



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

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

BACHELOR FINAL THESIS

Abstract

Acknowledgments

Preface

Scope and Objectives

Motivation and Justification

Methodological Approach

Document Structure

Contents

List of Figures

List of Tables

Acronyms

ASL Above Sea Level.

CONOPS CONceptual OPerationS.

DCM Direct Collocation Method.

EP Exit Point.

IPOPT Interior Point OPTimizer.

KKT Karush-Kuhn-Tucker.

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.

SP Starting Point.

STG STaGe.

TP Target Point.

TPS Throttle Position Sensor.

UAV Unmanned Aerial Vehicle.

Nomenclature

Symbol	Description	Units
<i>Note: Bold symbols denote vectors; dotted symbols denote time-derivatives ($\dot{x} = dx/dt$).</i>		
BEM	Basic Empty Mass	$[kg]$
C_{D_0}, k	Drag coefficients: parasitic drag and induced drag factor	$[-]$
$C_{L_0}, C_{L_\alpha}, C_{L_{\delta e}}$	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, force projected into negative x-wind axis (x_w)	$[N]$
$F_b(x_b, z_b)$	Body-fixed reference frame with origin at the Centre of Gravity (CG), x_b along the fuselage, and z_b pointing down	$-$
$F_h(x_h, z_h)$	Local-horizon reference frame (North-East-Down)	$-$
$F_w(x_w, z_w)$	Wind reference frame, x_w aligned with the aerodynamic velocity vector and z_w perpendicularly pointing down	$-$
I_y	Moment of inertia about the pitch axis (y_b)	$[kg \cdot m^2]$
L	Lift force, force projected into negative z-wind axis (z_w)	$[N]$
$MTOM$	Maximum Take-Off Mass	$[kg]$
S	Aircraft wing's surface	$[m^2]$
SFC	Specific Fuel Consumption	$[kg/s]$
T	Thrust force, 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)	$[N]$

Symbol	Description	Units
<i>Note: Bold symbols denote vectors; dotted symbols denote time-derivatives ($\dot{x} = dx/dt$).</i>		
$\Delta X_{CT}, \Delta Z_{CT}$	Horizontal and vertical distances from the Centre of Thrust (CT) to the Centre of Gravity (CG)	[m]
α	Angle of Attack, angle between the aircraft's x-body axis (x_b) and the projection of the 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]
γ	Flight path angle, angle between the aerodynamic velocity vector 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_b and the horizontal plane ($x_h - y_h$)	[rad]
ε	Thrust angle of attack, angle between thrust vector and x_b axis	[rad]
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	[m/s]

1 | Fundamentals of Optimal Control and its Implementation

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

1.1 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 related with calculus of variations, where optimal points -either maxima or minima- are found using little variations in functions or functionals **Courant1953**. 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 Non-linear Programming (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 **Kirk2004**. See its definition in expression (??).

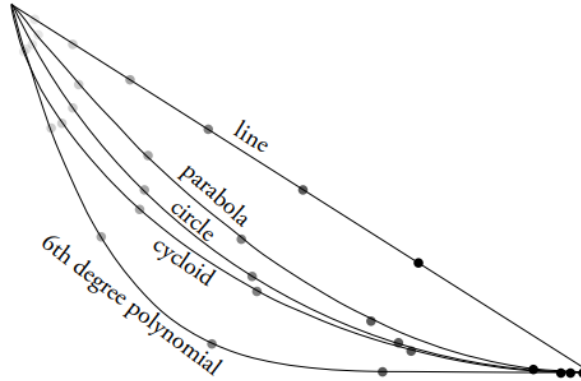


Figure 1.1: The brachistochrone problem. The ball rolls down the fastest on a cycloidal ramp. Points of the same shade correspond to the same moment in time. Extracted directly from **Blasjo2021**.

$$\begin{aligned}
 & \max_{x \in \mathbb{R}^n} f(x) \\
 & \text{s.t. } g_j(x) \leq b_j \text{ for each } j = 1, \dots, m \\
 & \quad x = (x_1, \dots, x_n) \\
 & \quad g = (g_1, \dots, g_m).
 \end{aligned} \tag{1.1}$$

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 Throttle Position Sensor (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 **Bernoulli1696**.

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 $F(y, u)$.

1.1.1 Mathematical formulation

The OCP is defined as follows.

$$\begin{aligned}
 \min_{y,u} \quad & J(y, u) := \Phi[y(t_f), t_f] + \int_{t_0}^{t_f} L[y(t), u(t), t] dt \\
 \text{s.t.} \quad & \dot{y} = f[y(t), u(t), t] \\
 & g[y(t), u(t), t] \leq 0 \\
 & \varphi[y(t_0), u(t_0), t_0] = 0 \\
 & y_{lb} \leq y(t) \leq y_{ub} \\
 & u_{lb} \leq u(t) \leq u_{ub} \\
 & t \in [t_0, t_f]
 \end{aligned} \tag{1.2}$$

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 **Betts2010IllinoisRobotics**. 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 condition. 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 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 Equation (ODE)s 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 constraints are another set of expressions that apply for a given time domain and that could refer to any type of extra restrictions 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 **BrysonHo1975**:
 - 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 $J(y, u) = \Phi[y(t_f), t_f]$, 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.

1.2 Direct and Indirect Methods

Once the basic problem has been set, it 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 implicates to discretise 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 **Betts2010**:

- 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 to explicit derivation of the equations and is a method subjected to initial conditions -p.e Pontryagin's maximum principle-.

1.2.1 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) **Becerra2012**.

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 this 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 **Bohme2017**.

$$u(t) = \begin{cases} U_k^u(u_k, u_{k+1}, t) & \forall t \in [t_k, t_{k+1}] \\ U_{N-1}^u(u_{N-1}, u_N, t) & \forall t \in [t_{N-1}, t_N] \end{cases} \quad (1.3)$$

$$\bar{u} = [u_0, u_1, u_2, \dots, u_N]^T \in \mathbb{R}^{(N+1) \cdot n_u} \quad (1.4)$$

The previous expressions (??) and (??) are the time-domain discretisation in N subintervals with specific nodes t_i with $i \in [0, N]$, what allows to obtain a decision vector for control. In other words, it is an interpolation that avoids sudden discontinuities that can generate instabilities 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:

$$\bar{y} = [y_0, y_1, y_2, \dots, y_N]^T \in \mathbb{R}^{(N+1) \cdot n_y} \quad (1.5)$$

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.

$$\begin{aligned} u(t) &\longrightarrow u_i = u(t_i) \quad \text{for each } t_i \text{ with } i \in [0, \dots, N] \\ y(t) &\longrightarrow y_i = y(t_i) \quad \text{for each } t_i \text{ with } i \in [0, \dots, N] \end{aligned}$$

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 evaluation 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 **Becerra2012** and **Jackson2018**, can be state as:

$$\int_{t_0}^{t_f} L(y, u, t) dt \approx \sum_{k=0}^{N-1} \frac{1}{2} h_k \cdot (L_k + L_{k+1})$$

where

$$\begin{aligned} h_k &:= t_{k+1} - t_k \\ L_k &:= L[y(t_k), u(t_k), t_k] \\ L_{k+1} &:= L[y(t_{k+1}), u(t_{k+1}), t_{k+1}] \\ k &\in [0, N - 1] \end{aligned} \tag{1.6}$$

Furthermore, the trapezoidal rule has to be applied to the constraints of the problem, now called collocation constraints. For the simple bounds, path constraints 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 **Jackson2018**.

$$\begin{aligned} \dot{y} &= f(y, u, t) \\ \int_{t_k}^{t_{k+1}} \dot{y} dt &= \int_{t_k}^{t_{k+1}} f(y, u, t) dt \end{aligned}$$

Resulting in:

$$y_{k+1} - y_k = \frac{1}{2} h_k \cdot (f_k + f_{k+1})$$

where

$$f_k = f(y_k, u_k, t_k) \text{ for each } k \in [0, N - 1]$$
(1.7)

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.

$$\begin{aligned}
\min_{y,u} \quad & J(y, u) := \Phi[y(t_N), u(t_N), t_N] + \sum_{k=0}^{N-1} \frac{1}{2} h_k \cdot (L_k + L_{k+1}) \\
\text{s.t.} \quad & y_{k+1} = y_k + \frac{1}{2} h_k \cdot (f_k + f_{k+1}) \text{ for } k \in [0, N-1] \\
& g[y(t_k), u(t_k), t_k] \leq 0 \text{ for } \forall k \\
& \varphi[y(t_0), u(t_0), t_0] = 0 \\
& y_{lb} \leq y(t_k) \leq y_{ub} \\
& u_{lb} \leq u(t_k) \leq u_{ub} \\
& t_k \text{ for each } k \in [0, N]
\end{aligned} \tag{1.8}$$

1.2.2 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 **UCLCalcVar**. 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 Karush-Kuhn-Tucker (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.

$$\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & f(x) \\
\text{s.t.} \quad & g_i(x) \leq 0 \text{ for each } i = 1, \dots, m \\
& h_j(x) = 0 \text{ for each } j = 1, \dots, p
\end{aligned} \tag{1.9}$$

$$\mathcal{L}(x, z, \mu) := f(x) + \sum_{i=1}^m z_i g_i(x) + \sum_{j=1}^p \mu_j h_j(x) \tag{1.10}$$

The KKT conditions are stationarity, complementary slackness, primal feasibility and dual feasibility. Its meaning and mathematical formulation can be seen below as an adaptation from **GordonTibshirani**.

- 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.

$$\nabla_x \mathcal{L}(x, z, \mu) := \nabla f(x) + \sum_{i=0}^m z_i \nabla g_i(x) + \sum_{j=0}^p \mu_j \nabla h_j(x) = 0 \quad (1.11)$$

- 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.

$$z_i g_i(x) = 0 \quad (1.12)$$

- Primal feasibility. 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.

$$\begin{aligned} g_i(x) &\leq 0 \\ h_j(x) &= 0 \end{aligned} \quad (1.13)$$

- Dual feasibility. 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.

$$z_i \geq 0 \quad (1.14)$$

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 (??), can be defined like in a boundary value problem in his canonical form with ODEs in all time domain $t_0 \leq t \leq t_f$ **Stryk1992Leonard1992**.

$$H(\lambda, \nu, t) := \lambda^T f + \nu^T g \quad (1.15)$$

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 **Stryk1992** and **Chachuat2009**.

$$\dot{\lambda} = -\frac{\partial H}{\partial y} = -\lambda^T \frac{\partial f}{\partial y} - \nu^T \frac{\partial g}{\partial y} \quad (1.16)$$

$$\lambda(t_f) = \frac{\partial \Phi}{\partial y(t_f)} - \nu^T \frac{\partial \varphi}{\partial y(t_f)} \quad (1.17)$$

Eventually, the PMP is a fundamental result of optimal control theory, since it establishes 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 **ColumbiaPMP**, 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 control 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.

$$\begin{aligned} \dot{\lambda}(t) &= -\frac{\partial H}{\partial y} \\ \lambda(t_f) &= \frac{\partial \Phi}{\partial y(t_f)} \end{aligned} \quad (1.18)$$

where

$$\begin{aligned} H(y, u, \lambda, t) &:= L(y, u, t) + \lambda^T f(y, u, t) \\ \dot{y}(t) &= f(y, u, t) \end{aligned}$$

Therefore, the PMP can be expressed as the minimisation of the Hamiltonian with respect to the control variables, as indicated in equation (??). 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 **BrysonHo1975_Blaissdell** -also called Pontryagin's Minimum Principle-. 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.

$$u(t) := \arg \min_{u \in U} H(y, u, \lambda, t) \quad (1.19)$$

1.3 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.

1.3.1 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.

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned} \tag{1.20}$$

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \varphi_\mu(x) := f(x) - \mu \sum_{i=0}^n \ln(x^{(i)}) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned} \tag{1.21}$$

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.

$$\mathcal{L}(x, \lambda, z) := f(x) + c(x)^T \lambda - x^T z \tag{1.22}$$

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 .

$$\nabla f(x) + \nabla c(x) \lambda - z = 0 \tag{1.23a}$$

$$c(x) = 0 \tag{1.23b}$$

$$XZe - \mu e = 0 \tag{1.23c}$$

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 **Wachter2004**.

$$E_\mu(x, \lambda, z) := \max \left\{ \frac{\|\nabla f(x) + \nabla c(x) - z\|_\infty}{s_d}, \frac{\|XZe - \mu e\|_\infty}{s_c} \right\} \quad (1.24)$$

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 each iteration, the update in barrier parameter is obtained following the expression (??). All three expressions are computed for given constants κ_ϵ , κ_μ , θ_μ and ϵ_{tol} , as indicated in **Wachter2004**.

$$E_{\mu,j}(\tilde{x}_{j+1}^*, \tilde{\lambda}_{j+1}^*, \tilde{z}_{j+1}^*) \leq \kappa_\epsilon \mu_j \quad (1.25)$$

$$E_0(\tilde{x}^*, \tilde{\lambda}^*, \tilde{z}^*) \leq \epsilon_{tol} \quad (1.26)$$

$$\mu_{j+1} = \max \left\{ \frac{\epsilon_{tol}}{10}, \min \left\{ \kappa_\mu \mu_j, \mu_j^{\theta_\mu} \right\} \right\} \quad (1.27)$$

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 $A_k := \nabla c(x_k)$, $W_k := \nabla_{xx} \mathcal{L}(x_k, \lambda_k, z_k)$ and k the inner-loop counter-

$$\begin{bmatrix} W_k & A_k^T & -I \\ A_k & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} d_x^k \\ d_\lambda^k \\ d_z^k \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + A_k^T \lambda_k - z_k \\ c(x_k) \\ X_k Z_k e - \mu_j e \end{pmatrix} \quad (1.28)$$

From the previous expression, gradients could be computed. Thus, the update on variables and multipliers will be set following the set of equations (??) once the step size parameter α_k is determined by a filter search or filter method. A deep explanation of the filter methods used can be found on **Wachter2004** and **Fletcher2002**.

$$x_{k+1} := x_k + \alpha_k d_x^k \quad (1.29a)$$

$$\lambda_{k+1} := \lambda_k + \alpha_k d_\lambda^k \quad (1.29b)$$

$$z_{k+1} := z_k + \alpha_k d_z^k \quad (1.29c)$$

Eventually, a summarised version of the general IPOPT algorithm has been adapted from the original source **Wächter2004**.

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  $\mu$  and return to Step 2.
  end if
  4. Solve KKT System: Compute search directions  $d_k$  via expression (??).
  5. Filter Line-Search: Determine  $\alpha_k$  via the filter method.
  6. Update Iterates: Compute them using equations (??).
end for
if  $k = N$  then
  Exit: Maximum iterations reached without convergence.
end if

```

1.3.2 Implementation of IPOPT using CasADi

IPOPT is one of the integrated solvers provided by CasADi inside the *ca.nlpso()* 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 feasible 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 **Andersson2018**. In this problem, there are three different variables, an objective function and an equality constraint.

$$\begin{aligned} \min_{x,y,z} \quad & f(x, z) := x^2 + 100z^2 \\ \text{s.t.} \quad & z + (1 - x)^2 - y = 0 \end{aligned} \quad (1.30)$$

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 (??).

$$F(x, z) := x_0^2 + 100z_0^2 + x_1^2 + 100z_1^2 + \dots + x_N^2 + 100z_N^2 \quad (1.31)$$

- 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.

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

Listing 1.1: Example definition using CasADi *ca.nlpsol()* block and IPOPT solver. Extracted from **Andersson2018**.

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 (??), 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.

2 | Benchmark Problem: A Level-Flight Cruise Trajectory

This chapter introduces the reader to flight dynamics using a level-cruise flight trajectory. Also, that 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.

2.1 UAV Kinematics and Equations of Motion

Firstly, the flight dynamics for a general Unmanned Aerial Vehicle (UAV) flight trajectory had to be introduced and described. Since the aim of the project is OCT development for firefighting manoeuvres, some hypothesis have to be applied. The hypotheses considered are the following 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 $\mathcal{L} = 0$, $\mathcal{N} = 0$, $\Phi = 0$, $\dot{\Phi} = 0$, $\Psi = 0$, $\dot{\Psi} = 0$, $y = \text{const.}$ 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 Atmosphere (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 hypotheses 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) -.

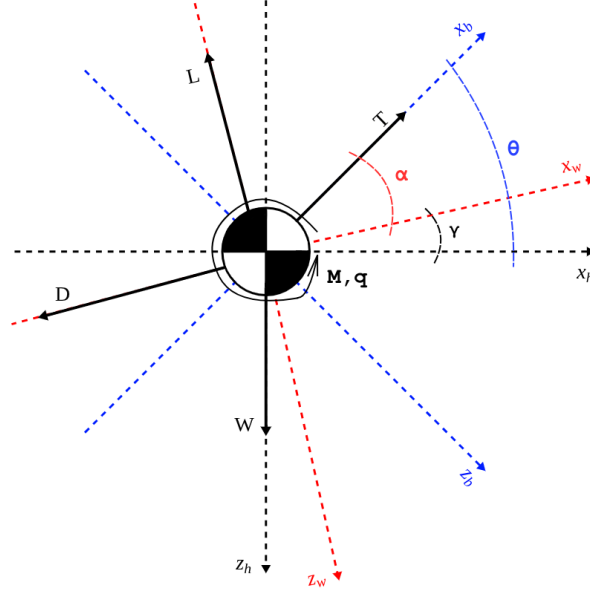


Figure 2.1: Free-body diagram of an aircraft bidimensional (2D) flight trajectory, involving forces and moments. Own source.

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.

$$\sum F_{x_b} = m \frac{du}{dt} = m\dot{u} = T + L \sin \alpha - D \cos \alpha - W \sin \theta - mqw \quad (2.1)$$

$$\sum F_{z_b} = m \frac{dw}{dt} = m\dot{w} = W \cos \theta - L \cos \alpha - D \sin \alpha + mqu \quad (2.2)$$

$$\sum M_{y_b} = I_y \frac{dq}{dt} = I_y \dot{q} = C_m \bar{q} S \bar{c} - T [\Delta X_{CT} \sin \varepsilon + \Delta Z_{CT} \cos \varepsilon] \quad (2.3)$$

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

$$\begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} := R_{hb} \begin{bmatrix} u \\ w \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} u \\ w \end{bmatrix} = \begin{bmatrix} u \cos \theta + w \sin \theta \\ -u \sin \theta + w \cos \theta \end{bmatrix} \quad (2.4)$$

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.

$$\frac{d\theta}{dt} := \dot{\theta} = q \quad (2.5)$$

$$\frac{dm}{dt} := \dot{m} = -SFC \quad (2.6)$$

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

$$L := \bar{q}SC_L = \bar{q}S \left(C_{L_0} + C_{L_\alpha}\alpha + C_{L_{\delta_e}}\delta_e \right) \quad (2.7)$$

$$D := \bar{q}SC_D = \bar{q}S \left[C_{D_0} + k \left(C_{L_0} + C_{L_\alpha}\alpha + C_{L_{\delta_e}}\delta_e \right)^2 \right] \quad (2.8)$$

$$C_m := C_{m_0} + C_{m_\alpha}\alpha + C_{m_{\delta_e}}\delta_e \quad (2.9)$$

2.2 Mathematical Formulation of the OCP

Next step is to transform the problem above into a set of equations using the Optimal Control (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.

2.2.1 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, respectively -i.e UAV speed is controlled by δ_{TPS} variations while UAV altitude is controlled by δ_e variations-.

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.

$$\mathbf{w} := \begin{bmatrix} u \\ w \\ q \\ \theta \\ x \\ z \\ m \\ \delta_{TPS} \\ \delta_e \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ u_1 \\ u_2 \end{bmatrix} \quad (2.10)$$

2.2.2 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 ??, they have to be transcribed using state space notation, which led to the following set of expressions.¹

$$\begin{aligned} \dot{x}_1 = \frac{1}{x_7} & \left[T(x_1, x_2, x_6, u_1) + L(x_1, x_2, x_6, u_2) \cdot s\alpha(x_1, x_2) \right. \\ & \left. - D(x_1, x_2, x_6, u_2) \cdot c\alpha(x_1, x_2) \right] - x_3x_2 - g \cdot sx_4 \end{aligned} \quad (2.11)$$

$$\begin{aligned} \dot{x}_2 = g \cdot cx_4 - \frac{1}{x_7} & \left[L(x_1, x_2, x_6, u_2) \cdot c\alpha(x_1, x_2) \right. \\ & \left. + D(x_1, x_2, x_6, u_2) \cdot s\alpha(x_1, x_2) \right] + x_3x_1 \end{aligned} \quad (2.12)$$

$$\begin{aligned} \dot{x}_3 = \frac{1}{I_y} & \left[C_m(u_2)\bar{q}(x_1, x_2, x_6)S\bar{c} \right. \\ & \left. - T(x_1, x_2, x_6, u_1) [\Delta X_{CT} \cdot s\varepsilon(x_1, x_2) + \Delta Z_{CT} \cdot c\varepsilon(x_1, x_2)] \right] \end{aligned} \quad (2.13)$$

$$\dot{x}_4 = x_3 \quad (2.14)$$

¹From 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.

$$\dot{x}_5 = x_1 \cdot cx_4 + x_2 \cdot sx_4 \quad (2.15)$$

$$\dot{x}_6 = -x_1 \cdot sx_4 + x_2 \cdot cx_4 \quad (2.16)$$

$$\dot{x}_7 = -SFC \quad (2.17)$$

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.

$$\mathbf{g}_{\text{dyn}} := \begin{bmatrix} \dot{x}_1 - f_1(x_1, x_2, x_3, x_4, x_6, x_7, u_1, u_2) \\ \dot{x}_2 - f_2(x_1, x_2, x_3, x_4, x_6, x_7, u_2) \\ \dot{x}_3 - f_3(x_1, x_2, x_6, u_1, u_2) \\ \dot{x}_4 - f_4(x_3) \\ \dot{x}_5 - f_5(x_1, x_2, x_4) \\ \dot{x}_6 - f_6(x_1, x_2, x_4) \\ \dot{x}_7 - SFC \end{bmatrix} = 0 \quad (2.18)$$

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-.

$$\begin{cases} 0.9 \cdot V_{TP} - \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} \leq 0 \\ \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} - 1.1 \cdot V_{TP} \leq 0 \end{cases} \quad (2.19)$$

$$\begin{cases} (h_{ref} - 3.0) + z \leq 0 \\ -z - (h_{ref} + 3.0) \leq 0 \end{cases} \quad (2.20)$$

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

$$\mathbf{g}_{\text{path}} := \begin{bmatrix} 0.9V_{tp} - \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} \\ \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} - 1.1V_{tp} \\ (h_{ref} - 3.0) + x_6 \\ -x_6 - (h_{ref} + 3.0) \end{bmatrix} \leq 0 \quad (2.21)$$

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.

$$\varphi_0 := \begin{bmatrix} x_1^0 - u_0 \\ x_2^0 - w_0 \\ x_3^0 \\ x_4^0 - \theta_0 \\ x_5^0 \\ x_6^0 - h_{ref} \\ x_7^0 - TOM \end{bmatrix} = 0 \quad (2.22)$$

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.

$$\mathbf{lbx} := \begin{bmatrix} u_{min} - x_1 \\ w_{min} - x_2 \\ -x_3 \\ \theta_{min} - x_4 \\ -x_5 \\ z_{min} - x_6 \\ BEM - x_7 \end{bmatrix} \leq 0 \quad (2.23)$$

$$\mathbf{ubx} := \begin{bmatrix} x_1 - u_{max} \\ x_2 - w_{max} \\ x_3 \\ x_4 - \theta_{max} \\ x_5 \\ x_6 - z_{max} \\ x_7 - MTOM \end{bmatrix} \leq 0 \quad (2.24)$$

2.2.3 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} .

$$J_\gamma := \bar{\gamma}^2 = \left(\frac{\gamma}{\gamma_{max}} \right)^2 = \left(\frac{\theta - \alpha}{\gamma_{max}} \right)^2 = \left(\frac{x_4 - \alpha(x_1, x_2)}{\gamma_{max}} \right)^2 \quad (2.25)$$

- 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 (??), 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 penalizing 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.

$$J_h := (\bar{h} - 1)^2 = \left(\frac{h - h_{ref}}{h_{ref}} \right)^2 = \left(\frac{-x_6 - h_{ref}}{h_{ref}} \right)^2 \quad (2.26)$$

- 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 ??, since it involves partial derivatives and a large mathematical development.

$$J_{\dot{\gamma}} := \left(q - \frac{\dot{w}u - \dot{u}w}{u^2 + w^2} \right)^2 / \dot{\gamma}_{max}^2 = \left(x_3 - \frac{\dot{x}_2 x_1 - \dot{x}_1 x_2}{x_1^2 + x_2^2} \right)^2 / \dot{\gamma}_{max}^2 \quad (2.27)$$

- 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 **stengel1994optimal**. 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-.

$$J_{\dot{\delta}_{TPS}} := \frac{(\delta_{TPS}^{k+1} - \delta_{TPS}^k)^2}{\Delta t} = \frac{(u_1^{k+1} - u_1^k)^2}{\Delta t} \quad (2.28)$$

$$J_{\dot{\delta}_e} := \frac{(\delta_e^{k+1} - \delta_e^k)^2}{\delta_{e,max}^2 \Delta t} = \frac{(u_2^{k+1} - u_2^k)^2}{\delta_{e,max}^2 \Delta t} \quad (2.29)$$

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.

$$L(x, u) := \sum_{k=0}^{N-1} (w_\gamma \cdot J_\gamma + w_h \cdot J_h + w_{\dot{\gamma}} \cdot J_{\dot{\gamma}} + w_{\dot{\delta}_{TPS}} \cdot J_{\dot{\delta}_{TPS}} + w_{\dot{\delta}_e} \cdot J_{\dot{\delta}_e}) \quad (2.30)$$

2.3 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 ??, 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-

$$\begin{aligned} \mathbf{x}^k &:= [x_1^k, x_2^k, x_3^k, x_4^k, x_5^k, x_6^k, x_7^k]^\top \\ \mathbf{u}^k &:= [u_1^k, u_2^k]^\top \\ \mathbf{w} &:= \begin{bmatrix} \mathbf{x}^0 \\ \mathbf{u}^0 \\ \mathbf{x}^1 \\ \mathbf{u}^1 \\ \vdots \\ \mathbf{x}^N \\ \mathbf{u}^N \end{bmatrix} \end{aligned} \quad (2.31)$$

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

$$\mathbf{g}(\mathbf{w}) := \begin{bmatrix} \mathbf{d}^0 \\ \vdots \\ \mathbf{d}^{N-1} \\ \mathbf{h}^0 \\ \vdots \\ \mathbf{h}^N \\ \varphi_0 \end{bmatrix}, \quad \mathbf{g}_{lb} := \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ -\infty_{4N} \\ \mathbf{0}_7 \end{bmatrix}, \quad \mathbf{g}_{ub} := \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ \mathbf{0}_{4N} \\ \mathbf{0}_7 \end{bmatrix} \quad (2.32)$$

$$\begin{aligned}\mathbf{w}_{lb} &:= \mathbf{1}_N \otimes \mathbf{lbx} \\ \mathbf{w}_{ub} &:= \mathbf{1}_N \otimes \mathbf{ubx}\end{aligned} \quad (2.33)$$

$$\begin{aligned}L^k &:= L(\mathbf{x}^k, \mathbf{u}^k) \\ J(\mathbf{w}) &:= \frac{\Delta t}{2} \sum_{k=0}^{N-1} (L^k + L^{k+1})\end{aligned} \quad (2.34)$$

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

$$\begin{aligned}\min_{\mathbf{w}} \quad & J(\mathbf{w}) := \frac{\Delta t}{2} \sum_{k=0}^{N-1} (L^k + L^{k+1}) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{w}) \in [\mathbf{g}_{lb}, \mathbf{g}_{ub}] \\ & \mathbf{w} \in [\mathbf{1}_N \otimes \mathbf{lbx}, \mathbf{1}_N \otimes \mathbf{ubx}]\end{aligned} \quad (2.35)$$

The algorithm that has been implemented to solve the NLP problem using IPOPT solver is the referred below. The whole structure of the implemented algorithm in Python can be seen in appendix [ANEXO CÓDIGO](#), referring to benchmark problem code section.

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 $\mathbf{w} := [\mathbf{x}^0, \mathbf{u}^0, \dots, \mathbf{x}^N, \mathbf{u}^N]^\top \in \mathbb{R}^{9N}$.
Apply variable bounds: $\mathbf{w} \in [\mathbf{1}_N \otimes \mathbf{lb}\mathbf{x}, \mathbf{1}_N \otimes \mathbf{ub}\mathbf{x}]$.

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: $\mathbf{g}(\mathbf{w}) := [\mathbf{d}^0, \dots, \mathbf{d}^{N-1}, \mathbf{h}^0, \dots, \mathbf{h}^N, \varphi_0]^\top$.

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.

2.4 Results

2.5 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.

2.5.1 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:

$$\mathbf{w} := \begin{bmatrix} \mathbf{x}^0 \\ \mathbf{u}^0 \\ \mathbf{x}^1 \\ \mathbf{u}^1 \\ \vdots \\ \mathbf{x}^N \\ \mathbf{u}^N \\ t_F \end{bmatrix}, \quad \mathbf{w}_{lb} := \begin{bmatrix} \mathbf{1}_N \otimes \mathbf{l} \mathbf{b} \mathbf{x} \\ t_{F,min} \end{bmatrix}, \quad \mathbf{w}_{ub} := \begin{bmatrix} \mathbf{1}_N \otimes \mathbf{u} \mathbf{b} \mathbf{x} \\ t_{F,max} \end{bmatrix} \quad (2.36)$$

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-.

$$\mathbf{h}_{dist} := \begin{bmatrix} V_{tp} \cdot t_{F,min} - (x_f - x_0) \\ (x_f - x_0) - V_{tp} \cdot t_{F,max} \end{bmatrix} \leq \mathbf{0} \quad (2.37)$$

$$\mathbf{g}(\mathbf{w}) := \begin{bmatrix} \mathbf{d}^0 \\ \vdots \\ \mathbf{d}^{N-1} \\ \mathbf{h}^0 \\ \vdots \\ \mathbf{h}^N \\ \mathbf{h}_{dist} \\ \varphi_0 \end{bmatrix}, \quad \mathbf{g}_{lb} := \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ -\infty_{4N} \\ -\infty_2 \\ \mathbf{0}_7 \end{bmatrix}, \quad \mathbf{g}_{ub} := \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ \mathbf{0}_{4N} \\ \mathbf{0}_2 \\ \mathbf{0}_7 \end{bmatrix} \quad (2.38)$$

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 $\Delta t = 1/(N - 1)$.

$$J(\mathbf{w}) := \frac{\Delta t \cdot t_F}{2} \sum_{k=0}^{N-1} (L^k + L^{k+1}) + w_{t_F} \cdot t_F \quad (2.39)$$

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

$$\begin{aligned} \min_{\mathbf{w}} \quad & J(\mathbf{w}) := \frac{\Delta t \cdot t_F}{2} \sum_{k=0}^{N-1} (L^k + L^{k+1}) + w_{t_F} \cdot t_F \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{w}) \in [\mathbf{g}_{lb}, \mathbf{g}_{ub}] \\ & \mathbf{w} \in \left[\begin{bmatrix} \mathbf{1}_N \otimes \mathbf{lbx} \\ t_{F,min} \end{bmatrix}, \begin{bmatrix} \mathbf{1}_N \otimes \mathbf{ubx} \\ t_{F,max} \end{bmatrix} \right] \end{aligned} \quad (2.40)$$

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

1. Initialise Parameters:

Define N nodes, time-step $\Delta t = 1/(N - 1)$, 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 $\mathbf{w} := [\mathbf{x}^0, \mathbf{u}^0, \dots, \mathbf{x}^N, \mathbf{u}^N, t_F]^\top \in \mathbb{R}^{9N+1}$.
Apply variable bounds: $\mathbf{w} \in \left[\begin{bmatrix} \mathbf{1}_N \otimes \mathbf{lbx} \\ t_{F,min} \end{bmatrix}, \begin{bmatrix} \mathbf{1}_N \otimes \mathbf{ubx} \\ t_{F,max} \end{bmatrix} \right]$.

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: $\mathbf{g}(\mathbf{w}) := [\mathbf{d}^0, \dots, \mathbf{d}^{N-1}, \mathbf{h}^0, \dots, \mathbf{h}^N, \mathbf{h}_{dist}, \varphi_0]^\top$.

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.

2.5.2 Results

3 | 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.

3.1 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.

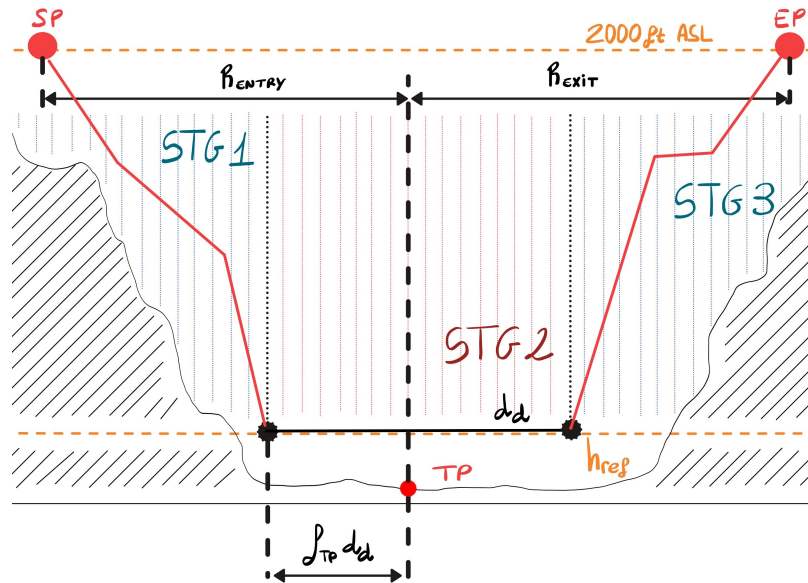


Figure 3.1: CONOPS of Flyox autonomous firefighting manoeuvres. Own source.

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 pre-defined 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 3.1: 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. $d_{2,min} := V_{TP} \cdot t_{2,min}$	2	[m]
$d_{2,max}$	Maximum discharge distance. $d_{2,max} := V_{TP} \cdot t_{2,max}$	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	[-]

3.2 OCP Formulation and NLP Implementation

This section contains the formulation of each manoeuvre's stage 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 ?? from expression (??) to (??). 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 (??).

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

3.2.1 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 end state at known altitude and horizontal distance. Initial state computation will be introduced at section ?? 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:

$$g_{path,1} := \begin{cases} V_S - \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} \leq 0 \\ \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} - V_{NO} \leq 0 \\ -z^N - h_{ref} = 0 \\ (x^N - x^0) - (R_{ENTRY} - f_{TP} \cdot d_{2,min}) = 0 \end{cases} \quad (3.1)$$

Converting the expressions to NLP notation, the equations can be stated as follows. In addition, \mathbf{d} and φ_0 are the dynamic constraints vector and initial state constraints vector, respectively -see expression (??) and its meaning-.

$$\mathbf{h}_1 := \begin{bmatrix} V_S - \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} \\ \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} - V_{NO} \end{bmatrix} \leq 0 \quad (3.2)$$

$$\mathbf{h}_{1,end} := \begin{bmatrix} -x_6^N - h_{ref} \\ (x_5^N - x_5^0) - (R_{ENTRY} - f_{TP} \cdot d_{2,min}) \end{bmatrix} = 0 \quad (3.3)$$

$$\mathbf{g}(\mathbf{w}) := \begin{bmatrix} \mathbf{d}^0 \\ \vdots \\ \mathbf{d}^{N-1} \\ \mathbf{h}_1^0 \\ \vdots \\ \mathbf{h}_1^N \\ \mathbf{h}_{1,end} \\ \varphi_0 \end{bmatrix}, \quad \mathbf{g}_{lb} := \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ -\infty_{2N} \\ \mathbf{0}_2 \\ \mathbf{0}_7 \end{bmatrix}, \quad \mathbf{g}_{ub} := \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ \mathbf{0}_{2N} \\ \mathbf{0}_2 \\ \mathbf{0}_7 \end{bmatrix} \quad (3.4)$$

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 (??). 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 maintain 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} -.

$$J_\gamma := (\bar{\gamma} - 1)^2 = \left(\frac{\gamma - \gamma_d}{\gamma_d} \right)^2 = \left(\frac{\theta - \alpha - \gamma_d}{\gamma_d} \right)^2 = \left(\frac{x_4 - \alpha(x_1, x_2)}{\gamma_d} - 1 \right)^2 \quad (3.5)$$

- 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.

$$J_{\dot{\gamma}} := \left(q - \frac{\dot{w}u - \dot{u}w}{u^2 + w^2} \right)^2 / \dot{\gamma}_{min}^2 = \left(x_3 - \frac{\dot{x}_2 x_1 - \dot{x}_1 x_2}{x_1^2 + x_2^2} \right)^2 / \dot{\gamma}_{min}^2 \quad (3.6)$$

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

$$J_{\dot{\delta}_{TPS}} := \frac{(\delta_{TPS}^{k+1} - \delta_{TPS}^k)^2}{\Delta t} = \frac{(u_1^{k+1} - u_1^k)^2}{\Delta t} \quad (3.7)$$

$$J_{\dot{\delta}_e} := \frac{(\delta_e^{k+1} - \delta_e^k)^2}{\delta_{e,max}^2 \Delta t} = \frac{(u_2^{k+1} - u_2^k)^2}{\delta_{e,max}^2 \Delta t} \quad (3.8)$$

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

$$L(x, u) := \sum_{k=0}^{N-1} (w_\gamma \cdot J_\gamma + w_{\dot{\gamma}} \cdot J_{\dot{\gamma}} + w_{\dot{\delta}_{TPS}} \cdot J_{\dot{\delta}_{TPS}} + w_{\dot{\delta}_e} \cdot J_{\dot{\delta}_e}) \quad (3.9)$$

$$\begin{aligned} L^k &:= L(\mathbf{x}^k, \mathbf{u}^k) \\ J(\mathbf{w}) &:= \frac{\Delta t}{2} \sum_{k=0}^{N-1} (L^k + L^{k+1}) + w_{t_{F,1}} \cdot t_{F,1} \end{aligned} \quad (3.10)$$

3.2.2 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:

$$\dot{x}_7 := \begin{cases} -SFC & x_5 < (V_{TP} \cdot f_{TP} \cdot t_{2,min} - t_d/2) \\ -SFC & x_5 > [V_{TP} \cdot (f_{TP} \cdot t_{2,min} + t_d/2)] \\ -SFC - \frac{PM}{t_d} & \text{otherwise} \end{cases} \quad (3.11)$$

Function above can be explained using distances because each stage has been defined as a separated problem and, therefore, distances are relative. 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 $t = f_{TP} \cdot t_{2,min} + t_d$.

3.2.3 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 ?? and they will only appear referred 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:

$$g_{path,3} := \begin{cases} V_S - \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} \leq 0 \\ \sqrt{(u - V_{x_w})^2 + (w - V_{z_w})^2} - V_{NO} \leq 0 \\ -z^N - h_{ref} = 0 \\ (x^N - x^0) - [R_{EXIT} - (1 - f_{TP}) \cdot d_{2,min}] = 0 \end{cases} \quad (3.12)$$

Then, applying the transformation into NLP notation it rests like the expressions below. As explained before, \mathbf{d} and φ_0 are the dynamic constraints vector and initial state constraints vector, respectively -see expression (??) and its meaning-.

$$\mathbf{h}_3 := \begin{bmatrix} V_S - \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} \\ \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} - V_{NO} \end{bmatrix} \leq 0 \quad (3.13)$$

$$\mathbf{h}_{3,end} := \begin{bmatrix} -x_6^N - h_{ref} \\ (x_5^N - x_5^0) - [R_{EXIT} - (1 - f_{TP}) \cdot d_{2,min}] \end{bmatrix} = 0 \quad (3.14)$$

$$\mathbf{g}(\mathbf{w}) := \begin{bmatrix} \mathbf{d}^0 \\ \vdots \\ \mathbf{d}^{N-1} \\ \mathbf{h}_3^0 \\ \vdots \\ \mathbf{h}_3^N \\ \mathbf{h}_{3,end} \\ \varphi_0 \end{bmatrix}, \quad \mathbf{g}_{lb} := \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ -\infty_{2N} \\ \mathbf{0}_2 \\ \mathbf{0}_7 \end{bmatrix}, \quad \mathbf{g}_{ub} := \begin{bmatrix} \mathbf{0}_{7(N-1)} \\ \mathbf{0}_{2N} \\ \mathbf{0}_2 \\ \mathbf{0}_7 \end{bmatrix} \quad (3.15)$$

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).

$$J_{\dot{\gamma}} := \left(q - \frac{\dot{w}u - \dot{u}w}{u^2 + w^2} \right)^2 / \dot{\gamma}_{max}^2 = \left(x_3 - \frac{\dot{x}_2 x_1 - \dot{x}_1 x_2}{x_1^2 + x_2^2} \right)^2 / \dot{\gamma}_{max}^2 \quad (3.16)$$

- 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 aggressive manoeuvres. It ensures the aircraft maintains a sufficient margin from the aerodynamic stall regime during the ascent stage. For the penalty defined below, margin is $\alpha_{mg} := 0.25 \cdot \theta_{max}$ and safe angle is $\alpha_{sf} := 0.75 \cdot \theta_{max}$ -in this case θ_{max} has been used as α_{stall} for simplicity-.

$$J_{sp} := \left(\frac{\max[0, \alpha(u, w) - \alpha_{sf}]}{\alpha_{mg}} \right)^2 = \left(\frac{\max[0, \alpha(x_1, x_2) - \alpha_{sf}]}{\alpha_{mg}} \right)^2 \quad (3.17)$$

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

$$J_{\dot{\delta}_{TPS}} := \frac{(\delta_{TPS}^{k+1} - \delta_{TPS}^k)^2}{\Delta t} = \frac{(u_1^{k+1} - u_1^k)^2}{\Delta t} \quad (3.18)$$

$$J_{\dot{\delta}_e} := \frac{(\delta_e^{k+1} - \delta_e^k)^2}{\delta_{e,max}^2 \Delta t} = \frac{(u_2^{k+1} - u_2^k)^2}{\delta_{e,max}^2 \Delta t} \quad (3.19)$$

- 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.

$$J_{\delta_{e,sat}} := \left(\frac{\delta_e}{\delta_{e,max}} \right)^4 = \left(\frac{u_2}{\delta_{e,max}} \right)^4 \quad (3.20)$$

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

$$L(x, u) := \sum_{k=0}^{N-1} \left(w_{\dot{\gamma}} \cdot J_{\dot{\gamma}} + w_{sp} \cdot J_{sp} + w_{\dot{\delta}_{TPS}} \cdot J_{\dot{\delta}_{TPS}} + w_{\dot{\delta}_e} \cdot J_{\dot{\delta}_e} + w_{\dot{\delta}_{e,sat}} \cdot J_{\dot{\delta}_{e,sat}} \right) \quad (3.21)$$

$$\begin{aligned} L^k &:= L(\mathbf{x}^k, \mathbf{u}^k) \\ J(\mathbf{w}) &:= \frac{\Delta t}{2} \sum_{k=0}^{N-1} \left(L^k + L^{k+1} \right) + w_{t_{F,3}} \cdot t_{F,3} \end{aligned} \quad (3.22)$$

3.3 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 in different files and functions. Essentially, it is possible to distinguish between:

- *simulation.py*, which is the function that extracts the information stored in JSON file *Simulation.json*. This file contains the information and problem parameters defined in ??, as well as the mission bounds to the defined problem. In addition, numerical values as the number of nodes can be modified, every simulation. In the JSON file, the aircraft model can be selected and this will load automatically all geometrical, physical and operational information about the aircraft. 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 up by different functions to plot the whole manoeuvre trajectories and cost objective or, individual trajectories and costs 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 successfully.

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 weights for each stage penalty.

Aircraft and atmosphere parameters retrieving by *aircraft.py* and *atmosphere.py* calls.

2. Compute initial conditions at SP:

Computation of initial state and control vector \mathbf{w}_0 .

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

Fix $q_0 = 0, x_0 = 0, z_0 = -h_{cruise}$ and $m_0 = TOM$. $\mathbf{x}_0 := [u_0, w_0, q_0, \theta_0, x_0, z_0, m_0]$

Using a static NLP solver obtain initial controls $\mathbf{u}_0 := [\delta_{TPS,0}, \delta_{e,0}]$

Stack variables to create STG1 $\mathbf{w}_0 := [\mathbf{x}_0^0, \mathbf{u}_0^0, \dots, \mathbf{x}_0^N, \mathbf{u}_0^N, t_1^0]$

3. NLP Creation for STG1:

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

Apply variable bounds: $\mathbf{w} \in \left[\begin{smallmatrix} \mathbf{1}_N \otimes \mathbf{lb}_x \\ t_{F,min} \end{smallmatrix} \right], \left[\begin{smallmatrix} \mathbf{1}_N \otimes \mathbf{ub}_x \\ t_{F,max} \end{smallmatrix} \right]$.

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

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

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

4. Solve STG1:

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

Extract state and control trajectories, cost objective per iteration and final computation time.

5. NLP Creation for STG2:

Extract end-state from STG1 $\mathbf{x}_f := \mathbf{w}_1^N \in \mathbb{R}^7$.

Using a static NLP solver obtain initial controls $\mathbf{u}_0 := [\delta_{TPS,0}, \delta_{e,0}]$

Generate linking condition $\mathbf{w}_0 := [\mathbf{x}_f^0, \mathbf{u}_0^0, \dots, \mathbf{x}_f^N, \mathbf{u}_0^N, t_2^0]$

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

6. Solve STG2:

Repeat **Step 4** but with STG2 NLP dictionary.

7. NLP Creation for STG3:

Extract end-state from STG2 $\mathbf{x}_f := \mathbf{w}_2^N \in \mathbb{R}^7$.

Using a static NLP solver obtain initial controls $\mathbf{u}_0 := [\delta_{TPS,0}, \delta_{e,0}]$

Generate linking condition $\mathbf{w}_0 := [\mathbf{x}_f^0, \mathbf{u}_0^0, \dots, \mathbf{x}_f^N, \mathbf{u}_0^N, t_2^0]$

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

6. Solve STG2:

Repeat **Step 4** but with STG3 NLP dictionary.

7. Post-processing:

Reconstruction of whole manoeuvre state and control trajectories. Stack all states and controls except from horizontal position which has to be stacked as follows:

$$\mathbf{x}_5 = [\mathbf{x}_{5,STG1}, \mathbf{x}_{5,STG2} + x_{5,STG1}^N, \mathbf{x}_{5,STG3} + x_{5,STG2}^N]$$

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

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

4 | Firefighting Manoeuvre: Results

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.

$$\gamma := \theta - \alpha \quad (\text{A.1})$$

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.

$$\dot{\gamma} := \frac{d}{dt}(\theta - \alpha) = \frac{d\theta}{dt} - \frac{d\alpha}{dt} = \dot{\theta} - \dot{\alpha} = q - \dot{\alpha} \quad (\text{A.2})$$

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.

$$\alpha(u_a, w_a) := \arctan(w_a, u_a) \approx \arctan(w, u) \quad (\text{A.3})$$

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

$$\dot{\alpha} := \frac{\partial \alpha}{\partial u} \frac{\partial u}{\partial t} + \frac{\partial \alpha}{\partial w} \frac{\partial w}{\partial t} = \frac{\partial \alpha}{\partial u} \dot{u} + \frac{\partial \alpha}{\partial w} \dot{w} \quad (\text{A.4})$$

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

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

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

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

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

$$\dot{\gamma} := q - \frac{\dot{w}u - \dot{u}w}{u^2 + w^2} \quad (\text{A.6})$$