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

Abstract

Acknoledgments

Preface

Scope and Objectives

Motivation and Justification

Methodological Approach

Document Structure

Contents

1 Fundamentals of Optimal Control and its Implementation **1**
1.1 Introduction to Optimal Control Theory 1
1.1.1 Mathematical formulation 3
1.2 Direct and Indirect Methods 4
1.2.1 Direct collocation method as direct method 5
1.2.2 Pontryagin's maximum principle as indirect method 7
1.3 Introduction to IPOPT 9
1.3.1 Mathematical formulation 10
1.3.2 Implementation of IPOPT using CasADi 12

2 Benchmark Problem: A Level-Flight Cruise Trajectory **14**
2.1 UAV Kinematics and Equations of Motion 14
2.2 Mathematical Formulation of the OCP 16
2.2.1 States and controls 16
2.2.2 Constraints and bounds 16
2.2.3 Cost functional 19
2.3 NLP Formulation and Implementation via CasADi 20
2.4 Results 23
2.5 Addition of the Free-End Condition 23
2.5.1 NLP formualtion and implementation 23
2.5.2 Results 23

List of Figures

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**. 2

List of Tables

Acronyms

DCM Direct Collocation Method.

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.

TPS Throttle Position Sensor.

UAV Unmanned Aerial Vehicle.

Nomenclature

Symbol	Description	Units
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	$[rad]$

1 | Fundamentals of Optimal Control and its Implementation

This chapter includes the mathematical formualtion 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 realted 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 once presented. 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**.

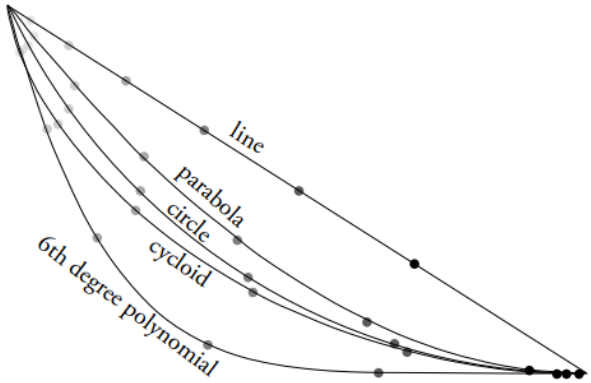

Figure 1.1: The brachistochrone problem. The ball rolls down the fastest on a cycloidal ramp. Points of the same shade correspond to the same moment in time. Extracted directly from **Blasjo2021**.

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

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.

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

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 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 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 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 states 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 $J(y, u) = \Phi[y(t_f), t_f]$, is it 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 $J(y, u) = \int_{t_0}^{t_f} L[y(t), u(t), t]\ dt$, is it said that the whole cost functional is in its Lagrange form.

1.2 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 discretized 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 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**.

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

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

The previous expressions (1.3) and (1.4) 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:

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

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.

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

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 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:

$$\int_{t_0}^{t_f} L(y,u,t)\ dt \approx \sum_{k=0}^{N-1} \frac{1}{2} h_k \cdot (L_k + L_{k+1})$$
$$\begin{aligned} &\textit{where} \\ &\quad h_k \coloneqq t_{k+1} - t_k \\ &\quad L_k \coloneqq L[y(t_k), u(t_k), t_k] \\ &\quad L_{k+1} \coloneqq L[y(t_{k+1}), u(t_{k+1}), t_{k+1}] \\ &\quad t_k \text{ for each } k \in [0, N-1] \end{aligned} \tag{1.6}$$

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

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

Resulting in:

$$\begin{aligned} &y_{k+1} - y_k = \frac{1}{2} h_k \cdot (f_k + f_{k+1}) \\ &\textit{where} \\ &\quad f_k = f(y_k, u_k, t_k) \text{ for each } k \in [0, N] \end{aligned} \tag{1.7}$$

Eventually, the terminal cost can be added taken 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.

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

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 thre 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.

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

$$
\mathcal{L}(x, z, \mu) \coloneqq f(x) + \sum_{i=0}^{m} z_i g_i(x) + + \sum_{j=0}^{p} \mu_j h_j(x) \tag{1.10}
$$

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 **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.

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

- 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.

$$z_i g_i(x) = 0 \tag{1.12}$$

- 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.

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

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

$$z_i \geq 0 \tag{1.14}$$

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's 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 (1.15), can be defined like in a boundary value problem in his canonical form with ODEs in all time domain $t_0 \leq t \leq t_f$ **Stryk1992Leonard1992**.

$$H\ (\lambda, \nu, t) \coloneqq \lambda^T f + \nu^T g \tag{1.15}$$

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**.

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

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

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 **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 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.

$$\begin{aligned} &\dot{\lambda}(t) = -\frac{\partial H}{\partial y} \\ &\lambda(T) = \frac{\partial \Phi}{\partial y(T)} \\ &\textit{where} \\ &\quad H(y, u, \lambda, t) \coloneqq L(y, u, t) + \lambda^T f(y, u, t) \\ &\quad \dot{y}(t) = f(y, u, t) \end{aligned} \tag{1.18}$$

Therefore, the PMP can be expressed as the minimisation of the Hamiltonian with respect to the control variables, as indicated in equation (1.19). 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 **BrysonHo1975_Blaisdell** -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 costate- 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.

$$u(t) \coloneqq \arg\min_{u \in U} \; H \; (y, u, \lambda, t) \tag{1.19}$$

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.

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

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

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.

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

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 **Waechter2004**.

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

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 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 (1.27). All three expresions are computed for given constants κ_ϵ, κ_μ, θ_μ and ϵ_{tol}, as indicated in **Waechter2004**.

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

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

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

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.

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

From the previous expression, gradients could be computed. Thus, the update on variables and multipliers will be set following the set of equations (1.27) once the step size parameter α_k is determined by a filter search or filter method. A deep explanation of filther methods used can be found on **Waechter2004** and **Fletcher2002**.

$$x_{k+1} := x_k + \alpha_k d_x^k \tag{1.29a}$$
$$\lambda_{k+1} := \lambda_k + \alpha_k d_\lambda^k \tag{1.29b}$$
$$z_{k+1} := z_k + \alpha_z^k d_z^k \tag{1.29c}$$

Eventually, a summarised version of the general IPOPT algorithm has been adapted from the original source **Waechter2004**.

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**
  // Start of the Iterative Process
  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 (1.28).
  5. Filter Line-Search: Determine α_k via the filter method.
  6. Update Iterates: $x_{k+1} = x_k + \alpha_k d_k^x$, etc.
end for
// End of the Iterative Process
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.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 **Andersson2018**. In this problem, there are three different variables, an objective function and an equality cosntraint.

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

The directrices 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, $w = (x_0, y_0, z_0, x_1, y_1, z_1 \ldots x_N, y_n, z_N)$-.
- 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 (1.31).

$$F(x,z) \coloneqq x_0^2 + 100z_0^2 + x_1^2 + 100z_1^2 + \ldots + x_N^2 + 100z_N^2 \tag{1.31}$$

- Constraint $g(x, y, z)$: 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.

```
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)
```

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 (1.8), 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 a vector -noted as *lbg* and *ubg* through all thesis and provided code-. That last remark also applies to simple bounds x_{lb} and x_{ub}, 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 notation is presented right after the equations of motion and, right before presenting the results. Eventually, the free-end condition is applied to the problem, 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 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 condistions shall be modelled using International Standard Atmoshpere (ISA) model. Only static bidimensional (2D) wind field could be sonsidered, which means that free-stream velocity has to be constant in all domain.
- Bidimensional (2D) level-flight cruise trajectory implies a nul flight path angle and nul 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)-.

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.

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

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

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

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

$$\begin{bmatrix}\dot{x}\\ \dot{z}\end{bmatrix} \coloneqq R_{hb}\begin{bmatrix}u\\ w\end{bmatrix} = \begin{bmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}u\\ w\end{bmatrix} = \begin{bmatrix}u\cos\theta + w\sin\theta\\ -u\sin\theta + w\cos\theta\end{bmatrix} \tag{2.4}$$

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 consuption is considered on mass variation equation.

$$\frac{d\theta}{dt} \coloneqq \dot{\theta} = q \tag{2.5}$$

$$\frac{dm}{dt} \coloneqq \dot{m} = -SFC \tag{2.6}$$

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

$$L \coloneqq \frac{1}{2}\bar{q}SC_L = \frac{1}{2}\bar{q}S\left(C_{L,0} + C_{L,\alpha}\alpha + C_{L,\delta_e}\delta_e\right) \tag{2.7}$$

$$D \coloneqq \frac{1}{2}\bar{q}SC_D = \frac{1}{2}\bar{q}S\left[C_{D,0} + k\left(C_{L,0} + C_{L,\alpha}\alpha + C_{L,\delta_e}\delta_e\right)^2\right] \tag{2.8}$$

$$C_m \coloneqq C_{m,0} + C_{m,\alpha}\alpha + C_{m,\delta_e}\delta_e \tag{2.9}$$

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 descriptions is 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 variations in TPS 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 to be states 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.

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

2.2.2 Constraints and bounds

After the state space is set, the constraints can be defined, following 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 prestablished order, which starts by dynamic constraints. Since dynamic constraints have been defined

in section 2.1, they have to be transcribed using state space notation, which led to the following set of expressions.[1]

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

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

$$\begin{aligned}\dot{x_3} = \frac{1}{I_y}\Big[&C_m(u_2)\bar{q}(x_1,x_2,x_6)S\bar{c}\\ &- T(x_1,x_2,x_6,u_1)\left[\Delta X_{CT}\cdot s\varepsilon(x_1,x_2) + \Delta Z_{CT}\cdot c\varepsilon(x_1,x_2)\right]\Big]\end{aligned} \tag{2.13}$$

$$\dot{x_4} = x_3 \tag{2.14}$$

$$\dot{x_5} = x_1\cdot cx_4 + x_2\cdot sx_4 \tag{2.15}$$

$$\dot{x_6} = -x_1\cdot sx_4 + x_2\cdot cx_4 \tag{2.16}$$

$$\dot{x_7} = -SFC \tag{2.17}$$

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

$$\mathbf{g_{dyn}} \coloneqq \begin{bmatrix}\dot{x_1} - f_1(x_1,x_2,x_3,x_4,x_6,x_7,u_1,u_2)\\ \dot{x_2} - f_2(x_1,x_2,x_3,x_4,x_6,x_7,u_2)\\ \dot{x_3} - f_3(x_1,x_2,x_6,u_1,u_2)\\ \dot{x_4} - f_4(x_3)\\ \dot{x_5} - f_5(x_1,x_2,x_4)\\ \dot{x_6} - f_6(x_1,x_2,x_4)\\ \dot{x_7} - SFC\end{bmatrix} = 0 \tag{2.18}$$

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

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 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 has to introduce some margins with target constant values-.

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

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

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

$$\mathbf{g_{path}} \coloneqq \begin{bmatrix} 0.9V_{tp} - \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} \\ \sqrt{(x_1 - V_{x_w})^2 + (x_2 - V_{z_w})^2} - 1.1V_{tp} \\ (h_{ref} - 3.0) + x_6 \\ -x_6 - (h_{ref} + 3.0) \end{bmatrix} \leq 0 \tag{2.21}$$

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 enforced also.

$$\varphi_{\mathbf{0}} \coloneqq \begin{bmatrix} x_1^0 - u_0 \\ x_2^0 - w_0 \\ x_3^0 \\ x_4^0 - \theta_0 \\ x_5^0 \\ x_6^0 - h_{ref} \\ x_7^0 - TOM \end{bmatrix} = 0 \tag{2.22}$$

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.

$$\mathbf{lbx} \coloneqq \begin{bmatrix} u_{min} - x_1 \\ w_{min} - x_2 \\ -x_3 \\ \theta_{min} - x_4 \\ -x_5 \\ z_{min} - x_6 \\ BEM - x_7 \end{bmatrix} \leq 0 \tag{2.23}$$

$$\mathbf{ubx} \coloneqq \begin{bmatrix} x_1 - u_{max} \\ x_2 - w_{max} \\ x_3 \\ x_4 - \theta_{max} \\ x_5 \\ x_6 - z_{max} \\ x_7 - MTOM \end{bmatrix} \leq 0 \tag{2.24}$$

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 can 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}.

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

- 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 hard path constraints defined in (2.20), 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 penalizing deviations from the centerline—mimics real-world flight control laws. It allows the

optimiser to prioritize control smoothness and actuator limits over rigid altitude tracking, resulting in a more physically realistic and numerically stable trajectory.

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

- 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 annex Annex:Gamma_Dot, since it involves partial derivatives and a large mathematical development.

$$J_{\dot{\gamma}} := \left(\frac{\dot{w}u - \dot{u}w}{u^2 + w^2}\right)^2 / \dot{\gamma}^2_{max} = \left(\frac{\dot{x_2}x_1 - \dot{x_1}x_2}{x_1^2 + x_2^2}\right)^2 / \dot{\gamma}^2_{max} \tag{2.27}$$

- 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 "lazy pilot" implementation. This implementation basically reduces pilot 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-.

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

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

Once all penalties have been defined, the whole cost functional ins its Lagrange form can be stated as follows. Notice that notation has been indicated in its discretised form, using time-instant t_k, and each cost term has a penalty weight associated.

$$L(x,u) := \sum_{k=0}^{N-1} \left(w_\gamma \cdot J^k_\gamma + w_h \cdot J^k_h + w_{\dot{\gamma}} \cdot J^k_{\dot{\gamma}} + w_{\dot{\delta}_{TPS}} \cdot J_{\dot{\delta}_{TPS}} + w_{\dot{\delta}_e} \cdot J_{\dot{\delta}_e}\right) \tag{2.30}$$

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 Lagrangian cost term, it will be evaluated also following a trapezoidal scheme.

These 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 1.3.2, which means to create state and controls, 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-

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

$$\mathbf{w} := \begin{bmatrix} \mathbf{x}^0 \\ \mathbf{u}^0 \\ \mathbf{x}^1 \\ \mathbf{u}^1 \\ \vdots \\ \mathbf{x}^N \\ \mathbf{u}^N \end{bmatrix} \tag{2.31}$$

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

$$\mathbf{g}(\mathbf{w}) := \left[\begin{array}{c} \mathbf{d}^0 \\ \vdots \\ \mathbf{d}^{N-1} \\ \hline \mathbf{h}^0 \\ \vdots \\ \mathbf{h}^N \\ \hline \varphi_0 \end{array}\right], \quad \mathbf{g}_{lb} := \begin{bmatrix} \mathbf{0}_{7N} \\ -\infty_{4(N+1)} \\ \mathbf{0}_7 \end{bmatrix}, \quad \mathbf{g}_{ub} := \begin{bmatrix} \mathbf{0}_{7N} \\ \mathbf{0}_{4(N+1)} \\ \mathbf{0}_7 \end{bmatrix} \tag{2.32}$$

$$\begin{aligned} \mathbf{w}_{lb} &:= \mathbf{1}_{N+1} \otimes \mathbf{lbx} \\ \mathbf{w}_{ub} &:= \mathbf{1}_{N+1} \otimes \mathbf{ubx} \end{aligned} \tag{2.33}$$

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

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

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

The algorithm that has been implemented to solve the NLP problem using IPOPT solver is the reffered below. The whole structure of the implemented algorithm in Python can be seen in appendix ANEXO CÓDIGO, referring to benckmark problem code section.

Algorithm 2 Trajectory Optimisation Algorithm for Level-Cruise Flight

1. Initialise Parameters:
Define N nodes, end-time t_F, 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 $\mathbf{w} := [\mathbf{x}^0, \mathbf{u}^0, \ldots, \mathbf{x}^N, \mathbf{u}^N]^\top \in \mathbb{R}^{9N}$.
Apply variable bounds: $\mathbf{w} \in [\mathbf{1}_{N+1} \otimes \mathbf{lbx},\ \mathbf{1}_{N+1} \otimes \mathbf{ubx}]$.
for $k = 0$ **to** N **do**
3. Evaluate NLP Functions:
Compute trapezoidal Cost: $J(\mathbf{w}) = \frac{\Delta t}{2} \sum_{k=0}^{N-2} (L^k + L^{k+1})$.
Evaluate dynamics $\mathbf{d}^k$ and path Constraints $\mathbf{h}^k$.
Stack constraints: $\mathbf{g}(\mathbf{w}) := [\mathbf{d}^0, \ldots, \mathbf{d}^{N-1}, \mathbf{h}^0, \ldots, \mathbf{h}^N, \varphi_0]^\top$.
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_k.
6. Filter Line-Search: Determine step size α_k to ensure decrease in J or constraint violation $\|\mathbf{g}\|$.
7. Update Iterates: $\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k d_k^w$.
end for
8. Post-processing: Extract state and control trajectories for plotting.

2.4 Results

2.5 Addition of the Free-End Condition

2.5.1 NLP formualtion and implementation

2.5.2 Results