and intensity of wildfires have highlighted the urgent need for more effective and autonomous aerial firefighting solutions [1]. From a global perspective, enhancing the autonomy and operational efficiency of Unmanned Aerial System (UAS) for emergency response represents a major step toward safer and more sustainable fire suppression operations. In particular, fixed-wing UAVs offer significant advantages over rotary-wing systems, including greater range, endurance, and payload capacity, making them suitable for long-duration firefighting missions in hazardous environments [2].

This project is developed within the framework of an extracurricular internship at Singular Aircraft, an aerospace company founded by Luis Carrillo with over twelve years of experience in the design, manufacture, operation, and certification of Remotely Piloted Aircraft (RPA) and UAV. The company has designed and fully developed the Flyox, currently the largest civilian fixed-wing UAV designed for firefighting, humanitarian aid, and surveillance missions [3]. Several previous projects and studies carried out within the company have focused on the improvement of the Flyox's flight systems, payload integration, and mission control capabilities. Building upon those developments, this thesis contributes to the new generation of the Flyox platform, which aims to introduce advanced levels of autonomy for critical mission phases.

Specifically, this new generation seeks to enable the aircraft to perform the water-drop manoeuvre autonomously, reducing operator workload and enabling missions under low-visibility or high-risk conditions. To achieve this, the aircraft must be equipped with a trajectory computation algorithm capable of planning and re-planning the discharge trajectory in real time, considering aircraft dynamics, environmental constraints, and safety margins, without human validation. This capability represents a fundamental step toward full operational autonomy in firefighting UAS.

The current work focuses on developing such an algorithm through optimal control and numerical optimisation techniques. The approach aims to fill existing gaps in trajectory generation software for UAS by integrating non-standard operational constraints such as DEM, NFZ, atmospheric conditions, and aircraft operational limits. Although not all these constraints will be implemented in the present study, they define the broader framework in which this research is embedded.

From a critical standpoint, the proposed approach offers clear advantages: it enables a

higher degree of autonomy, improved safety by reducing human involvement in dangerous environments, and adaptability to real-time mission changes. However, potential limitations include the computational cost of real-time optimisation, the need for accurate environmental data, and the complexity of integrating such systems into certified aircraft architectures. Despite these challenges, this research represents an essential step toward achieving the next level of autonomy in firefighting UAS operations.

From my viewpoint as the author of the thesis and the project, I was thrilled to be able to contribute to a real project, such as Singular Aircraft and the guidance of manoeuvres in the Flyox UAV, as well as to learn about a subject that was new to me and develop my skills in it.

Methodological Approach

The methodological framework employed to solve the firefighting trajectory optimisation problem has been a combination of OCT and NLP. Specifically, the physical system has been modelled through differential-algebraic equations, which are subsequently transformed into NLP problem via the direct transcription method and a trapezoidal integration scheme. Then, the NLP have been implemented and solved using a CasADi framework, including the definition of objective functions, constraint handling, and solver configurations.

Document Structure

This document is structured in seven (7) different chapters. Firstly, in the State-of-the-art chapter, some research has been done in matters of trajectory optimisation problems and solutions until actual limitations and research gaps have been identified. Then, the second chapter introduces the reader to OCT through characteristic equations and definitions until reaching the NLP implementation using CasADi, a dedicated optimisation library. Third chapter contains a level-cruise flight trajectory defined as an Optimal Control Problem (OCP) and solved using NLP implementation. The chapter describes and solves two different problems, one without and one with the free-end condition. In fourth place, the whole firefighting manoeuvre has been described and modelled using Optimal Control (OC) notation. Then, some simulations have been done and the results have been presented on Chapter 5. In sixth place, a budget and enviromental impact analysis of this project has been included. Eventually, the main contributions have been summarised on Conlusions dedicated chapter. The bibliography and all appendices have been added right after the concluding remarks.

Contents

List of Figures

List of Tables

Acronyms

ASL Above Sea Level.

CONOPS CONceptual OPerationS.
COTS Commercial-of-the-Shelf.

DC Direct Collocation.
DCM Direct Collocation Method.
DEM Data Elevation Models.
DOF Degrees-of-Freedom.

EP Exit Point.

IPOPT Interior Point OPTimizer.

KKT Karush-Kuhn-Tucker.

NFZ No-Fly Zones.
NLP Non-linear Programming.

OC Optimal Control.
OCP Optimal Control Problem.
OCT Optimal Control Theory.
ODE Ordinary Differential Equation.

PDE Partial Differential Equation.
PMP Pontryagin's Maximum Principle.

RPA Remotely Piloted Aircraft.

SP Starting Point.
STG STaGe.

TP Target Point.
TPS Throttle Position Sensor.

UAS Unmanned Aerial System.
UAV Unmanned Aerial Vehicle.

Nomenclature

Symbol	Description	Units
Note: Bold symbols denote vectors; dotted symbols denote time-derivatives.		
AR	Aspect Ratio of the aircraft -computed as $AR = \frac{b^2}{S}$-	$[-]$
BEM	Basic Empty Mass	$[kg]$
C_{D_0}, k	Drag coefficients: parasitic drag and induced drag factor -computed as $k = \frac{1}{\pi ARe}$-	$[-]$
$C_{L_0}, C_{L_\alpha}, C_{L_{\delta e}}$	Lift coefficients: zero-alpha lift, lift-curve slope, and elevator lift contribution	$[-], [rad^{-1}], [rad^{-1}]$
$C_{m_0}, C_{m_\alpha}, C_{m_{\delta e}}$	Pitching moment coefficients: static stability and elevator moment contribution	$[-], [rad^{-1}], [rad^{-1}]$
D	Drag force, projected into negative x-wind axis (x_w)	$[N]$
D_p	Propeller's diameter	$[m]$
FM	Fuel Mass	$[kg]$
$F_b(x_b, y_b, z_b)$	Body-fixed reference frame with origin at the CG, x_b along the fuselage, y_b along right semi-wing and z_b pointing down	–
$F_h(x_h, y_h, z_h)$	Local-horizon reference frame (North-East-Down)	–
$F_w(x_w, y_w, z_w)$	Wind reference frame, x_w aligned with the aerodynamic velocity vector, y_w perpendicularly pointing rightwards and z_w perpendicularly pointing down	–
I_y	Moment of inertia about the y-body axis (y_b)	$[kg \cdot m^2]$
L	Lift force, projected into negative z-wind axis (z_w)	$[N]$
$MTOM$	Maximum Take-Off Mass	$[kg]$
PM	Payload Mass	$[kg]$

Symbol	Description	Units
Note: Bold symbols denote vectors; dotted symbols denote time-derivatives.		
RPM	Indicates the propeller RPMs for thrust computations	$[rpm]$
S	Aircraft wing's surface	$[m^2]$
SFC	Specific Fuel Consumption	$[kg/s]$
T	Thrust force, projected into positive x-body axis (x_w)	$[N]$
TOM	Take-Off Mass	$[kg]$
V_{tp}	Target speed for level-flight cruise trajectory or water-discharge level-flight stage	$[m/s]$
W	Weight force, force projected into positive z-local-horizon axis (z_h)	$[N]$
$[CG_x, CG_z]$	Centre of Gravity (CG) defined from wing's leading edge in body reference frame; point where weight force is applied	$[m]$
$[CP_x, CP_z]$	Centre of Pressure (CP) defined from wing's leading edge in body reference frame; point where aerodynamic forces are applied	$[m]$
$[CT_x, CT_z]$	Centre of Thrust (CT) defined from wing's leading edge in body reference frame; point where thrust force is applied	$[m]$
$[C_{T_0}, C_{T_1}, C_{T_2}, C_{T_3}]$	Thrust coefficients, to compute thrust $T(J)$ and torque $Q(J)$ as functions of the advance ratio J	$[-]$
$[C_{\eta_0}, C_{\eta_1}]$	Thrust coefficients, to compute propeller's efficiency $\eta(J)$ as a function of the advance ratio J	$[-]$
$\Delta X_{CT}, \Delta Z_{CT}$	Horizontal and vertical distances from CT to the CG	$[m]$
α	Angle-of-Attack, angle between the aircraft's x-body axis (x_b) and the projection of the aerodynamic velocity vector onto the aircraft's plane of symmetry ($x_b - z_b$) plane	$[rad]$
$\bar{c}$	Mean Aerodynamic Chord of aircraft's wing	$[m]$
$\bar{q}$	Dynamic pressure -computed as $\bar{q} = \frac{1}{2}\rho V_a^2$ where ρ is air's density and V_a is the aerodynamic velocity-	$[Pa]$
δ_{FLAPS}	Flaps deflection, fixed value during the mission	$[rad]$
δ_{TPS}	Throttle position sensor, variable associated to TPS	$[rad]$

Symbol	Description	Units
Note: Bold symbols denote vectors; dotted symbols denote time-derivatives.		
δ_e	Elevator deflection, trailing-edge down is defined as positive	$[rad]$
γ	Flight path angle, angle between the aerodynamic velocity vector (x_w) and the horizontal plane ($x_h - y_h$)	$[rad]$
$\mathbf{V}_w$	Bidimensional wind field vector, defined by horizontal (V_{x_w}) and vertical (V_{z_w}) components in the local-horizon frame	$[m/s]$
θ	Pitch angle, angle between x-body axis (x_b) and the horizontal plane ($x_h - y_h$)	$[rad]$
ε, ε_0	Thrust angle-of-attack and its initial value; they are the angle between aerodynamic velocity and the angle between thrust vector and x-body axis (x_b), repectively	$[rad]$
b	Span of the aircaft's wing	$[m]$
h_{ref}	Reference altitude for level-flight cruise trajectory or water-discharge level-flight stage	$[m]$
m	Aircraft's mass	$[kg]$
q	Pitch rate, angular velocity about the y-body axis (y_b)	$[rad/s]$
u, w	Velocity components in the body-fixed frame (x_b, z_b)	$[m/s]$

1 | State-of-the-Art

This first chapter provides a general overview of the current state-of-the-art in trajectory optimisation frameworks and solutions, ranging from industrial and commercial software to academic-oriented tools. The chapter concludes with a brief summary of identified limitations and the primary research gaps within the current trajectory optimisation landscape.

1.1 Overview of Frameworks and Solutions

Trajectories are time-continuous functions that describe the evolution and behaviour of a system from an initial state until it reaches a specific objective. This objective is typically defined by a cost function within an optimisation framework. The following discussion explores various solutions for trajectory optimisation problems, with a specific focus on aerospace applications.

1.1.1 Commercial-of-the-shelf (COTS) solutions

Commercial-of-the-Shelf (COTS) refers to commercially developed tools widely used in the industry, accessible via licensing fees. While some of these tools originated as open-source or research projects, their reliability led to commercialization. In this context, the most prominent COTS tools for trajectory optimization are NASA OTIS4 and ASTOS.

NASA OTIS4 was initially developed for missile trajectories in 1986 by the United States Air Force (USAF). It is a robust framework that supports both 3-DOF and 6-DOF modeling, allowing users to solve for the complete state of an aircraft -including position and attitude- or a reduced point-mass version. The framework is designed to handle highly constrained problems or those with sparse initial data. It features four distinct solution modes:

- Discrete Mode: Used for simple integration of the equations of motion given a fixed control history (simulation only).
- Target Mode: Adjusts a set of free parameters to satisfy specific terminal conditions or goals (boundary value problem).

- Optimal Constrained Mode: Minimises or maximises a cost function while strictly satisfying path and terminal constraints using NLP.
- Optimal Open-Loop Mode: Solves for the optimal control history without feedback, typically used for generating baseline reference trajectories.

OTIS4 relies on Direct Collocation Method (DCM), pseudospectral methods, and implicit integration, interfaced with solvers such as SLSQP, SNOPT, or NPOPT. However, it remains limited regarding the integration of dynamic DEM or complex NFZ in real-time. Furthermore, it is characterised by a high computational demand [4] [5].

ASTOS is perhaps the most widely recognised tool for launch vehicle trajectory computation. It provides a comprehensive environment for modeling all flight phases, from liftoff to re-entry. Developed by the DLR in 1989 and written in C, it utilises DCM and multiple shooting methods, such as Sequential Quadratic Programming (SQP) and general NLP solvers. Despite its high fidelity and versatility, ASTOS shares the limitations of high computational overhead and lack of suitability for real-time onboard computation [6].

1.1.2 Open-source frameworks

Open-source frameworks have gained significant traction in the research community due to their flexibility, transparency, and the ability to be integrated into custom software pipelines without licensing constraints. Unlike COTS solutions, these tools often allow for deeper modification of the underlying numerical methods. Two examples are PSOPT and TAOS.

PSOPT is an open-source, C++ and Python-based optimal control library developed in 2010 by Victor M. Becerra. It was designed to solve a wide range of multi-phase engineering problems by approximating continuous-time optimal control problems into finite-dimensional NLP problems. It is distinguished by its versatility, offering three primary numerical approaches:

- DCM: Solving the state and control variables simultaneously at discrete points.
- Local Discretisation Methods: Utilising implicit integration schemes, such as the Runge-Kutta, for high-accuracy local approximations.
- Pseudospectral Methods: Employing global orthogonal polynomials, specifically Chebyshev and Legendre approximations, to achieve exponential convergence rates for smooth problems.

Despite its robust mathematical core, PSOPT remains limited in modern operational contexts; it lacks native support for DEM integration or complex NFZ constraints, and its architecture is not optimized for real-time on-board applications [7].

TAOS is a specialiSed framework designed to handle a spectrum of flight regimes, from low-speed atmospheric vehicles to high-speed orbital trajectories. The core philosophy of TAOS is based on Newtonian mechanics: it estimates system performance by calcu-

lating the position, velocity, and acceleration through a segmentation method. In this approach, the trajectory is divided into discrete segments where the equations of motion are integrated. To achieve optimisation, TAOS typically employs Powell's Method, a derivative-free optimisation algorithm used to find local minima of the cost function. While efficient for specific mission profiles, TAOS shares the same limitations as earlier frameworks regarding the lack of integrated DEM and NFZ data handling, which restricts its use in complex terrain-avoidance scenarios [8].

1.1.3 Academic and research oriented tools

Academic tools are often at the forefront of algorithmic innovation, focusing on computational efficiency, real-time feasibility, and the implementation of advanced mathematical theories. In this context, ACADO Toolkit and OpenAp.opt are two (2) great examples of academic tools.

ACADO Toolkit, developed in C++ in 2011, is a comprehensive software library designed for solving OCP in non-linear dynamic systems. It is specifically engineered for versatility and efficiency, making it a primary choice for embedded systems and industrial engineering applications. ACADO supports a wide array of problems, including non-linear optimal control, multi-objective optimization, and state/parameter estimation. Its core strength lies in its ability to handle both Model Predictive COntrol (MPC) and Non-linear Model Predictive Control (NMPC). The underlying algorithms utilize DCM, single shooting, and multiple shooting methods. While ACADO does not natively support DEM or NFZ integration, it is one of the few frameworks explicitly prepared for real-time applications. However, it is noted in the community for having a very steep learning curve, requiring significant expertise in C++ and numerical optimisation [9].

OpenAp.opt is a specialized Python-based tool introduced in 2019 as part of the OpenAP ecosystem for aviation purposes. It provides a complete environment for aircraft performance modeling, the calculation of key aerodynamic parameters, and trajectory generation and optimization. The tool solves non-linear OCP through DCM and fourth-order Runge-Kutta integration. It leverages the CasADi framework and the Interior Point OPTimizer (IPOPT) solver to handle large-scale non-linear programming. Unlike ACADO, OpenAP.opt is designed for strategic planning and high-fidelity performance analysis rather than tactical maneuvers; consequently, real-time computation is not supported [10].

1.2 Research Gaps

The analysis identifies a clear void in the state-of-the-art: the integration of Digital Elevation Models (DEM) and No-Fly Zones (NFZ) within a real-time, adaptable environment. To address this gap, this research adopts CasADi -via Python- as a flexible baseline. This choice provides the necessary mathematical foundation -done in OCT- and algorithmic differentiation required to develop a baseline algorithm that will serve as the starting

point for future real-time, terrain-aware developments, which are necessary for autonomous firefighting missions.

2 | Fundamentals of Optimal Control and its Implementation

This chapter includes the mathematical formualtion of the OCP, which is the base of the OCT. It also includes a careful explanation about direct and indirect methods, which are two different approaches to solve optimal control problems, highlighting the DCM and Pontryagin's Maximum Principle (PMP). Eventually, the IPOPT method is presented as the used approach, including the mathematical formulation of the method and its implementation from CasADi's optimisation library.

2.1 Introduction to Optimal Control Theory

The OCT is a control field related branch where an objective function has to be optimised -most of the times, it has to be minimised- in order to find the control trajectory for a determined dynamical system. This theory is historically realted with calculus of variations, where optimal points -either maxima or minima- are found using little variations in functions or functionals [11]. This field of study, proposed by Isaac Newton, was developed as a solving approach for the brachistochrone problem, posed by Bernoulli in 1696. The brachistochrone curve is defined as the fastest descent path -most optimal path for minimising time- between two points A and B under a uniform gravitational field. Counterintuitively, it was found that the most optimal path was not but the cycloidal ramp, as can be seen in the image below.

In essence, the OCT responds the need to solve continuous time optimisation problems, as the brachistochrone one. Thus, the OCP can be understood as a n-dimensional extension of the NLP problem. The NLP problem can be defined as the minimisation of a given function subject to a different set of equations or inequations that act as restrictions -also called constraints- and simple bounds [12]. See its definition in expression (2.1).

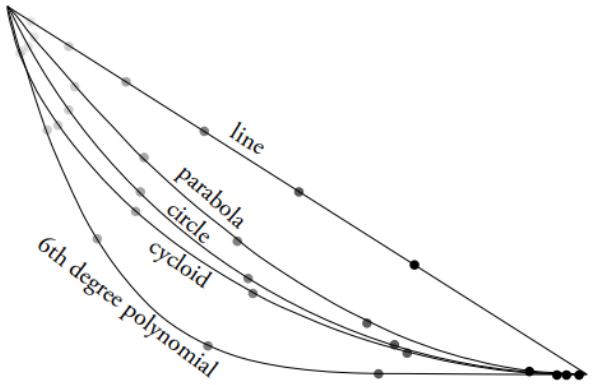


Figure 2.1: The brachistochrone problem. The ball rolls down the fastest on a cycloidal ramp. Points of the same shade correspond to the same moment in time. Extracted directly from [13].

$$\begin{aligned} \max_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g_j(x) \leq b_j \text{ for each } j = 1, \ldots, m \\ & x = (x_1, \ldots, x_n) \\ & g = (g_1, \ldots, g_m). \end{aligned} \tag{2.1}$$

The NLP problem is limited to a finite number of state variables and restrictions, since it has a discretised treatment. At this point, it has to be mentioned that any NLP problem goes through three different phases, which are applicable to the OCP or any optimisation problem. The first phase is modelling, which is, in fact, a mathematical description of a dynamic system through simple equations that allow future state predictions of a system thanks to a given set of control variables. While state variables $y_i(t)$ are the variables that describe system conditions at a given time, control variables $u_i(t)$ are the variables that can be manipulated through time because they have an influence on the system. In other words, state variables are consequence of the control variables -p.e for a flight trajectory, the position and velocity of the aircraft would be the state variables for a set of different control variables as could be the α or the TPS-. Secondly, the restrictions have to be applied to the system in the form of equality or inequality constraints to describe its physical or operational limitations. In addition, the admissible trajectory and admissible control are described, what means that state and control variables satisfy the restrictions within all time domain, respectively -later will be referred to as simple bounds-. Third and last phase is optimisation, which means the maximisation or minimisation of the cost function or functional associated with the system while complying with all the constraints and bounds set for a determined optimisation criteria. For example, in the brachistochrone problem introduced before, there are several feasible paths between points A and B, but only the cycloidal path complies with the minimum time optimisation criteria [14].

It has to be pinpointed that the OCP is a n-dimensional interpretation of the NLP because

state and control variables are time-continuous functions and, therefore, the problem's objective is to find the n_k-dimensional vector of state functions $y_k(t)$ and control functions $u_k(t)$ that optimise a determined functional $F(y, u)$.

2.1.1 Mathematical formulation

The OCP is defined as follows.

$$\begin{aligned} \min_{y,u} \quad & J(y,u) \coloneqq \Phi[y(t_f), u(t_f), t_f] + \int_{t_0}^{t_f} L[y(t), u(t), t]\ dt \\ \text{s.t.} \quad & \dot{y} = f[y(t), u(t), t] \\ & g[y(t), u(t), t] \leq 0 \\ & \varphi[y(t_0), u(t_0), t_0] = 0 \\ & y_{lb} \leq y(t) \leq y_{ub} \\ & u_{lb} \leq u(t) \leq u_{ub} \\ & t \in [t_0, t_f] \end{aligned} \tag{2.2}$$

The previous description is the formal description for the OCP, since it contains a cost functional $J(y, u)$ that has to be minimised subject to a set of constraints and simple bounds [15][16]. Specifically, the description above has the following terms:

- State variables $y(t)$: As explained before, state variables are time-continuous functions that describe the system condition and allow to predict its future coindition. This state variables are limited by lower and upper bounds, y_{lb} and y_{ub}, respectively. Those bounds are the physical or operational limits that each state variable has; for example, on the brachistochrone problem, the horizontal position of the ball -when movement is restricted to the vertical plane- cannot be lower than the point A horizontal coordinate and bigger than the point B horizontal coordinate -those are the simple bounds for horizontal position state variable in that case-.

- Control variables $u(t)$: Control variables are time-continuous functions that describe the changes on the system to achieve a determined state or condition through time. Similarly, they are simple bounded, in this case by lower bound u_{lb} and upper bound u_{ub}; for example, for the flight trajectory previously described, control variable α could be lower bounded by a null value and upper bounded by the stall angle, α_{stall} -those are the simple bounds for the α control variable in that case-.

- Time domain t: Time domain is the closed interval of time, defined on real numbers, between initial time t_0 -where initial state condition is applied- and the time horizon t_f -that can be fixed or free-. On OCP, time horizon can be set to a known value if its a problem's parameter or can be free if it is set as a decision variable -also called control variable- of the problem through end-time constraints or inside the cost functional.

- Dynamic constraints $\dot{y}$: Dynamic equations are a set of Ordinary Differential Equa-

tion (ODE)s defined as equality constraints that describe the physics of the problem. This set of constraints are always active, which means that they apply for all time domain.

- Path constraints $g[y(t), u(t), t]$: Path contsraints are another set of expressions that apply for a given time domain and that could refer to any type of extra restrcitions to the problem. They could be equalities or inequalities that could both restrict state or controls in a determined way.

- Initial state conditions $\varphi[y(t_0), u(t_0), t_0]$: The initial state constraints are a set of equalities that restrict the initial value of state variables to a known value -the initial condition of the system is known-. They can also include restrictions for the initial value of control variables; nevertheless, it is not a common practice because the problem could be overrestricted and the solver may be unable to found a feasible solution.

- Cost functional $J(y, u)$: The cost functional is the cost objective function that has to be minimised to found the optimal solution. This cost is defined by two different terms that contribute differently to the whole objective: terminal and running costs. If both contributions are present, then the cost objective is defined as in its Bolza form. As clarified in [17]:

  - Terminal cost is the contribution that only penalises the final value of state variables. Is represented by $\Phi[y(t_f), u(t_f), t_f]$ and, if cost objective $J(y, u) = \Phi[y(t_f), u(t_f), t_f]$, it is said that the whole cost functional is in its Mayer form.

  - Running cost is the contribution that penalises each state and control variable trough time and, therefore, accumulates the whole penalisation. Is represented by $\int_{t_0}^{t_f} L[y(t), u(t), t]\ dt$ and, if cost objective $J(y, u) = \int_{t_0}^{t_f} L[y(t), u(t), t]\ dt$, it is said that the whole cost functional is in its Lagrange form.

2.2 Direct and Indirect Methods

Once the basic problem has been set, is has to be noticed that although the variables are time-continuous, they have to be discretised in order to solve the problem. This discretisation is necessary because the problem is not affordable analytically and it has to be treated numerically. This implies a discretisation of the time domain, converting the infinite-dimensional treatment inherited from the running cost into a sum of different time-instants. There are two different types of approaches, as stated by John T. Betts on [15]:

- Direct methods, which are based in the evaluation of the images from a function and the comparison between different values at different instants to find the derivative -p.e Direct collocation method-.

- Indirect methods, which are based on direct evaluation of the derivative and its proximity to zero value. In this case, there is need of explicit derivation of the equations and is a method subjected to initial conditions -p.e Pontryagin's maximum principle-.

2.2.1 Direct collocation method as direct method

Direct collocation methods are a type of direct method in which the functions are discretised in a set of predefined points called collocation points. These collocation points are part of a collocation grid that is set along the domain in both state and control variables. It has to be pinpointed that between collocation methods two different types can be distinguished: low-order collocation methods, applied to solve ODEs, and pseudospectral methods, applied to solve Partial Differential Equation (PDE)s. Since this bachelor thesis adresses the flight mechanics of an aircraft for low-flight manoeuvres, which dynamic equations consists on different ODEs, the wording will focus on low-order collocation methods (DCM from now on) [18].

Essentially, in the DCM the state and the control variables are both discretised in different collocation points. This collocation points are a set of N time-instants that form the collocation grid (in an equivalent way, the reader can think about the collocation grid as the element grid usually made for finite element purposes). On these points, the dynamic of the problem will be evaluated, or forced, to solve the equations and, therefore, the whole problem. Thus, the DCM for the control variables is defined as follows, extracted from [19].

$$u(t) = \begin{cases} U_u^k(u^k, u^{k+1}, t) & \forall t \in [t^k, t^{k+1}] \\ U_u^{N-1}(u^{N-1}, u^N, t) & \forall t \in [t^{N-1}, t^N] \end{cases} \tag{2.3}$$

$$\bar{u} = [u^0, u^1, u^2, \ldots, u^N]^T \in \mathbb{R}^{(N+1)\cdot n_u} \tag{2.4}$$

The previous expressions (2.3) and (2.4) are the time-domain discretisation in N subintervals with specific nodes t^i with $i \in [0, N]$, what allows to obatin a decision vector for control. In other words, it is an interpolation that avoids sudden discontinuities that can generate inestabilities on the integrator or solver, allowing a more precise capture of the different variations. In an equivalent way, the discretisation for state variables can be obtained as:

$$\bar{y} \;=\; [y^0, y^1, y^2, \ldots, y^N]^T \in \mathbb{R}^{(N+1)\cdot n_y} \tag{2.5}$$

For a better comprehension, the mathematical formulation behind discretisation can be simplified to the following scheme, which exemplifies the conversion of a time-continuous function to its discretised form.

$$u(t) \longrightarrow u^i = u(t^i) \quad \text{for each } t^i \text{ with } i \in [0, \dots, N]$$
$$y(t) \longrightarrow y^i = y(t^i) \quad \text{for each } t^i \text{ with } i \in [0, \dots, N]$$

At this point, a trapezoidal scheme is proposed to be used on the numerical approach since it takes into account not only the present time-instant but the following time-instant for the function evaulation and its integration. This numerical integration scheme is preferred above others such as Runge-Kutta or Hermite-Simpson because of its simplicity, its easy applicability and its precision, which is enough for the scope of this project. The trapezoidal integration scheme for the running cost, adapted from [18] and [20], can be state as:

$$\int_{t_0}^{t_f} L(y,u,t)\ dt \approx \sum_{k=0}^{N-1} \frac{1}{2} h^k \cdot (L^k + L^{k+1})$$
$$\textit{where}$$
$$h^k := t^{k+1} - t^k \tag{2.6}$$
$$L^k := L[y(t^k), u(t^k), t^k]$$
$$L^{k+1} := L[y(t^{k+1}), u(t^{k+1}), t^{k+1}]$$
$$k \in [0, N-1]$$

Furthermore, the trapezoidal rule has to be applied to the constraints of the problem, now called collocation constraints. For the simple bounds, path cosntraints and initial state constraints, the application of the discretisation scheme is easy to reach because the equations have to be evaluated at the given collocation points directly, without need of trapezoidal rule. About the dynamic equations, which are the only ones that have to be integrated by applying the fundamental theorem of calculus, the following development can be obtained, extracted from [20].

$$\dot{y} := f(y,u,t)$$
$$\int_{t^k}^{t^{k+1}} \dot{y}\ dt = \int_{t^k}^{t^{k+1}} f(y,u,t)\ dt$$

Resulting in:

$$y^{k+1} - y^k = \frac{1}{2} h^k \cdot (f^k + f^{k+1})$$
$$\textit{where} \tag{2.7}$$
$$f^k := f(y^k, u^k, t^k) \text{ for each } k \in [0, N-1]$$

Eventually, the terminal cost can be added taking into account its contribution on final time-instant only. Thus, the optimal control problem can be written in its discretised form in a way such that could be solved applying a DCM.

$$\begin{aligned}
\min_{y,u} \quad & J(y,u) \coloneqq \Phi[y(t^N), u(t^N), t^N] + \sum_{k=0}^{N-1} \frac{1}{2} h^k \cdot (L^k + L^{k+1}) \\
\text{s.t.} \quad & y^{k+1} = y^k + \frac{1}{2} h^k \cdot (f^k + f^{k+1}) \text{ for } k \in [0, N-1] \\
& g[y(t^k), u(t^k), t^k] \leq 0 \text{ for } \forall k \\
& \varphi[y(t^0), u(t^0), t^0] = 0 \\
& y_{lb} \leq y(t^k) \leq y_{ub} \\
& u_{lb} \leq u(t^k) \leq u_{ub} \\
& t^k \text{ for each } k \in [0, N]
\end{aligned} \tag{2.8}$$

2.2.2 Pontryagin's maximum principle as indirect method

As said before, the indirect methods search a solution for the optimality conditions directly. In this context, they doesn't need to discretise the control variables but they will need an approximation; thus, they are subject to the initial conditions. Moreover, since the indirect methods are based on the calculus of variations, they also need to comply a set of necessary conditions called transversality conditions. These remarks about conditions are important and are stated before presenting the PMP.

The optimality conditions are consequence of the application of the Euler-Lagrange equation, which can be found on [21]. They mean to be the necessary but not sufficient conditions for the resolution of the optimisation problem; in other words, they are the proof that the solution found is actually the optimal one. They are careless of meaning until they are directly found for the problem. They are also called KKT conditions. Essentially, they are a set of equations related to the gradient or derivative of a functional respect to other parameters and variables that have to be null in order to verify optimality. A general optimisation problem and its Lagrangian can be defined as follows.

$$\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & f\ (x) \\
\text{s.t.} \quad & g_i\ (x) \leq 0 \text{ for each } i = 1, \ldots, m \\
& h_j\ (x) = 0 \text{ for each } j = 1, \ldots, p
\end{aligned} \tag{2.9}$$

$$\mathcal{L}(x, z, \mu) \coloneqq f(x) + \sum_{i=0}^{m} z_i g_i(x) + + \sum_{j=0}^{p} \mu_j h_j(x) \tag{2.10}$$

The KKT conditions are stationarity, complementary slackness, primal feasability and dual feasability. Its meaning and mathematical formulation can be seen below as an adaptation from [22].

- Stationarity. Referred to the balanced effect of the gradient of the active constraints and the gradient of the objective functions. In other words, it describes how each

constraint influences the cost function itself, ensuring that at the optimum no further improvement is possible without violating constraints.

$$\nabla_x \mathcal{L}(x, z, \mu) \coloneqq \nabla f(x) + \sum_{i=0}^{m} z_i \nabla g_i(x) + \sum_{j=0}^{p} \mu_j \nabla h_j(x) = 0 \tag{2.11}$$

- Complementary slackness. The multiplier reflects how much the constraint influences the optimal solution. For each inequality constraint, either the constraint is active and its Lagrange multiplier can be positive, or the constraint is inactive and its multiplier is zero.

$$z_i g_i(x) = 0 \tag{2.12}$$

- Primal feasability. Ensures that the found solution satisfies the original constraints of the problem, which means that the inequality conditions must be held and the equality conditions must be satisfied totally.

$$\begin{aligned} g_i(x) &\leq 0 \\ h_j(x) &= 0 \end{aligned} \tag{2.13}$$

- Dual feasability. Related to inequality constraints multipliers, they verify they are non-negative values and describe how would the cost and the solution change if these constraints are relaxed or inactivated.

$$z_i \geq 0 \tag{2.14}$$

On its part, the transversality conditions are the boundary conditions for the co-state variables λ. They are a necessary condition for optimisation in state free-end problems, which means that there is no end known for the trajectory. These transversality conditions can be defined as the existence of a n-vectorial function of the co-state variables $\lambda(t)$ and a m-vectorial function $\nu(t)$ such that the Hamiltonian function $H(\lambda, \nu, t)$, see equation (2.15), can be defined like in a boundary value problem in his canonical form with ODEs in all time domain $t_0 \leq t \leq t_f$ [23][24].

$$H\ (\lambda, \nu, t) \coloneqq \lambda^T f + \nu^T g \tag{2.15}$$

Once the Hamiltonian is defined, both transversality conditions can be found as the partial derivative of the Hamiltonian respect to the state variables and, the evaluation of the co-state at the end of time domain, respectively. Formulation has been adapted from [23] and [25].

$$\dot{\lambda} \coloneqq -\frac{\partial H}{\partial y} = -\lambda^T \frac{\partial f}{\partial y} - \nu^T \frac{\partial g}{\partial y} \tag{2.16}$$

$$\lambda(t_f) \coloneqq \frac{\partial \Phi}{\partial y(t_f)} - \nu^T \frac{\partial \varphi}{\partial y(t_f)} \tag{2.17}$$

Eventually, the PMP is a fundamental result of optimal control theory, since it stablishes the necessary conditions to find the optimal trajectory for control variables. In other words, this principle gives the strategy that has to be followed on the control variables, the ones that can be modified through time on the whole problem, to minimise the cost functional while the system dynamic and its constraints are satisfied. From [26], the PMP is defined as the partial derivative of the Hamiltonian function respect to the control variables for a problem where $y(0) = y_0 \in \mathcal{S}$ and $u(t)$, a contol trajectory, has a state trajectory $y(t)$ associated for a given dynamic system. Thus, a co-state trajectory $\lambda(t)$ can be defined as follows.

$$\begin{aligned} &\dot{\lambda}(t) \coloneqq -\frac{\partial H}{\partial y} \\ &\lambda(t_f) \coloneqq \frac{\partial \Phi}{\partial y(t_f)} \\ &\textit{where} \\ &\qquad H(y,u,\lambda,t) \coloneqq L(y,u,t) + \lambda^T f(y,u,t) \\ &\qquad \dot{y}(t) \coloneqq f(y,u,t) \end{aligned} \tag{2.18}$$

Therefore, the PMP can be expressed as the minimisation of the Hamiltonian with respect to the control variables, as indicated in equation (2.19). This formulation arises because the original statement of PMP is given as a maximisation problem, but it can be equivalently rewritten in minimisation form by changing the sign of the Hamiltonian -also called Pontryagin's Minimum Principle- [27]. In essence, the principle reformulates the original optimal control problem into an equivalent system involving the state and an adjoint -or co-state- system. The optimal control trajectory is then obtained by ensuring that, at each instant of time, the Hamiltonian is minimised with respect to the admissible controls.

$$u(t) \coloneqq \arg\min_{u \in U} \; H\ (y,u,\lambda,t) \tag{2.19}$$

2.3 Introduction to IPOPT

The IPOPT is a robust interior-point line-search algorithm designed to solve large-scale NLP problems. This method is particularly well-suited for high-dimensional trajectory

optimisation, such as the low-altitude flight manoeuvres required during water-discharge phases in firefighting missions -the primary focus of this thesis-.

IPOPT identifies local optimal solutions by employing a primal-dual barrier approach to handle inequality constraints and a filter-based line-search method to ensure global convergence. The following subsections detail the mathematical formulation of the IPOPT algorithm and its practical implementation via the CasADi numerical optimisation framework.

2.3.1 Mathematical formulation

Below there is a general formulation of an optimisation problem followed by its analogue in barrier problem notation. The barrier problem method is a technique used to transform inequalities in constrained optimisation problems into an unconstrained problem, which is easy to solve.

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned} \tag{2.20}$$

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \varphi_\mu(x) \coloneqq f(x) - \mu \sum_{i=0}^{n} ln(x^{(i)}) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned} \tag{2.21}$$

The barrier method transforms the inequality constraints of a problem into a sequence of equality-constrained subproblems that are easier to solve numerically. Once the barrier problem is defined, the IPOPT algorithm computes an approximate solution for a fixed value of the barrier parameter μ. This process is iterative: for each μ_j, the algorithm seeks a local solution to the subproblem. Upon reaching a specific inner tolerance, the barrier parameter is decreased, and the previous solution is used as a 'warm start' for the next subproblem.

$$\mathcal{L}(x, \lambda, z) \coloneqq f(x) + c(x)^T\lambda - x^T z \tag{2.22}$$

Following the Lagrangian, the KKT conditions can be defined for the barrier problem using diagonal matrices for x variables and z multipliers on complementary slackness definition, which is defined using a tolerance e.

$$\nabla f(x) + \nabla c(x)\lambda - z = 0 \tag{2.23a}$$
$$c(x) = 0 \tag{2.23b}$$
$$XZe - \mu e = 0 \tag{2.23c}$$

At this point, optimality error of the barrier problem can be defined using KKT conditions. The error is the ratio that would be used to compare with tolerances to determine if local and global optimal points are reached. Full definition of s_d and s_c can be found on [28].

$$E_\mu(x,\lambda,z) \coloneqq \max\left\{\frac{||\nabla f(x)+\nabla c(x)-z||_\infty}{s_d}, \frac{||XZe-\mu e||_\infty}{s_c}\right\} \tag{2.24}$$

The expression above is used on each iteration of the IPOPT algorithm to compute the optimality error and, therefore, check local convergence before stepping onto the following point. The checks for local and global convergence are the following, respectively. As can be seen, these checks use an approximate solution and an approximation to multipliers $(\tilde{x}^*, \tilde{\lambda}^*, \tilde{z}^*)$. It also has to be pinpointed that at each iteration, the update in barrier parameter is obtained following the expression (2.27). All three expresions are computed for given constants κ_ϵ, κ_μ, θ_μ and ϵ_{tol}, as indicated in [28].

$$E_{\mu,j}(\tilde{x}^*_{j+1}, \tilde{\lambda}^*_{j+1}, \tilde{z}^*_{j+1}) \leq \kappa_\epsilon \mu_j \tag{2.25}$$

$$E_0(\tilde{x}^*, \tilde{\lambda}^*, \tilde{z}^*) \leq \epsilon_{tol} \tag{2.26}$$

$$\mu_{j+1} = \max\left\{\frac{\epsilon_{tol}}{10}, \min\left\{\kappa_\mu \mu_j, \mu_j^{\theta_\mu}\right\}\right\} \tag{2.27}$$

As said before, the IPOPT is based on a primal-dual framework that iterates between variables and multipliers by solving a linearised set of the KKT conditions via Newton's method. This method allows to find gradients for each variable or multiplier, as stated below -with $A_k \coloneqq \nabla c(x_k)$, $W_k \coloneqq \nabla_{xx}\mathcal{L}(x_k,\lambda_k,z_k)$ and k the inner-loop counter-.

$$\begin{bmatrix} W_k & A_k^T & -I \\ A_k & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} d_x^k \\ d_\lambda^k \\ d_z^k \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + A_k^T\lambda_k - z_k \\ c(x_k) \\ X_k Z_k e - \mu_j e \end{pmatrix} \tag{2.28}$$

From the previous expression, gradients could be computed. Thus, the update on variables and multipliers will be set following the set of equations (2.27), once the step size parameter α_k is determined by a filter search or filter method. A deep explanation of the filther methods used can be found on [28] and [29].

$$x_{k+1} \coloneqq x_k + \alpha_k d_k^x \tag{2.29a}$$
$$\lambda_{k+1} \coloneqq \lambda_k + \alpha_k d_k^\lambda \tag{2.29b}$$
$$z_{k+1} \coloneqq z_k + \alpha_k d_k^z \tag{2.29c}$$

Eventually, a summarised version of the general IPOPT algorithm has been adapted from the original source [28].

Algorithm 1 Interior-Point Filter Line-Search Algorithm (IPOPT).

1. Initialise: Choose $x_0 > 0$, $\lambda_0, z_0 > 0$, $\mu > 0$, and iteration limit N.
for $k = 0$ **to** N **do**
  2. Compute Errors: Calculate $E_\mu(x_k, \lambda_k, z_k)$ and $E_0(x_k, \lambda_k, z_k)$.
  if $E_0 \leq \epsilon_{tol}$ **then**
    STOP: Optimal solution found at iteration k.
  end if
  if $E_\mu \leq \kappa_\epsilon \mu$ **then**
    3. Update Barrier: Decrease μ and return to Step 2.
  end if
  4. Solve KKT System: Compute search directions d_k via expression (2.28).
  5. Filter Line-Search: Determine α_k via the filter method.
  6. Update Iterates: Compute them using equations (2.29).
end for
if $k = N$ **then**
  Exit: Maximum iterations reached without convergence.
end if

2.3.2 Implementation of IPOPT using CasADi

IPOPT is one of the integrated solvers provided by CasADi inside the *ca.nlpsol()* block. This solver blocks needs from a very specific definition of the problem initial guess, its variables, constraints and simple bounds using symbolic notation. This specific definition has to end in the creation of a *nlp{}* dictionary that contains all the discretised symbolic variables that form the NLP problem. In other words, from the OCP, a discretisation has to be applied to reach a fesaible NLP definition with initial guess, constraints and simple bounds defined for each variable at each collocation point -all those defined using symbolic notation-. In addition, this dictionary has to include the definition of the cost function as the sum of all variables, when needed.

Since the needs explained above, suppose a simple problem directly extracted from NLP CasADi documentation section [30]. In this problem, there are three different variables, an objective function and an equality constraint.

$$\begin{aligned} \min_{x,y,z} \quad & f(x,z) \coloneqq x^2 + 100z^2 \\ \text{s.t.} \quad & z + (1-x)^2 - y = 0 \end{aligned} \tag{2.30}$$

The guidelines for the creation of NLP problem dictionary are:

- State and control vector w. The state and control vector has to contain all the states and controls in symbolic notation for each collocation point, from the first to the last, following an order. On the example set, since there is a unique collocation point and there are three variables, the definition will be $w = (x, y, z)$ -in the case there were more than one collocation point, $w = (x_0, y_0, z_0, x_1, y_1, z_1 \dots x_N, y_n, z_N)$-.
- Cost function $f(x, z)$: The cost function has to include the penalties defined for each variable when needed. In the example, there is a unique cost defined. Nevertheless, if it was defined for more than one collocation point it would be translated into *ca.nlpsol()* needs, like stated in (2.31).

$$F(x, z) := x_0^2 + 100z_0^2 + x_1^2 + 100z_1^2 + \ldots + x_N^2 + 100z_N^2 \tag{2.31}$$

- Constraint vector g: The constraint vector has to the different constraints by order of appearance and, following the remarks on cost function, they would have to be defined for each collocation point variables when needed.

From the remarks above, the dictionary for *ca.nlpsol()* can be defined. Its definition is inserted inside an end-to-end example of programming implementation. The bases set are the ones that would be later used for the thesis problem definition and implementation.

Listing 2.1: Example definition using CasADi *ca.nlpsol()* block and IPOPT solver. Extracted from [30].

```
import casadi as ca
x = ca.SX.sym('x'); y = ca.SX.sym('y'); z = ca.SX.sym('z')
nlp = {'x':vertcat(x,y,z), 'f':x**2+100*z**2, 'g':z+(1-x)**2-y}
S = ca.nlpsol('S', 'ipopt', nlp)
```

Another important remark lays into the formulation of numeric initial guess, which has to be initialised in each variable defined on state and control vector w. In other words, solver would need an analogue initial state and control vector w_0. Furthermore, constraints are defined following the order provided in discretised OCP definition, see (2.8), which is: dynamic constraints, path constraints and initial state constraints. As state before, those remarks apply for all variables and would have to be defined following the same order that has been defined in w. CasADi does not accept matrices easily, which means that constraints vector g will need g_{lb} and g_{ub} provided as vectors too -noted as *lbg* and *ubg* through all thesis and provided code-. That last remark also applies to simple bounds x_{lb} and x_{ub}, which also have to be provided as vectors and which are noted as *lbx* and *ubx*, respectively.

3 | Benchmark Problem: A Level-Flight Cruise Trajectory

This chapter introduces the reader to flight dynamics using a level-cruise flight trajectory. Also, the trajectory is formulated as an OCP and solved via an implementation using CasADi. Therefore, a mathematical formulation on OCP and NLP notation is presented right after the equations of motion and, right before presenting the results. Eventually, the free-end condition is applied and solved, acting as a preamble of the following chapter.

3.1 UAV Kinematics and Equations of Motion

Firstly, the flight dynamics for a general UAV flight trajectory has to be introduced and described. Since the aim of the project is OCT development for firefighting manoeuvres, some hypotheses have to be applied. The hypotheses considered are the follwoing ones.

- A bidimensional (2D) trajectory without tridimensional (3D) effects will be considered. Thus, all lateral stability and movements will be neglected and only longitudinal static stability will be considered -i.e $\mathcal{L} = 0$, $\mathcal{N} = 0$, $\Phi = 0$, $\dot{\Phi} = 0$, $\Psi = 0$, $\dot{\Psi} = 0$, $y = const.$ and $\dot{y} = 0$-.
- All dynamic modes are neglected and only static stability phenomena will be included.
- The atmospheric conditions shall be modelled using International Standard Atmoshpere (ISA) model. Only static bidimensional (2D) wind field would be considered, which means that free-stream velocity has to be constant in all domain.
- Bidimensional (2D) level-flight cruise trajectory implies a null flight path angle and null variation of the same -i.e $\gamma = 0$ and $\dot{\gamma} = 0$ -.

Once the hypostheses have been written, its time to introduce the flight dynamics that are involved in bidimensional (2D) flight trajectories. The image below shows the free-body diagram and the relation between plane reference systems -i.e including body axes (x_b,z_b), wind axes (x_w,z_w) and local-horizon axes (x_h,z_h)-.

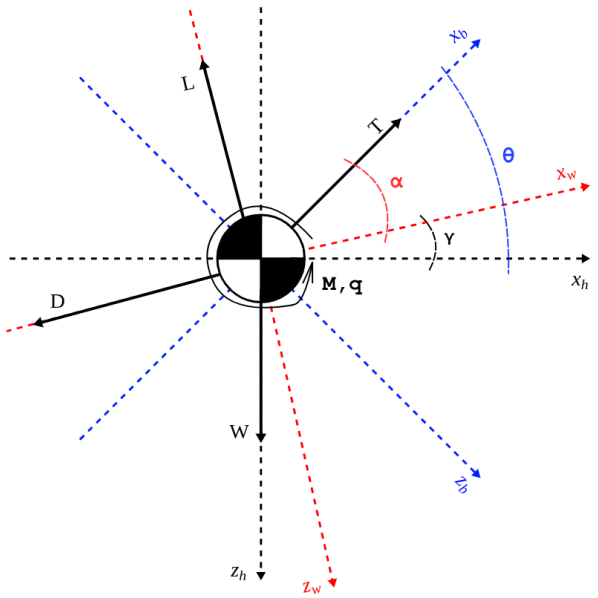


Figure 3.1: Free-body diagram of an aircraft bidimensional (2D) flight trajectory, involving forces and moments. Own source.

Applying Newton's second law, the equilibrium of forces and moments can be obtained. To see the different elements involved in each equation, the reader can refer to nomenclature's section.

$$\sum F_{x_b} = m\frac{du}{dt} = m\dot{u} = T + L\sin\alpha - D\cos\alpha - W\sin\theta - mqw \tag{3.1}$$

$$\sum F_{z_b} = m\frac{dw}{dt} = m\dot{w} = W\cos\theta - L\cos\alpha - D\sin\alpha + mqu \tag{3.2}$$

$$\sum M_{y_b} = I_y\frac{dq}{dt} = I_y\dot{q} = C_m\bar{q}S\bar{c} - T\left[\Delta X_{CT}\sin\varepsilon + \Delta Z_{CT}\cos\varepsilon\right] \tag{3.3}$$

Along to the dynamic equations, the UAV kinematics can be described using the rotation matrix between body and local-horizon axes as follows.

$$\begin{bmatrix}\dot{x}\\ \dot{z}\end{bmatrix} := R_{hb}\begin{bmatrix}u\\ w\end{bmatrix} = \begin{bmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}u\\ w\end{bmatrix} = \begin{bmatrix}u\cos\theta + w\sin\theta\\ -u\sin\theta + w\cos\theta\end{bmatrix} \tag{3.4}$$

Eventually, pitch and mass variations have to be considered to complete the full dynamic and kinematics set of equations. It has to be pinpointed that for this benchmark problem, only specific fuel consumption is considered on mass variation equation.

$$\frac{d\theta}{dt} \coloneqq \dot{\theta} = q \tag{3.5}$$

$$\frac{dm}{dt} \coloneqq \dot{m} = -SFC \tag{3.6}$$

As a clarification on the equation terms, below are the complete formulation of aerodynamic forces and moments, with all proper aerodynamic coefficients considered.

$$L \coloneqq \bar{q}SC_L = \bar{q}S\left(C_{L_0} + C_{L_\alpha}\alpha + C_{L_{\delta_e}}\delta_e\right) \tag{3.7}$$

$$D \coloneqq \bar{q}SC_D = \bar{q}S\left[C_{D_0} + k\left(C_{L_0} + C_{L_\alpha}\alpha + C_{L_{\delta_e}}\delta_e\right)^2\right] \tag{3.8}$$

$$C_m \coloneqq C_{m_0} + C_{m_\alpha}\alpha + C_{m_{\delta_e}}\delta_e \tag{3.9}$$

3.2 Mathematical Formulation of the OCP

Next step is to transform the problem above into a set of equations using the OC notation. To do so, the three phases of optimisation problems description are used, which means that the problem has to be modelled, constrained and then optimised.

3.2.1 States and controls

First step is modelisation, which means that state and control variables have to be defined for the problem. Since the benchmark problem is a level-flight cruise trajectory, the states involved in the previous equations of motion are: local-horizon position (x,z); body velocities (u,w); pitch (θ); pitch rate (q); and mass (m). Those seven states are the ones that allow describing the trajectory and UAV condition through time, since they indicate position, attitude and mass.

The previous states are defined for a bidimensional (2D) frame using only longitudinal stability equations, as has been indicated before. Therefore, the controls that shall be used are no other ones but TPS, denoted as δ_{TPS}, and elevator deflection, denoted as δ_e. That pair of control variables are the minimum controllable conditions for the longitudinal flight given the conditions of the problem and they are responsible of controlling speed and altitude.

Eventually, the state and control vector can be constructed, followed by its form using state space notation -where x_i are states and u_i are controls-. Notice that the order followed to construct the vector has been state variables followed by control variables;

nevertheless, is indistinct which state or control goes first in each group. The order has been chosen by convenience for later NLP formulation.

$$\mathbf{w} \coloneqq \begin{bmatrix} u \\ w \\ q \\ \theta \\ x \\ z \\ m \\ \delta_{TPS} \\ \delta_e \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ u_1 \\ u_2 \end{bmatrix} \tag{3.10}$$

3.2.2 Constraints and bounds

After the state space is set, the constraints can be defined, following the second step on OCP definition. At this point, previous remarks on constraint definition towards OCP notation could be recovered. That means that constraints are described following the preset order, which starts by dynamic constraints. Since dynamic constraints have been defined in section 3.1, they have to be transcribed using state space notation, which led to the following set of expressions.[1]

$$\begin{aligned} \dot{x}_1 = \frac{1}{x_7}\Big[& T(x_1, x_2, x_6, u_1) + L(x_1, x_2, x_6, u_2) \cdot s\alpha(x_1, x_2) \\ & - D(x_1, x_2, x_6, u_2) \cdot c\alpha(x_1, x_2)\Big] - x_3 x_2 - g \cdot s x_4 \end{aligned} \tag{3.11}$$

$$\begin{aligned} \dot{x}_2 = g \cdot c x_4 - \frac{1}{x_7}\Big[& L(x_1, x_2, x_6, u_2) \cdot c\alpha(x_1, x_2) \\ & + D(x_1, x_2, x_6, u_2) \cdot s\alpha(x_1, x_2)\Big] + x_3 x_1 \end{aligned} \tag{3.12}$$

$$\begin{aligned} \dot{x}_3 = \frac{1}{I_y}\Big[& C_m(u_2)\bar{q}(x_1, x_2, x_6) S\bar{c} \\ & - T(x_1, x_2, x_6, u_1)\left[\Delta X_{CT} \cdot s\varepsilon(x_1, x_2) + \Delta Z_{CT} \cdot c\varepsilon(x_1, x_2)\right]\Big] \end{aligned} \tag{3.13}$$

$$\dot{x}_4 = x_3 \tag{3.14}$$

[1] From now on, sine operation sin() is denoted as s and, cosine operation cos() is denoted as c. This has been done for an easy readability of some equations and operations.

$$\dot{x}_5 = x_1 \cdot cx_4 + x_2 \cdot sx_4 \tag{3.15}$$

$$\dot{x}_6 = -x_1 \cdot sx_4 + x_2 \cdot cx_4 \tag{3.16}$$

$$\dot{x}_7 = -SFC \tag{3.17}$$

The set of equations above can be summarised to a reduced version if RHS is expressed as functionals of the states and controls involved only.

$$\mathbf{g_{dyn}} \coloneqq \begin{bmatrix} \dot{x}_1 - f_1(x_1, x_2, x_3, x_4, x_6, x_7, u_1, u_2) \\ \dot{x}_2 - f_2(x_1, x_2, x_3, x_4, x_6, x_7, u_2) \\ \dot{x}_3 - f_3(x_1, x_2, x_6, u_1, u_2) \\ \dot{x}_4 - f_4(x_3) \\ \dot{x}_5 - f_5(x_1, x_2, x_4) \\ \dot{x}_6 - f_6(x_1, x_2, x_4) \\ \dot{x}_7 - SFC \end{bmatrix} = 0 \tag{3.18}$$

Once dynamic constraints have been fully described, its time to set the path constraints and define them using state space notation. As said before, path constraints are the ones that could affect state and control trajectories during all time domain. For a level-flight cruise trajectory it is important to introduce a target speed (V_{TP}) and a target altitude (h_{ref}) as a constraints, see expressions below. It has to be pinpointed that flight path angle (γ) could be introduced as a constraint, but it has been chosen to introduce it later as a penalty into cost functional definition. Both constraints for target speed and target altitude are introduced using inequality expressions, so as not to be too restrictive with the problem -i.e the expressions have to introduce some margins with target constant values-.

$$\begin{cases} 0.9 \cdot V_{TP} - \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} \leq 0 \\ \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} - 1.1 \cdot V_{TP} \leq 0 \end{cases} \tag{3.19}$$

$$\begin{cases} (h_{ref} - 3.0) + z \leq 0 \\ -z - (h_{ref} + 3.0) \leq 0 \end{cases} \tag{3.20}$$

Transcribing the equations using state space notation leads to the following set of expressions.

$$\mathbf{g_{path}} \coloneqq \begin{bmatrix} 0.9 \cdot V_{TP} - \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} \\ \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} - 1.1 \cdot V_{TP} \\ (h_{ref} - 3.0) + x_6 \\ -x_6 - (h_{ref} + 3.0) \end{bmatrix} \leq 0 \tag{3.21}$$

At this point, the only remaining constraints are initial state conditions, which will be enforced to a known state value. Thus, the following equations will have to be fulfilled.

$$\varphi_{\mathbf{0}} \coloneqq \begin{bmatrix} x_1^0 - u_0 \\ x_2^0 - w_0 \\ x_3^0 \\ x_4^0 - \theta_0 \\ x_5^0 \\ x_6^0 + h_{ref} \\ x_7^0 - TOM \end{bmatrix} = 0 \tag{3.22}$$

Eventually, the simple bounds for each state and control variable have to be declared. The lower bounds and upper bounds are separated onto two different groups denoted by *lbx* and *ubx*, respectively. They have been already denoted using state space notation.

$$\mathbf{lbx} \coloneqq \begin{bmatrix} u_{min} - x_1 \\ w_{min} - x_2 \\ -x_3 \\ \theta_{min} - x_4 \\ -x_5 \\ z_{min} - x_6 \\ BEM - x_7 \end{bmatrix} \leq 0 \tag{3.23}$$

$$\mathbf{ubx} \coloneqq \begin{bmatrix} x_1 - u_{max} \\ x_2 - w_{max} \\ x_3 \\ x_4 - \theta_{max} \\ x_5 \\ x_6 - z_{max} \\ x_7 - MTOM \end{bmatrix} \leq 0 \tag{3.24}$$

3.2.3 Cost functional

Third phase of an OCP definition is defining which variables have to be optimised and below which optimisation criteria. For this benchmark case, five different criteria have been chosen.

- Firstly, the minimisation of the flight path angle (γ) is introduced as a quadratic penalty in the objective function rather than a hard equality constraint. This formulation encourages the solver to converge on a solution where $\gamma \approx 0$, effectively approximating a level-flight trajectory. This approach was chosen because enforcing $\gamma = 0$ as an equality constraint could lead to numerical stiffness, significantly increasing the difficulty of finding a feasible solution. By using it as a penalty, the solver can more efficiently negotiate the trade-offs between system dynamics and path objectives, leading to a more robust and faster convergence toward the optimal solution. The expression introduced in the running cost is the following one, where $\bar{\gamma}$ is the normalised γ term using γ_{max}.

$$J_\gamma := \bar{\gamma}^2 = \left(\frac{\gamma}{\gamma_{max}}\right)^2 = \left(\frac{\theta - \alpha}{\gamma_{max}}\right)^2 = \left(\frac{x_4 - \alpha(x_1, x_2)}{\gamma_{max}}\right)^2 \tag{3.25}$$

- Secondly, the minimisation of altitude error relative to the reference altitude (h_{ref}) is implemented as a quadratic penalty to promote a level-flight profile. Unlike the path constraints defined in (3.20), which establish strict safety limits, the penalty allows for transient deviations during manoeuvres. This dual-layer approach -constraining the aircraft within a feasible corridor while penalising deviations from the centerline- mimics real-world flight control laws. It allows the optimiser to prioritise control smoothness and actuator limits over rigid altitude tracking, resulting in a more physically realistic and numerically stable trajectory.

$$J_h := (\bar{h} - 1)^2 = \left(\frac{h - h_{ref}}{h_{ref}}\right)^2 = \left(\frac{-x_6 - h_{ref}}{h_{ref}}\right)^2 \tag{3.26}$$

- Thirdly, the minimisation of flight path angle rate ($\dot{\gamma}$) prevents the aircraft from rapidly oscillating up and down while forces the solver to find a solution that is not only level ($\gamma \approx 0$) but also steady ($\dot{\gamma} \approx 0$). The full derivation of the expression below can be found on appendix A, since it involves partial derivatives and a large mathematical development.

$$J_{\dot{\gamma}} := \left(q - \frac{\dot{w}u - \dot{u}w}{u^2 + w^2}\right)^2 / \dot{\gamma}_{max}^2 = \left(x_3 - \frac{\dot{x}_2 x_1 - \dot{x}_1 x_2}{x_1^2 + x_2^2}\right)^2 / \dot{\gamma}_{max}^2 \tag{3.27}$$

- Eventually, control variable rates minimisation have been introduced using a quadratic penalty, in each case, for preventing a bang-bang -rapidly control changing- behaviour, since it is a non-feasible behaviour in real systems. Furthermore, control rate minimisation is the normal criteria for real pilots when facing low-altitude flight manoeuvres, also known as the 'lazy pilot' implementation. This implementation basically reduces pilot's workload while preventing overcontrol risks, as stated by Stengel in [31]. The cost expressions for both TPS and elevator deflection are the following ones, respectively -t_{k+1} and t_k, refer to next and actual time-instants-.

$$J_{\dot{\delta}_{TPS}} := \frac{\left(\delta_{TPS}^{k+1} - \delta_{TPS}^{k}\right)^2}{\Delta t} = \frac{\left(u_1^{k+1} - u_1^{k}\right)^2}{\Delta t} \tag{3.28}$$

$$J_{\dot{\delta}_e} := \frac{\left(\delta_e^{k+1} - \delta_e^{k}\right)^2}{\delta_{e,max}^2 \Delta t} = \frac{\left(u_2^{k+1} - u_2^{k}\right)^2}{\delta_{e,max}^2 \Delta t} \tag{3.29}$$

Once all penalties have been defined, the whole cost functional in its Lagrange form can be stated as follows. Notice that each cost term has a penalty weight associated.

$$L(x,u) := \sum_{k=0}^{N-1} \left(w_\gamma \cdot J_\gamma + w_h \cdot J_h + w_{\dot{\gamma}} \cdot J_{\dot{\gamma}} + w_{\dot{\delta}_{TPS}} \cdot J_{\dot{\delta}_{TPS}} + w_{\dot{\delta}_e} \cdot J_{\dot{\delta}_e} \right) \tag{3.30}$$

3.3 NLP Formulation and Implementation via CasADi

After defining the whole OCP, it is time to discretise the equations using trapezoidal rule to achieve a NLP problem formulation and, therefore, implement it into CasADi blocks. To do the discretisation it is important to remark two different points: firstly, the dynamic constraints will be evaluated for the actual time-instant and next time-instant -following trapezoidal integration scheme- while initial constraints will be enforced at time-instant t_0 only, and path constraints will be enforced at each time-instant t_k separately; referring to Lagrange cost term, it will be evaluated also following a trapezoidal scheme. This leads to the following constraints, simple bounds and cost terms. It has to be pinpointed that, in this case, end-time is known and, therefore, time-step is defined as $\Delta t = t_f/(N-1)$, where N is the number of collocation points -or nodes-. Moreover, its implementation in CasADi has to follow the statements done in section 2.3.2, which means to create state and control, constraints and simple bounds vectors as large as the number of states and control variables per node -seven (7) states plus two (2) controls, per node-

$$\begin{aligned} \mathbf{x}^k &:= [x_1^k, x_2^k, x_3^k, x_4^k, x_5^k, x_6^k, x_7^k]^\top \\ \mathbf{u}^k &:= [u_1^k, u_2^k]^\top \end{aligned}$$

$$\mathbf{w} := \begin{bmatrix} \mathbf{x}^0 \\ \mathbf{u}^0 \\ \mathbf{x}^1 \\ \mathbf{u}^1 \\ \vdots \\ \mathbf{x}^N \\ \mathbf{u}^N \end{bmatrix} \tag{3.31}$$

$$\begin{aligned}\mathbf{d}^k &:= \mathbf{x}^{k+1} - \mathbf{x}^k - \frac{\Delta t}{2}\left(\mathbf{f}^k + \mathbf{f}^{k+1}\right) \in \mathbb{R}^7 \\ \mathbf{h}^k &:= \mathbf{g}_{\text{path}}(\mathbf{x}^k, \mathbf{u}^k) \in \mathbb{R}^4 \\ \varphi_0 &:= \mathbf{x}^0 - \mathbf{x}_{\text{init}} \in \mathbb{R}^7\end{aligned}$$

$$\mathbf{g}(\mathbf{w}) := \left[\begin{array}{c} \mathbf{d}^0 \\ \vdots \\ \mathbf{d}^{N-1} \\ \hline \mathbf{h}^0 \\ \vdots \\ \mathbf{h}^N \\ \hline \varphi_0 \end{array}\right], \quad \mathbf{g}_{lb} := \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ -\infty_{4N} \\ \mathbf{0}_7 \end{bmatrix}, \quad \mathbf{g}_{ub} := \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ \mathbf{0}_{4N} \\ \mathbf{0}_7 \end{bmatrix} \tag{3.32}$$

$$\begin{aligned}\mathbf{w}_{lb} &:= \mathbf{1}_N \otimes \mathbf{lbx} \\ \mathbf{w}_{ub} &:= \mathbf{1}_N \otimes \mathbf{ubx}\end{aligned} \tag{3.33}$$

$$\begin{aligned}L^k &:= L(\mathbf{x}^k, \mathbf{u}^k) \\ J(\mathbf{w}) &:= \frac{\Delta t}{2} \sum_{k=0}^{N-1} \left(L^k + L^{k+1}\right)\end{aligned} \tag{3.34}$$

Next, the NLP problem can be defined properly using almost CasADi notation.

$$\begin{aligned}\min_{\mathbf{w}} \quad & J(\mathbf{w}) := \frac{\Delta t}{2} \sum_{k=0}^{N-1} \left(L^k + L^{k+1}\right) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{w}) \in [\mathbf{g}_{lb}, \mathbf{g}_{ub}] \\ & \mathbf{w} \in [\mathbf{1}_N \otimes \mathbf{lbx},\ \mathbf{1}_N \otimes \mathbf{ubx}]\end{aligned} \tag{3.35}$$

The algorithm that has been implemented to solve the NLP problem using IPOPT solver is the refered below. The whole structure of the implemented algorithm in Python can be seen in appendix B.

Algorithm 2 Trajectory Optimisation Algorithm for Level-Cruise Flight.

1. Initialise Parameters:
Define N nodes, end-time t_F, time-step Δt, and symbolic vectors for $\mathbf{x}, \mathbf{u}$.
Obtain trim state $\mathbf{x}_{trim}$ via static NLP solver.
Construct initial guess $\mathbf{w}_0$ using $\mathbf{w}_{trim}$ and linear propagation.
2. Assemble Decision Vector:
Formulate $\mathbf{w} \coloneqq [\mathbf{x}^0, \mathbf{u}^0, \ldots, \mathbf{x}^N, \mathbf{u}^N]^\top \in \mathbb{R}^{9N}$.
Apply variable bounds: $\mathbf{w} \in [\mathbf{1}_N \otimes \mathbf{lbx},\ \mathbf{1}_N \otimes \mathbf{ubx}]$.
for $i = 0$ **to** N_{iter} **do**
3. Evaluate NLP Functions:
Compute trapezoidal Cost: $J(\mathbf{w}) = \frac{\Delta t}{2} \sum_{k=0}^{N-1} (L^k + L^{k+1})$.
Evaluate dynamics $\mathbf{d}^k$ and path constraints $\mathbf{h}^k$.
Stack constraints: $\mathbf{g}(\mathbf{w}) \coloneqq [\mathbf{d}^0, \ldots, \mathbf{d}^{N-1}, \mathbf{h}^0, \ldots, \mathbf{h}^N, \varphi_0]^\top$.
4. Check Convergence:
if $E_0(J, \mathbf{g}, \mathbf{w}) \leq \epsilon_{tol}$ **then**
STOP: Optimal trajectory found.
end if
5. Compute Step (KKT System): Form the Lagrangian $\mathcal{L}(\mathbf{w}, \lambda, z)$ and solve the linearised primal-dual system for search direction d_i.
6. Filter Line-Search: Determine step size α_i to ensure decrease in J or constraint violation $\|\mathbf{g}\|$.
7. Update Iterates: $\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha_i d_i^w$.
end for
8. Post-processing: Extract state and control trajectories for plotting.

3.4 Results

The following simulation has been done for the initial states indicated in the table below, which summarises the *Simulation.json* for benchmark problem. The aircraft used for this simulation has been the UAV Flyox, which parameters can be seen on appendix C table C.1. In addition, the simulation has been performed with 150 collocation points -or nodes- for a end-time of 600 s.

As can be seen on figure 3.2, the generated trajectory is a level-flight cruise trajectory for 600 s, leading the UAV to a distance of 30 km when flying at 50 m/s -or 100 kts-. States seem typical for a level-flight since flight path angle is null ($\gamma \approx 0$) and there are no apparent oscillations on altitude. Mass decreases as fuel is consumed, and velocity is mantained at target velocity. Moreover, figure 3.3 shows the TPS and elevator deflection (δ_e) trim points. Elevator deflection, which is positive-defined as trailing edge goes down -if the aircarft does pitch down-, shows a trim point that compensates the natural tendency of Flyox, which is a pitch up moment due to its positive C_{m_0}. It has to be pinpointed that the end value for states is abrupt and increases due to minor changes on controls.

Table 3.1: Benchmark simulation parameters. See reference in appendix B, *Simulation.json*. Own source.

Parameter	Value	Units
N -collocation points-	150	[-]
Initial State -or guesses-	[50.00, 0.07, -, 600.00, -]	[m/s, rad, s, s, s]
Wind Speed	[0.00, 0.00]	[m/s]
Target Point parameters	[-, -, -, 50.00, -]	[km, km, -, m/s, s]
Mission Bounds	LB: [-, -, -, -608.00, -] UB: [-, -, -, -608.00, -]	[s, s, s, m, m/s]
Weights STG1	[0.0, 0.3, 0.2, 1.0, 1.5]	[-]
Weights STG2	[-]	[-]
Weights STG3	[-]	[-]

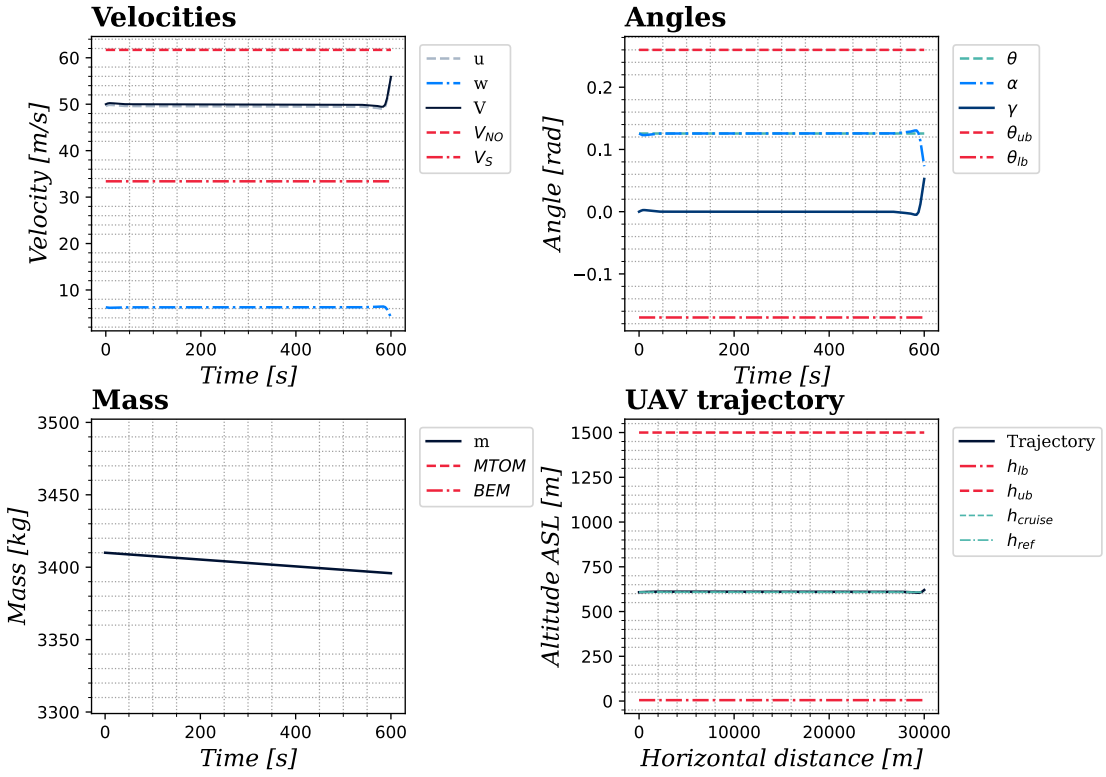


Figure 3.2: State trajectories, benchmark. States included are velocities, angles, mass and UAV trajectory. Own source.

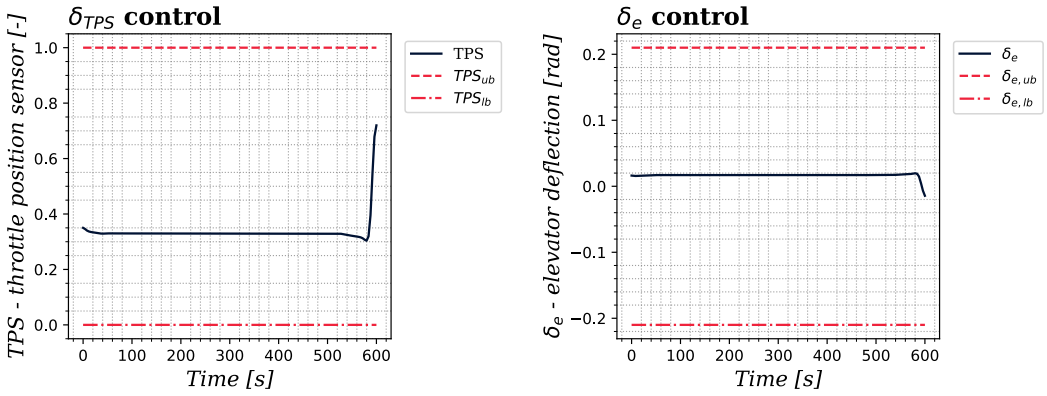


Figure 3.3: Control trajectories, benchmark. Controls included are TPS and elevator deflection. Own source.

3.5 Addition of the Free-End Condition

Next step in OCP formulation is to set the free-end condition to the problem stated before. Free-end condition implies adding end-time (t_f) as a control variable and to let the solver compute its value as part of the dynamic constraints, path constraints -if needed specific end-time conditions- and as part of the cost functional -at this point, terminal cost or Mayer term, has to be remembered-.

To do so, the following section includes the modification on the NLP formulation and on the implementation made. Afterwards, the results of the state and control trajectories for a free-end condition will be presented.

3.5.1 NLP formulation and implementation

As said before, end-time participates in all trapezoidal schemes -so it participates in all dynamic equations-, can participate as path constraints when a terminal state condition is set and, participates into the cost functional implicitly at running cost, because there is also a trapezoidal scheme integrated, and explicitly as a terminal cost. In this case, it would be implemented as both implicit and explicit forms into cost functional.

Firstly, the state and control vector will incorporate a unique symbolic term or control variable that will refer to the final time (t_F). It would also have lower and upper bounds associated, so the modification rests like this:

$$\mathbf{w} \coloneqq \begin{bmatrix} \mathbf{x}^0 \\ \mathbf{u}^0 \\ \mathbf{x}^1 \\ \mathbf{u}^1 \\ \vdots \\ \mathbf{x}^N \\ \mathbf{u}^N \\ t_F \end{bmatrix}, \quad \mathbf{w}_{lb} \coloneqq \begin{bmatrix} \mathbf{1}_N \otimes \mathbf{lbx} \\ t_{F,min} \end{bmatrix}, \quad \mathbf{w}_{ub} \coloneqq \begin{bmatrix} \mathbf{1}_N \otimes \mathbf{ubx} \\ t_{F,max} \end{bmatrix} \tag{3.36}$$

Secondly, the constraints could add some path constraints referring to final state conditions. Those, would only be evaluated at final node N. In this case, the following equations have been added to the path constraints block of constraints as they indicate the minimum and maximum distance to perform the level-flight cruise trajectory -the expressions are also expressed along to previous path constraints and integrated with all NLP constraints vector-.

$$\mathbf{h}_{dist} \coloneqq \begin{bmatrix} V_{TP} \cdot t_{F,min} - (x_f - x_0) \\ (x_f - x_0) - V_{TP} \cdot t_{F,max} \end{bmatrix} \leq \mathbf{0} \tag{3.37}$$

$$\mathbf{g}(\mathbf{w}) \coloneqq \left[\begin{array}{c} \mathbf{d}^0 \\ \vdots \\ \mathbf{d}^{N-1} \\ \hline \mathbf{h}^0 \\ \vdots \\ \mathbf{h}^N \\ \hline \mathbf{h}_{dist} \\ \hline \varphi_0 \end{array}\right], \quad \mathbf{g}_{lb} \coloneqq \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ -\infty_{4N} \\ -\infty_2 \\ \mathbf{0}_7 \end{bmatrix}, \quad \mathbf{g}_{ub} \coloneqq \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ \mathbf{0}_{4N} \\ \mathbf{0}_2 \\ \mathbf{0}_7 \end{bmatrix} \tag{3.38}$$

Next, the cost functional integrates terminal cost term, only applies to end-time, and running cost term, corrected by adding t_F and restringing the problem to $N-1$ time-intervals with a fixed value of nodes, since time-step is defined as $\Delta t = 1/(N-1)$.

$$J(\mathbf{w}) \coloneqq \frac{\Delta t \cdot t_F}{2} \sum_{k=0}^{N-1} \left(L^k + L^{k+1} \right) + w_{t_F} \cdot t_F \tag{3.39}$$

Eventually, the NLP problem can be defined properly using almost CasADi notation before presenting the scheme algorithm so solve it using IPOPT.

$$\begin{aligned} \min_{\mathbf{w}} \quad & J(\mathbf{w}) := \frac{\Delta t \cdot t_F}{2} \sum_{k=0}^{N-1} \left(L^k + L^{k+1} \right) + w_{t_F} \cdot t_F \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{w}) \in [\mathbf{g}_{lb}, \mathbf{g}_{ub}] \\ & \mathbf{w} \in \left[\begin{bmatrix} \mathbf{1}_N \otimes \mathbf{lbx} \\ t_{F,min} \end{bmatrix}, \begin{bmatrix} \mathbf{1}_N \otimes \mathbf{ubx} \\ t_{F,max} \end{bmatrix} \right] \end{aligned} \tag{3.40}$$

Algorithm 3 Updated Version of **Algorithm 2**, Free-End Condition Added.

1. Initialise Parameters:
Define N nodes, time-step Δt, and symbolic vectors for $\mathbf{x}, \mathbf{u}, t_F$.
Obtain trim state $\mathbf{x}_{trim}$ via static NLP solver.
Construct initial guess $\mathbf{w}_0$ using $\mathbf{w}_{trim}$ and linear propagation.
2. Assemble Decision Vector:
Formulate $\mathbf{w} := [\mathbf{x}^0, \mathbf{u}^0, \ldots, \mathbf{x}^N, \mathbf{u}^N, t_F]^\top \in \mathbb{R}^{9N+1}$.
Apply variable bounds: $\mathbf{w} \in \left[\begin{smallmatrix} \mathbf{1}_N \otimes \mathbf{lbx} \\ t_{F,min} \end{smallmatrix} \right], \left[\begin{smallmatrix} \mathbf{1}_N \otimes \mathbf{ubx} \\ t_{F,max} \end{smallmatrix} \right]$.
for $i = 0$ **to** N_{iter} **do**
3. Evaluate NLP Functions:
Compute trapezoidal Cost: $J(\mathbf{w}) = \frac{\Delta t \cdot t_F}{2} \sum_{k=0}^{N-1} (L^k + L^{k+1}) + w_{t_F} \cdot t_F$.
Evaluate scaled dynamics $\mathbf{d}^k$ and path constraints $\mathbf{h}^k$.
Stack constraints: $\mathbf{g}(\mathbf{w}) := [\mathbf{d}^0, \ldots, \mathbf{d}^{N-1}, \mathbf{h}^0, \ldots, \mathbf{h}^N, \mathbf{h}_{\mathbf{dist}}, \varphi_0]^\top$.
4. Check Convergence:
if $E_0(J, \mathbf{g}, \mathbf{w}) \leq \epsilon_{tol}$ **then**
STOP: Optimal trajectory found.
end if
5. Compute Step (KKT System): Form the Lagrangian $\mathcal{L}(\mathbf{w}, \lambda, z)$ and solve the linearised primal-dual system for search direction d_i.
6. Filter Line-Search: Determine step size α_i to ensure decrease in J or constraint violation $\|\mathbf{g}\|$.
7. Update Iterates: $\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha_i d_i^w$.
end for
8. Post-processing: Extract state trajectories, control trajectories and optimal t_F for plotting.

3.5.2 Results

For the free-end simulation, the parameters from table 3.1 remain unchanged except from the definition of a lower-bound $t_{F,lb} = 600\ s$ and $t_{F,ub} = 800\ s$ for end-time control variable. What is more, a penalty weight on terminal cost for end-time has been added with a 0.5 value, in order to see if there are relevant differences.

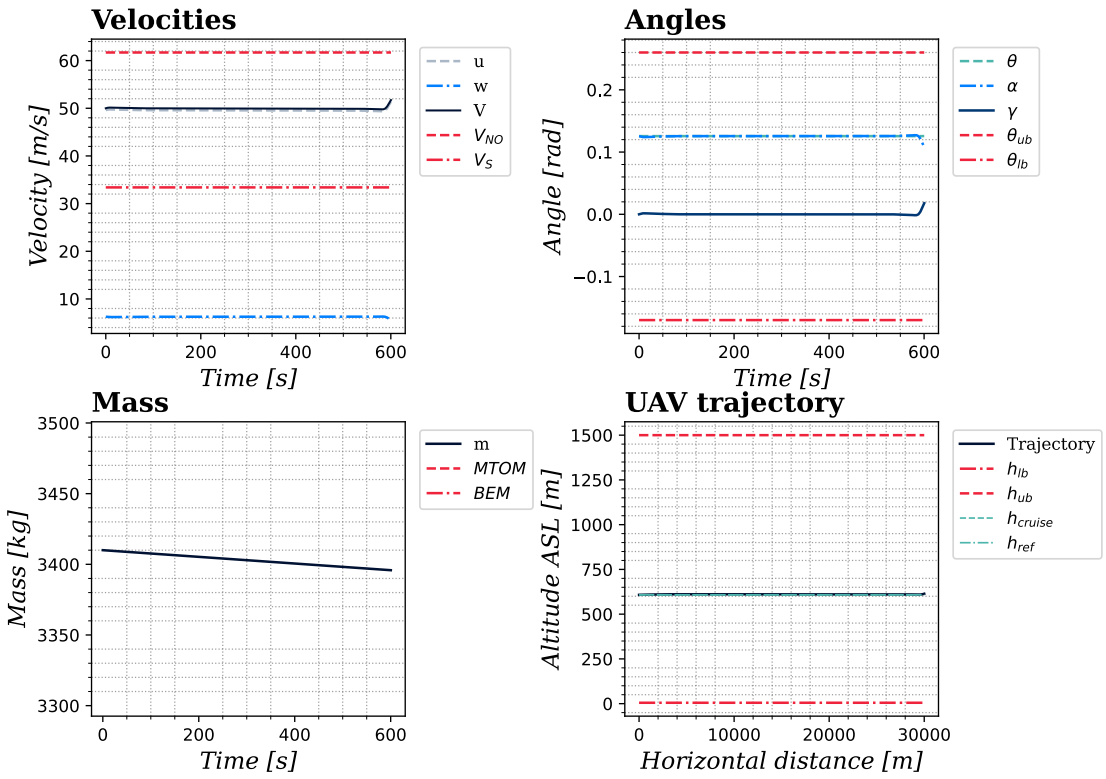


Figure 3.4: State trajectories, benchmark with free-end condition. States included are velocities, angles, mass and UAV trajectory. Own source.

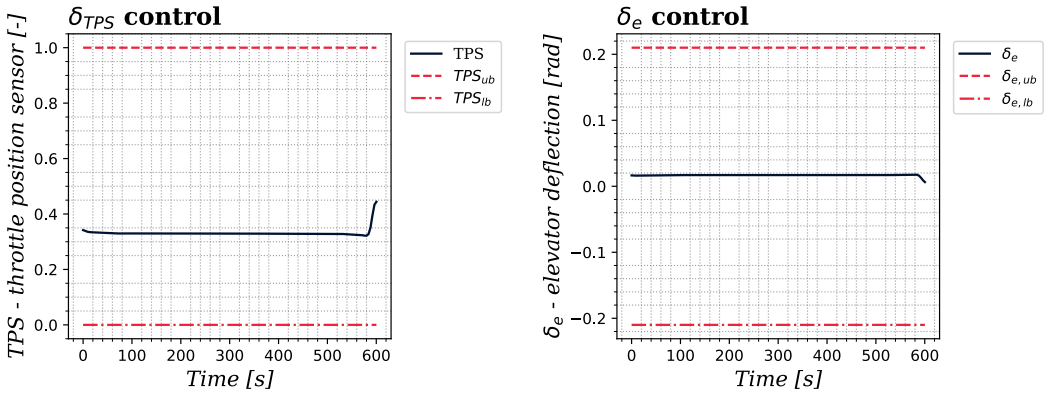
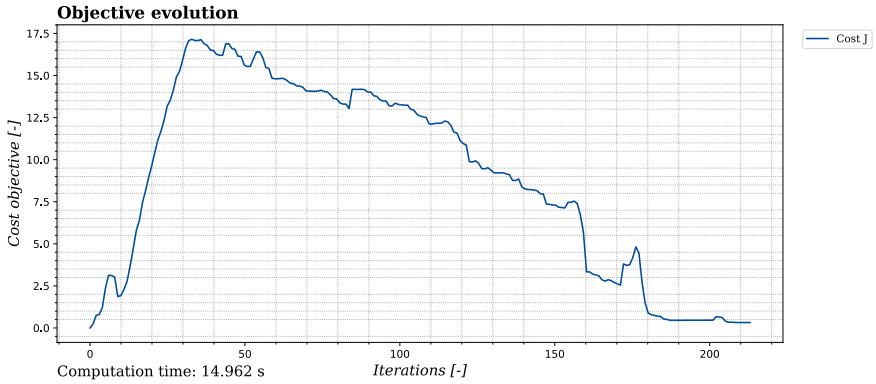


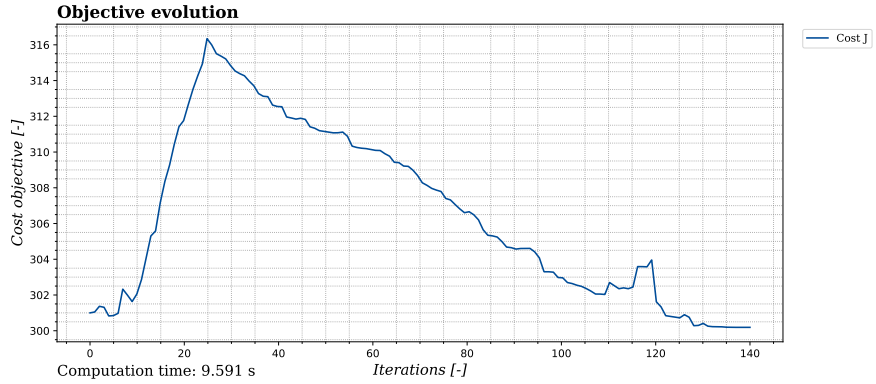
Figure 3.5: Control trajectories, benchmark with free-end condition. Controls included are TPS and elevator deflection. Own source.

As can be spotted on previous figures, there are no eye-sight differences between the state trajectories from this case with the previous benchmark case. That is because the

problem is, in essentia, the same. However, it can be pinpointed that endpoint of both state and control trajectories are slightly smoother and continuous. That minor change is consequence of adding free-end condition with a penalty weight in terminal cost because the optimiser can seek a better adjustment, reducing end-time. This reduction translates into a slightly modification of end states and controls which produce a bit of pitch down -invaluable in the flight path- that consume a bit of more fuel. Nonetheless, the important change has been produced in cost objective graph, which has increased due to terminal cost pressence -its profile is similar because running cost has not changed- and time computation reduction. This increase is unnoticeable in states or controls because the problem is a simple validation problem; however, this could provoke huge differences in more realistic scenarios because the optimiser could not minimise properly the cost functional when terminal cost is present and, therefore, it could lead to appreciable differences in states or controls.



(a) No free-end condition.



(b) Free-end condition.

Figure 3.6: Comparison between benchmark problem cost objective per iteration without or with free-end condition. Own source.

4 | Firefighting Manoeuvre: Modelling and Implementation

The following chapter includes a brief description of the firefighting manoeuvre that has to be modelled as an OCP in order to find the optimal trajectory to perform the water-discharge. Consequently, the NLP formulation and implementation is also included right after the OCP definition. Eventually, the whole algorithm's framework is presented.

4.1 User Case and CONOPS

Firstly, the user case has to be presented. As stated on the thesis' motivations, this project arises from internal needs in Singular Aircraft, specifically from the need to implement a more autonomous and safer planification of the water-discharge that the UAV Flyox has to follow during routinaire firefighting operations. This leads to the algorithm that is defined at the end of this chapter, starting from the following CONceptual OPerationS (CONOPS) diagram.

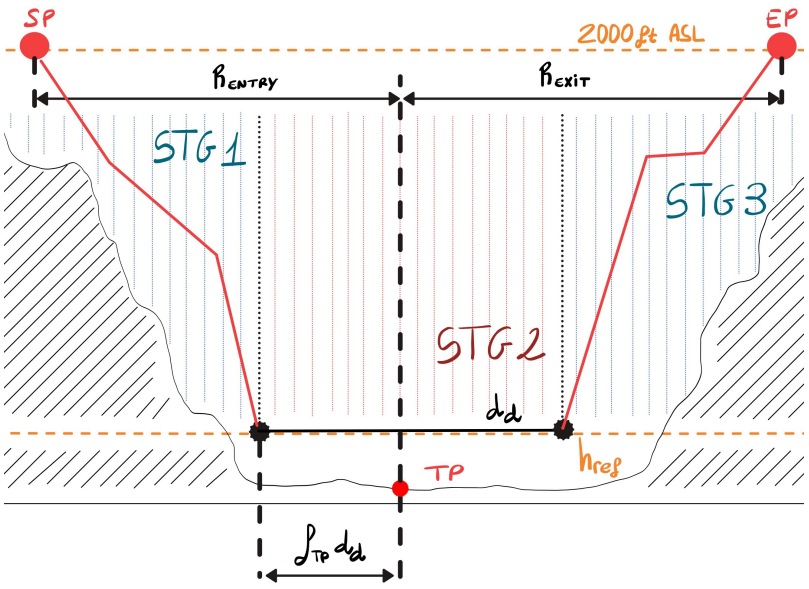


Figure 4.1: Flyox UAS autonomous firefighting generic CONOPS. Own source.

As can be seen, the UAV starts from a cruising level at 2000ft Above Sea Level (ASL) from the Starting Point (SP). First stage is a descent stage (STG1) from cruising level to reference altitude (h_{ref}), which is the altitude from where the discharge will be done to extinguish the flames, located at Target Point (TP). Once the aircraft is levelled at reference altitude, the discharge is performed along a pre-defined distance named discharge distance (d_d). TP is located at a known fraction f_{TP} of this preset distance. During the water-drop stage (STG2), the aircraft levels its flight, performs the discharge and correct its attitude to reach level-flight condition again. Thirdly, the UAV enters the climb stage (STG3) and climbs safely to cruising level 2000ft ASL, eventually reaching the Exit Point (EP) where the firefighting manoeuvre ends. It has to be pinpointed that the horizontal distances where SP and EP are located are defined radially from TP using an entry (R_{ENTRY}) and exit radius (R_{EXIT}), respectively.

All the mentioned parameters have been summarised in the following table. It should be noted that these parameters can be modified to accommodate any feasible scenario, supporting the main objective: a path planning algorithm designed to perform firefighting operations safely and with a higher grade of autonomy.

Table 4.1: Summary of the firefighting manoeuvre involved parameters per stages. Own source.

Parameter	Description	Stage	Units
$t_{2,min}$	Minimum time needed to perform the water-discharge safely.	2	[s]
$t_{2,max}$	Maximum time needed to perform the water-discharge safely.	2	[s]
t_d	Discharge time. Time during which discharge gates are open.	2	[s]
V_{TP}	Desired velocity to discharge safe and accurately.	2	[m/s]
d_d	Discharge distance. Bounded by $d_{2,min}$ and $d_{2,max}$. $d_{2,min} \leq d_d \leq d_{2,max}$	2	[m]
$d_{2,min}$	Minimum discharge distance needed. $d_{2,min} \coloneqq V_{TP} \cdot t_{2,min}$	2	[m]
$d_{2,max}$	Maximum discharge distance. $d_{2,max} \coloneqq V_{TP} \cdot t_{2,max}$	2	[m]
f_{TP}	Fraction of $d_{2,min}$ where TP is located.	2	[-]
h_{cruise}	Cruising level altitude.	-	[m]
h_{ref}	Level-flight altitude where water is discharged.	-	[m]
R_{ENTRY}	Horizontal distance between TP and SP.	1 & 2	[m]
R_{EXIT}	Horizontal distance between TP and EP.	2 & 3	[m]
TP	Target Point.	2	[-]
SP	Starting Point.	1	[-]
EP	Exit Point.	3	[-]

4.2 OCP Formulation and NLP Implementation

This section contains the formulation of each stage's manoeuvre as an OCP and the subsequent implementation as NLP separated problems. This means that for each stage a NLP problem will be defined and solved. Initial conditions for STG2 and STG3 will be the linking conditions for the overall manoeuvre, since they will be enforced to be the end-state from previous stage.

It has to be pinpointed that any of the stages dedicated sections will not contain the definition of the dynamic equations, because they are the same as the ones defined for the benchmark case, referring section 3.2.2 from expression (3.11) to (3.17). In case of the initial state constraints, its definition does not change and therefore, they will not be added on this section. Specifically, the only change is the initial value for state and control vector, which will be the linking condition between stages, but not its formal definition. Moreover, state and control vectors -and simple bounds by extension- will not be specified, since they are the same as the ones explained for free-end addition to benchmark problem; see expression (3.36).

Alternatively, the path constraints and cost functionals are the major changes per stage. Eventually, new constraint vector, constraint bounds vectors and cost functional will be defined per stage. Wording will focus on that. Eventually, each stage constitutes a free-end NLP problem using expression (3.40).

4.2.1 Descent stage (STG1)

Descent stage is characterised by a negative flight path angle ($\gamma \leq 0$) that enforces the aircraft to decrease its altitude. In terms of the manoeuvre, it is also characterised by initial state -given by the user initial conditions- and an end state at known altitude and horizontal distance. Initial state computation will be introduced at section 4.3 because, as said before, it does not have effect on mathematical formulation.

Then, path constraints for the descent stage are related with aerodynamic velocity limits -stall velocity V_S and never-operate velocity V_{NO}- and with end-time altitude and horizontal distance, which have to be proper, respectively. Translating to expressions it can be written like:

$$g_{path,1} \coloneqq \begin{cases} V_S - \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} \leq 0 \\ \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} - V_{NO} \leq 0 \\ -z^N - h_{ref} = 0 \\ (x^N - x^0) - (R_{ENTRY} - f_{TP} \cdot d_{2,min}) \leq 0 \end{cases} \tag{4.1}$$

Converting the expressions to NLP notation, the equations can be stated as follows. In addition, $\mathbf{d}$ and $\boldsymbol{\varphi_0}$ are the dynamic constraints vector and initial state constraints vector, respectively -see expression (3.32) and its meaning-.

$$\mathbf{h}_1 \coloneqq \begin{bmatrix} V_S - \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} \\ \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} - V_{NO} \end{bmatrix} \leq 0 \tag{4.2}$$

$$\mathbf{h}_{1,end} \coloneqq \begin{bmatrix} -x_6^N - h_{ref} = 0 \\ (x_5^N - x_5^0) - (R_{ENTRY} - f_{TP} \cdot d_{2,min}) \leq 0 \end{bmatrix} \tag{4.3}$$

$$\mathbf{g}(\mathbf{w}) \coloneqq \begin{bmatrix} \mathbf{d}^0 \\ \vdots \\ \mathbf{d}^{N-1} \\ \hline \mathbf{h_1}^0 \\ \vdots \\ \mathbf{h_1}^N \\ \hline \mathbf{h_{1,end}} \\ \hline \varphi_0 \end{bmatrix}, \quad \mathbf{g}_{lb} \coloneqq \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ -\infty_{2N} \\ \mathbf{0}_1 \\ -\infty_1 \\ \mathbf{0}_7 \end{bmatrix}, \quad \mathbf{g}_{ub} \coloneqq \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ \mathbf{0}_{2N} \\ \mathbf{0}_1 \\ \mathbf{0}_1 \\ \mathbf{0}_7 \end{bmatrix} \tag{4.4}$$

Once the constraints have been defined, it is time to express the cost functional that governs descent stage. In this case, it includes both running and terminal cost terms. Terminal cost only includes the end-time penalisation, as expressed in equation (4.10). Referring to running cost, it is made by four (4) different penalties, which are the following ones.

- Firstly, the minimisation of error between the current and desired flight path angle $(\gamma - \gamma_d)$ is introduced as a quadratic penalty in the objective function. This penalty encourages the solver to mantain the desired descent rate whenever possible. The expression introduced in the running cost is the following one, where $\bar{\gamma}$ is the normalised γ term using γ_d -which is in fact γ_{min}-.

$$J_\gamma \coloneqq (\bar{\gamma} - 1)^2 = \left(\frac{\gamma - \gamma_d}{\gamma_d}\right)^2 = \left(\frac{\theta - \alpha - \gamma_d}{\gamma_d}\right)^2 = \left(\frac{x_4 - \alpha(x_1, x_2)}{\gamma_d} - 1\right)^2 \tag{4.5}$$

- Secondly, the minimisation of flight path angle rate $(\dot{\gamma})$ has been added too in order to prevent huge oscillations and to provide smoothness on flight angle profile. This penalty was already used in benchmark problems and the only change introduced is the type of normalisation done on the penalty term.

$$J_{\dot{\gamma}} \coloneqq \left(q - \frac{\dot{w}u - \dot{u}w}{u^2 + w^2}\right)^2 / \dot{\gamma}_{min}^2 = \left(x_3 - \frac{\dot{x}_2 x_1 - \dot{x}_1 x_2}{x_1^2 + x_2^2}\right)^2 / \dot{\gamma}_{min}^2 \tag{4.6}$$

- Eventually, control variable rates minimisation have been introduced using a quadratic penalty, as has been done in benchmark problems.

$$J_{\dot{\delta}_{TPS}} := \frac{\left(\delta_{TPS}^{k+1} - \delta_{TPS}^{k}\right)^2}{\Delta t} = \frac{\left(u_1^{k+1} - u_1^{k}\right)^2}{\Delta t} \tag{4.7}$$

$$J_{\dot{\delta}_e} := \frac{\left(\delta_e^{k+1} - \delta_e^{k}\right)^2}{\delta_{e,max}^2 \Delta t} = \frac{\left(u_2^{k+1} - u_2^{k}\right)^2}{\delta_{e,max}^2 \Delta t} \tag{4.8}$$

Then, the formal expression of running cost and overall cost objective for descent stage are the following ones.

$$L(x,u) := \sum_{k=0}^{N-1} \left(w_\gamma \cdot J_\gamma + w_{\dot{\gamma}} \cdot J_{\dot{\gamma}} + w_{\dot{\delta}_{TPS}} \cdot J_{\dot{\delta}_{TPS}} + w_{\dot{\delta}_e} \cdot J_{\dot{\delta}_e} \right) \tag{4.9}$$

$$\begin{aligned} L^k &:= L(\mathbf{x}^k, \mathbf{u}^k) \\ J(\mathbf{w}) &:= \frac{\Delta t}{2} \sum_{k=0}^{N-1} \left(L^k + L^{k+1} \right) + w_{t_{F,1}} \cdot t_{F,1} \end{aligned} \tag{4.10}$$

4.2.2 Level-flight stage with water-discharge (STG2)

The second stage is characterised by a level-flight cruise stage where a sudden change of mass is produced -water is discharged over the fire flames-. The rest of properties of this stage are the ones defined on benchmark problem, which was essentially level-flight condition ($\gamma = 0$). Thus, no changes have been made to the NLP formulation, in terms of path constraints or cost objective function.

The only change is produced at mass related dynamic constraint, all other dynamic constraints remain equal. Since there is a sudden mass variation, the dynamic equation associated to $\dot{m}$ has to reflect this variation. After testing multiple solutions and approaches, it has been decided to introduce a piecewise-defined function that incorporates water discharge only based on the detection that the aircraft is above the discharge point. It can be mathematically defined as:

$$\dot{x}_7 := \begin{cases} -SFC & x_5 < [V_{TP} \cdot (f_{TP} \cdot t_{2,min} - t_d/2)] \\ -SFC & x_5 > [V_{TP} \cdot (f_{TP} \cdot t_{2,min} + t_d/2)] \\ -SFC - \frac{PM}{t_d} & \text{otherwise} \end{cases} \tag{4.11}$$

Function above can be explained using distances because each stage has been defined as a separated problem and, therefore, distances are realtive. Essentially, the mass variation is only SFC when the UAV is not discharging water. In addition, discharge rate is applied only when target point (TP) is detected and only for a period equal to discharge time (t_d). In other words, once the problem starts, discharge is initiated at a distance equal to TP and it is finalised once the open-gates distance is over, defining this distance as the flying distance from $t = f_{TP} \cdot t_{2,min}$ to $t = f_{TP} \cdot t_{2,min} + t_d$.

4.2.3 Climb stage (STG3)

Climb stage is the antagonist of descent stage; thus, it is defined by a positive flight path angle ($\gamma \geq 0$) that enforces gaining altitude. As happened in the previous two stages, it is also characterised by a end state. In addition, it has to be pinpointed that climb stage is the most sensitive stage in the whole manoeuvre because velocity can easily be near the stall velocity and sometimes controls tend to the limit in order to achieve the objective.

Similarly to descent stage, the major changes that have been produced in path constraints and in cost objective function. Therefore, state and control vector, dynamic constraints vector and initial state constraints vector remain equal to the ones defined in section 3.2.2 and they will only appear reffered on this section.

In this case, the path constraints are similar to the ones defined in descent stage but with minor changes in end-time constraints. Following the structure previously used, the path constraints in OCP notation are:

$$g_{path,3} \coloneqq \begin{cases} V_S - \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} \leq 0 \\ \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} - V_{NO} \leq 0 \\ -z^N - h_{ref} = 0 \\ (x^N - x^0) - [R_{EXIT} - (1 - f_{TP}) \cdot d_{2,min}] \leq 0 \end{cases} \tag{4.12}$$

Then, applying the transformation into NLP notation it rests like the expressions below. As explained before, $\mathbf{d}$ and $\boldsymbol{\varphi_0}$ are the dynamic constraints vector and initial state constraints vector, respectively -see expression (3.32) and its meaning-.

$$\mathbf{h}_3 \coloneqq \begin{bmatrix} V_S - \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} \\ \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} - V_{NO} \end{bmatrix} \leq 0 \tag{4.13}$$

$$\mathbf{h}_{3,end} \coloneqq \begin{bmatrix} -x_6^N - h_{ref} = 0 \\ (x_5^N - x_5^0) - [R_{EXIT} - (1 - f_{TP}) \cdot d_{2,min}] \leq 0 \end{bmatrix} \tag{4.14}$$

$$\mathbf{g}(\mathbf{w}) \coloneqq \begin{bmatrix} \mathbf{d}^0 \\ \vdots \\ \mathbf{d}^{N-1} \\ \hline \mathbf{h_3}^0 \\ \vdots \\ \mathbf{h_3}^N \\ \hline \mathbf{h_{3,end}} \\ \hline \varphi_0 \end{bmatrix}, \quad \mathbf{g}_{lb} \coloneqq \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ -\infty_{2N} \\ \mathbf{0}_1 \\ -\infty_1 \\ \mathbf{0}_7 \end{bmatrix}, \quad \mathbf{g}_{ub} \coloneqq \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ \mathbf{0}_{2N} \\ \mathbf{0}_1 \\ \mathbf{0}_1 \\ \mathbf{0}_7 \end{bmatrix} \tag{4.15}$$

Next, the terms that form part of the running cost are explained. In this case, running cost is made up to five (5) different penalties.

- In first place, the minimisation of flight path angle rate ($\dot{\gamma}$) has been added too in order to prevent huge oscillations and to provide smoothness on flight angle profile, as done in the previous stages. In this case, normalisation of flight path angle rate has been done using γ_{max}, as done in benchmark problem and in level-flight stage (STG2).

$$J_{\dot{\gamma}} \coloneqq \left(q - \frac{\dot{w}u - \dot{u}w}{u^2 + w^2}\right)^2 / \dot{\gamma}_{max}^2 = \left(x_3 - \frac{\dot{x}_2 x_1 - \dot{x}_1 x_2}{x_1^2 + x_2^2}\right)^2 / \dot{\gamma}_{max}^2 \tag{4.16}$$

- In second place, an extra protection has been added in terms of the stall angle (α_{stall}). In this case, α has been penalised quadratically once it enters the warning zone -defined using a 25% safety margin in stall angle-. To prevent and avoid angle-of-attack stall, the solver is forced to prioritise lift-force over agressive manoeuvres. It ensures the aircraft maintains a sufficient margin from the aerodynamic stall regime during the ascent stage. For the penalty defined below, margin angle is $\alpha_{mg} \coloneqq 0.25 \cdot \theta_{max}$ and safe angle is $\alpha_{sf} \coloneqq 0.75 \cdot \theta_{max}$ -in this case θ_{max} has been used as α_{stall} for simplicity-.

$$J_{sp} \coloneqq \left(\frac{\max[0, \alpha(u, w) - \alpha_{sf}]}{\alpha_{mg}}\right)^2 = \left(\frac{\max[0, \alpha(x_1, x_2) - \alpha_{sf}]}{\alpha_{mg}}\right)^2 \tag{4.17}$$

- Thirdly, control variable rates minimisation have been introduced using a quadratic penalty, as has been done in previous cases.

$$J_{\dot{\delta}_{TPS}} \coloneqq \frac{\left(\delta_{TPS}^{k+1} - \delta_{TPS}^{k}\right)^2}{\Delta t} = \frac{\left(u_1^{k+1} - u_1^{k}\right)^2}{\Delta t} \tag{4.18}$$

$$J_{\dot{\delta}_e} \coloneqq \frac{\left(\delta_e^{k+1} - \delta_e^{k}\right)^2}{\delta_{e,max}^2 \Delta t} = \frac{\left(u_2^{k+1} - u_2^{k}\right)^2}{\delta_{e,max}^2 \Delta t} \tag{4.19}$$

- In last place, a saturation penalty has been introduced in elevator deflection ($\delta_{e,sat}$). As can be seen, it has a higher power because it is a soft barrier function. Unlike a quadratic penalty, this penalty remains very low when the elevator is near its neutral position but increases explosively as it approaches its physical limits ($\delta_{e,max}$), enforcing the solver to not reach those limits frequently. In other words, it prevents the solver to reach a solution that starts applying a lot of elevator deflection to win altitude faster or that prioritises pull-ups -heavy-load manoeuvres- very often.

$$J_{\delta_{e,sat}} \coloneqq \left(\frac{\delta_e}{\delta_{e,max}}\right)^4 = \left(\frac{u_2}{\delta_{e,max}}\right)^4 \tag{4.20}$$

Then, the formal expression of running cost and overall cost objective for climb stage are the following ones.

$$L(x,u) := \sum_{k=0}^{N-1} \left(w_{\dot{\gamma}} \cdot J_{\dot{\gamma}} + w_{sp} \cdot J_{sp} + w_{\dot{\delta}_{TPS}} \cdot J_{\dot{\delta}_{TPS}} + w_{\dot{\delta}_e} \cdot J_{\dot{\delta}_e} + w_{\dot{\delta}_{e,sat}} \cdot J_{\dot{\delta}_{e,sat}} \right) \tag{4.21}$$

$$\begin{aligned} L^k &:= L(\mathbf{x}^k, \mathbf{u}^k) \\ J(\mathbf{w}) &:= \frac{\Delta t}{2} \sum_{k=0}^{N-1} \left(L^k + L^{k+1} \right) + w_{t_{F,3}} \cdot t_{F,3} \end{aligned} \tag{4.22}$$

4.3 Algorithm General Framework

Finally, the algorithm developed can be summarised in the following algorithm lines, see next page. As can be seen, the main structure consists of different files and functions. Essentially, it is possible to distinguish between:

- *simulation.py* is the function that extracts the information storaged in JSON file *Simulation.json*. This file contains the information and problem parameters defined in 4.1, as well as the mission bounds. In addition, numerical values and the aircraft model can be selected. Eventually, wind conditions could be added -they have been not used in any of the results for this thesis, which means that all results have been generated in a calm atmosphere-.

- *aircraft.py* is a function that loads and process all information storage in a determined aircraft model JSON file. It is called by *simulation.py* using the JSON file path of the aircraft model. It also contains a function to compute maximum thrust and torque at each iteration when the solver is running.

- *atmosphere.py* is another function that loads atmospheric conditions from *Atmos.json* and that contains an ISA density computation function used during the solver iterations.

- *plotterfunction.py* is the postprocessing function to generate all trajectories and cost plots. It is made by different functions to plot the whole manoeuvre trajectories and cost objective or, individual trajectories and cost per stage. It is a clean solution used to not overload main function with postprocessing activities.

- *nonlinearprogramming.py* is the function where all NLP dictionaries are created. Essentially, there are different functions to create state and control vectors, constraints vectors, bounds vectors and cost objective function for all symbolic variables. For each stage, a class has been created with the different functions and they only had to be called from main algorithm to create the vectors and dictionaries.

Algorithm 4 Main algorithm (*main_IPOPT.py*)

1. Problem's Configuration:

Using *Simulation.json*.

Define initial conditions, mission parameters, mission bounds and aircraft file.

Define number of nodes N and penalty weights for each stage.

Retrieving of aircraft and atmosphere parameters.

2. Compute initial conditions at SP:

Computation of initial state and control vector $\mathbf{w_0}$.

$(V_0, \theta_0) \longrightarrow u_0, w_0, \theta_0$.

Fix $q_0 = 0$, $x_0 = 0$, $z_0 = -h_{cruise}$ and $m_0 = TOM$. $\mathbf{x_0} \coloneqq [u_0, w_0, q_0, \theta_0, x_0, z_0, m_0]$.

Using a static NLP solver obtain initial controls $\mathbf{u_0} \coloneqq [\delta_{TPS,0}, \delta_{e,0}]$.

Stack variables to create STG1 $\mathbf{w_0} \coloneqq [\mathbf{x}_0^0, \mathbf{u}_0^0, \ldots, \mathbf{x}_0^N, \mathbf{u}_0^N, t_1^0]$.

3. NLP Creation for STG1:

Formulate $\mathbf{w} \coloneqq [\mathbf{x}^0, \mathbf{u}^0, \ldots, \mathbf{x}^N, \mathbf{u}^N, t_F]^\top \in \mathbb{R}^{9N+1}$.

Apply variable bounds: $\mathbf{w} \in \begin{bmatrix} \mathbf{1}_N \otimes \mathbf{lbx} \\ t_{F,min} \end{bmatrix}, \begin{bmatrix} \mathbf{1}_N \otimes \mathbf{ubx} \\ t_{F,max} \end{bmatrix}$.

Compute trapezoidal Cost: $J(\mathbf{w}) = \frac{\Delta t \cdot t_F}{2} \sum_{k=0}^{N-1} (L^k + L^{k+1}) + w_{t_F} \cdot t_F$.

Evaluate scaled dynamics $\mathbf{d}^k$ and path constraints $\mathbf{h}^k$.

Stack constraints: $\mathbf{g}(\mathbf{w}) \coloneqq [\mathbf{d}^0, \ldots, \mathbf{d}^{N-1}, \mathbf{h}^0, \ldots, \mathbf{h}^N, \varphi_0]^\top$.

4. Solve STG1:

IPOPT solver using $N_{iter} = 3000$, $\epsilon_{tol} = 10^{-6}$ and $\epsilon_{acc.tol.} = 10^{-6}$.

Extract state and control trajectories and cost objective per iteration.

5. NLP Creation for STG2:

Extract end states from STG1 $\mathbf{x_f} \coloneqq \mathbf{w_1^N} \in \mathbb{R}^7$.

Using a static NLP solver obtain initial controls $\mathbf{u_0} \coloneqq [\delta_{TPS,0}, \delta_{e,0}]$.

Generate linking condition $\mathbf{w_0} \coloneqq [\mathbf{x_f}^0, \mathbf{u}_0^0, \ldots, \mathbf{x_f}^N, \mathbf{u}_0^N, t_2^0]$.

Create $\mathbf{w}$, $\mathbf{g}$, $J(\mathbf{w})$ and bounds using NLP_STG2 class. Repeat Step 3.

6. Solve STG2:

Repeat Step 4 but with STG2 NLP dictionary.

7. NLP Creation for STG3:

Extract end states from STG2 $\mathbf{x_f} \coloneqq \mathbf{w_2^N} \in \mathbb{R}^7$.

Using a static NLP solver obtain initial controls $\mathbf{u_0} \coloneqq [\delta_{TPS,0}, \delta_{e,0}]$.

Generate linking condition $\mathbf{w_0} \coloneqq [\mathbf{x_f}^0, \mathbf{u}_0^0, \ldots, \mathbf{x_f}^N, \mathbf{u}_0^N, t_3^0]$.

Create $\mathbf{w}$, $\mathbf{g}$, $J(\mathbf{w})$ and bounds using NLP_STG3 class. Repeat Step 3.

6. Solve STG3:

Repeat Step 4 but with STG3 NLP dictionary.

7. Post-processing:

Reconstruction of whole manoeuvre state and control trajectories.

Stack all states and controls.

Horizontal position: $\mathbf{x_5} = [\mathbf{x_{5,STG1}}, \mathbf{x_{5,STG2}} + x_{5,STG1}^N, \mathbf{x_{5,STG3}} + x_{5,STG2}^N]$.

Generate time vectors using end-time values and linspace function.

Generate trajectories and cost objectives plots using *plotterfunction.py*.

5 | Firefighting Manoeuvre: Results

Next chapter includes a comprehensive analysis of the results obtained for the different simulations performed. Since the firefighting manoeuvre is a multi-parameter optimisation problem, only a few set of parameter variants have been tried. Then, a full review has been done and some remarks of the results have been added.

5.1 Simulations

In order to compare a few things among the different simulations, it is important to describe a base set of parameters to stablish later comparisons. The following table summarises the value of the parameters set on *Simulation.json*. The aircraft used for this simulation and the following ones has been the UAV Flyox, which parameters can be seen on appendix C table C.1. In addition, all simulations have been performed with 150 collocation points -or nodes-, which are enough to capture water-discharge stage.

Table 5.1: Base case simulation parameters. See reference in appendix B, *Simulation.json*. Own source.

Parameter	**Value**	**Units**
N -collocation points-	150	[-]
Initial State -or guesses-	[50.00, 0.07, 20.00, 40.00, 60.00]	[m/s, rad, s, s, s]
Wind Speed	[0.00, 0.00]	[m/s]
Target Point parameters	[10.00, 10.00, 0.35, 45.00, 1.0]	[km, km, -, m/s, s]
Mission Bounds	LB: [5.00, 20.00, 40.00, -608.00, -5.08] UB: [360.00, 180.00, 360.00, -15.00, 5.08]	[s, s, s, m, m/s]
Weights STG1	[0.0, 0.3, 0.2, 1.0, 1.5]	[-]
Weights STG2	[0.0, 0.5, 0.3, 0.5, 1.0, 1.5]	[-]
Weights STG3	[0.0, 0.2, 2.0, 1.0, 1.5, 1.0]	[-]

Once all parameters have been properly set, the state and control trajectories are the following ones. As can be seen on figure 5.1, the water-discharge has been modelled as a discontinuity for this base case -indeed, it is a one-second discharge but, since PM

is two-hundred (200) kg, it can be suddenly interpreted-. Descent and climb stages are almost continuous, as can be seen by nearly constant value on flight path angle (γ) or directly on the UAV trajectory. In terms of velocity, its profile may be suprisingly, but it denotes a key aspect of Flyox behaviour: longitudinal stability is closely coupled, as can be seen on velocities subgraph -difference between body velocity along x_b-axis (u) and velocity modulus V-. Thus, the aircraft tends to align the descent or ascent path, provoking changes on velocity's module. In terms of controls, discontinuities are unavoidable because they correspond with the jump between stages; nevertheless, there are discontinuities affordable to a control feedback loop proper of an autopilot on an UAV. They will not be further mentioned on the report. Moreover, control seems smooth in each stage and they are far from the operational limits. Essentially, controls reach a trim point in each stage, more or less.

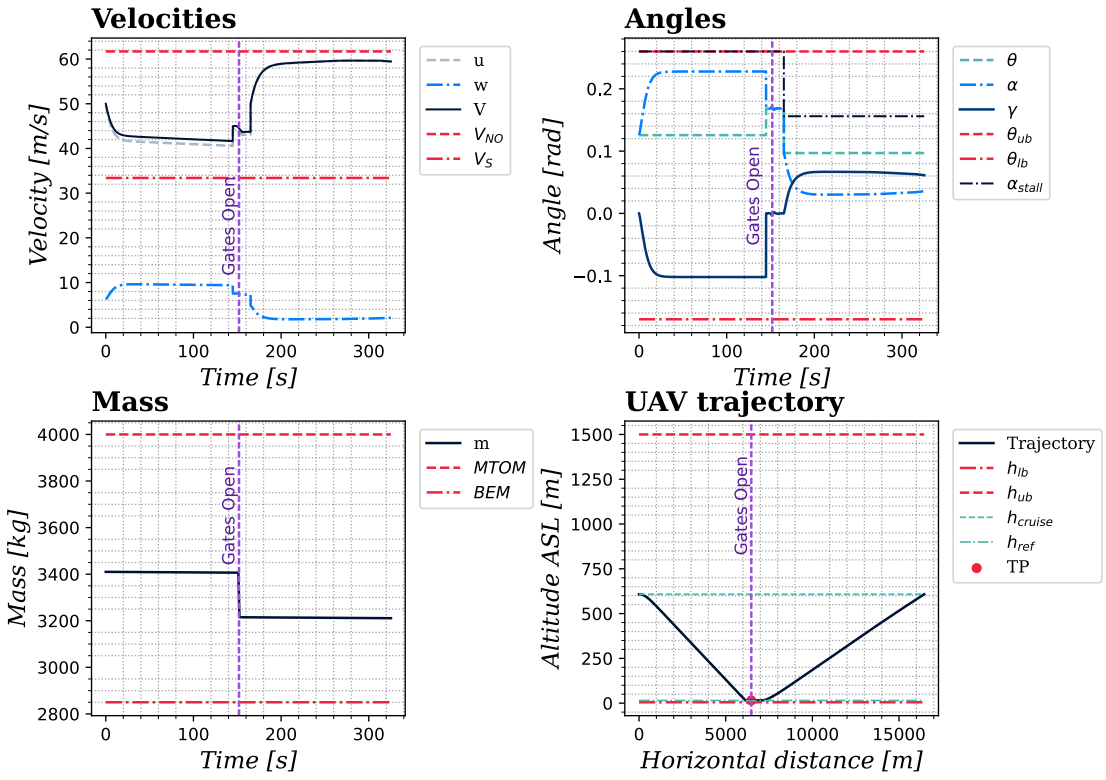


Figure 5.1: State trajectories, base case. States included are velocities, angles, mass and UAV trajectory. Own source.

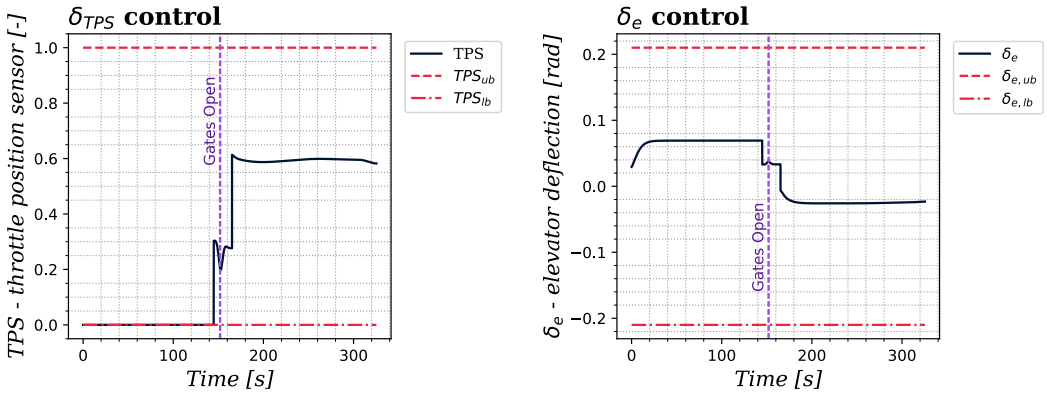


Figure 5.2: Control trajectories, base case. Controls included are TPS and elevator deflection. Own source.

Following figure shows the evolution of cost objective per iteration for each stage. As can be seen, the demanding stages in terms of cost are the most restrictive ones: descent and climb stages, specifically descent one. This means that some constraints have been breached or nearly breached -i.e the solution found in the iteration which cost is highest is not feasible, constraints or penalties have been not fulfilled and the control turns agressive to avoid constraint breach-. After this, control variables tend to be smoother, because cost is slowly relaxed until reaching the final objective near zero value.

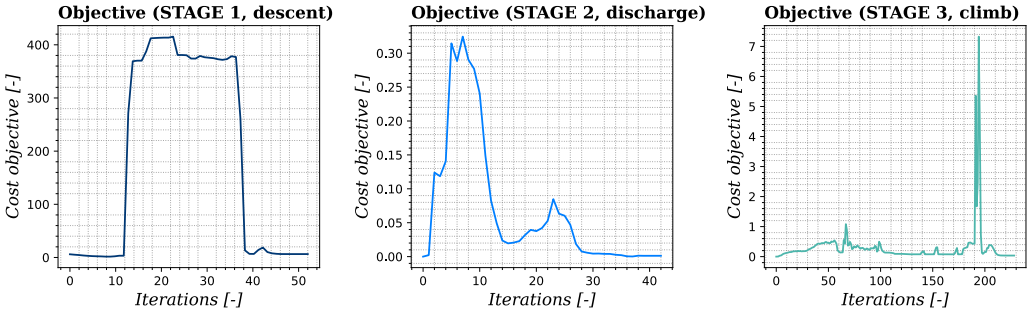


Figure 5.3: Cost objective per iteration, base case. Each stage cost is represented. Own source.

5.1.1 Case 1: continuous water-discharge

This case has been done using the same parameters that have been indicated on table 5.1 except for the discharge time (t_d), which has been incremented to perform a more continuous and controlled discharge above TP. The discharge time has been set to a value equal to $t_d = 7.2\ s$, which is near to firefighting controlled drops -as specified by Singular Aircraft engineers-.

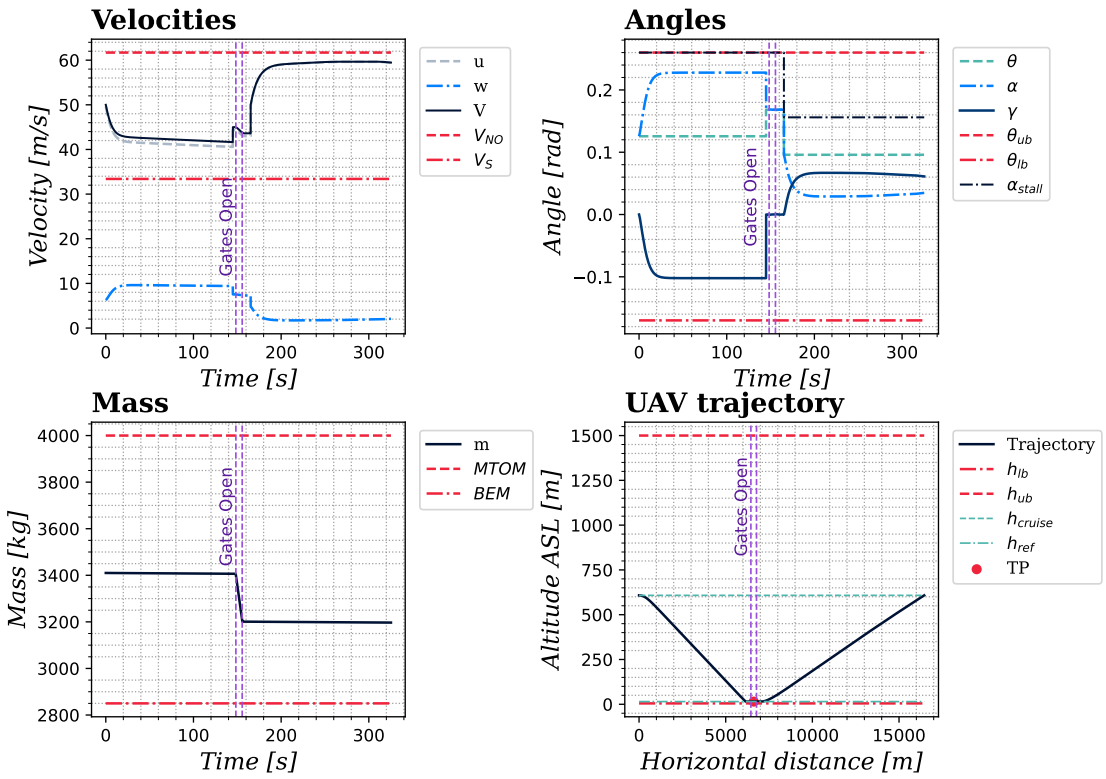


Figure 5.4: State trajectories, case 1. States included are velocities, angles, mass and UAV trajectory. Own source.

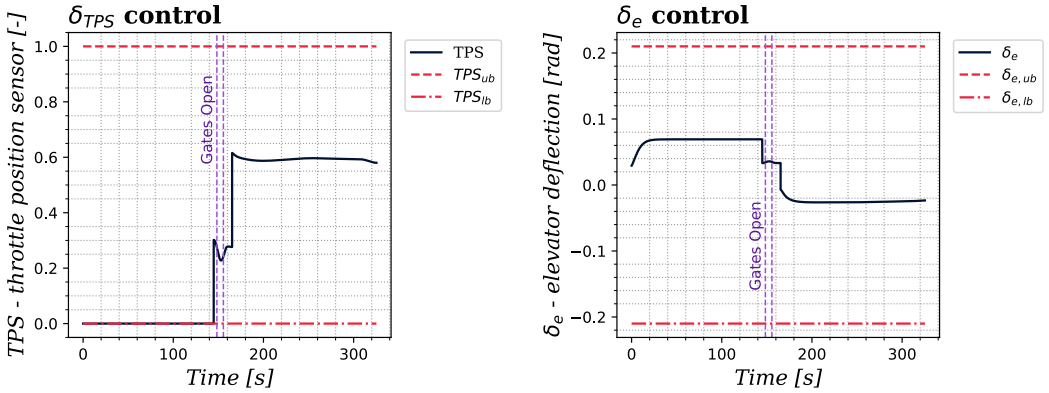
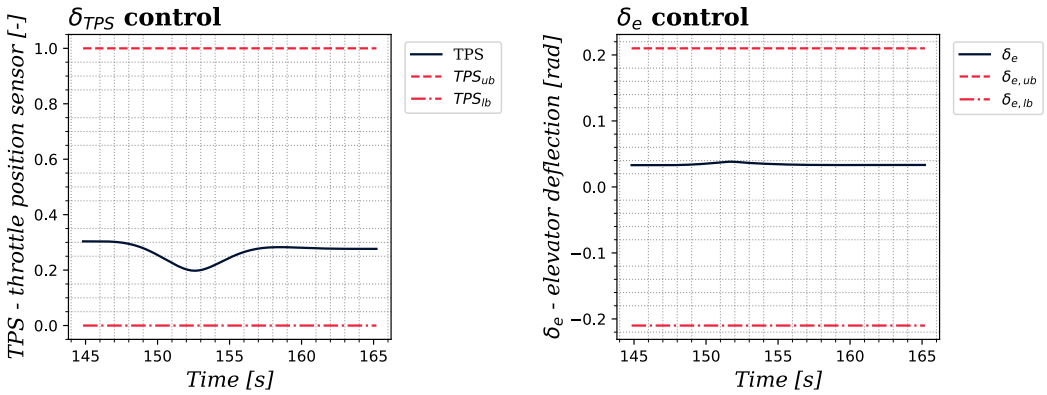


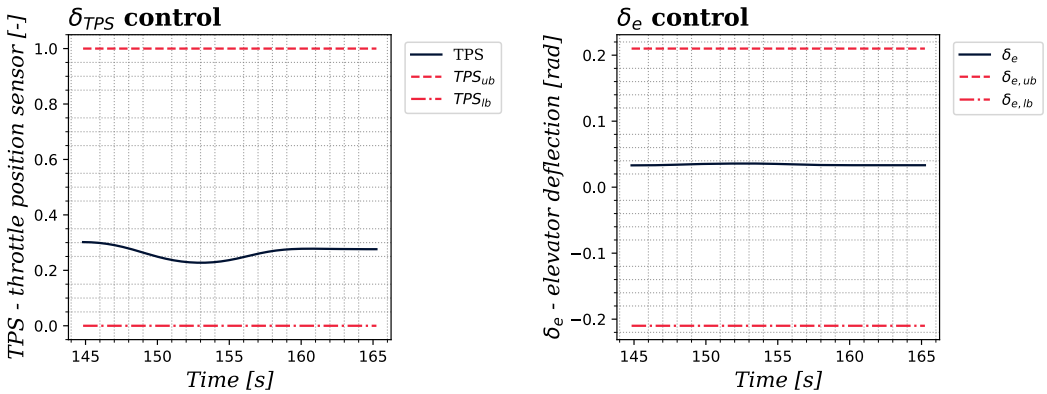
Figure 5.5: Control trajectories, case 1. Controls included are TPS and elevator deflection. Own source.

As can be seen on previous figures, there are no noticeable changes on state trajectories. However, there is a minor change in term of controls. Since the gates are open for a longer

period of time, control corrections are soft and smooth than in the previous case. The following figure shows a comparison between the second stage control from base case -top- and this case -bottom-. Eventually, this smoother controls are traduced in a reduction of the flight path angle rate ($\dot{\gamma}$) during discharge phase because vertical speed (w) is more stable during the discharge, as can be seen in state comparison just after controls comparison.

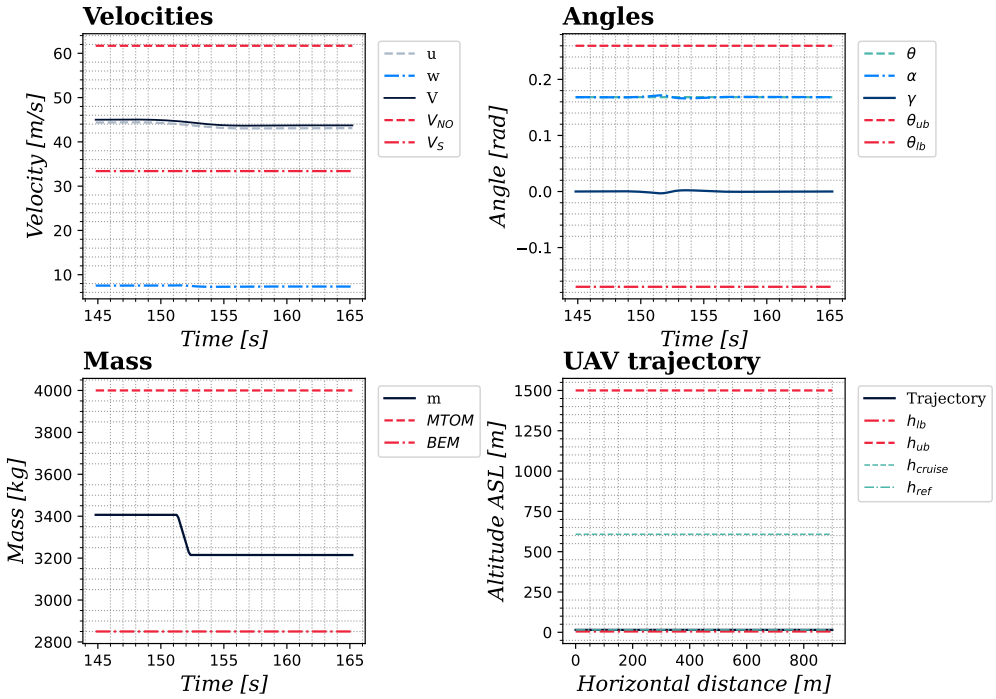


(a) Base case, discontinuous discharge. $t_d = 1.0\ s$

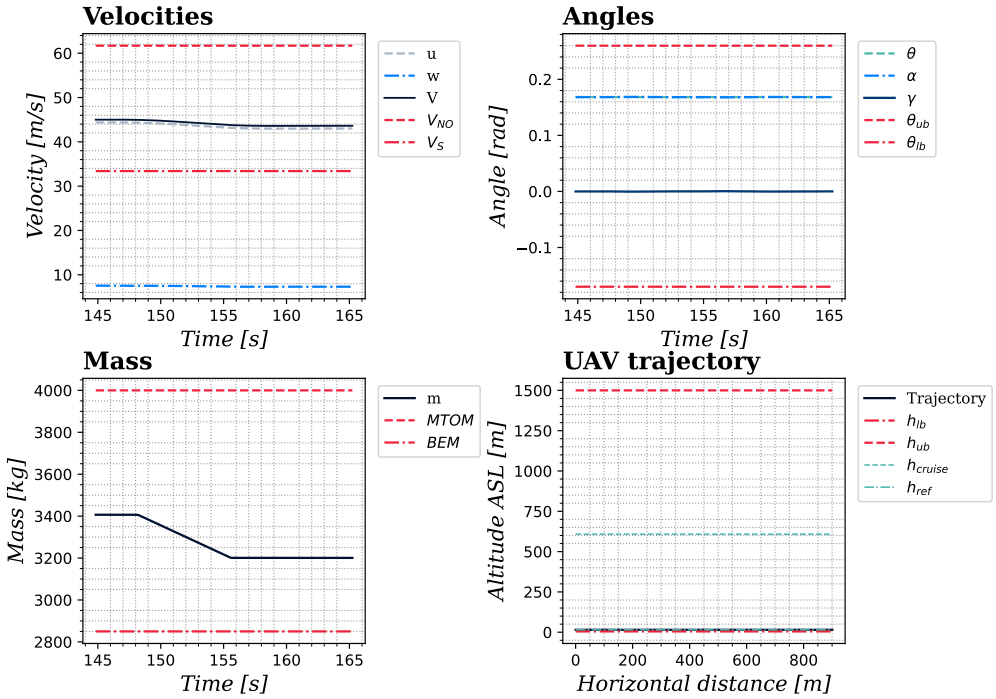


(b) Case 1, continuous discharge. $t_d = 7.2\ s$

Figure 5.6: Comparison between control trajectories during discharge stage for different discharge times (t_d). Own source.



(a) Base case, discontinuous discharge. $t_d = 1.0\ s$



(b) Case 1, continuous discharge. $t_d = 7.2\ s$

Figure 5.7: Comparison between state trajectories during discharge stage different discharge times (t_d). Own source.

5.1.2 Case 2: restrictive distances and end-time penalties

Next simulation tries to see which are the effects when entry and exit distances -i.e R_{ENTRY} and R_{EXIT}- are constrained to the limit. From base case it has been seen that minimum needed distances for $f_{TP} = 0.35$ are $R_{ENTRY} = 6500\ m$ and $R_{EXIT} = 9800\ m$, respectively. This simulation uses this values to restrict the movement and adds penalty weight to end-time penalisations -$w_{t_{F,1}} = w_{t_{F,2}} = w_{t_{F,3}} = 0.5$-.

As can be seen on the graph below, the huge change respect to base case appears during climb stage. In this case, it seems that transitions between stages are more abrupt in terms of velocity (V), flight path angle (γ) and angle-of-attack (α) although stationary moudulus seems equal in all cases. The reason after this behaviour is TPS control, which experiments a quick change between stage 2 and stage 3. Specifically, it reaches upper bound, holds for a few seconds and then drops until it reaches trim point. Nevertheless, this peak was not present on base case and, the transition was smoother -see figure 5.2-. In terms of elevator deflection, both trajectories from base case and this case are similar. Eventually, it could be said that pull-up manoeuvre at the begin of stage 3 is agressive because of the request to minimise end-time -the solver aims to reach exit point (EP) the fastest-.

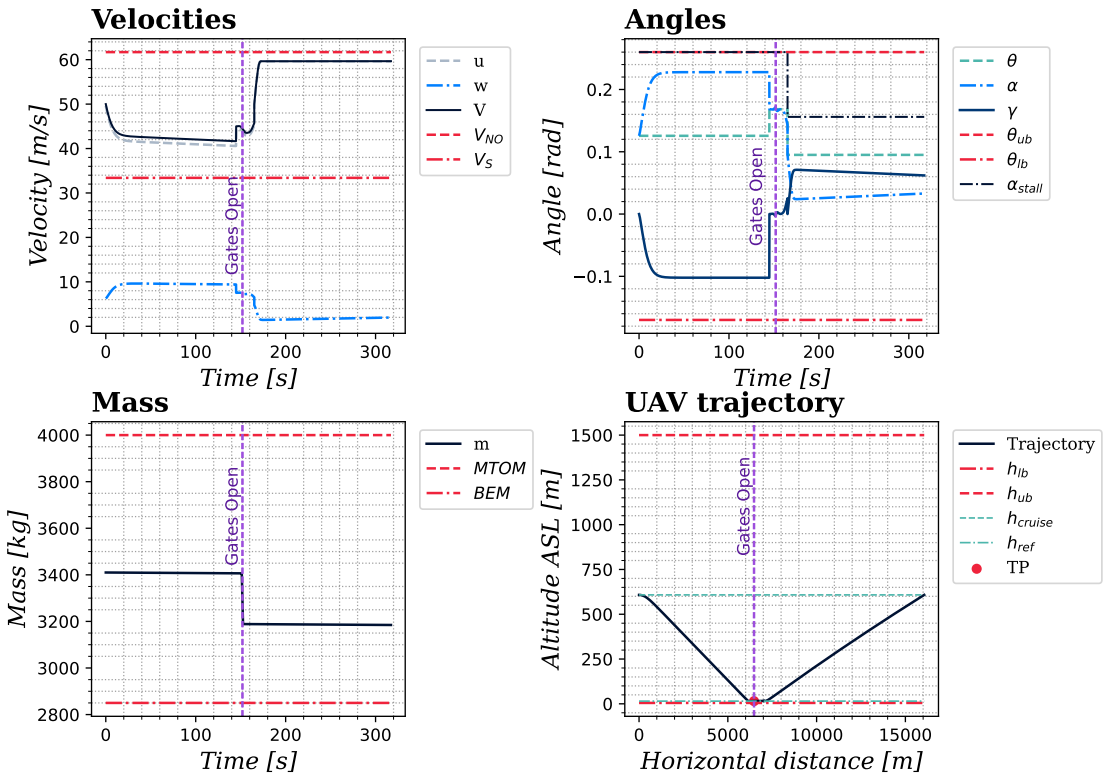


Figure 5.8: State trajectories, case 2. States included are velocities, angles, mass and UAV trajectory. Own source.

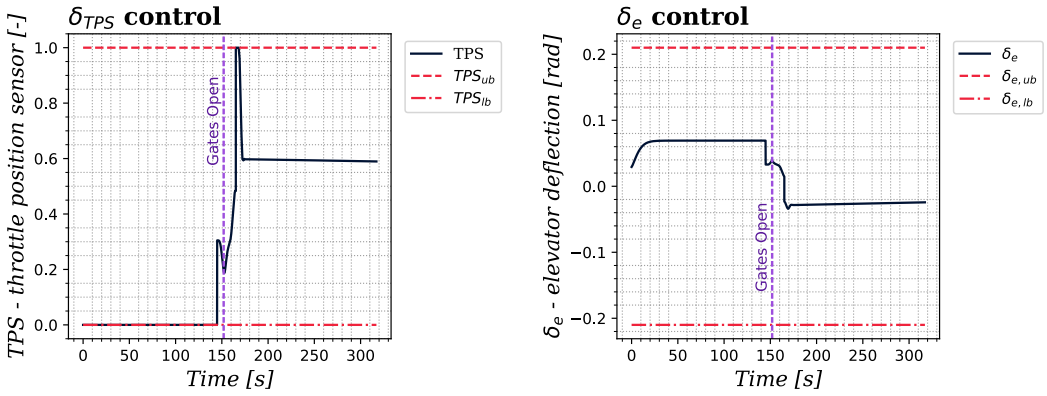


Figure 5.9: Control trajectories, case 2. Controls included are TPS and elevator deflection. Own source.

It has to be pinpointed that in this case, penalty weight on terminal cost has been added. Therefore, this has produced some changes in cost objective plots. The shape of the objective profile has changed for both stages 2, the discharge, and 3, the climb. This change could be due to the change in distance constraints when entry and exit distances have been modified. Despite the shape change in cost objective profiles, the most noticeable change is in cost objective magnitude for all stages, which is a result from adding a terminal penalty directly related to end-time value -usually a higher value-.

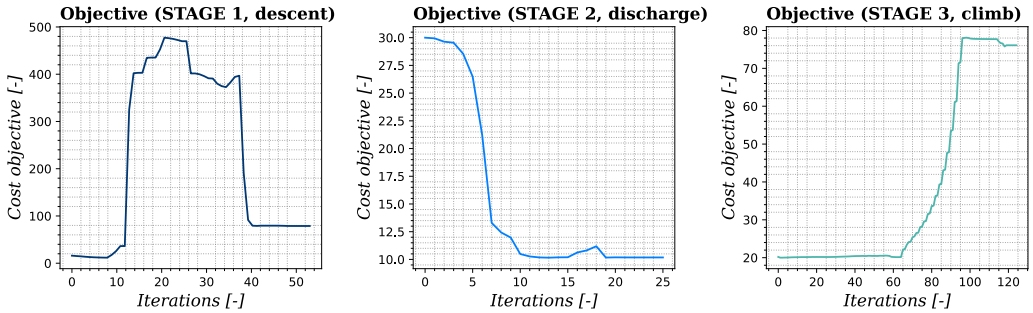


Figure 5.10: Cost objective per iteration, case 2. Each stage cost is represented. Own source.

5.1.3 Case 3: controls and desired attitudes not penalised

This section contains a simulation where all related control weights have been set to zero -$w_{\dot{\delta}_{TPS}} = w_{\dot{\delta}_e} = 0.0$- in all stages. In addition, the weight associated to the desired flight path angle error$(\gamma - \gamma_d)$ during descent stage and, stall protection and elevator deflection saturation weights during ascent stage have been set to zero too. In other words, this simulation is trying to minimise flight path angle rate $(\dot{\gamma})$ -during all stages- and altitude error $(h - h_{ref})$ -during second stage-. No terminal cost has been considered

because end-time penalty weights are null this time. The rest of parameters remain equal as specified in table 5.1 except from distances, which have been set to high values $R_{ENTRY} = R_{EXIT} = 20000\ m$, and end-time upper bound, which have been set to $t_{F,ub} = 600\ s$ for each stage. In this case, a $t_d = 7.2\ s$ has been used for visualisation purposes, so comparison will be done against case 1.

The following states and control trajectories are very different from figures 5.4 and 5.5. Firstly, manoeuvre is longer in distance and time because of the new value set on entry and exit radii. Descent stage is more progressive as flight path angle (γ) descend smoother at the start. Velocity profile is also softer at the beggining. This is a result of more continuous profile on both controls, δ_{TPS} and δ_e. Eventually, pull-up manoeuvre at the end of descent stage is more gentle and the union between stage trajectories is flatter -the discontinuous jump is of lesser magnitude, especially in elevator deflection-. Discharge stage is longer but has not noteworthy differences. Eventually, in climb stage there are some significant decreases in terms of magnitude, principally on flight path angle (γ), angle-of-attack (α) and velocity module (V). It has to be highlighted that all state trajectories seems more stable, continuous and softer than the first case, and the reason are more continuous and flatter control profiles -since there is no rush to achieve objectives, magnitude on all variables is lower and trim points are easy to find and mantain-.

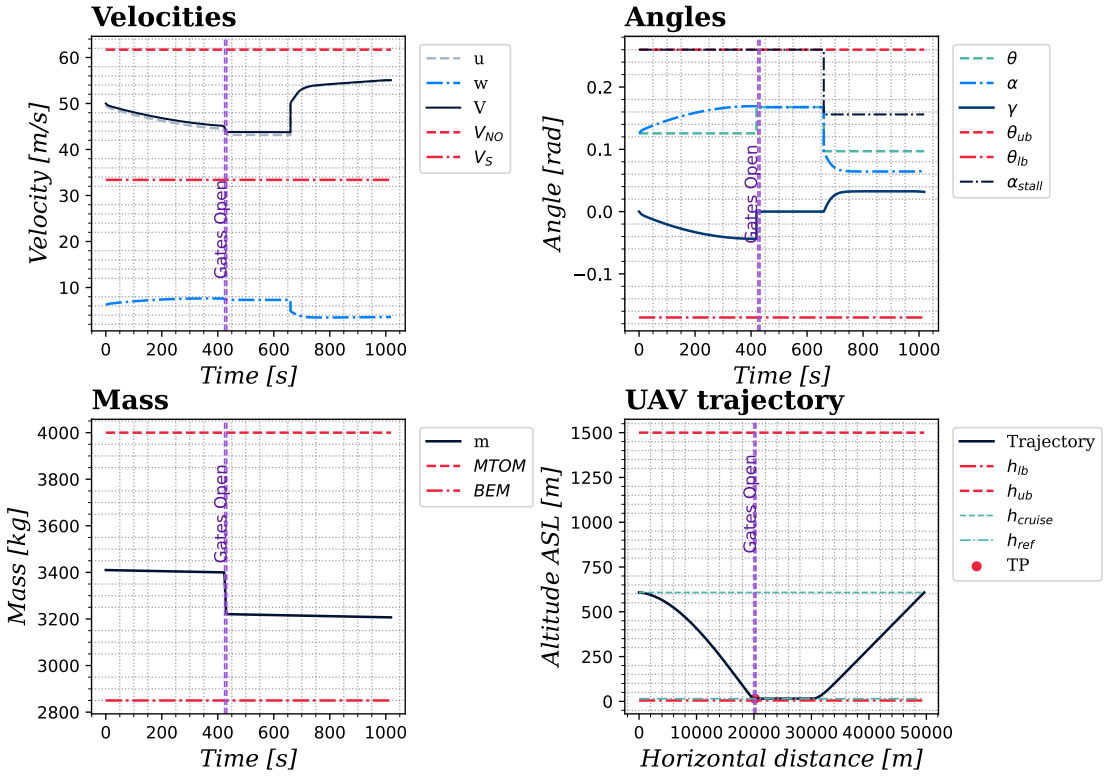


Figure 5.11: State trajectories, case 3. States included are velocities, angles, mass and UAV trajectory. Own source.

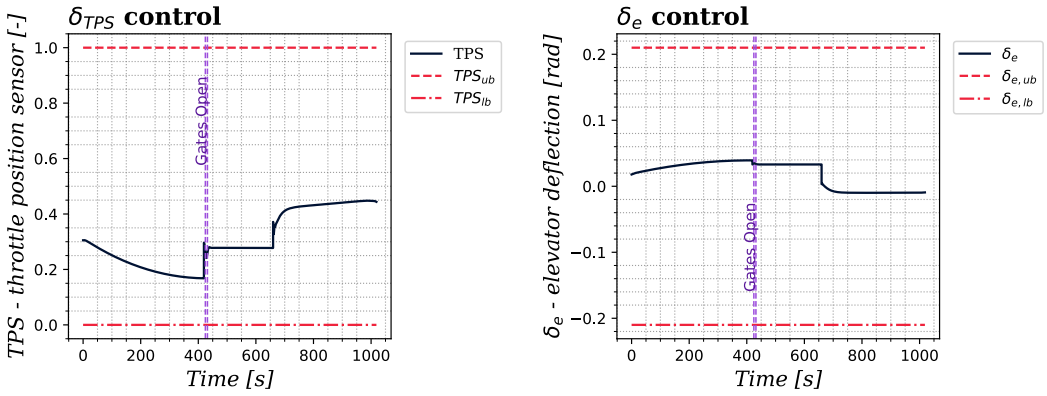
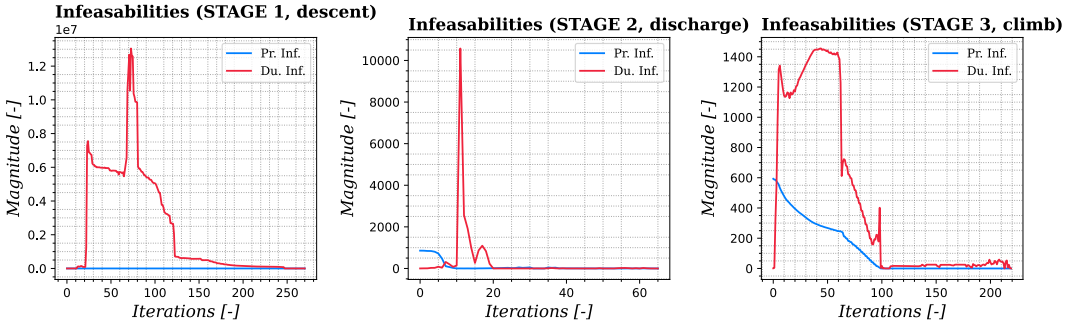
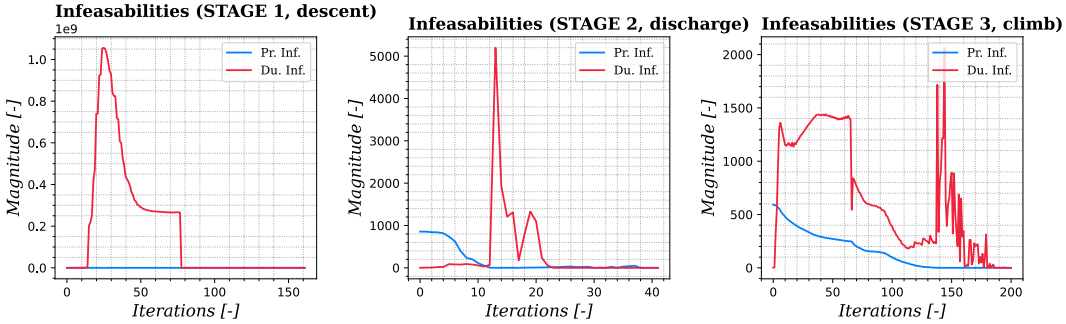


Figure 5.12: Control trajectories, case 3. Controls included are TPS and elevator deflection. Own source.

There is no coincidence on the behaviour observed before. Controls are softer because the overall problem has seen an increment in optimal manoeuvrability - KKT conditions have changed, see figure below-.



(a) Case 1. With controls and desired attitude.



(b) Case 3. No controls nor desired attitude.

Figure 5.13: Comparison between KKT conditions per iteration for cases 1 and 3. Own source.

Essentially, the primal infeasability magnitude remained unchanged between case 1 and case 3. Primal infeasability refers to maximum violation of equality or inequality constraints, which means that if its value is high, the trajectories will be near a constraint wall -they will be facing constraint limits or simple bound limits-. On the other hand, there is dual infeasability, which is related to the violation of stationarity condition -see equation (2.11)- and which means how great is the capacity for change and optimisation without breaking the constraints. As can be seen in the comparison above, the dual infeasability value has increased noticeable between case 1 and case 3, remarking that there is more optimisation margin on this second problem -principally at stages 1 and 3-. Since there are less parameters to optimise, but with a bigger margin, the solver can choose smoother solutions. In other words, the problem has smooth solution because in some manners, is less cost constrained and controls are more free to adapt circumstances, apart from the fact that trajectories are longer.

5.1.4 Case 4: terminal cost only

Fourth case has been set to determine which is the effect on setting all running costs to zero and leaving only a terminal cost. Comparison is also done with case 1, since the parameters have been taken from previous case.

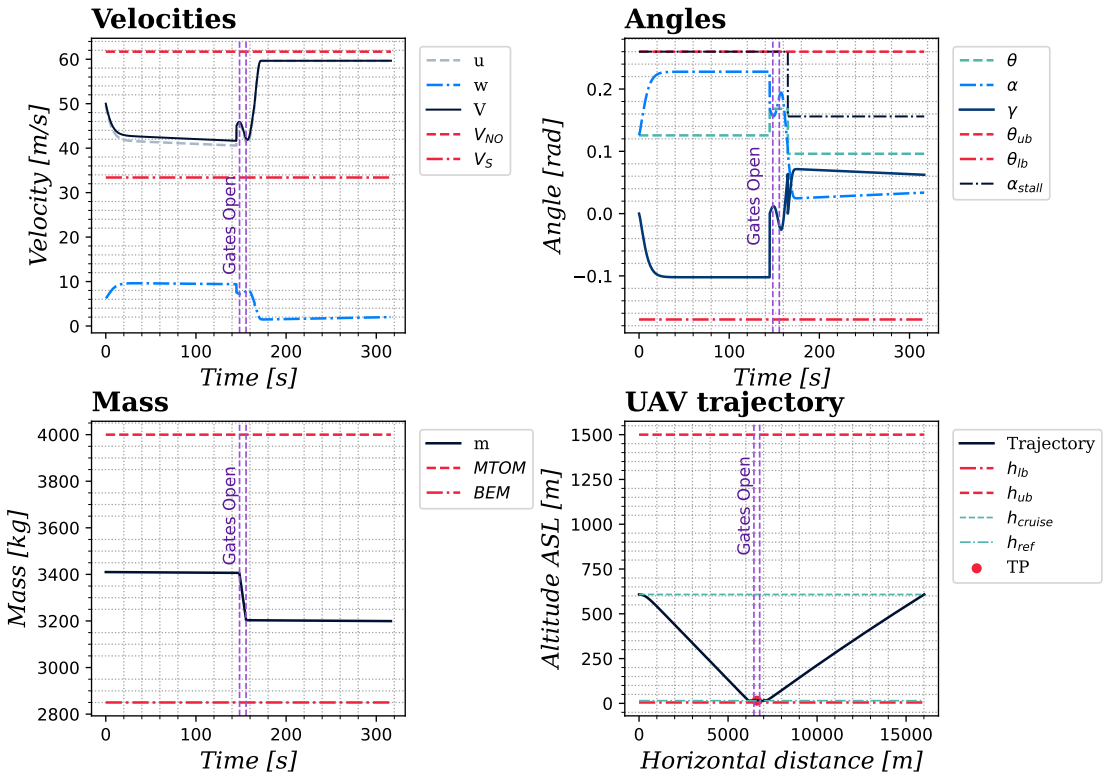


Figure 5.14: State trajectories, case 4. States included are velocities, angles, mass and UAV trajectory. Own source.

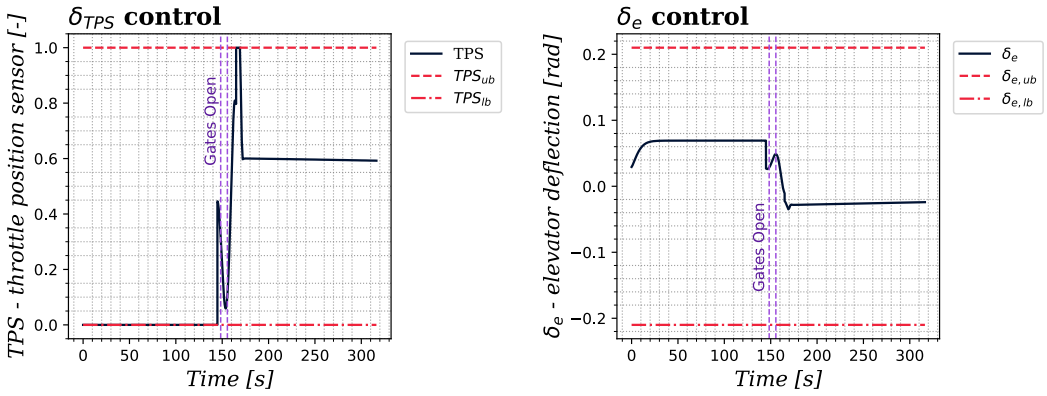
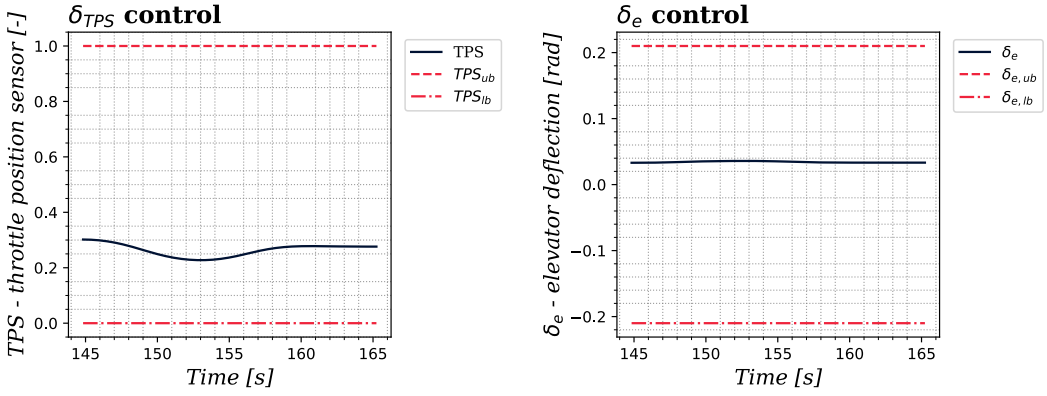
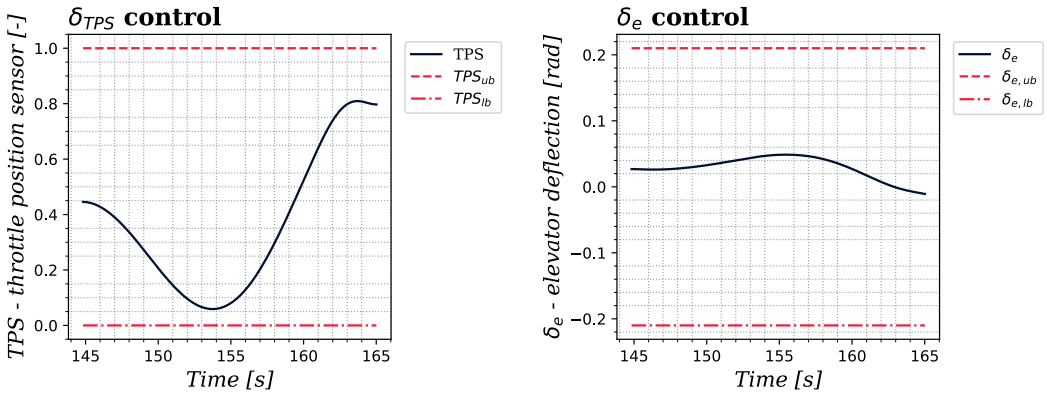


Figure 5.15: Control trajectories, case 4. Controls included are TPS and elevator deflection. Own source.

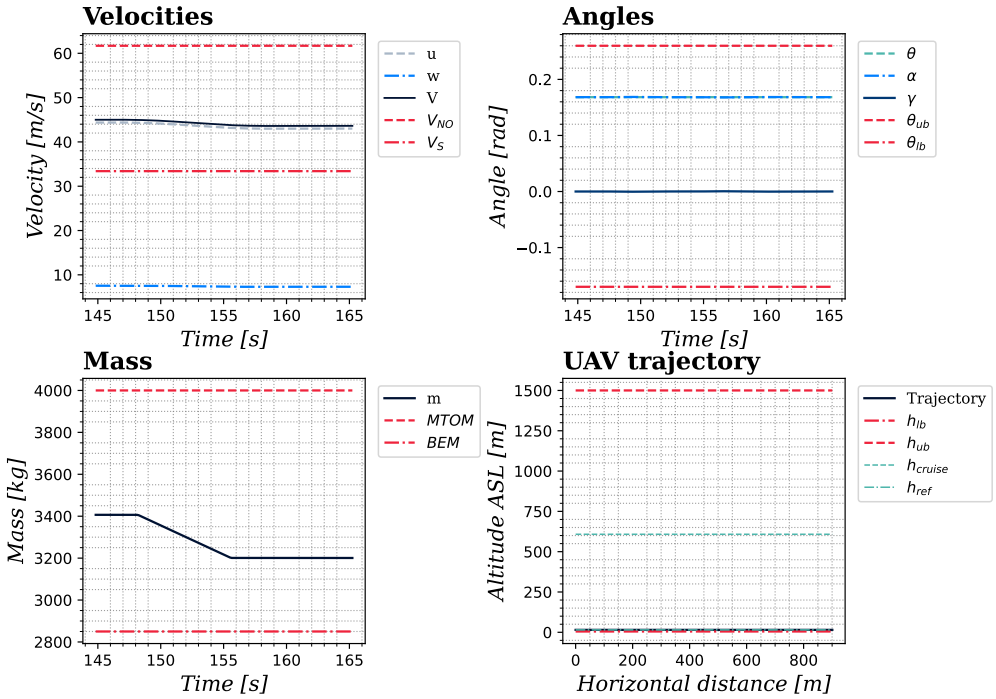


(a) Case 1. All running costs.

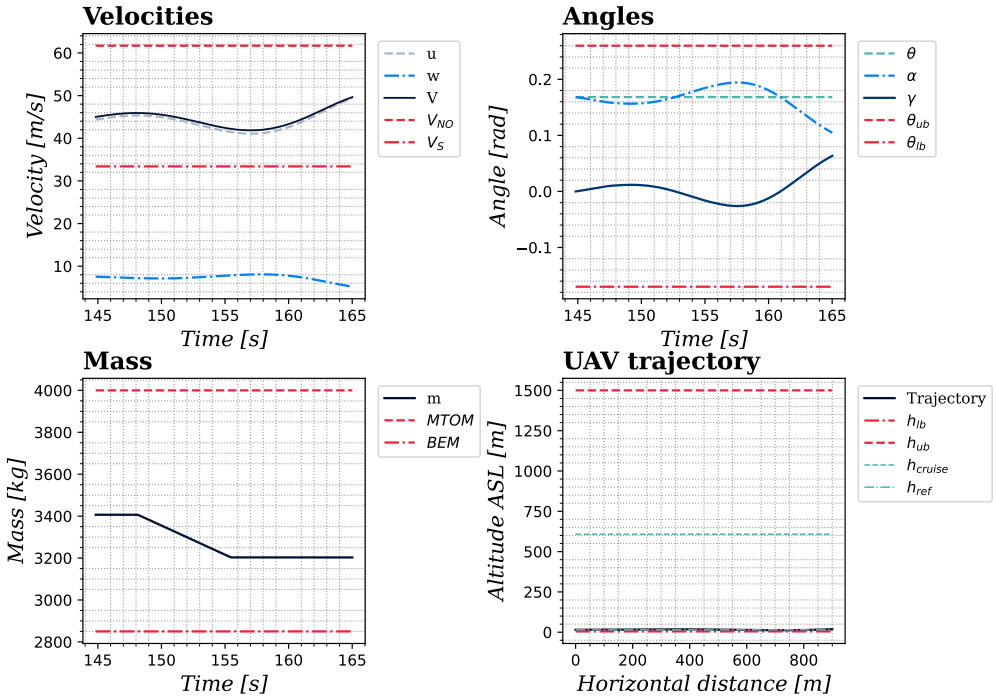


(b) Case 4. Terminal cost only.

Figure 5.16: Comparison between control trajectories during discharge stage for different costs. Own source.



(a) Case 1. All running costs.



(b) Case 4. Terminal cost only.

Figure 5.17: Comparison between state trajectories during discharge stage for different costs. Own source.

As can be seen, there are no huge differences between state trajectories. Nonetheless, control trajectories are more unstable and spiky. Essentially, big differences seem to be at discharge stage. Figure 5.16 shows a deep comparison between control trajectories during discharge stage. As mentioned, TPS variations are accentuated and prolonged in time while elevator deflection (δ_e) fluctuates considerably; in any case, it is not possible to find a trim point -perhaps the stage is short to find trim controls or, the sudden discharge is substantial this time that controls or states have not been penalised-. Eventually, fluctuations in control trajectories provoke oscillations in states, particularly in angles and altitude -although this last cannot be seen due to graph limits, which represent operational ceiling altitude-. Figure 5.17 enhance those state differences and the mirror behaviour on state trajectories with changes in control variables for each case, respectively.

5.1.5 Case 5: constraints only

Next, a simulation has been set without any type of costs -all weights have been fixed to zero values-. Therefore, the purpose of this simulation is to observe the state and control trajectories in a problem without a cost function but with many constraints. It is about observing how would the trajectories look like when there are no optimisation variables. To do so, the rest of parameters have been set as done in case 3.

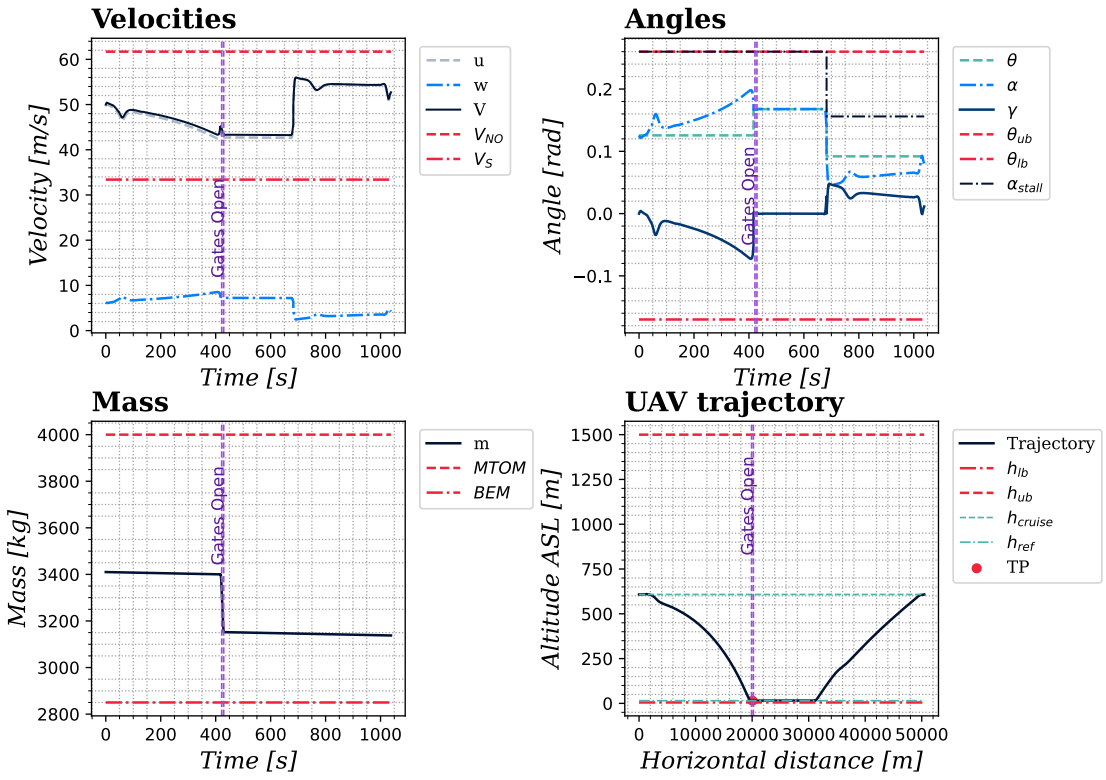


Figure 5.18: State trajectories, case 5. States included are velocities, angles, mass and UAV trajectory. Own source.

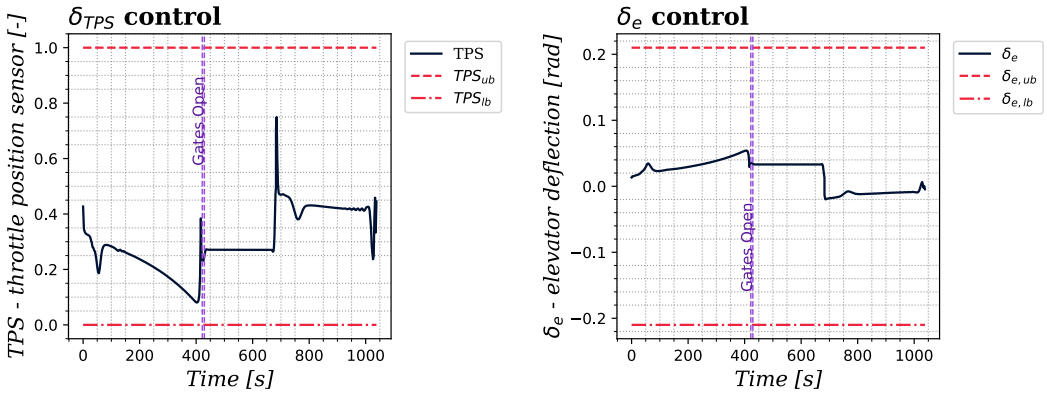


Figure 5.19: Control trajectories, case 5. Controls included are TPS and elevator deflection. Own source.

In this case the UAV trajectory does not appear chaotic. However, the states are discontinuous and aggressive in all sections because the controls are neither smooth nor do they tend towards a trim point: they are oscillating, abrupt and discontinuous. In other words, if the trajectory were shorter, it would surely be a case of irrecoverable bang-bang control, physically impossible due to the inertia of the systems. Since there is no penalty on the control effort or state deviations, the solver has no incentive to produce smooth or efficient results. This leads to a scenario where the optimiser merely seeks 'any' mathematical path that respects the constraints, regardless of physical or operational logic.

5.2 Summary of Observations

Finally, all the observations made during the set of simulations have been summarised.

- Firstly, a disturbance has been created by reducing or extending the discharge's duration. It has been observed that this system is apparently robust to instantaneous disturbances; nonetheless, control is softer and smoother in nominal conditions, which also means flatter state trajectories. Certainly, states are a consequence of control variables.

- Secondly, end-time penalisation in the terminal cost has been added. It must be noted that if an end-time is free and it is penalised in the terminal cost term, the resulting trajectories would be faster, but manoeuvres will be aggressive. In other words, demanding a terminal cost composed only of an end-time penalty leads to spiky control trajectories -regardless of penalising control rates- and, therefore, to less smooth state trajectories. For the simulation set, velocity and flight path angle trajectories have been the most affected. However, it is worth noting that introducing an end-time penalty provokes a more stable stationary state.

- Next, desired states -or conditions- and control rates have been unpenalised, while

certain restriction margins have been eased. This led to smoother and flatter control trajectories and continuous state trajectories as a consequence. The reason behind this is a greater optimality margin in KKT conditions, especially in the dual feasibility condition. In other words, the solver finds solutions which can move away from the wall of constraints while generating greater optimal control. Nevertheless, it must be said that this has been possible because, at the same time as the control parameters' effect was reduced in the cost functional, constraint margins were larger; if the margins had not been extended, the optimality margin would probably have decreased.

- Fourthly, a simulation with only terminal cost has been carried out. Specifically, it has been observed that this could lead to a bang-bang control situation if the problem is more constrained—because constraint margins were amplified for the simulation. If a bang-bang control situation arose, the states would begin to oscillate.

- Lastly, a constraints-only scheme has been tried, revealing that longer manoeuvres are the optimal solution found; nevertheless, manoeuvres are far from smooth because the control becomes aggressive: the solver has no motivation to reduce or smooth control. In other words, the absence of a cost objective gives a feasible solution for the problem but it is not optimal—it could be 'any' solution complying with the constraints and limits of the problem.

6 | Budget and Impact

6.1 Budget

6.2 Environmental Impact

6.3 Social Impact

7 | Conclusions and Future Work

7.1 Main Contributions

In this project, an algorithm has been developed for generating optimal trajectories in firefighting manoeuvres, fulfilling the primary objective of this thesis. The methodology successfully synthesised Optimal Control Theory (OCT) and Non-linear Programming (NLP) techniques, employing a Direct Collocation Method (DCM) and a trapezoidal integration scheme to model generic firefighting operations for unmanned aerial systems. This theoretical framework was implemented using the CasADi optimisation suite to develop an IPOPT-based solver, which consistently yielded feasible and optimal solutions tailored to the operational parameters of the Flyox UAV.

Firstly, a reduced version of a cruise flight has been modelled and implemented, enabling the validation of the acquired knowledge about OCT and NLP. Subsequently, the requirements for the extinguishing manoeuvre for a UAV were identified and mathematically modelled. After mathematically modelling the extinguishing manoeuvre in three different phases—descent, cruise with water discharge, and ascent—various simulations were carried out to observe the criticality of the integration as well as the operability of the Flyox under optimal control.

Essentially, it has been possible to corroborate that the integrated system has a high degree of robustness against possible instantaneous disturbances. This verification was carried out by measuring the opening time of the gates through which water is discharged onto the flames. When the system was disturbed by a sudden decrease in mass, the algorithm was able to find an optimal and feasible solution to compensate for this effect and thus minimise the variation in control and states. However, under nominal operating conditions—a longer discharge regime—the control inputs are smoother and result in flatter state trajectories. Therefore, it is confirmed that the algorithm or solver maintains stability under certain stress, although further testing should be carried out.

Subsequently, a pivot analysis was performed on the effect of the end-time within the terminal cost. While a heavily penalised free end-time reduces the total duration of the manoeuvre -critical in forest firefighting manoeuvres- it leads to more spiky control trajectories and, therefore, less smooth state profiles. However, despite this aggressiveness, including the end-time penalty helps to facilitate transitions between stages, suggesting

that for firefighting operations, this penalty should be balanced. The aim is to prioritise speed while maintaining safety and efficiency in all stages.

Then, by easing constraint margins and reducing the penalisation in state deviations, a significant improvement in control profile continuity has been observed. This phenomenon is mathematically grounded in the expansion of the Karush-Kuhn-Tucker (KKT) optimality margins, particularly regarding dual feasibility. Specifically, the solver finds solutions which can move away from the wall of constraints while generating greater optimal control. Nevertheless, it must be said that this has been possible because, at the same time as the control parameters' effect was reduced in the cost functional, constraint margins were larger; if the margins had not been extended, the optimality margin would probably have decreased. This highlight that in high-dimensional NLP rigid constraint boundaries can often hinder the search for truly optimal control, whereas broader margins allow the IPOPT solver to find solutions with better state profiles.

In fourth place, the comparative study between different objective schemes underscores the necessity of a well-defined cost functional. Simulations relying solely on terminal costs revealed a tendency toward bang-bang control behaviour, which triggers undesirable state oscillations as the system nears its constraint limits. Conversely, a constraint-only approach proved insufficient for practical application; while it yields feasible solutions, the absence of an objective goal results in unnecessarily long and aggressive manoeuvres. These findings confirm that for a platform like the Flyox UAS, the objective function must be a multi-variable compromise that motivates the solver not just to find a solution, but to find the smoothest possible path.

In conclusion, this project establishes a robust mathematical and computational baseline for the autonomous operation of the Flyox UAV in firefighting missions. While the current 2D point-mass model and environmental simplifications represent a necessary starting point for stability analysis, the successful implementation of the IPOPT-based solver proves that non-linear programming is a viable path for real-time trajectory generation. In addition, this work has provided a framework that enhances both the efficiency and safety of aerial firefighting, laying the groundwork for future integration into high-fidelity environments.

7.2 Limitations and Future Research

The developed algorithm is far from being the perfect solution to the problem, as it has some limitations in terms of integration and development, as listed below.

- **Bi-dimensional Approach.** The current algorithm has been planned to solve bi-dimensional (2D) vertical plane restricted trajectories; nevertheless, real-world scenario includes lateral stability effects or dynamics, which is a limitation in firefighting manoeuvres.
- **Environmental Simplification.** All atmospheric conditions, terrain elevation or obstacles have been neglected or simplified to offer a quick solution to the problem

but real scenarios does not have those simplifications. They are critical elements that have to be considered in low-altitude flight manoeuvres.

- **Point-Mass Dynamics.** The mathematical formulation and implementation behind the algorithm has reduced a real aircraft to a point-mass model, simplifying all the dynamics and kinematics.
- **Dynamic Stability and Control Feed-Back.** No dynamic stability has been considered during the algorithm's development, which means that all transitory have been negelected because there is no control feed-back.
- **Real-Time Computation.** The algorithm has been developed in Python, oferring a sufficient solution to the needed base. Nonetheless, a real-world mission would need a faster and embedded algorithm, which implies migration to other codes that have CasADi optimisation libraries or similars.

Eventually, the following aspects have to be treated in the near future.

- **Integration of Data Elevation Models (DEM), No-Fly Zones (NFZ) and Real Atmospheric Conditions**. By using pre-loaded or on-board data, the obstacles and the terrain shall be converted into updated constraints of the optimal control algorithm. In addition, the atmosphere effects shall be included in real trajectory computation.
- **Stochastic Dynamic Modelisation.** A real aircraft dynamic stability should be modelled including stationary and transitory stability remarks and effects, including a stochastic optimal control approach that deals with non-ideal disturbances atmospheric conditions.
- **On-Board Real-Time Computation.** An embedded solution is the best approach to the firefighting scenario, since the manoeuvres are performed in a very dynamic environment. Within this context, a communication protocol between the output of the algorithm and the UAV autopilot shall be included by waypoint generation or other specifical needs.

References

1. FOOD AND AGRICULTURE ORGANIZATION OF THE UNITED NATIONS. *Global Forest Fire Assessment 2023*. 2023. Available also from: https://www.fao.org/forest-resources-assessment/fire/en/. Accessed: Jan. 2, 2026.

2. RESTAS, Agoston. Drone Applications for Wildfire Management: From Detection to Suppression. *Fire Technology*. 2022, vol. 58, no. 3, pp. 1011–1036. Available from DOI: 10.1007/s10694-021-01169-9.

3. SINGULAR AIRCRAFT S.L. *Flyox I – The World's Largest Civilian UAV*. [N.d.]. Available also from: https://singularaircraft.com/. Accessed: Jan. 2, 2026.

4. HARGREAVES, C.; PARIS, S.; LASES, W. OTIS past, present and future. In: *AIAA Atmospheric Flight Mechanics Conference*. 2012. Available from DOI: 10.2514/6.1992-4530. Online.

5. RÖSMANN, T.; BERNING, T.; WERNER, R. Trajectory Simulations, Qualitative Analyses and Differential Flatness of Constrained Systems. In: *AIP Conference Proceedings*. 2006, vol. 830, pp. 522–529. No. 1. Available from DOI: 10.1063/1.2203055. Online.

6. ASTOS SOLUTIONS GMBH. *ASTOS - Analysis, Simulation and Trajectory Optimization Software* [online]. 2024. [visited on 2026-01-06]. Available from: https://www.astos.de/products/astos. Online.

7. BECERRA, Victor M. *PSOPT – An Open Source Optimal Control Solver for C++* [online]. 2010. [visited on 2026-01-06]. Available from: https://www.psopt.net/home. Online.

8. SALGUERO, David E. *Trajectory Analysis and Optimization System (TAOS) User's Manual* [online]. Albuquerque, New Mexico, United States, 1995 [visited on 2026-01-06]. Tech. rep., SAND-95-1652. Sandia National Laboratories. Available from DOI: 10.2172/162896. Online.

9. HOUSKA, B.; FERREAU, H. J.; DIEHL, M. ACADO Toolkit: An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*. 2011, vol. 32, no. 3, pp. 298–312. Available from DOI: 10.1002/oca.939.

10. SUN, J.; ELLERBROEK, J.; HOEKSTRA, J. OpenAP: An Open-source Aircraft Performance Model in Python. *Aerospace*. 2020, vol. 7, no. 8, p. 104. Available from DOI: 10.3390/aerospace7080104. Reference for the underlying OpenAP framework used by OpenAP.opt.

11. COURANT, R.; HILBERT, D. *Methods of Mathematical Physics*. Vol. I. New York, NY, USA: John Wiley & Sons, 1953. ISBN 0-471-50447-5.

12. KIRK, E. D. Optimal Control Theory: An Introduction. In: Mineola, NY, USA: Dover Publications, 2004, chap. 1, pp. 3–28. ISBN 0-486-43484-2.

13. BLÅSJÖ, V. *Galileo, ignoramus: Mathematics versus philosophy in the scientific revolution*. 2021. Available also from: https://doi.org/10.48550/arXiv.2102.06595. arXiv:2102.06595 [math.HO].

14. BERNOULLI, J. Problema novum ad ejus solutionem Mathematica invitantur. *Acta Eruditorum*. 1696, p. 269. 18th ed.

15. BETTS, J. T. Practical Methods for Optimal Control and Estimation Using Nonlinear Programming. In: 2nd. Philadelphia, PA, USA: SIAM, 2010, chap. 4, pp. 199–205. ISBN 978-0-89871-688-7.

16. ILLINOIS ROBOTICS SYSTEMS. *Chapter 17: Optimal Control (Section 1)*. Section IV: Dynamics and Control. Intelligent Motion Lab. Available also from: https://motion.cs.illinois.edu/RoboticSystems/Book.html. Accessed Dec. 23, 2025.

17. BRYSON, A. E.; HO, Y. C. *Applied Optimal Control: Optimization, Estimation, and Control*. Washington, DC, USA: Hemisphere Publishing Corp, 1975.

18. BECERRA, V. M. Practical Direct Collocation Methods for Computational Optimal Control. In: FASANO, G.; PINTÉR, J. D. (eds.). *Modeling and Optimization in Space Engineering*. Springer, 2012, pp. 33–60. Available from DOI: 10.1007/978-1-4614-4469-5_2.

19. BÖHME, J. T.; FRANK, B. Hybrid Systems, Optimal Control and Hybrid Vehicles: Theory, Methods and Applications. In: 1st. Cham, Switzerland: Springer, 2017, chap. 8, pp. 245–247. ISBN 978-3-319-51316-4.

20. JACKSON, B. *Exploring quadrature rules in direct collocation methods for trajectory optimization*. 2018. Available also from: https://bjack205.github.io/assets/AA203_project.pdf. Principles of optimal control (AA203). Accessed Dec. 23, 2025.

21. UNIVERSITY COLLEGE LONDON. *The Euler-Lagrange Equation, or Euler's Equation*. Available also from: https://www.ucl.ac.uk/~ucahmto/latex_html/chapter2_latex2html/node5.html. Accessed Dec. 23, 2025.

22. GORDON, G.; TIBSHIRANI, R. *Karush-Kuhn-Tucker conditions* [PowerPoint slides, School of Comput. Sci., Carnegie Mellon Univ.]. [N.d.]. Available also from: http://www.cs.cmu.edu/~ggordon/10725-F12/slides/. Accessed Dec. 23, 2025.

23. STRYK, O. von; BULIRSCH, R. Direct and indirect methods for trajectory optimization. *Ann. Oper. Res.* 1992, vol. 37, no. 1–4, pp. 357–373. Available from DOI: 10.1007/BF02071065.

24. LÉONARD, D.; LONG, N. van. Endpoint constraints and transversality conditions. In: *Optimal Control Theory and Static Optimization in Economics*. Cambridge, UK: Cambridge Univ. Press, 1992, pp. 221–262.

25. CHACHUAT, B. *Optimal Control: Lecture 28—Indirect solution methods* [PowerPoint slides, Dept. Chem. Eng., McMaster Univ.]. 2009. Available also from: https://www.epfl.ch/labs/la/wp-content/uploads/2018/08/Slides28.pdf. Accessed Dec. 23, 2025.

26. COLUMBIA UNIVERSITY. *10. Pontryagin's Minimum Principle*. 2018. Lecture Notes. Columbia Univ. Available also from: https://blogs.cuit.columbia.edu/zp2130/files/2019/03/Lecture10_Pontryagins-Minimum-Principle.pdf. Accessed Dec. 23, 2025.

27. BRYSON, A. E.; HO, Y. C. Applied Optimal Control: Optimization, Estimation and Control. In: Waltham, MA, USA: Blaisdell, 1975, pp. 1–39. ISBN 0-89116-228-3.

28. WÄCHTER, A.; BIEGLER, L. T. *On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming*. 2004. Tech. rep. Optimization Online. Available also from: https://optimization-online.org/wp-content/uploads/2004/03/836.pdf. Accessed Dec. 23, 2025.

29. FLETCHER, R.; LEYFFER, S. Nonlinear programming without a penalty function. *Math. Program.* 2002, vol. 91, no. 2, pp. 239–269. Available from DOI: 10.1007/s101070100244.

30. ANDERSSON, Joel A E; GILLIS, Joris; HORN, Greg; RAWLINGS, James B; DIEHL, Moritz. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*. 2018.

31. STENGEL, Robert F. *Optimal Control and Estimation*. New York, NY, USA: Dover Publications, 1994. ISBN 978-0486682006.

Appendices

A | Mathematical Derivation of Path Angle Rate's ($\dot{\gamma}$) Expression

This appendix section includes the formal mathematical derivation of the flight path angle variation ($\dot{\gamma}$), used in the definition of cost functional during benchmark problem and firefighting manoeuvre problem.

Firstly, the flight path angle (γ) can be defined as the difference between the pitch angle (θ) and the angle-of-attack -AoA- (α), as stated below.

$$\gamma \coloneqq \theta - \alpha \tag{A.1}$$

This leads to the definition of flight path angle variation ($\dot{\gamma}$) as the time derivative of the expression above. Since both angles, pitch and AoA are defined in the same frame, the derivative can be easily defined as the difference between time derivatives.

$$\dot{\gamma} \coloneqq \frac{d}{dt}(\theta - \alpha) = \frac{d\theta}{dt} - \frac{d\alpha}{dt} = \dot{\theta} - \dot{\alpha} = q - \dot{\alpha} \tag{A.2}$$

Then α is defined as the arctan relation between aerodynamic velocity components (u_a,w_a), which are defined as the difference between the body velocity and the wind velocity. Nevertheless, for the application that concerns this thesis, wind velocity has been neglected on aerodynamic velocity definition and, therefore, the expression for α can be written as follows.

$$\alpha(u_a, w_a) \coloneqq \arctan(w_a, u_a) \approx \arctan(w, u) \tag{A.3}$$

As stated above, AoA has a relation with the body velocities (u, w) which implies the use of chain rule during time derivatives. Thus:

$$\dot{\alpha} \coloneqq \frac{\partial \alpha}{\partial u}\frac{\partial u}{\partial t} + \frac{\partial \alpha}{\partial w}\frac{\partial w}{\partial t} = \frac{\partial \alpha}{\partial u}\dot{u} + \frac{\partial \alpha}{\partial w}\dot{w} \tag{A.4}$$

Deriving AoA respect to the body velocities (u, w), the following expressions can be found.

$$\frac{\partial \alpha}{\partial u} := \frac{\partial}{\partial u} \arctan(w, u) = \frac{-w}{u^2 + w^2} \tag{A.5a}$$

$$\frac{\partial \alpha}{\partial w} := \frac{\partial}{\partial w} \arctan(w, u) = \frac{u}{u^2 + w^2} \tag{A.5b}$$

Eventually, the relations can be substituted into the whole expression and flight path angle variation ($\dot{\gamma}$) can be written as:

$$\dot{\gamma} := \dot{\theta} - \dot{\alpha} = q - \left(\frac{-w}{u^2 + w^2} \dot{u} + \frac{u}{u^2 + w^2} \dot{w} \right) = q - \frac{u\dot{w} - w\dot{u}}{u^2 + w^2} \dot{u}$$

$$\dot{\gamma} := q - \frac{\dot{w}u - \dot{u}w}{u^2 + w^2} \tag{A.6}$$

B | Code Snippets and Files

To see the last version of all code snippets, the reader can access to the main branch at the following repository: `https://github.com/OverfittedPenguin/TFG_Campillo_Abimael`. This GitHub repository contains both source code of the project's algorithm and thesis report. Essentially, all code snippets have been comented during its development and no further description is required. Nevertheless, this section includes a brief description of the structure used in each JSON file.

The configuration files or JSON files are three, as mentioned in 4.3. Atmos.json is the atmospheric conditions dedicated configuration file. Its structure is self-descriptive, since it only contains variables and its value, so no further comments will be done.

Listing B.1: Atmospheric conditions configuration file (*Atmos.json*).

```
{
    "GRAVITY_ACC": 9.81,
    "SL_PRESS": 101325.0,
    "SL_TEMP": 15.0,
    "SL_RHO": 1.225,
    "ATMOS_GRADIENT": -0.0065,
    "AIR_CONSTANT": 287.053
}
```

Then, the structure of the aircraft configuration file. In this case, the file contains the description of all geometrical, physical and operational parameters as stated on C. The structure of the JSON is descriptive along the tables of the mentioned appendix.

Listing B.2: Aircraft configuration file (*Flyox.json*).

```
{
   "NAME": "FLYOX_MSN011",
    "BEM": 2850.0,
    "MTOM": 4000.0,
    "FM": 360.0,
    "PM": 200.0,
    "SFC": 0.0236,
    "AR": 7.0,
    "S": 28.0,
```

```
    "b": 14.0,
    "c": 2.0,
    "e": 0.84,
    "CGx": 0.759,
    "CGz": -0.090,
    "CPwx": 0.537,
    "CPwz": -1.100,
    "CTx": 0.000,
    "CTz": -1.010,
    "Dp": 2.03,
    "nENG": 2.0,
    "EPS0": 0.0,
    "Iyy": 20980.0,
    "LIFT_DRAG_COEFFS": [[0.410, 3.340, 0.048],[0.510, 4.584,
        0.060],[0.700, 5.730, 0.080]],
    "MOMENT_COEFFS": [0.052, -1.030],
    "ELEVATOR_COEFFS": [0.750, -2.640],
    "THRUST_COEFFS": [0.156, 0.036, -0.201, 0.093],
    "EFFICIENCY_COEFFS": [1.0, 0.0],
    "OPERATIONAL_LIMITS": [[33.40, -0.17, 0.00, -3000.00,
      0.00, -0.21],[61.70, 0.26, 50000.00, -5.00, 1.00,
      0.21]],
    "MISSION_SPECS": [0.0, 2700.0]
}
```

Eventually, there is the simulation configuration file, which deserves a bit of explanation. JSON structure does not accept comments but, in this case comments have been added to this example to explain the structure -comments are after // in green color-. As stated before, initial guesses, mission parameters, mission bounds and all numerical needed values are defined in this configuration file.

Listing B.3: Simulation configuration file (*Simulation.json*).

```
{
    "N":150, // Nodes
    "INITIAL_STATE": [50.00,0.0,120.00,110.00,120.00],
    // Initial guesses [V, Pitch, t1, t2, t3]
    "WIND_SPEED": [0.00,0.00],
    // Wind speed [Vwx, Vwz]
    "TARGET_POINT": [10000.00, 10000.00, 0.25, 52.00, 5.0],
    // TP Parameters [RENTRY, REXIT, fTP, VTP, td]
    "MISSION_BOUNDS": [[5.00,10.00,15.00,-608.00,-7.08],
    [600.0,600.00,600.0,-25.0,7.08]],
    // Mission bounds. [t1, t2, t3, h, ROC]
    "STG1_WEIGHTS":[0.0,0.3,0.2,1.0,1.5],
    // STG1 Penalty Weights. Time weight with running cost
```

```
14      // weights per order of appearance on Section 4.2.1
15      "STG2_WEIGHTS":[0.1,0.5,0.3,0.5,1.0,2.0],
16      // STG2 Penalty Weights. Time weight with running cost
17      // weights per order of appearance on Section 4.2.2
18      "STG3_WEIGHTS":[0.0,0.2,5.0,0.5,1.5,1.0],
19      // STG3 Penalty Weights. Time weight with running cost
20      // weights per order of appearance on Section 4.2.3
21      "AIRCRAFT_FILE": "configs/Flyox.json"
22      // Path of the aircraft.
23  }
```

C | Aircraft Parameters: Flyox

This chapter contains the geometrical, physical and operational parameters of the UAV used in the different simulations. This parameters have been provided by Singular Aircraft and are subjected to a confidentiality agreement. Thus, any misuse, unauthorised dissemination or commercialisation may be subject to legal actions. In case of there is any doubt on an aircraft parameter, reader can refer to nomenclature's section.

Table C.1: Geometrical, physical and operational parameters of aircraft model Flyox. Provided by Singular Aircraft.

Parameter	Value	Units
BEM	2850.00	kg
MTOM	4000.00	kg
FM	360.00	kg
PM	200.00	kg
SFC	0.0236	kg/s
AR	7.00	-
S	28.00	m^2
b	14.00	m
$\bar{c}$	2.00	m
e	0.84	-
I_y	$2.098 \cdot 10^4$	$kg \cdot m^2$
$[CG_x, CG_z]$	[0.759, -0.090]	m
$[CP_x, CP_z]$	[0.537, -1.100]	m
C_{L_0} , C_{L_α}, C_{D_0}	F0: [0.410, 3.340, 0.048] F15: [0.510, 4.584, 0.060] F40: [0.700, 5.730, 0.080]	[-, $1/rad$,-]
C_{m_0}, C_{m_α}	[0.052, -1.030]	[-, $1/rad$]
$C_{L_{\delta e}}$, $C_{m_{\delta e}}$	[0.750, -2.640]	[$1/rad$, $1/rad$]
$[CT_x, CT_z]$	[0.000, -1.010]	m
Engines	2	-
Propeller Diameter (D_p)	2.03	m
Initial thrust AoA (ε_0)	0.00	rad
Thrust Coeffs. $[C_{T_0}, C_{T_1}, C_{T_2}, C_{T_3}]$	[0.156, 0.036, -0.201, 0.093]	-
Efficiency Coeffs. $[C_{\eta_0}, C_{\eta_1}]$	[1.000, 0.000]	-
Operational Limits $[V, \theta, x, z, \delta_{TPS}, \delta_e]$	LBX: [33.40, -0.17, 0.00, -1500.00, 0.00, -0.21] UBX: [61.70, 0.26, 50000.00, -5.00, 1.00, 0.21]	[m/s, rad, m, m, -, rad]
Mission Specifications $[\delta_{FLAPS}, RPM]$	[0, 2700]	[-, rpm]