UNIVERSIDADE
BEIRA INTERIOR

B-spline Parameterization Based Flight Trajectory Optimization

Rose Avelino Correia Teixeira

Dissertação para obtenção do Grau de Mestre em

Engenharia Aeronáutica

(Ciclo de estudos integrados)

Orientador: Prof. Doutor Kouamana Bousson

Junho de 2022

Dedication

To my family, for all their support.

Acknowledgements

I would like to thank Professor K. Bousson for all his guidance during the development of this work, and along all my academic road at UBI. His constant motivation on the different subjects, engaging the students on the importance of practical applications of the lectured contents, greatly contributed to the choice of the object of study.

I am eternally grateful to *Camões I. P* for the scholarship during my entire journey at UBI which enabled me to attend the course.

I would like to express my gratitude to all my family, my parents Atanásia and Avelino, my sisters Yara and Sandra, my brother Elisandro, and my wife Ivana, for all their support throughout the years.

Finally, my great appreciation for each Professor at UBI that I had the honor to learn with, thank you for sharing your knowledge and wisdom.

Resumo

Materializar o primeiro voo de um dispositivo mais pesado do que o ar durante o século XX constituiu certamente um grande marco na história da humanidade. Contudo, o voo em si raramente constitui o objetivo final. Para se executar uma missão de forma efetiva, é necessário determinar a trajetória a ser seguida de acordo com o objetivo da missão, por exemplo minimizando o tempo decorrido, o comprimento da trajetória, etc. Como se lida com tais problemas? Quais são as técnicas adequadas para resolvê-las? A Otimização de Trajetórias é a disciplina que lida com estes tipos de problemas, e também o objeto de estudo desta dissertação. Proponho o uso da Parametrização B-spline para a Otimização de Trajetórias de Voo. Foi desenvolvido um código em *Python* que implementa o método proposto, tendo sido obtido resultados promissores para os três exemplos estudados, provando assim a robustez e a versatilidade do método proposto. Trabalhos futuros devem explorar diferentes tipos de missões, incluindo trajetórias espaciais, assim como diversas combinações de *waypoints*. O estudo e a implementação dos sinais de controlo que façam com que o veículo desejado siga a trajetória ótima determinada deverá constituir também um exercício interessante.

Palavras-chave

Parametrização B-spline, Python, Controlo Ótimo, Otimização de Trajetórias de voo.

Resumo alargado

Introdução

Os pioneiros da aviação estavam principalmente interessados em atingir o impensável: fazer com que um dispositivo mais pesado do que o ar voasse. O primeiro voo de um dispositivo mais pesado do que o ar foi conseguido no século XX, constituindo um grande marco para a humanidade no que diz respeito à engenharia aeronáutica.

Apesar do grande entusiasmo causado por estes primeiros passos, a verdade é que os primeiros voos foram muito curtos e revelaram pouco controlo sobre o dispositivo, exigindo grandes progressos nos princípios básicos de voo por parte de cientistas, e também diversos experimentos usando planadores, para que se atingisse o voo prático.

Tendo sido atingido esse tal voo prático, novos desafios se revelaram, nomeadamente executar um voo de um determinado ponto A até um ponto B de uma determinada forma, seja minimizando o tempo decorrido, o comprimento da trajetória ou até o combustível necessário. Este é exatamente o objeto de estudo da Otimização de Trajetórias, e também o objeto de estudo do presente trabalho, estudar as técnicas de otimização de trajétorias, focando mais tarde na Parametrização usando B-splines.

Diversos fatores têm vindo a dar nova vida à investigação acerca da Otimização de Trajetórias (OT), sendo que na verdade foi sempre um tema de grande relevo. A crescente preocupação com o impacto da emissão de poluentes atmosféricos, assim como da poluição sonora nas proximidades dos aeroportos, traz ímpeto reforçado à OT, pois esta pode ser uma chave importante. O novo paradigma do controlo de tráfego aéreo, tendendo para rotas mais flexíveis e mais otimizadas, e o crescente desenvolvimento de veículos aéreos não tripulados, completam o leque de motivos que levaram à escolha da OT como tema para o presente trabalho, que se proprõe três objetivos fundamentais;

- Prover uma breve revisão acerca dos métodos de otimização;
- Implementar um código em *Python* que resolva os casos de estudos apresentados. Pretende-se que o programa calcule a trajetória ótima (posição: latitude, longitude e altitude) em função do tempo para um conjunto de *waypoints*.
- Contribuição do autor: um *framework* universal para otimização preliminar de trajetórias 4D usando parametrização B-spline.

A dissertação está estruturada em cinco capítulos:

- **Capítulo 1:** Introdução.
- **Capítulo 2:** Teoria do Controlo Ótimo.
- **Capítulo 3:** Metodologia e Ferramentas.
- **Capítulo 4:** Resultados da simulação.
- **Capítulo 5:** Conclusão.

Teoria do Controlo Ótimo

O problema de otimização de trajetória é formulado em termos de problema de controlo ótimo neste trabalho, pelo que se didicou um capítulo à sua teoria. A Teoria do Controlo Ótimo é resultado do Cálculo de variações (ou cálculo variacional), os seus primeiros passos remontam ao século XVII. No entanto, conheceu grande êxito durante a década de 1960 com as suas aplicações no sector aeroespacial.

O objetivo de um Problema de Controlo Ótimo (PCO) diz respeito à *determinação dos sinais de controlo que fazem com que um processo satisfaça as restrições impostas e ao mesmo tempo minimize (ou maximize) uma determinada medida de performance.* A formulação de um PCO requer:

1. uma descrição matemática (um modelo) do processo a ser controlado. O objetivo é a obtenção da descrição matemática mais simples possível, que ajude no cálculo preciso da resposta física do sistema a todos os sinais de entrada esperadas. De forma geral o modelo é construído com recurso a equações diferenciais ordinárias;
2. a identifição das restrições físicas. Estas podem dizer respeito ao estado ou ao controlo. Dependem efetivamente do sistema em estudo. As restrições do controlo representam a capacidade do sistema em alterar o seu estado;
3. a especificação de alguma medida de performance. A medida de performance refere-se a um determinado parâmetro ou combinação de parâmetros que traduza a qualidade desejada em quantidade numérica.

Tendo sido formulado o problema, é necessário identificar um método que seja adequado à sua resolução. Os métodos capazes de resolver PCO são variados, sendo que diferentes

autores procedem à sua cetegorização de diferentes formas. Uma das possíveis categorizações divide as técnicas de otimização para resolução de PCO em três categorias:

- *Métodos de Programação Dinâmica:* usam os critérios de Hamilton-Jacobi-Bellman para determinar o controlo ótimo.
- *Métodos Indiretos:* usam o cálculo de variações e o Princípio de Mínimo de Pontryagin para derivar as condições necessárias de otimalidade.
- *Métodos Diretos:* consistem na discretização do PCO contínuo através da construção de uma sequência de pontos. Um conjunto finito de variáveis é obtido, em que podem ser determinadas usando ferramentas de otimização adequadas.

A *programação dinâmica* é relativamente fácil de se entender e implementar. No entanto não é adequada a problemas de grandes dimensões (i. e. com grande número de variáveis). Os *métodos indiretos* são capazes de produzir resultados altamente precisos, no entando a derivação das condições de otimalidade pode ser muito complexa e suscetível a erros. Por fim, os *métodos diretos* apresentam-se como fortes candidatos à resolução de problemas multidimensionais e de grandes dimensões, embora a implementação possa ser desafiante. Tendo em conta os aspectos apresentados, foi decidido que um método direto seria adequado ao tipo de problema em estudo.

Metodologia e Ferramentas

B-splines são o conceito central do método proposto neste trabalho, por este motivo é importante apresentar os aspectos fundamentais dessa função matemática tão versátil. Comecemos por considerar um determinado vetor $T = (T_0, T_1, ..., T_{n-1}, T_n, T_{n+1}, ..., T_{n+k}), k \geq 1$ e $n \geq 0$. As *funções B-spline normalizadas* correspondentes N_{ik} de ordem k (grau $k-1$) são definidas por:

$$N_{i1}(t) = \begin{cases} 1, & T_i \leq t \leq T_{i+1} \\ 0, & \text{caso contrário.} \end{cases}$$

para $k = 1$, e

$$N_{ik}(t) = \frac{(t - T_i)}{(T_{i+k-1} - T_i)} N_{i,k-1}(t) + \frac{(T_{i+k} - t)}{(T_{i+k} - T_i + 1)} N_{i+1,k-1}(t)$$

para $k > 1$, e $i = 0, 1, ..., n$. Usando tais funções, é possível contruir uma *curva B-spline*. Considerando uma lista ordenada de pontos de controlo $d_i \in \mathbb{R}^m$, $m \geq 1$, uma *curva*

B-spline é definida por

$$X(t) = \sum_{i=0}^{n} N_{ik}(t) d_i, \text{para } T_0 \leq t \leq t_{n+k}$$

As curvas B-spline podem ser bastante úteis tanto para interpolação quanto para parametrização.

O problema específico que se pretende resolver neste trabalho é a determinação da *trajetória ótima* para um veículo aéreo, minimizando uma medida de performance que será apresentada de seguida. Consideremos as equações de navegação em coordenadas cartesianas, apresentadas sob forma matricial.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$$

Em que x, y e x correspondem às coordenadas cartesianas, enquanto que v_x, v_y e v_z correspondem à velocidade em cada eixo e a_x, a_y e a_z às acelerações. Empregando uma linguagem dos PCO, o vetor de estado será

$$X = \begin{bmatrix} x & y & z & v_x & v_y & v_z \end{bmatrix}^T$$

e o vetor de controlo será

$$u = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T$$

Assim, pretende-se determinar a trajetória ótima X^* correspondente ao controlo ótimo u^* que minimize a medida de performance

$$MinJ(x, u) = \int_{t_0}^{t_f} (\dot{x}^T Q \dot{x} + u^T R u) dt$$

sujeita às equações diferenciais de navegação já apresentadas, assim como outras limitações em termos de velocidades e acelerações máximas. Pretende-se também que a solução passe por um conjunto de *waypoints* $W_j = (\phi_j, \lambda_j, h_j, t_j), j = 1, 2, ..., p, p \geq 2$.

O método proposto consiste na parametrização tanto do controlo quanto do estado, re-

presentando ambos em termos de curvas B-spline, sendo obtido assim:

$$\begin{cases} x(t) = \sum_{i=1}^{n} N_{ik}(t) d_i^{(x)} \\ y(t) = \sum_{i=1}^{n} N_{ik}(t) d_i^{(y)} \\ z(t) = \sum_{i=1}^{n} N_{ik}(t) d_i^{(z)} \end{cases}, \begin{cases} v_x(t) = \sum_{i=1}^{n} N_{ik}(t)' d_i^{(x)} \\ v_y(t) = \sum_{i=1}^{n} N_{ik}(t)' d_i^{(y)} \\ v_z(t) = \sum_{i=1}^{n} N_{ik}(t)' d_i^{(z)} \end{cases}, \begin{cases} a_x(t) = \sum_{i=1}^{n} N_{ik}(t)'' d_i^{(x)} \\ a_y(t) = \sum_{i=1}^{n} N_{ik}(t)'' d_i^{(y)} \\ a_z(t) = \sum_{i=1}^{n} N_{ik}(t)'' d_i^{(z)} \end{cases}$$

O problema é assim convertido num problema de programação não linear e pode ser resolvido usando ferramentas adequadas. O integral da medida de performance é calculado recorrendo à regra de Simpson composta.

É importante referir-se que foi utilizado o sistema de coordenadas geodético para questões de visualização dos resultados assim como a definição dos waypoints. No entanto, os cálculos internos do programa utilizam um referencial cartesiano designado por *Earth Fixed Earth Centered*, por possibilitar o uso de equações de navegação bastante simples. Assim sendo, naturalmente foram utilizadas equações de conversão entre um referencial e o outro.

Conclusão

Por forma a verificar a robustez do método proposto, foi implementado um código em *Python*, tendo sido estudado três exemplos distintos de missões. Embora os resultados sejam distintos em termos de qualidade e suavidade das trajetórias ótimas calculadas, os resultados para os três exemplos estudados são considerados satisfatórios, comprovando assim a robustez e a versatilidade do método proposto.

Foram identificados como possíveis trabalhos futuros a exploração de diferentes tipos de missões (incluindo trajetórias espaciais) e combinações de conjuntos de *waypoints*, assim como o estudo e a implementação dos sinais de controlo que façam com que o veículo desejado siga a trajetória ótima determinada.

Abstract

Achieving the first heavier-than-air flight during the 20^{th} century was certainly a great landmark in human history. However, flight itself generally is not the final objective. To perform the desired mission effectively, it is necessary to determine the path to follow according to the objective, for instance minimizing the elapsed time, the path length, etc. How do we deal with such problems? Which are the techniques to solve them? Trajectory Optimization is the subject that deals with such problems, and the object of study of the present dissertation. I propose to study the use of B-spline Parameterization for Flight Trajectory Optimization. A Python code was developed to implement the proposed method, revealing promising results for all three presented examples, proving the robustness and versatility of the proposed method. Future studies should explore different types of missions, including spatial trajectories, and various sets of waypoints. The study and implementation of the control to stimulate the desired flying device to follow the determined optimal trajectory would also constitute an interesting exercise.

Keywords

B-spline Parameterization, Flight Trajectory Optimization, Python, Optimal Control Problem.

Contents

List of Figures

List of Tables

List of Acronyms

ETA	Estimated Time of Arrival
OFP	Operational Flight Planning
ICAO	International Civil Aviation Organization
ATC	Air Traffic Control
EU	European Union
SES	Single European Sky
ATM	Air Traffic Management
FRA	Free Route Airspace
CCO	Continuous Climb Operations
CDO	Continuous Descent Operations
EASA	European Union Aviation Safety Agency
OCT	Optimal Control Theory
OCP	Optimal Control Problem
ODE	Ordinary Differential Equations
NLP	Nonlinear Programming
BVP	Boundary Value Problem
ECEF	Earth Centered Earth Fixed

Chapter 1

Introduction

The development of consistent research work requires deep knowledge of the studied field, together with good mastery of the technical skills required for the practical applications. However, it is also of great importance to present the main interest of the object of study, which will be exactly the objective of the first section, to present the framework where the present work fits, justifying its great significance today.

1.1 Motivation

The aviation pioneers were mainly interested in achieving the unthinkable, making a heavier-than-air device fly. The first heavier-than-air powered flights were achieved in the course of the 20^{th} century [1], constituting a great achievement for mankind in aeronautical engineering. Despite the excitement caused by those first steps of the aviation history, the truth is that the first flights were not practical at all, there was very little control over the flying devices. Furthermore, the available power plants were not robust or reliable, making the task of achieving practical and controlled flight very difficult at the time.

Great progress on the basic principles of flight by scientists and inventors, together with experimentation using gliders and steam-powered flying machines led aviation to a period of extremely fast development. Whereas in the early years of the 20^{th} century pioneers like the Wright brothers worked hard to perform some short and poorly controlled flights [2], a couple of decades later the first commercial flights were taking-off, opening a new era of fast development worldwide.

Achieving controlled powered flight led scientists and engineers to new challenges, namely performing a flight from point A to point B in a certain way, which could be minimizing the elapsed time, the trajectory length, or even fuel consumption. That is exactly the object of study of the present work, to study the trajectory optimization techniques, and later focusing on B-spline parameterization.

1.1.1 Current Commercial Flights Planning

Commercial air transport operation is a term commonly used to refer to an aircraft operation involving the transport of passengers, cargo, or mail for remuneration or hire [3]. In this context, to ensure financial viability in addition to environmental concerns, one needs to possess suitable tools in order to find an optimal trajectory for the aircraft. In formal terms, this necessity is filled through the elaboration and submission of an *operational flight plan*, which consists of the operator's plan for the safe conduct of the flight based on considerations of the aircraft performance, other operating limitations, and relevant conditions on the route to be followed and at the aerodromes concerned.

For engineering purposes, one must translate airlines' interest parameters such as cost and pollutants emissions into more engineering-related parameters like fuel consumption, noise levels, etc. In the present work, we will not be (directly) concerned about the financial aspects of flight planning, our goal is to study mathematical trajectory optimization methods.

1.1.2 Flight Phases

An aircraft flight may be divided into several phases [4] according to the required detail. In a simple way, one can describe the different phases of a commercial flight as follows.

1. Pre-departure: preparation for flight. Includes all required services (fueling, cleaning, catering, ramp inspections, etc.), passenger boarding, cargo unloading and/or loading, and other aspects depending on the purpose of each flight.

2. Taxi and Take-off: once all the passengers are on board, all cargo is loaded and all doors are securely closed, the pilots will ask and eventually obtain clearance from the airport control tower to taxi. Arriving at the runway, it is time to take off.

3. Climb: this phase includes the initial climb and the climb to cruise altitude. The take-off requires a lot of power from the engines, therefore consuming a good amount of fuel. The climb also requires great power, but in the climb phase, the power may be reduced to slightly smaller power levels than in the take-off phase providing fuel saving and noise reduction.

4. Cruise altitude: defined as the period following the initial climb during which the aircraft is in level flight. In general terms, the cruise is the longest phase of flight, depending on the traffic congestion it may include changing flight level to lower

altitudes, but generally the aircraft climbs as it gets lighter due to fuel consumption.

5. Descent: the descent consists of the decrease in altitude, from cruise altitude to the initial approach altitude. In a more theoretical way, it is defined as any time before approach during which the aircraft has a negative rate of climb for an extended period. It starts around 20 minutes before the estimated time of arrival (ETA).
6. Approach: goes from around 1000 feet above the runway elevation to the beginning of the landing flare.
7. Landing: goes from the precedent phase until the aircraft touches down and exits the runway. It is also frequently referred to as Touchdown.
8. Taxi and post-flight: after landing, the aircraft rolls on the ground to get to a stand. After this stage, the aircraft is prepared for a new flight (restarting the cycle) or is stored at an appropriate place.

In addition to the described phases, in some cases, landing is abandoned due to some impossibility. Then the Go-around phase takes place, naturally, it is an undesirable event. The go-around is a situation where the pilot is about to touchdown but decides to apply full power before the landing gear touches the ground.

1.1.3 Operational Flight Planning

Previously, a more theoretical definition of Operational Flight Planning (OFP) was given; however, in practical terms, and for engineering purposes, the minimization of flight cost must be seen as the determination of an appropriate route while carrying the minimum legal and necessary fuel.

In fact, an OFP is an essential piece of document to make possible the realization of commercial flights. First, is necessary the vehicle (an aircraft), second, the demand for flight (passengers and/or cargo), and then a flight plan, which is filled by an airline in a standard format, called ICAO format [5]. The flight plan is sent to the Air Traffic Control Authority.

A flight plan is an important tool for Air Traffic Control since it is used to assure that all flights are assigned to a certain time slot to prevent mid-flight collision. If the time slot required by the airline is already allocated to another flight, ATC proposes another time slot.

The augmentation of air traffic flow during the past decade, as shown in Figure 1.1 [6], is pushing ATC toward a more cost-effective and flexible model, where more optimized and

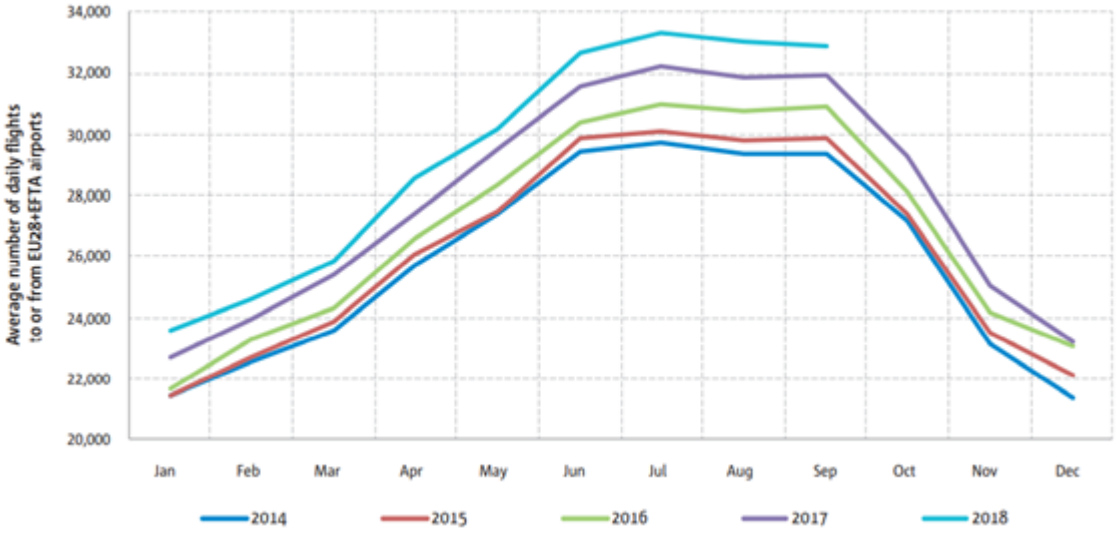


Figure 1.1: Number of daily flights increases every year between 2014 and 2019 in EU (source: EASA).

less rigid routes can be used to increase airspace capacity [7]. Despite the present decrease in the number of flights worldwide due to the Covid-19 pandemic, it is predictable that the air traffic flow is going to continue to grow after this difficult period.

As a response to the aspect mentioned previously, the European Union (EU), through the European Commission, launched an ambitious initiative in 2004 titled the Single European Sky (SES) to reform the architecture of European Air Traffic Management (ATM). The key objectives of this European initiative are [8]:

- To restructure European airspace.
- To create additional capacity.
- To increase the overall efficiency of the ATM system.

The SES is a very ambitious project which is currently in the deployment phase. Since our goal is not to investigate ATM aspects, and also a complete analysis of the SES project would make content for an entire dissertation, it was decided to mention it considering that the new paradigm introduced by SES supports the choice of *Trajectory Optimization* as the subject of study. In fact, flexible and optimized trajectories seem to be one of the keys to the future of ATM.

1.1.4 Environmental and Noise Concerns

In the beginning, the aviation sector was a symbol of power and wealth, only rich people used to fly, for pleasure or business. As time passed the paradigm changed, the number

of passengers, companies and operators multiplied throughout the years, and commercial flights assumed a major role in global mobility. Today, air travel is not an extravagance, it is part of our lives as any other type of transportation.

As the number of passengers and flights increased, environmental concerns became more and more important since the great majority of commercial aircraft run on fossil fuel, consequently discharging pollutants into the atmosphere. The augmentation of the number of flights has clearly impacted the total emission.

Figure 1.2 [6] shows the emissions from a typical two-engine jet aircraft during 1-hour flight carrying 150 passengers, according to the European Environmental Report (2019). It is possible to notice that the greatest part of the emissions is associated with carbon dioxide, which is a well-known gas due to its greenhouse effect.

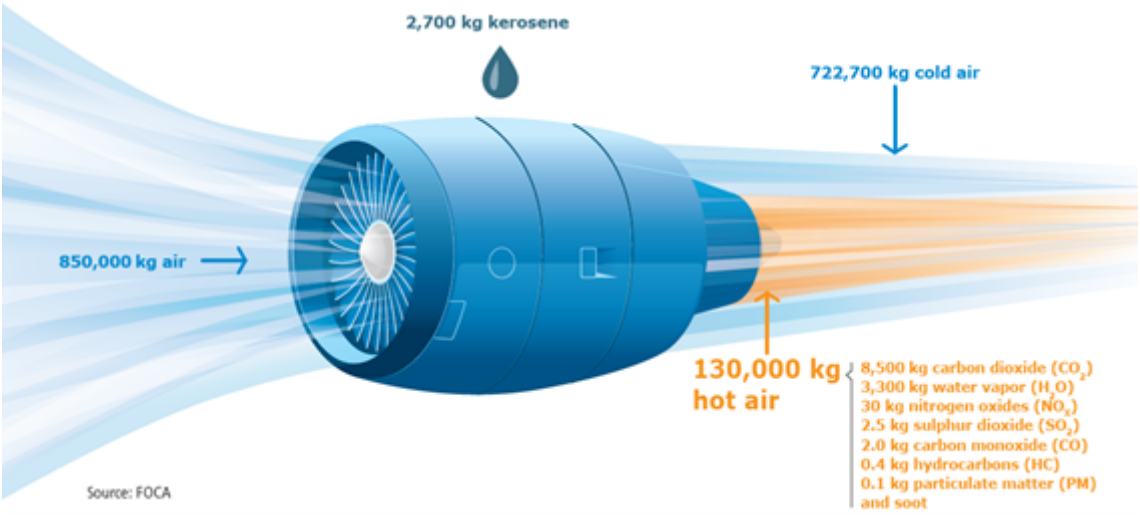


Figure 1.2: Emissions from a typical two-engine jet aircraft during 1-hour flight carrying 150 passengers (source: EASA).

Pollutants emissions reduction can be achieved either by performing improvements on the aircraft aerodynamics (thus reducing the drag), improvements on the engines, or through operational procedures improvements. A good example of operational improvement is the case of Free Route Airspace (FRA), which consists of specified airspace within which users may freely plan a route between a defined entry point and a defined exit point. The estimated saved emissions between 2014 and 2019 due to the implementation of this concept are more than 2.6 million tonnes of CO_2 [6]. This fact shows that operational procedure improvements still have a major role to play in terms of pollutants emission reduction.

Now that the environmental concerns are referred to, let us superficially look at noise as-

pects. Certain noise levels have a serious impact on both human and wildlife well-being and health, as is acknowledged in several studies on the subject [9,10]. Since aviation operations are generally quite noisy, it is necessary to observe closely the noise levels around airports and aerodromes. For this purpose, on one hand, there are regulations by the responsible authorities such as EASA. On the other hand, there are the noise contours that can be used in order to determine if the legal noise levels are not exceeded.

Figure 1.3 shows an example of an airport noise contour, an important tool. The main axis of the noise contour is aligned with the direction of the head of the runway as is expected since it corresponds to the major direction of flight, along which the exhaust gases are expelled. The engines are the main source of the noise.

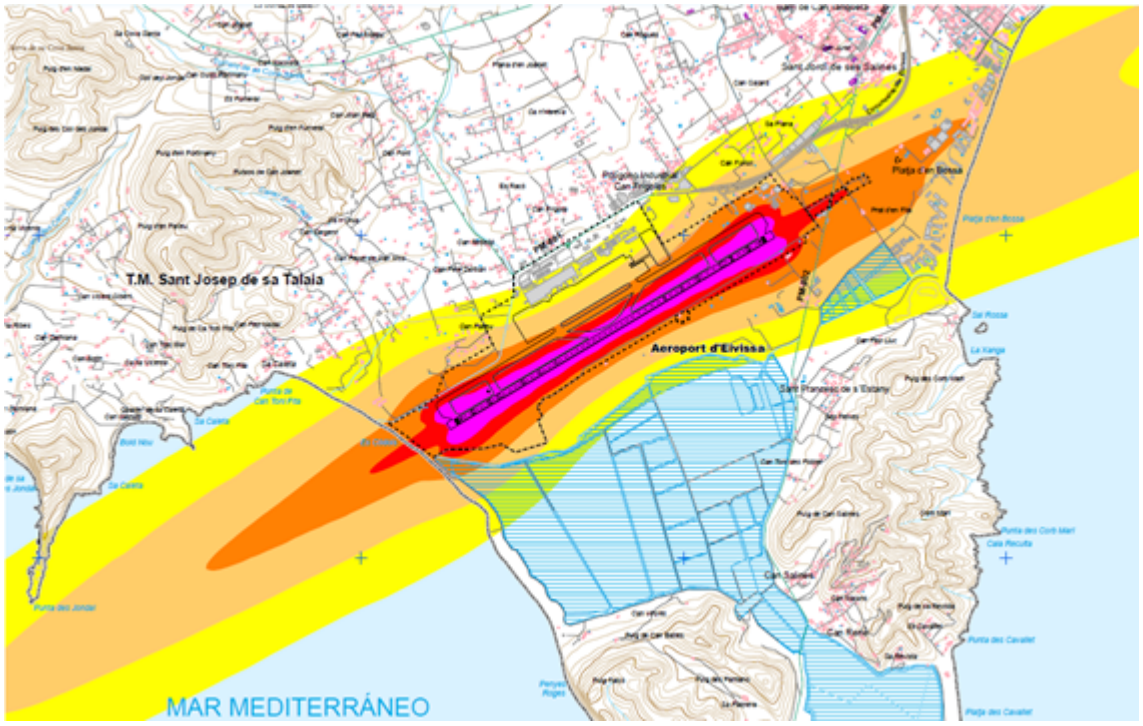


Figure 1.3: Example of an airport noise contour (source: EASA).

Even though aircraft became less noisy throughout the years due to technological improvements mainly in the engines, the growing amount of air traffic in Europe suggests that an important part of the population is exposed to problematic noise levels. Aircraft noise is the third biggest source of noise exposure after road and rail traffic.

Fuel and noise concerns can be addressed in two ways:

- a) **Technological improvements on aircraft:** this solution consists of the development of new and quieter aircraft. This can be achieved through the development of new engines and/or improvements to the aerodynamics of the aircraft.

- b) **Operational improvements:** several operational improvements can be implemented, one of them is the Continuous Climb Operations (CCO) / Continuous Descent Operations (CDO).

We will clearly be focused on operational improvements, through trajectory optimization. CCO and CDO are important concepts since they make possible valuable fuel savings and noise reduction during the approach compared to the traditional non-precision approach.

Figure 1.4 shows a *Balanced Approach* to airport noise management suggested by EASA. Numbers 1 and 2 are mainly related to technological improvements while 3 and 4 are related to operations.

Figure 1.4: Balanced Approach for airport noise management by EASA.

To conclude, the choice of trajectory optimization as the subject of study for the present work is justified by its increasingly high importance namely referred on the SESAR project, as well as its importance in terms of noise management included in numbers 3 and 4 of Balanced Approach for airport noise management presented by EASA.

1.2 Objectives

The main objectives of this dissertation are pointed out below.

1. **Optimization techniques**: to provide an overview of the main optimization techniques for solving Optimal Control Problems.

2. **Practical implementation**: to implement a *Python* code to solve the cases of study that are going to be presented. The code is intended to provide the optimal path (position: latitude, longitude, and altitude) as a function of time for a given set of waypoints.
3. **Author's contribution**: a universal framework for preliminary 4D trajectory optimization using B-spline parameterization.

1.3 Thesis outline

The present work is structured as described below, comprising five chapters.

- **Chapter 1:** Introduction. Presents the motivation, as well as the context and importance of flight trajectory optimization today.
- **Chapter 2:** Optimal control theory. Provides an overview of OCT, including the theoretical aspects, an illustrative example, and also a brief review of the optimization methods for solving OCP.
- **Chapter 3:** Methodology and tools. This chapter intends to present all methods and tools used to reach the desired results. Includes a description of B-splines, the specific problem statement, the proposed method, the coordinate systems used and their conversion equations, and a brief description of the implemented code.
- **Chapter 4:** Simulation Results. Presents the graphical results obtained for the three presented examples, using the proposed method and the implemented code.
- **Chapter 5:** Conclusion. Synthesizes the initial objectives and summarizes the obtained results.

Chapter 2

Optimal Control Theory

Since the trajectory optimization problem is going to be formulated in terms of Optimal Control Problem (OCP) in this work, it is pertinent to present the theoretical fundamentals behind it.

Optimal Control Theory represents the outcome of calculus of variations, and its first steps go back to the 17^{th} century; however, OCT knew its greatest success during the 1960s through its aerospace applications [11].

The objective of an optimal control problem is *to determine the control signals that will cause a process to satisfy constraints and at the same time minimize (or maximize) some performance measure* [12]. The formulation of an OCP requires:

1. A mathematical description (or model) of the process to be controlled.
2. A Statement of the physical constraints.
3. Specification of some performance measure.

2.1 The Mathematical Model

Any control/trajectory optimization problem includes a modeling phase of the process. The main objective is to obtain the simplest mathematical description that helps to accurately compute the response of the physical system to all expected inputs. The mathematical models are generally described by ordinary differential equations (ODE) in the state variable form.

$$\dot{x}(t) = f(x(t), u(t), t) \quad (2.1)$$

Working with the mathematical model in the state variable form is convenient because:

1. The differential equations are ideally suited for digital or analog solutions.
2. The state form provides a unified framework for the study of nonlinear and linear systems.

3. The state variable form is invaluable in theoretical investigations.

A history of control input values during the interval $[t_0, t_f]$ is called a *control history* or simply *control*, and a history of state values in the interval $[t_0, t_f]$ is called a *state trajectory* and is denoted by $\mathbf{x}$.

2.2 Physical Constraints

After defining the mathematical model, it is necessary to identify the physical constraints on the state and control values. These constraints depend on the problem, if the system under study is for instance an aircraft, the state constraints could be related to the position (obstacles), velocity (due to transonic condition), or acceleration (structural limitations).

The control limitations represent the capacity of the system to change its state. Those constraints are often related to the capability of the power plant (engines), the maximum acceleration, etc.

A control history which satisfies the control constraints during the entire time interval $[t_0, t_f]$ is called an *admissible control*; a state trajectory which satisfies the state variable constraints during the entire time interval $[t_0, t_f]$ is called an *admissible trajectory*.

2.3 The Performance Measure

In order to evaluate the performance of a system quantitatively, it is necessary to select a performance measure. The performance measure refers to a given parameter or combination of parameters that expresses the desired quality into numerical quantity. In some cases, the problem statement may suggest the performance measure to be selected; however, in other cases, the selection may be difficult and a more subjective matter may be required.

To better visualize the concept, let us consider a simple example: let us assume that an aircraft is required to move from one point to another as quickly as possible. In this case, the statement suggests that the elapsed time is the ideal performance measure to be minimized:

$$J = t_f - t_i \quad (2.2)$$

2.3.1 Minimum-Time

Problem: to transfer a system from an arbitrary initial state $x(t_0) = x_0$ to a specified target set *S* in minimum time. The performance measure to be minimized is:

$$J = t_f - t_0 = \int_{t_0}^{t_f} dt \quad (2.3)$$

t_f being the first instant of time when $x(t)$ and *S* intersect. A good example of this type of problem is the interception of an aircraft or missile, generally, in combat scenes, one is required to take an attacking aircraft down as fast as possible, from the decision moment.

2.3.2 Terminal Control

Problem: to minimize the deviation of the final state of a system x_f from its target value **r**(t_f). A possible performance measure is:

$$J = [x(t_f) - r(t_f)]^T [x(t_f) - r(t_f)] \quad (2.4)$$

Or

$$J = [x(t_f) - r(t_f)]^T H [x(t_f) - r(t_f)] \quad (2.5)$$

Where H is a real symmetric positive semi-definite $n \times n$ weighting matrix.

2.3.3 Minimum-Control-Effort

Problem: To transfer a system from an arbitrary initial state x_0 to a given target set S, with minimum spending of control effort. The sense of the term ”minimum control effort” depends upon the particular physical application, consequently, the performance measure may take several forms. A possible example may be the total energy:

$$J = \int_{t_0}^{t_f} u^2(t)dt \quad (2.6)$$

2.3.4 Tracking

Problem: to maintain the system state $x(t)$ as close as possible to the target state $r(T)$ in the interval $[t_0, tf_f]$.

Many other performance measures may be selected, based on the specific studied problem.

2.4 Systems Classification

In general terms, systems are described by the terms *linear, nonlinear, time-invariant* and *time-varying*. Notations for each type of system are as follows.

- Nonlinear and time-varying

$$\dot{x}(t) = a(x(t), u(t), t)$$

- Nonlinear and time-invariant

$$\dot{x}(t) = a(x(t), u(t))$$

- Linear and time-varying

$$\dot{x}(t) = A(t)x(t) + B(t)u(t)$$

- Linear and time-invariant

$$\dot{x}(t) = Ax(t) + Bu(t)$$

2.5 General Optimal Control Problem

The OCP aims to find an admissible control u^* which causes the system

$$\dot{x}(t) = a(x(t), u(t), t) \quad (2.7)$$

to follow an *admissible trajectory* x^* that minimizes the performance measure

$$J = h(x(t_f), t_f) + \int_{t_0}^{t_f} g(x(t), u(t), t)dt \quad (2.8)$$

u^* is called an *optimal control* and x^* an *optimal trajectory*.

2.6 An illustrative example

To summarise all the presented concepts, let us consider a simple example described in Ref. [13]. As illustrated in Figure 2.1, the objective is to move the vehicle from one position and velocity to another position and velocity.

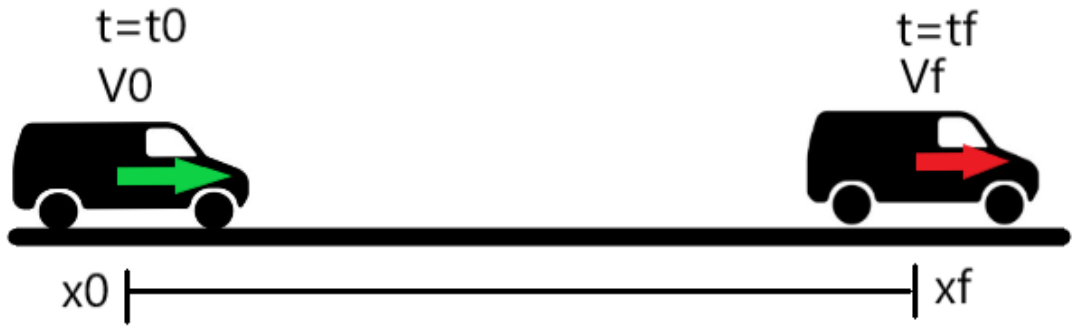


Figure 2.1: Illustration for the example of vehicle motion.

x denotes the position, thus the motion of the vehicle is governed by the differential equation $\ddot{x} = a$, where a is the controlled acceleration. It is usual to write the system of equations governing the motion in the first order, which allows a uniform framework for all types of problems.

Considering a fixed final time problem, the objective could be to make the transfer in a given time while minimizing the control energy. The formal statement of the problem is: find the acceleration history $a(t)$ that minimizes the performance measure

$$J = \frac{1}{2}\int_{t_0}^{t_f} a^2 dt \quad (2.9)$$

subject to the vehicle equations of motion

$$\begin{cases} \dot{x} = v \\ \dot{v} = a \end{cases} \quad (2.10)$$

the prescribed initial conditions

$$t(0) = t_0, x(t_0) = x_0, v(t_0) = v_0 \quad (2.11)$$

and the prescribed final conditions

$$t(f) = t_f, x(t_f) = x_f, v(t_f) = v_f \quad (2.12)$$

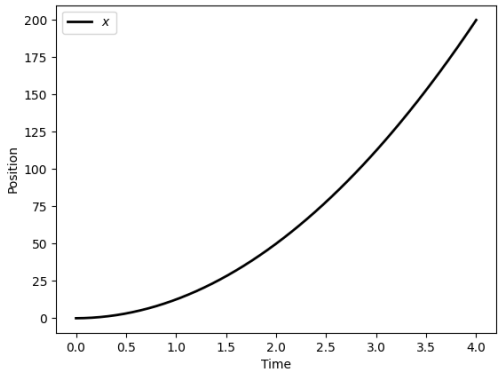
To better visualize the example and compute the results, let us consider the values presented in Table 2.1 for the prescribed initial and final conditions:

Parameter	Initial	Final
t [s]	0	4
x [m]	0	200
v [m/s]	0	100

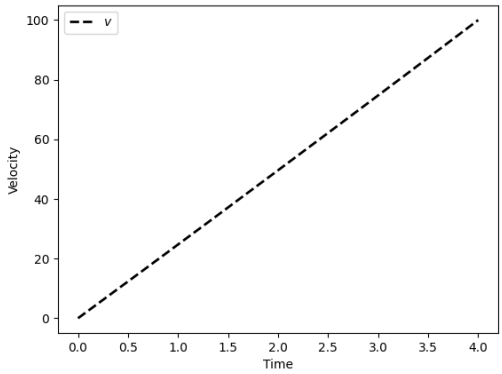
Table 2.1: Prescribed initial and final conditions for the vehicle's motion example.

Figure 2.2 presents the results obtained for the example of the motion of a vehicle, assuming that the acceleration is controlled, using the initial and final conditions presented in Table 2.1.

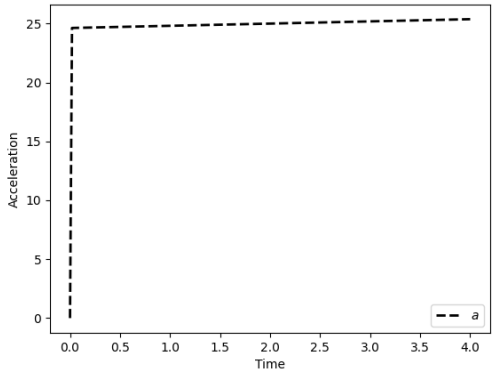
Even though simple, this example is illustrative of what an optimal control problem consists of. One is referred to [12–15] for further reading on OCT.



(a) Position vs time.



(b) Velocity vs time.



(c) Acceleration vs time.

Figure 2.2: Graphical results obtained for the vehicle motion example.

2.7 Optimization methods review

The main goal of this study is to perform optimization of the flight trajectory of commercial flights, general aviation flights, and UAVs flights. The most common way of formulating trajectory optimization problems is using OCT, which ultimate goal is to find the control $u(t)$ that minimizes (or maximizes) the performance measure along the optimal trajectory. Once the optimal control law is found, the optimal trajectory can be easily obtained. The following section provides a brief overview of the main categories for solving trajectory optimization problems, for further reading one is referred to Ref. [16].

In general, the hardest part of dealing with trajectory optimization problems is solving the OCP, which formulation itself has its level of difficulty but finding the right (numerical) method to solve OCP, and then implementing it may be challenging. Having the mentioned in mind, let us identify the methods that are suitable for solving OCP, which can be categorized in several ways, one of them is presented below, dividing the techniques into 3 categories:

- **Dynamic Programming methods:** uses Hamilton-Jacobi-Bellman optimality criteria.
- **Indirect methods:** use the calculus of variations and Pontryagin's Minimum Principle to derive necessary conditions of optimality.
- **Direct methods:** constituted by shooting methods and collocation methods. It consists in discretizing the continuous optimal control problem and constructing a sequence of points. A finite set of variables is obtained that can be solved using adequate optimization tools. One possible way is to convert the problem into a nonlinear programming problem (NLP) which can be solved using suitable optimization tools.

2.7.1 Dynamic Programming

In the dynamic programming method, the principle of optimality is used to find an optimal control law. The problem is divided into sub-problems that are linked by a recurrence relation. For the sake of a better understanding, let us consider the multistage decision process of Figure 2.3.

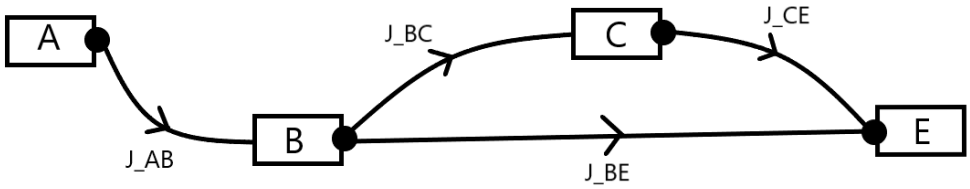


Figure 2.3: Multistage decision process illustration.

Considering the example, there are two ways to go from A to E: one A-B-E directly and the other passing through C. Being J_{AB} the cost for the section A-B, we have:

$$J^*_{AE} = J_{AB} + J_{BE} \quad (2.13)$$

If A-B-E (directly) is the optimal path from A to E, then B-E (directly) is also the optimal path from B to E. This way we are guided into Bellman's Principle of Optimality [17]:

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Dynamic programming is a computational technique that extends the multi decision-making using the principle of optimality to sequences of decisions, which combined make an optimal policy and trajectory. This technique is simple and very effective for small problems. However, when the problem to be solved involves large numbers of optimization parameters as is the case of flight trajectory problems, it becomes computationally ineffective, for this reason, we will not be particularly interested in this technique.

2.7.2 Indirect Methods

The core philosophy behind the indirect methods is finding an optimal solution by satisfying optimality conditions instead of directly minimizing a cost criterion as it is done in the case of direct methods, referred forward. Depending on the OCP, the optimality conditions may lead to a two-point or multi-point boundary value problem (BVP). In a few very particular cases, such as purely continuous, linear systems and quadratic costs, the analytic solution of the BVP can be found. However, in most cases, it is necessary to iteratively approach a solution using numerical methods.

As it is not reasonable to approach indirect methods without referring to Pontryagin's maximum (minimum) principle, let us briefly present its basic aspects. It was formulated in 1956 by the mathematician Lev Pontryagin, being initially used for the maximization of the terminal speed of a rocket. The formulation of this principle uses notions from the classical calculus of variations, also used by Isaac Newton.

To illustrate Pontryagin's Minimum Principle let us consider the problem of determining the control function $u(t)$ that minimizes the performance index given by

$$J = \Phi(x(t_f)) + \int_{t_0}^{t_f} F(x(t), u(t))dt \quad (2.14)$$

Subject to

$$\dot{x} = f(x, u), \quad x(t_0) = x_0, \quad x(t_f) = x_f \quad (2.15)$$

The Hamiltonian function $H(x, u, p)$ is defined as

$$H(x,u,p) = F(x,u) + p^T f(x,u) \quad (2.16)$$

where p is the co-state vector. Even though the entire derivation of the principle will not be presented, it is worth mentioning that p is selected to correspond to the solution of the differential equation

$$\frac{\partial H}{\partial x} + \dot{p}^T = 0^T \quad (2.17)$$

Then, the necessary conditions for $u \in U$ to minimize (2.14) subject to (2.15) are

$$\dot{p} = -\left(\frac{\partial H}{\partial x}\right)^T \quad (2.18)$$

$$H(x^*,u^*,p^*) = \min_{u \in U} H(x,u,p) \quad (2.19)$$

If the final state x_{t_f} is free, then in addition to (2.18) and (2.19) conditions, it is required that (2.20) end-point condition is satisfied.

$$p(t_f) = \nabla_x \Phi|_{t=t_f} \quad (2.20)$$

The equation (2.18) is called the *adjoint* or *costate* equation.

2.7.3 Direct Methods

Direct methods consist of the transcription of the infinite-dimensional problem into a finite-dimensional Nonlinear Programming (NLP) problem. This transcription can be achieved by implementing a control parameterization constructed using arbitrarily chosen analytical function, as in *shooting* methods, or by implementing a piecewise approx-

imation of both control and state variables based on a polynomial sequence of arbitrary degree, as in *collocation* methods.

In both cases, the transcribed dynamical system is integrated along the time interval $[t_0, t_f]$. The search and determination of the optimal set of discretization parameters are formulated as an NLP problem, which is then solved computationally using efficient numerical optimization tools. Figure 2.4 illustrates the described categorization for direct methods.

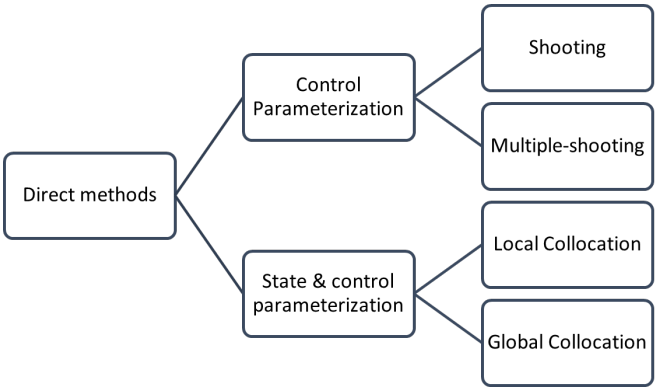


Figure 2.4: Direct methods categorization.

2.7.3.1 Direct shooting

In the direct shooting (and multiple direct shooting), the parameterization is performed on the controls $u(t)$ only, the dynamic constraints are integrated making use of conventional numerical methods for instance Runge-Kutta family, and the Lagrange term in the cost function is approximated by a quadrature. The parameterization of the control variables is expressed as

$$u(t) = \sum_{k=1}^{N} c_k q_k(t) \quad \textbf{(2.21)}$$

where $q_k(t)$ are known functions and c_k are the parameters to be determined from the optimization. Algorithm 1 presents a basic algorithm for the Direct Shooting Method [16].

Algorithm 1 Basic Algorithm of the Direct Shooting Method

1: **Input**: Initial Guess of Parameters in Control Parameterization.
2: **Output**: Optimal Values of Parameters and Optimal Trajectory.
3: **while** Cost is Not at a Minimum and Constraints Not Satisfied **do**
4: Integrate Trajectory from t_0 to t_f;
5: Compute Error in Terminal Conditions;
6: Update Unknown Initial Conditions By Driving Cost to Lower Value;

In general terms, the multiple direct shooting diverges from direct shooting as the studied interval is divided into subintervals, where the direct shooting is then applied to each subinterval, as illustrated by Figure 2.5 [16].

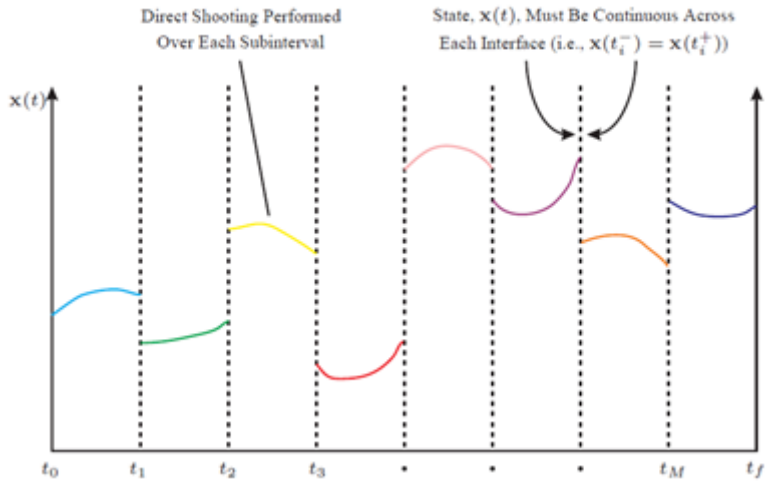


Figure 2.5: Illustration of the Direct Multiple Shooting Method (source: A. V. Rao).

In this case, continuity of the state is imposed at the interfaces as

$$x(t_i^-) = x(t_i^+) \quad \textbf{(2.22)}$$

2.7.3.2 Collocation methods

Local collocation is one of the two *direct collocation methods*. This family of methods is considered by many experts as the most powerful methods for solving general OCP. Direct collocation consists of state and control parameterization, where the state and control are approximated using a specified functional form.

For *global collocation*, the direct solution for the OCP is searched by enforcing the evaluation of the state and control vectors in discrete collocation points throughout the entire problem domain.

2.7.4 Solving Method discussion

Having formulated the problem, it is essential to choose a method to solve it. Choosing a solving method depends highly on the nature of the problem (consequently on its size) and also on the amount of time that one can invest in implementing the solution.

In general, methods for solving an OCP are divided into two categories, the *indirect methods* and the *direct methods*. The first category entails the derivation of the optimality conditions, while in the second category the solution to the problem is approximated in some suitable way. In this section a third category was mentioned, the *dynamic programming*.

Now let us evaluate the *attributes* and the *limitations* of each category of numerical methods for solving OCP, which are presented on Table 2.2, to decide on which technique to use.

Method category	Attributes	Limitations
Dynamic Programming	Easy to understand and implement.	Exponential increase of its size -> unsuitable for use om flight optimization.
		Problems arise from nonsmoothness of the value function.
Indirect Methods	Capable of producing highly accurate results	Entails derivation of optimality conditions (susceptible to errors).
	After establishing the BVP, the solution is quite straightforward.	Co-state variables are not representative of real physical entities -> challenging initial guess.
		Analytical derivation to establish BVP may vary significantly based on the problem -> limited applicability and flexibility of the technique.
Direct Methods	Does not require derivation of optimality conditions, avoiding possible errors.	Implementation may be more challenging compared to other categories.
	After converting the problem into NLP, it can be easily solved since NLP solvers are well developed and very efficient.	
	Can deal with large multidimensional problems as is the case of flight trajectory optimization.	

Table 2.2: Comparative table of Attributes and Limitations of each numerical method for solving OCP.

Considering all the aspects, it was decided that a *Direct Method* would fit well the purpose of the present work.

Chapter 3

Methodology and Tools

This chapter aims to present all the tools used to build the proposed method as well as those used to produce the results of the implemented examples; these are the problem statement, the basics of B-splines, the proposed method, the coordinates systems, and the respective conversion equations, and a brief description of the implementation's architecture.

3.1 Problem Statement

Our objective is to determine the state (trajectory or path) $x(t) \in \mathbb{R}^n$, and the control $u(t) \in \mathbb{R}^m$ that optimizes a given performance measure

$$J = \Phi[x(t_f), t_f] + \int_{t_0}^{t_f} \mathcal{L}[x(t), u(t), t]dt \quad (3.1)$$

subject to dynamic constraints,

$$\dot{x}(t) = f[x(t), u(t), t] \quad (3.2)$$

path constraints,

$$C_{min} \leq C[x(t), u(t), t] \leq C_{max} \quad (3.3)$$

and boundary conditions

$$\phi_{min} \leq \phi[x(t_0), t_0, x(t_f), t_f] \leq \phi_{max} \quad (3.4)$$

3.1.1 The Navigation Equations

Equation 3.5 presents the equations of navigation, in terms of matrix representation.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \quad (3.5)$$

Employing the OCP language, the *state variables* are

$$X = \begin{bmatrix} x & y & z & v_x & v_y & v_z \end{bmatrix}^T \quad (3.6)$$

where

- x: Cartesian x coordinate.
- y: Cartesian y coordinate.
- z: Cartesian z coordinate.
- v_x: velocity in the x axis.
- v_y: velocity in the y axis.
- v_z: velocity in the z axis.

and the *control variables* are

$$u = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T \quad (3.7)$$

where

- a_x: acceleration in the x axis.
- a_y: acceleration in the y axis.
- a_z: acceleration in the z axis.

As a result, trajectory optimization may be formulated in terms of Optimal Control Problem as follows.

Our goal is to determine an *optimal state trajectory*, X, corresponding to the optimal control, u,

$$MinJ(x,u) = \int_{t_0}^{t_f} (\dot{x}^T Q \dot{x} + u^T R u) dt \quad (3.8)$$

subject to

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{z} = v_z \\ \dot{v}_x = a_x \\ \dot{v}_y = a_y \\ \dot{v}_z = a_z \end{cases} \quad (3.9)$$

and other constraints, such as maximum velocities and/or accelerations.

3.2 B-splines

As B-splines are going to be a fundamental aspect of the proposed method, it is essential to present its basics, otherwise, it would be difficult or impossible for some readers to understand the presented solution.

To begin, consider a given knot vector $T = (T_0, T_1, ..., T_{n-1}, T_n, T_{n+1}, ..., T_{n+k})$, $k \geq 1$ and $n \geq 0$, the corresponding normalized ***B-spline functions*** N_{ik} of order k (degree $k-1$) is defined by

$$N_{i1}(t) = \begin{cases} 1, & T_i \leq t \leq T_{i+1} \\ 0, & \text{otherwise} \end{cases} \quad (3.10)$$

for $k = 1$, and

$$N_{ik}(t) = \frac{(t - T_i)}{(T_{i+k-1} - T_i)} N_{i,k-1}(t) + \frac{(T_{i+k} - t)}{(T_{i+k} - T_i + 1)} N_{i+1,k-1}(t) \quad (3.11)$$

for $k > 1$, and $i = 0, 1, ..., n$.

Using *B-spline functions* it is possible to build a *B-spline curve*. Let us consider a given set of ordered *control points* $d_i \in \mathbb{R}^m$, $m \geq 1$, $0 \leq i \leq n$, and the vector of knots points T, a *B-spline curve* of k order is defined by

$$X(t) = \sum_{i=0}^{n} N_{ik}(t) d_i \text{for} T_0 \leq t \leq t_{n+k} \quad (3.12)$$

where $N_{ik}(t)$ is the B-spline function defined in equations 3.10 and 3.11.

B-spline curves are a very useful tool for both *interpolation* and *parameterization*. Considering a set of data points $P_i \in \mathbb{R}^m$, $0 \leq i \leq n$, the b-spline of k order interpolation problem is stated as follows [18]:

- (1) to find the knot vector $T = (T_0, T_1, ..., T_{n-1}, T_n, T_{n+1}, ..., T_{n+k})$,
- (2) the parameter value t_i for each $P_i, 0 \leq i \leq n$, and
- (3) the control points d_i

such that the resulting b-spline curve defined by 3.12 satisfies the relation

$$X(t_i) = P_i, \forall i = 0, 1, ..., n. \quad (3.13)$$

To compute a B-spline interpolation, first we choose the knot vector T, so that the B-spline functions $N_{ik}(t)$ can be defined. For open curves, the first k T_i's must be equal and also the last k T_i's. The knots may be equally spaced for example. Applying 3.12 together with 3.13 we obtain

$$d_0 N_{0k}(t_i) + d_1 N_{1k}(t_i) + ... + d_n N_{nk}(t_i) = P_i. \quad (3.14)$$

Engaging the matrix form, the equation becomes

$$Ad = P, \quad (3.15)$$

where

$$A = \begin{pmatrix} N_{0k}(t_0) & N_{1k}(t_0) & ... & N_{nk}(t_0) \\ N_{0k}(t_1) & N_{1k}(t_1) & ... & N_{nk}(t_1) \\ \vdots & \vdots & \vdots & \vdots \\ N_{0k}(t_n) & N_{1k}(t_n) & ... & N_{nk}(t_n) \end{pmatrix}, \quad (3.16)$$

$d = (d_0, d_1, ..., d_n)^T$ and $P = (P_0, P_1, ..., P_n)^T$.

Now that B-spline functions and curves are presented, let us introduce the derivative. The first derivative of a B-spline curve is given by

$$X(t)' = \sum_{i=1}^{n} N_{ik}(t)' d_i \quad (3.17)$$

where

$$N_{ik}(t)' = k\frac{N_{i,k-1}(t)}{t_{i+k} - t_i} - k\frac{N_{i+1,k-1}(t)}{t_{i+k+1} - t_{i+1}} \quad (3.18)$$

Note that to define the derivative of a B-spline curve the only data needed are the control points d_i. The proof of 3.17 is given by [19].

3.3 Proposed method

We propose to solve 4D Trajectory Optimization Problem through a series of waypoints. Consider a set of ordered waypoints $W_j = (\phi_j, \lambda_j, h_j, t_j), j = 1, 2, ..., p, p \geq 2$. The state 3.6 and the control 3.7 are parameterized as B-spline curves

$$\begin{cases} x(t) = \sum_{i=1}^{n} N_{ik}(t) d_i^{(x)} \\ y(t) = \sum_{i=1}^{n} N_{ik}(t) d_i^{(y)} \\ z(t) = \sum_{i=1}^{n} N_{ik}(t) d_i^{(z)} \end{cases} . \quad (3.19)$$

Recalling from 3.17 that the derivative of a B-spline curve depends on the same control points that the b-spline curve itself, we have

$$\begin{cases} v_x(t) = \sum_{i=1}^{n} N_{ik}(t)' d_i^{(x)} \\ v_y(t) = \sum_{i=1}^{n} N_{ik}(t)' d_i^{(y)} \\ v_z(t) = \sum_{i=1}^{n} N_{ik}(t)' d_i^{(z)} \end{cases} . \quad (3.20)$$

and

$$\begin{cases} a_x(t) = \sum_{i=1}^{n} N_{ik}(t)'' d_i^{(x)} \\ a_y(t) = \sum_{i=1}^{n} N_{ik}(t)'' d_i^{(y)} \\ a_z(t) = \sum_{i=1}^{n} N_{ik}(t)'' d_i^{(z)} \end{cases} . \quad (3.21)$$

This means that having defined the knot vector, the optimization problem parameters are going to be the control points d_i. The performance measure integral is performed numerically using the *Composite Simpson's rule*, presented forward; this method is simple, provides fast calculations, and yet produces sufficiently accurate results for the purpose of the present work. Consider an interval over $[a, b]$; we have

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f_a + 2\sum_{j=1}^{\frac{n}{2}-1} f(x_{2j}) + 4\sum_{j=1}^{\frac{n}{2}} f(x_{2j-1}) + f_b\right] \quad (3.22)$$

n being an even integral that verifies $h = (b-a)/n$, and $x_j = a + jh$, for $j = 0, ..., n$. The associated error term is $-((b-a)/180)h^4 f^{(4)}(\mu)$, $\mu \in (a, b)$, $f_a = f(a)$, and $f_b = f(b)$. Having the parameterized problem, it becomes a simple Nonlinear Programming Problem that can be solved using available implemented techniques.

3.4 Coordinate Systems and Conversion

3.4.1 Coordinate Systems

In this work, two coordinate systems are used: the geodetic coordinate system and the Earth Centered Earth Fixed (ECEF) coordinate system. The geodetic coordinates are used to define the waypoints and to visualize the results, mainly because the default coordinate system used for navigation is the geodetic coordinate system and also because it enables better understanding and interpretation of the results.

The ECEF coordinate system is used mainly for internal calculations. Figure 3.1 illustrates both ECEF and geodetic coordinate systems.

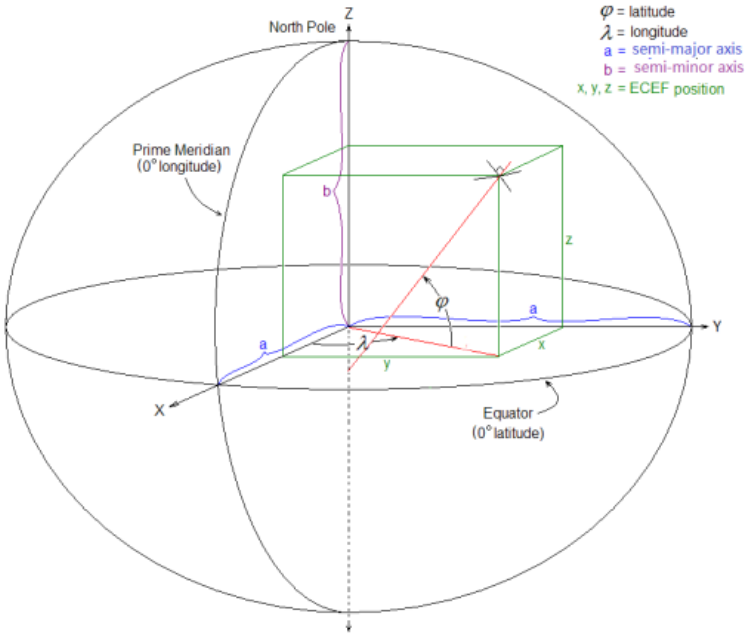


Figure 3.1: Geodetic and ECEF coordinate systems (Creative Commons license, source).

3.4.2 Conversion between Geodetic and Geocentric

The geocentric coordinates are related to the geodetic coordinates by the following formulae [20]:

$$X = (h + n)\cos\phi\cos\lambda \quad (3.23)$$

$$Y = (h + n)\cos\phi\sin\lambda \quad (3.24)$$

$$Z = (h + n - e^2 n) \sin \phi \quad (3.25)$$

where

$$n = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}} \quad (3.26)$$

3.4.3 Conversion between Geocentric and Geodetic

The algorithm to compute geodetic coordinates ϕ, λ, h from Cartesian coordinates X, Y, Z starts with the following sequence of formulae:

$$p = \frac{X^2 + Y^2}{a^2} \quad (3.27)$$

$$q = \frac{1 - e^2}{a^2} Z^2 \quad (3.28)$$

$$r = \frac{p + q - e^4}{6} \quad (3.29)$$

$$s = e^4 \frac{pq}{4r^3} \quad (3.30)$$

$$t = \sqrt[3]{1 + s + \sqrt{s(2 + s)}} \quad (3.31)$$

$$u = r\left(1 + t + \frac{1}{t}\right) \quad (3.32)$$

$$v = \sqrt{u^2 + e^4 q} \quad (3.33)$$

$$w = e^2 \frac{u + v - q}{2v} \quad (3.34)$$

$$k = \sqrt{u + v + w^2} - w \quad (3.35)$$

$$D = \frac{k\sqrt{X^2 + Y^2}}{k + e^2} \quad (3.36)$$

Then, the geodetic coordinates ϕ, λ, and h are computed by

$$\phi = 2 \arctan \frac{Z}{D + \sqrt{D^2 + Z^2}} \quad (3.37)$$

$$\lambda = 2 \arctan \frac{Y}{X + \sqrt{X^2 + Y^2}} \quad (3.38)$$

$$h = \frac{k + e^2 - 1}{k} \sqrt{D^2 + Z^2} \quad (3.39)$$

3.5 Implementation Architecture

Now that the mathematical aspects of the proposed solution are presented, let us look at the structure of the implemented solution. The examples presented in the next chapter were solved using a *Python* code implemented for this specific purpose. Figure 3.2 presents a synthesis of the structure of the implemented code, which can be found in Appendix B, containing the waypoints for example 1.

First, the necessary modules are loaded: *Numpy, Scipy* and *pymap3d* (for referential conversion), and also some specific functions. Then, the waypoints' matrix is created, containing the time, the latitude, the longitude, and the altitude, each line of the matrix corresponding to one waypoint.

Follows the creation of the knot vector, according to the desired number of discretization points, greater than or equal to the number of waypoints. The knot vector necessarily contains the time values of each waypoint, guaranteeing that the solution will include the waypoints. Then the objective functions are defined. It is important to mention that the calculations were made for each axis independently since the equations of each axis are not related.

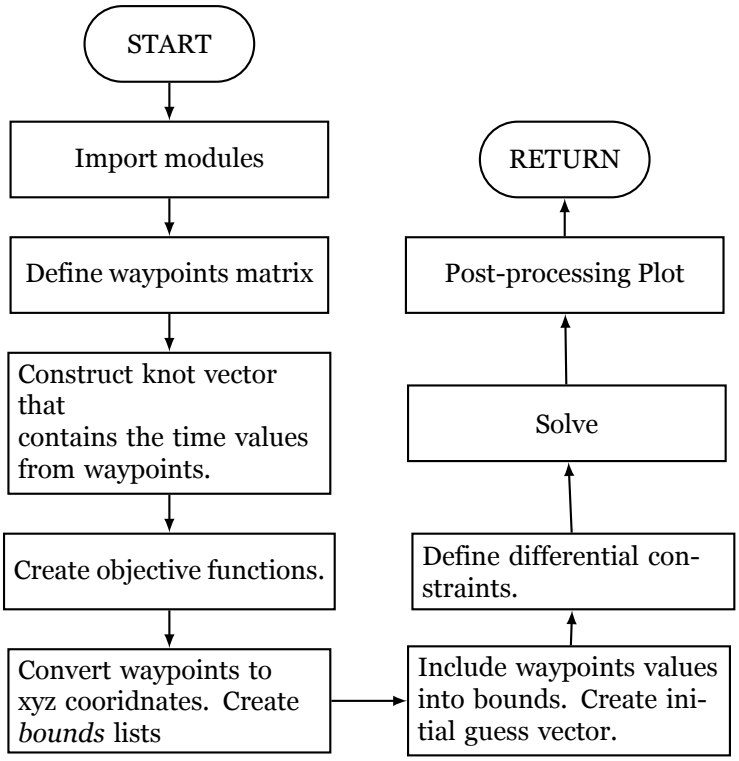


Figure 3.2: Synthesis of the code implemented to solve the Optimization Problem using the proposed method.

The waypoints are converted using the formulae 3.23-3.25 from geodetic to geocentric coordinates because the equations of navigation require such conversion. A list of *bounds* for each state and control is created to guarantee that each state and control do not assume values out of reach. At each discretization point, upper bound and lower bound are imposed; at waypoints, the values corresponding to the position, upper bounds and lower bounds have the same value.

Next, the waypoints values are included in the *bounds*, guaranteeing that the solution will necessarily include the waypoints. To promote a faster convergence to the solution, initial guesses vectors are created. For the position, the initial guess between each waypoint corresponds to the value of the later waypoint, and for the velocity, an average velocity is calculated using the waypoints.

The last steps include the definition of the functions representing the differential constraints, solving the optimization problem using the function *minimize* from *Scipy*, and plotting the results using *matplotlib*. For further information, one is invited to analyze the full code in the Appendix B.

Chapter 4

Simulation Results

To demonstrate the capacity and robustness of the proposed method as well as the implemented code, let us consider two examples from [21], the first representing a typical civil flight, and the second presents a round trip about Covilhã city. Finally, the last example uses waypoints from a commercial flight from Funchal (Madeira) to Lisbon.

4.1 Example 1

The first example refers to a typical civil flight. Table 4.1 presents the waypoints for this first example. Table 4.2 presents the convergence data for example 1. It is worth noticing that no major constraints violation was obtained, meaning that the solution passes exactly through the defined waypoints.

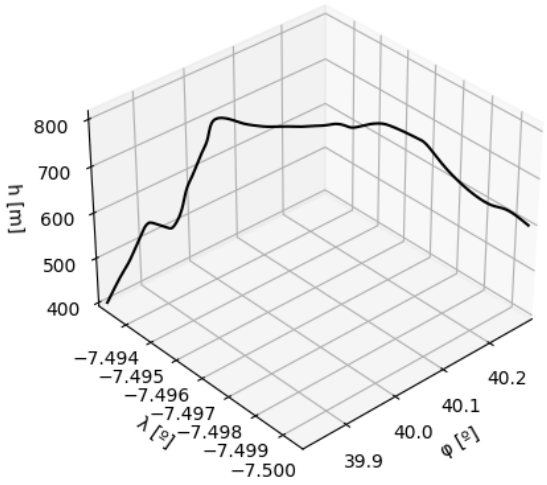
#	$t[s]$	λ[deg]	ϕ[deg]	$h[m]$
1	0	-7.493055556	39.82380833	400
2	126	-7.493611111	39.84300556	500
3	252	-7.494166667	39.85927222	600
4	288	-7.494722222	39.87773889	600
5	432	-7.494861111	39.91396111	700
6	594	-7.495000000	39.94871667	800
7	756	-7.495833333	39.98751111	800
8	882	-7.496388889	40.02142222	800
9	1008	-7.496944444	40.06275556	800
10	1170	-7.497500000	40.09205000	800
11	1332	-7.498055556	40.13682222	800
12	1494	-7.498611111	40.18511944	750
13	1620	-7.500000000	40.23539722	650
14	1728	-7.500555556	40.28389444	600

Table 4.1: Waypoints for Example 1.

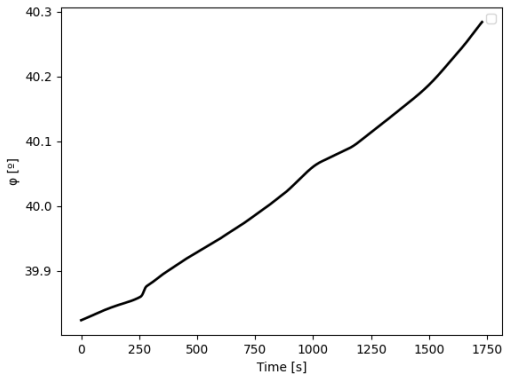
Axis	Iterations	Function Evaluations	Optimality	Constraints violation	Execution time $[s]$
x	178	20933	9.88e-05	4.72e-10	24
y	44	4719	9.99e-05	2.12e-08	5.8
z	179	21054	9.87e-05	1.16e-09	24

Table 4.2: Convergence data of example 1.

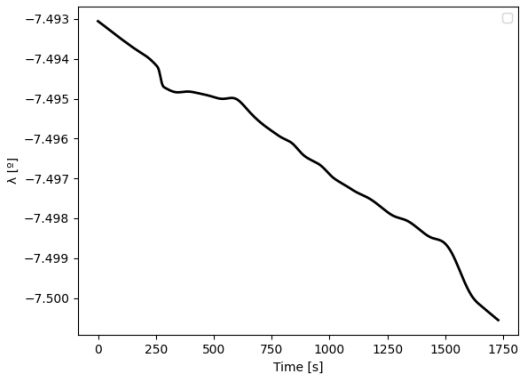
Figure 4.1 presents the optimal trajectory obtained for example 1, overall, the optimal trajectory is smooth.



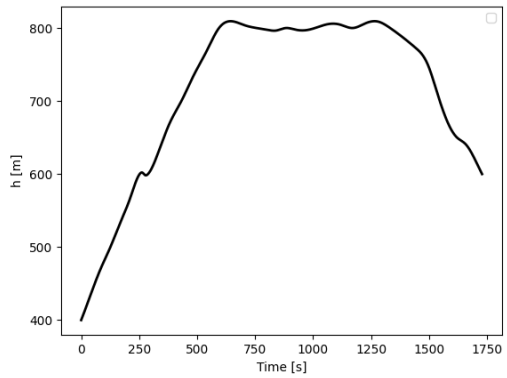
(a) 3D Trajectory.



(b) Latitude vs time.



(c) Longitude vs time.



(d) Altitude vs time.

Figure 4.1: Graphical results obtained for example's 1 optimal trajectory.

4.2 Example 2

The second example refers to a round trip around Covilhã City. Table 4.4 presents the waypoints for this second example. Table 4.3 presents the convergence data for example 2.

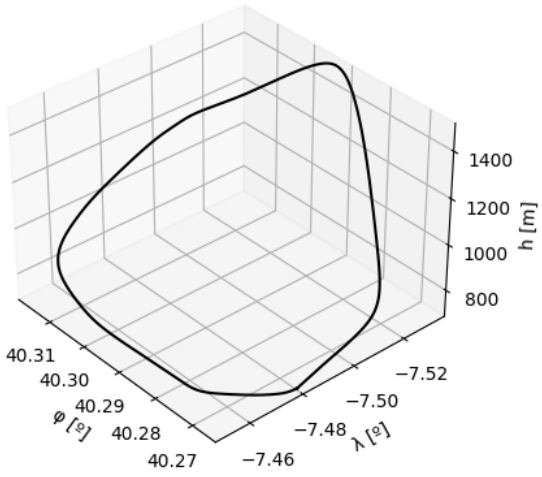
Axis	Iterations	Function Evaluations	Optimality	Constraints violation	Execution time $[s]$
x	131	16380	5.12e-05	6.09e-10	18
y	232	29510	6.76e-05	6.83e-10	31
z	614	79170	1.00e-04	2.58e-10	84

Table 4.3: Convergence data of example 2.

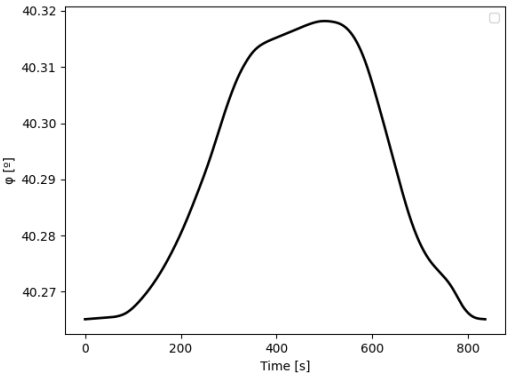
Figure 4.2 presents the optimal trajectory obtained for example 2, overall, the optimal trajectory is smooth. This example is the one that presents the best results in terms of trajectory quality and smoothness.

#	$t[s]$	λ[deg]	ϕ[deg]	$h[m]$
1	0	-7.47935	40.26508056	700
2	50.4	-7.493844444	40.26541667	750
3	82.8	-7.507930556	40.26605278	800
4	183.6	-7.526755556	40.27744167	1100
5	259.2	-7.534744444	40.29329444	1500
6	306	-7.524202778	40.30540833	1350
7	349.2	-7.5091	40.31254167	1250
8	432	-7.495969444	40.31637778	1150
9	489.6	-7.475677778	40.31813333	1000
10	565.2	-7.456869444	40.31474444	850
11	630	-7.449011111	40.29787778	810
12	694.8	-7.452633333	40.27941944	760
13	766.8	-7.460391667	40.27061111	730
14	795.6	-7.470616667	40.26656944	710
15	835.2	-7.47935	40.26508056	700

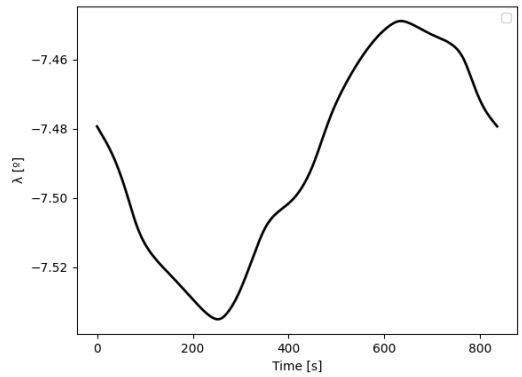
Table 4.4: Waypoints for Example 2.



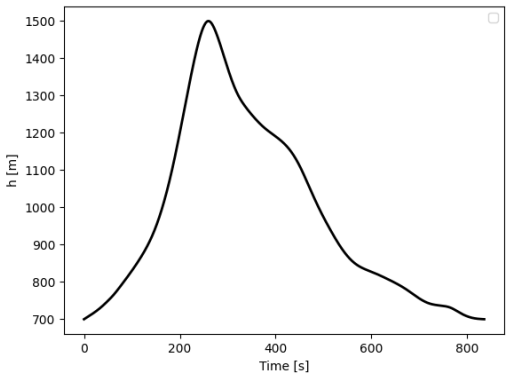
(a) 3D Trajectory.



(b) Latitude vs time.



(c) Longitude vs time.



(d) Altitude vs time.

Figure 4.2: Graphical results obtained for example's 2 optimal trajectory.

4.3 Example 3

The last example refers to a commercial flight route between Funchal (Madeira) and Lisbon. Table 4.5 presents the waypoints for this last example. Table 4.6 presents the convergence data for example 2.

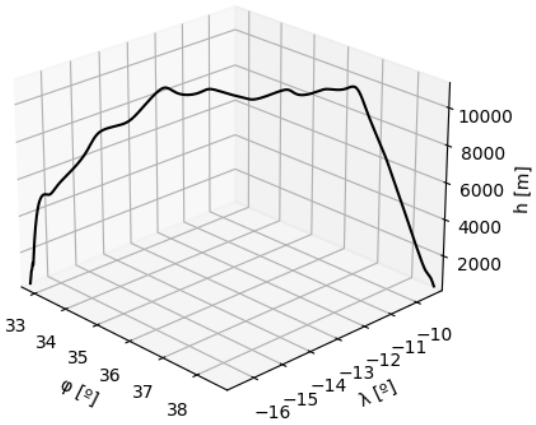
Overall, this example is the one that presented less satisfactory results. This could be related to the trajectory length, meaning that more knot points would be needed. This solution was investigated but the size of the problem increases greatly embarrassing the program's conversion to the solution.

#	$t[s]$	λ[deg]	ϕ[deg]	$h[m]$
1	0	-16.758467	32.711281	304.8
2	62	-16.695576	32.705135	982.98
3	103	-16.644567	32.691029	1234.44
4	128	-16.613459	32.704807	1584.96
5	520	-16.144129	33.107113	5623.56
6	1015	-15.391731	33.780537	8549.64
7	1701	-14.263916	34.762093	10972.8
8	2196	-13.392849	35.497421	10972.8
9	2949	-12.02631	36.583282	10972.8
10	3325	-11.293011	37.10408	10972.8
11	3635	-10.682294	37.529018	10972.8
12	3947	-10.089949	37.935974	7330.44
13	4198	-9.706839	38.232933	4396.74
14	4512	-9.290833	38.551212	1242.06
15	4651	-9.195855	38.669266	746.76
16	4765	-9.158298	38.739395	312.42

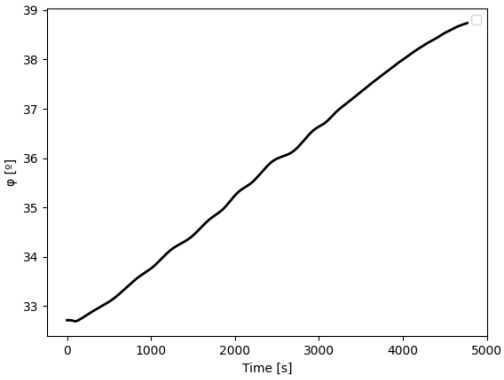
Table 4.5: Waypoints for Example 3.

Axis	Iterations	Function Evaluations	Optimality	Constraints violation	Execution time $[s]$
x	377	70612	6.31e-05	2.84e-11	74
y	1719	238107	9.70e-06	4.07e-08	360
z	2993	415193	3.50e-06	2.16e-08	460

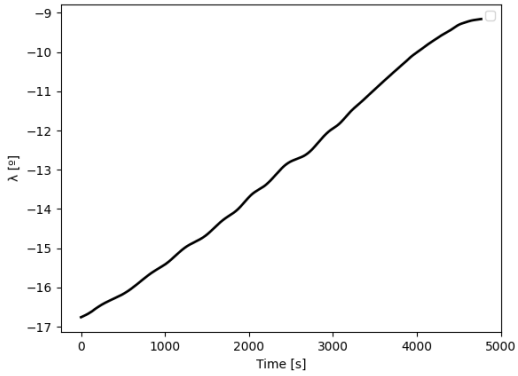
Table 4.6: Convergence data of example 3.



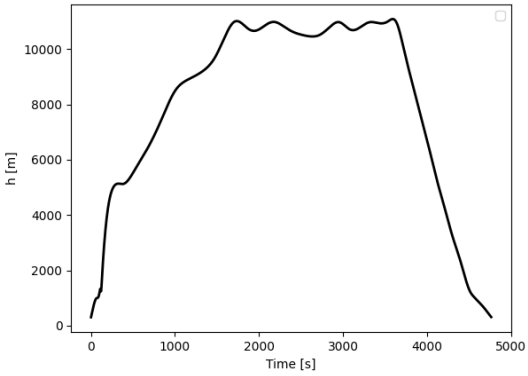
(a) 3D Trajectory.



(b) Latitude vs time.



(c) Longitude vs time.



(d) Altitude vs time.

Figure 4.3: Graphical results obtained for example's 3 optimal trajectory.

Chapter 5

Conclusion

The present work intended to study the use of B-spline Parameterization for Flight Trajectory Optimization problems formulated in terms of Optimal Control Problem. The basic aspects of Optimal Control Theory were presented, together with an overview of the suitable methods for solving OCP. All methods and tools that were used to compute the results were also presented.

To test and prove the robustness of the proposed method, three flight trajectory examples were presented: one civil aviation flight, one round trip around Covilhã City, and one commercial flight route between Funchal (Madeira) and Lisbon. Overall, the implemented *Python* code, using the proposed method, was able to obtain satisfactory results, producing smooth trajectories with minimal constraints violation.

Among the three examples, the second example presented the most accurate results, which may be related to the total time. Longer trajectories presented less satisfactory results, which may be improved by adding more discretization points; however, this leads to an enormous increase in the problem's size.

Future studies should include multiple simulations with different sets of waypoints to assess the best configuration of waypoints and convergence tolerance to produce good trajectories for different cases. Furthermore, the use of the proposed method for space trajectories optimization should be considered, as well as the study and implementation of the control to stimulate the desired flying device to follow the determined optimal trajectory for both atmospheric and spatial trajectories.

Bibliography

[1] R. G. Grant, *Flight: The Complete History of Aviation*, ser. ISBN: 978-1-46546-327-2. Dorling Kindersley, 2017. 1

[2] F. M. Howard, "Unlocking the secrets of the first airplane to fly: the wright flyer project story," *Technology in Society*, vol. 23, pp. 1–9, 2001. 1

[3] ICAO, *Annex 6 to the Convention on International Civil Aviation*, ser. ISBN: 978-92-9265-282-1. INTERNATIONAL CIVIL AVIATION ORGANIZATION, 2018. 2

[4] K. M. Valentine Goblet, Nicoletta Fala, "Identifying phases of flight in general aviation operations," in *15th AIAA Aviation Technology, Integration, and Operations Conference*. AIAA, 2015. 2

[5] Eurocontrol Experimental Centre, *SOFT - Study of Operational Flight Plans and Trajectories; Requirements for Advanced Flight Plan Information*. EUROCONTROL, 1998. 3

[6] EASA, EAA, *European Aviation Environmental Report*, 2019. 3, 5

[7] SESAR Joint Undertaking, *European ATM Master Plan, Executive view*, 2020. 4

[8] SESAR. (2022) Single european sky, about. [Online]. Available: https://www.sesarju.eu/background-ses 4

[9] A. H. Tremper et al, "Sources of particle number concentration and noise near london gatwick airport," *Environment International*, vol. 161, 2022. 6

[10] R. D. Alquezar and R. H. Macedo, "Airport noise and wildlife conservation: What are we missing?" *Perspectives in ecology and conservation*, vol. 17, pp. 163–171, 2019. 6

[11] R. W. H. Sargent, "Optimal control," *Journal of Computational and Applied Mathematics*, vol. 124, pp. 361–371, 2000. 9

[12] D. E. Kirk, *Optimal Control Theory, An Introduction*, ser. ISBN: 0-486-43484-2. Dover Publications, Inc., 1971. 9, 14

[13] D. G. Hull, *Optimal Control Theory for Applications*. Springer, 2003. 13, 14

[14] E. Todorov, *Optimal Control Theory*. MIT Press, 2006. 14

[15] A. E. Bryson and Tu-Chi Ho, *Applied Optimal Control*. Taylor & Francis, 1975. 14

[16] A. V. Rao, "A survey of numerical methods for optimal control," in *2009 AAS/AIAA Astrodynamics Specialist Conference*. AAS, 2009. 15, 19, 20

[17] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957. 16

[18] Choong-Gyoo Lim, "A universal parameterization in b-spline curve and surface interpolation," *Computer Aided Geometric Design*, vol. 16, pp. 407–422, 1999. 26

[19] J. Procházkhová, "Derivative of b-spline function," in *25 Konference o Gometrii a Pocitacove Grafice*, 2005, pp. 199–204. 27

[20] H. Vermeille, "Direct transformation from geocentric coordinates to geodetic coordinates," *Journal of Geodesy*, vol. 76, pp. 451–454, 2002. 29

[21] K. Bousson and P. F. F. Machado, "4d trajectory generation and tracking for waypoint-based aerial navigation," *WSEAS Transactions on Systems and Control*, vol. 8, no. 3, pp. 105–119, 2013. 33

Appendix A

Conference Paper

Conference paper submitted to the International Symposium on Aircraft Technology, MRO & Operations.

B-spline Parameterization Based Flight Trajectory Optimization

Rose A. C. Teixeira and K. Bousson

Abstract: Achieving the first heavier-than-air powered flight in the course of the 20th century was certainly a great landmark in human history. However, flight itself generally is not the final objective. To perform the desired mission effectively, it is necessary to determine the path to follow according to the objective, for instance minimizing time, path length, etc. Trajectory Optimization is the subject that deals with such problems, and the object of study of the present work. We propose the use of B-spline as a parameterization method for flight trajectory optimization problems formulated in terms of Optimal Control problems; satisfactory results were obtained using the *Python* language.

Keywords: B-spline Parameterization, Optimal Control, Trajectory Optimization

1. Introduction

1.1. Motivation

The aviation pioneers were mainly interested in achieving the unthinkable, making a heavier-than-air device fly. The first heavier-than-air powered flights were achieved during the 20th century [1], constituting a great achievement for mankind in aeronautical engineering. Great progress on the basic principles of flight by scientists and inventors and experimentations using gliders and steam-powered flying machines led aviation to a period of extremely fast development. Whereas in the early years of the 20th century pioneers like the Wright brothers worked hard to perform some short and poorly controlled flights [2], a couple of decades later the first commercial flights were taking-off, opening a new era of fast development worldwide. Achieving controlled powered flight led scientists and engineers to new challenges, namely performing a flight from point A to point B in a certain way, which could be minimizing the elapsed time, the trajectory length, or even the fuel consumption. Trajectory optimization is exactly the subject of the present work.

The augmentation of air traffic flow during the past decade in the European Union [3] is pushing Air Traffic Control toward a more cost-effective and flexible model, where more optimized and less rigid routes can be used to increase the airspace capacity [4]. As a response, the European Union, through the European

Corresponding A. K. Bousson
LAETA-UBI/AEROG, Covilhã, Portugal
e-mail: bousson@ubi.pt,
Supported by LAETA-AEROG in the framework of the Project UIDB/50022/2020.

T.H. Karakoç et al. (eds.), Name of Symposium Proceedings,
DOI 10.1007/…….

Commission, launched an ambitious initiative in 2004 titled the Single European Sky (SES) to reform the architecture of the European Air Traffic Management (ATM). The key objectives of this European initiative are [5]: to restructure European airspace, create additional capacity, and increase the overall efficiency of the ATM.

In addition to ATM performance improvement, environmental and noise concerns may also be addressed through trajectory optimization. Commercial flight pollutants emissions are a major concern today. In fact, reductions in pollutants and noise emissions can be achieved either by improving aircraft aerodynamics (thus reducing drag), improving engines, or through operational procedures. An example of operational improvement is the Free Route Airspace (FRA) concept, and both Continuous Climb Operations (CCO) and Continuous Descent Operations (CDO). Although trajectory optimization from the point of view of ATM is not the subject of the present paper, all aspects presented show the great importance of flight trajectory optimization. In addition, the fast-growing market of unmanned aerial vehicles also contributes to the necessity of consistent research on flight trajectory optimization. We will be looking at the mathematical aspects of trajectory optimization, fundamentally using Optimal Control principles.

1.2. Goals

- Optimization techniques – a brief overview of optimization techniques.
- Practical implementation – to implement a Python code to solve the case study problem.
- Authors' contribution: a universal framework for preliminary 4D trajectory optimization using B-spline parameterization.

2. Optimal Control Theory

We are going to formulate the trajectory optimization problem in terms of Optimal Control Problem (OCP). Optimal control theory (OCT) represents the outcome of the calculus of variations, and its first steps go back to the 17th century; however, OCT knew its greatest success during the 1960s through its aerospace applications [6]. The objective of an OCP is to determine the control signals that will cause a process to satisfy constraints and at the same time minimize (or maximize) a specified performance index [7]. The formulation of an OCP requires:

1. A mathematical description (or model) of the process to be controlled.
2. A statement of the physical constraints.
3. Specification of the performance measure.

2.1. The mathematical model

Any control/trajectory optimization problem includes a modeling phase of the process. The main objective is to obtain the simplest mathematical description that helps to accurately compute the response of the physical system to all expected input. The mathematical model is generally described by ordinary differential equations (ODE) in the state variable space form.

$$\dot{x}(t) = f(x(t), u(t), t). \tag{1}$$

2.2. Physical constraints

After defining the mathematical model, it is necessary to identify the *physical constraints* on the *state* and *control* values. These constraints depend on the problem, for instance, if the studied system is an aircraft the state constraints could be related to the position (obstacles), velocity (due to transonic condition), or acceleration (structural limitations). The control limitations represent the capacity of the system to change its state. These constraints are often related to the powerplant (engines), the maximum acceleration, etc.

2.3. The performance measure

To evaluate the performance of a system quantitatively, a performance measure is necessary. The performance measure refers to a given parameter or combination of parameters that expresses the desired quality into numerical quantity. Let us consider an example: let us assume that an aircraft is required to move from one point to another as quickly as possible. In this case, the statement clearly suggests the elapsed time as an ideal performance measure

$$J = t_f - t_0 = \int_{t_0}^{t_f} dt\,. \tag{2}$$

2.4. Optimization methods

Solving the OCP is, in general, the hardest part of dealing with trajectory optimization problems. Finding the right method to solve the OCP, and implementing it, is also part of the process, and is challenging. The methods suitable for solving OCP are various, one possible categorization divides the techniques into 3 categories:

- *Dynamic programming methods*: use Hamilton-Jacobi-Bellman optimality criteria.
- *Indirect methods*: use calculus of variations and Pontryagin's Minimum Principle (PMP) to derive the necessary conditions of optimality.
- *Direct methods*: consists of discretizing the continuous OCP and constructing a sequence of points. A finite set of variables is then obtained which can be solved using adequate optimization tools.

For further reading on OCT, one is referred to [6]-[10].

3. Methodology and Tools

3.1. B-splines

As B-splines are going the be a fundamental aspect of the proposed method, it is essential to present its basics. To begin, consider a given vector $T = (T_0, T_1, \dots, T_{n-1}, T_n, T_{n+1}, \dots, T_{n+k}), k \geq 1$ and $n \geq 0$, the corresponding normalized *B-spline functions* N_{ik} of order k (degree $k-1$) are defined by [11]:

$$N_{i1}(t) = \begin{cases} 1, & T_i \leq t \leq T_{i+1}, \\ 0, & \text{otherwise} \end{cases}, \text{for } k = 1 \text{ and} \tag{3}$$

$$N_{ik}(t) = \frac{(t - T_i)N_{i,k-1}(t)}{(T_{i+k-1} - T_i)} + \frac{(T_{i+k} - t)N_{i+1,k-1}(t)}{(T_{i+k} - T_{i+1})} \tag{4}$$

for $k > 1$ and $i = 0, 1, \dots, n$. Using *B-spline functions* it is possible to build a *B-spline curve*. Given an ordered list of control points $d_i \in \mathbb{R}^m$ $(m \geq 1), 0 \leq i \leq n$, and a knot vector T, a B-spline curve of k order is defined by

$$X(t) = d_i N_{ik}(t) \text{ for } T_0 < t < T_{n+k} \,. \tag{5}$$

Now that B-spline functions and curves are presented, let us introduce the derivative. The first derivative of a B-spline curve [12] is given by

$$X(t)' = \sum_{i=0}^{n} N_{ik}(t)' d_i \,, \tag{6}$$

$$N_{ik}(t)' = \frac{(k)\left(N_{i,k-1}(t)\right)}{t_{i+k} - t_i} - \frac{(k)\left(N_{i+1,k-1}(t)\right)}{t_{i+k+1} - t_{i+1}} \,. \tag{7}$$

3.2. Problem formulation

Let us begin presenting the equations of navigation, represented by (8) in terms of matrix notation.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \tag{8}$$

Employing an OCP language, the state vector X, and the control vector u, are

$$X = [x \quad y \quad z \quad v_x \quad v_y \quad v_z]^T \tag{9}$$

$$u = [a_x \quad a_y \quad a_z]^T \,, \tag{10}$$

x: cartesian x coordinate	v_x: velocity in the x-axis	a_x: acceleration in the x-axis.
y: cartesian y coordinate	v_y: velocity in the y-axis	a_y: acceleration in the y-axis.
z: cartesian z coordinate	v_z: velocity in the z-axis	a_z: acceleration in the z-axis.

As a result, trajectory optimization problem is formulated in terms of OCP as follows. Our goal is to determine an optimal state trajectory, X, corresponding to the optimal control, u, that minimizes the performance measure specified forward,

$$Min\, J(X, u) = \int_{t_0}^{t_f} (\dot{x}^T Q \dot{x} + u^T R\, u)\, dt \,, \tag{11}$$

subject to (8), and other constraints as maximum velocities and/or accelerations.

3.3. Proposed method

We propose to solve 4D Trajectory Optimization Problem through a series of waypoints. Consider a set of waypoints $W_j = (\varphi_j, \lambda_j, h_j, t_j), j = 1, 2, \dots, p, p \geq 2$. The state (9) and control (10) are parameterized as B-spline curves:

$$\begin{cases} x(t) = \sum_{i=1}^{n} N_{ik}(t) d_i^{(x)} \\ y(t) = \sum_{i=1}^{n} N_{ik}(t) d_i^{(y)} \\ z(t) = \sum_{i=1}^{n} N_{ik}(t) d_i^{(z)} \end{cases}, \begin{cases} v_x(t) = \sum_{i=1}^{n} N_{ik}(t)' d_i^{(x)} \\ v_y(t) = \sum_{i=1}^{n} N_{ik}(t)' d_i^{(y)} \\ v_z(t) = \sum_{i=1}^{n} N_{ik}(t)' d_i^{(z)} \end{cases}, \begin{cases} a_x(t) = \sum_{i=1}^{n} N_{ik}(t)'' d_i^{(x)} \\ a_y(t) = \sum_{i=1}^{n} N_{ik}(t)'' d_i^{(y)} \\ a_z(t) = \sum_{i=1}^{n} N_{ik}(t)'' d_i^{(z)} \end{cases}. \quad (12)$$

This means that having defined the knot vector, the optimization parameters are going to be the control points d_i. The performance measure integral is performed numerically using the *Composite Simpson's rule*. Having the problem parameterized, it becomes a simple Nonlinear Programming Problem that can be solved using available implemented techniques.

4. Simulations

To demonstrate the performance of the proposed method as well as the developed code, let us present one example, from [13]. Table 1 presents the list of waypoints. Figures 1, 2, 3, and 4 present respectively the results for the 3D trajectory, the latitude as a function of time, the longitude as a function of time, and the altitude as a function of time. The results obtained present smooth trajectories, there were no constraint violations and the solver achieved fast convergence. For the discretization, 45 points were used meaning that 2 points were added between each waypoint. The stop tolerance was set to $gtol = 10^{-4}$.

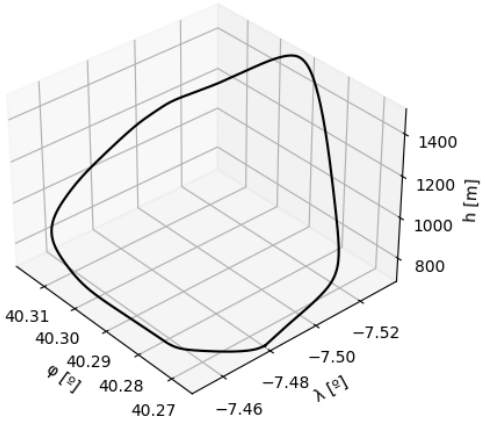


Fig. 1: 3D trajectory.

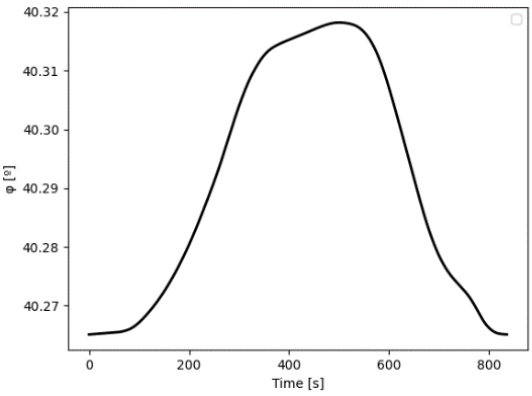


Fig. 2: Latitude vs Time.

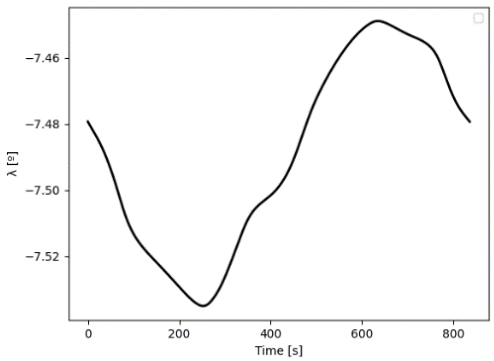


Fig. 3: Longitude vs Time.

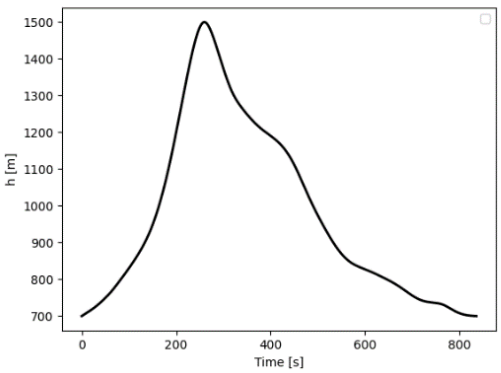


Fig. 4: Altitude vs Time.

Table 1: List of waypoints for the example.

Time [s]	Latitude [°]	Longitude [°]	Altitude [m]
0	-7,47935	40,26508	700
50,4	-7,49384	40,26542	750
82,8	-7,50793	40,26605	800
183,6	-7,52676	40,27744	1100
259,2	-7,53474	40,29329	1500
306	-7,5242	40,30541	1350
349,2	-7,5091	40,31254	1250
432	-7,49597	40,31638	1150
489,6	-7,47568	40,31813	1000
565,2	-7,45687	40,31474	850
630	-7,44901	40,29788	810
694,8	-7,45263	40,27942	760
766,8	-7,46039	40,27061	730
795,6	-7,47062	40,26657	710
835,2	-7,47935	40,26508	700

Conclusion

In this work, we proposed the use of B-splines as parameterization method for flight trajectory optimization problems. A code was implemented to solve the example, producing satisfactory results. B-splines are powerful tools as they enable the definition of smooth curves using a relatively low number of control points. Additionally, the derivatives are easily calculated, making them ideal for the studied problem. The work and the code developed are the ideal tools for preliminary determination of optimal trajectory for UAVs for example. Further studies may include the project of the controller to make the vehicle follow the optimal route.

References

[1] Grant, R. G., 2017. Flight: The Complete History of Aviation. Dorling Kindersley.
[2] Howard, F. M., 2001. Unlocking the secrets of the first airplane to fly: the wright flyer project history. Technology in Society, 23, pp. 1-9.
[3] EASA, EAA, 2019. European Aviation Environmental Report.
[4] SESAR Joint Undertaking, 2020. European ATM Master Plan, Executive view.
[5] SESAR (2022). Single European Sky. Available: https://www.sesarju.eu/background-ses
[6] Sargent, R. W. H., 2000. Optimal Control. Journal of Computational and Applied Mathematics, 124, pp. 361-371.
[7] Kirk, D. E., 1971. Optimal Control Theory, An Introduction. Dover Publications Inc.
[8] Todorov, E., 2006. Optimal Control Theory. Bayesian Brain.
[9] Bryson, A. E., and Yu-Chi Ho, 1975. Applied Optimal Control. Taylor & Francis.
[10] Hull, D. G., 2003. Optimal Control Theory for Applications. Springer.
[11] Choong-Gyoo Lim, 1999. A universal parameterization in B-spline curve and surface interpolation. Computer Aided Geometric Design, 16, pp. 407-422.
[12] Procházkhová, J., 2005. Derivative of B-spline function. Konference o Gometrii a Pocitacove Grafice, 25, pp. 199-204.
[13] Bousson, K., and Machado, F. F., 2013. 4D Trajectory Generation and Tracking for Waypoint-Based Aerial Navigation. WSEAS Transactions on systems and control, 8 (3), pp. 105-119.

Appendix B

Python Code

Python code implemented to solve the three examples, and validate the robustness of the proposed method.

```
#-------------------------------------Modules-------------------------------------#
import numpy
import numpy as np
import scipy
from scipy.optimize import minimize
from scipy.interpolate import BSpline
from scipy.interpolate import splrep
from scipy.optimize import NonlinearConstraint
from scipy.integrate import simps
from pymap3d import geodetic2ecef, ecef2geodetic

#-------------------------------------Data----------------------------------------#
#waypoints [time, lat, long, h]
#Example 1
Wps1 = np.array([[0,     39.82380833, -7.493055556,    400],
                 [126,   39.84300556, -7.493611111,    500],
                 [252,   39.85927222, -7.494166667,    600],
                 [288,   39.87773889, -7.494722222,    600],
                 [432,   39.91396111, -7.494861111,    700],
                 [594,   39.94871667, -7.495,          800],
                 [756,   39.98751111, -7.495833333,    800],
                 [882,   40.02142222, -7.496388889,    800],
                 [1008,  40.06275556, -7.496944444,    800],
                 [1170,  40.09205,    -7.4975,         800],
                 [1332,  40.13682222, -7.498055556,    800],
                 [1494,  40.18511944, -7.498611111,    750],
                 [1620,  40.23539722, -7.5,            650],
                 [1728,  40.28389444, -7.500555556,    600]])

#Example 2
Wps2 = np.array([[0,     40.26508056, -7.47935,        700],
                 [50.4,  40.26541667, -7.493844444,    750],
                 [82.8,  40.26605278, -7.507930556,    800],
                 [183.6, 40.27744167, -7.526755556,    1100],
                 [259.2, 40.29329444, -7.534744444,    1500],
                 [306,   40.30540833, -7.524202778,    1350],
                 [349.2, 40.31254167, -7.5091,         1250],
                 [432,   40.31637778, -7.495969444,    1150],
                 [489.6, 40.31813333, -7.475677778,    1000],
                 [565.2, 40.31474444, -7.456869444,    850],
                 [630,   40.29787778, -7.449011111,    810],
                 [694.8, 40.27941944, -7.452633333,    760],
                 [766.8, 40.27061111, -7.460391667,    730],
                 [795.6, 40.26656944, -7.470616667,    710],
                 [835.2, 40.26508056,  -7.47935,       700]])

#Example 3
Wps3 = np.array([[0,    32.711281,  -16.758467, 304.8],
                 [62,   32.705135,  -16.695576, 982.98],
                 [103,  32.691029,  -16.644567, 1234.44],
                 [128,  32.704807,  -16.613459, 1584.96],
                 [520,  33.107113,  -16.144129, 5623.56],
                 [1015, 33.780537,  -15.391731, 8549.64],
                 [1701, 34.762093,  -14.263916, 10972.8],
                 [2196, 35.497421,  -13.392849, 10972.8],
                 [2949, 36.583282,  -12.02631,  10972.8],
                 [3325, 37.10408,   -11.293011, 10972.8],
                 [3635, 37.529018,  -10.682294, 10972.8],
                 [3947, 37.935974,  -10.089949, 7330.44],
                 [4198, 38.232933,  -9.706839,  4396.74],
                 [4512, 38.551212,  -9.290833,  1242.06],
                 [4651, 38.669266,  -9.195855,  746.76],
                 [4765, 38.739395,  -9.158298,  312.42]])

Opts = [1, 2, 3, 4] #options
```

```
opt = 0

while True:
    try:
        while not (opt in Opts):
            print("Please, select the desired option, entering the \
corresponding #:\n",
              "1. Example 1: typical civil flight.\n",
              "2. Example 2: round trip around Covilhã.\n",
              "3. Example 3: Commercial flight Funchal -> Lisbon.\n",
              "4. Exit")
            opt=int(input())
        break
    except:
        print("Option invalid. Try again...")

if opt == 1:
    Wps=Wps1
    vmin = -100
    vmax = 100
    amin = -10
    amax = 10
    gtol = 0.0001
    maxiter = 1500
elif opt == 2:
    Wps=Wps2
    vmin = -220
    vmax = 220
    amin = -10
    amax = 10
    gtol = 0.0001
    maxiter = 2000
elif opt == 3:
    Wps=Wps3
    vmin = -220
    vmax = 200
    amin = -10
    amax = 10
    gtol = 0.00001
    maxiter = 5000
elif opt == 4:
    exit()



now = len(Wps) #number of waypoints
ti = Wps[0, 0]
tf = Wps[now-1, 0]
nop = (now-1)*3 #number of discretization points

vec_conc_point = [1/(now-1)]*(now-1)

for i in range(now-1):
    aux = np.linspace(Wps[i,0], Wps[i+1,0], int(nop*vec_conc_point[i])+1)
    if i ==0:
        T = aux
    else:
        T = np.concatenate((T, aux[1:len(aux)]))

phi_i, lam_i, h_i = (Wps[0, 1], Wps[0, 2], Wps[0, 3])
phi_f, lam_f, h_f = (Wps[now-1, 1], Wps[now-1, 2], Wps[now-1, 3])

q1=q2=q3=1/(vmax**2)
r1=r2=r3=1/(amax**2)
```

```
#-------------------------------------------------------------------------------#
#------------------------------Main program-------------------------------------#
#-------------------------------------------------------------------------------#

#-------------------------------Objective Function-----------------------------#
def objectivex(Bsolx):
    X = Bsolx[0:len(T)]
    vx = Bsolx[len(T):2*len(T)]

    new_T, dx, k = splrep(T, X)
    X_dot = BSpline(new_T,dx,k).derivative(1)(T)
    U1 = BSpline(new_T,dx,k).derivative(2)(T)
    return(  simps( q1*X_dot**2 + r1*(U1)**2 , T) )

def objectivey(Bsoly):
    Y = Bsoly[0:len(T)]
    vy = Bsoly[len(T):2*len(T)]

    new_T, dy, k = splrep(T, Y)
    Y_dot = BSpline(new_T,dy,k).derivative(1)(T)
    U2 = BSpline(new_T,dy,k).derivative(2)(T)
    return( simps( q2*Y_dot**2 + r2*(U2)**2 , T) )

def objectivez(Bsolz):
    Z = Bsolz[0:len(T)]
    vz = Bsolz[len(T):2*len(T)]

    new_T, dz, k = splrep(T, Z)
    Z_dot = BSpline(new_T,dz,k).derivative(1)(T)
    U3 = BSpline(new_T,dz,k).derivative(2)(T)
    return( simps( q3*Z_dot**2 + r3*(U3)**2 , T) )

#---------------------------------Bounds----------------------------------------#
xi, yi, zi = geodetic2ecef(phi_i, lam_i, h_i, deg=True)
xf, yf, zf = geodetic2ecef(phi_f, lam_f, h_f, deg=True)


#convert waypoints to XYZ
for i in range(now):
    phi = Wps[i, 1]
    lam = Wps[i, 2]
    h = Wps[i, 3]
    x, y, z = geodetic2ecef(phi, lam, h, deg=True)

    Wps[i, 1] = x
    Wps[i, 2] = y
    Wps[i, 3] = z

bx = (None, None)
bvx = (vmin, vmax)
bax = (amin, amax)
by = (None, None)
bvy = (vmin, vmax)
bay = (amin, amax)
bz = (None , None)
bvz = (vmin, vmax)
baz = (amin, amax)

#X axis
bndsx = [bx]*len(T) + [bvx]*len(T) + [bax]*len(T)

bndsx[0] = (xi, xi)
bndsx[len(T)-1] = (xf, xf)
```

```
#Y axis
bndsy = [by]*len(T) + [bvy]*len(T) + [bay]*len(T)

bndsy[0] = (yi, yi)
bndsy[len(T)-1] = (yf, yf)

#Z axis
bndsz = [bz]*len(T) + [bvz]*len(T) + [baz]*len(T)

bndsz[0] = (zi, zi)
bndsz[len(T)-1] = (zf, zf)

#Include waypoints into boundary vector & initial guesses improving
X0 = np.zeros(3*len(T), float)
Y0 = np.zeros(3*len(T), float)
Z0 = np.zeros(3*len(T), float)

count = 0
count_local = 0
i=0
for j in range (len(T)):
    if (i < now ):
        t = Wps[i, 0]
        if t == T[j]:
            bndsx[j] = (Wps[i, 1], Wps[i, 1])
            bndsy[j] = (Wps[i, 2], Wps[i, 2])
            bndsz[j] = (Wps[i, 3], Wps[i, 3])
            if i > 0 and count > 0 :
                X0[j-count_local:count]= np.ones(len(X0[j-count_local:count]))\
                *Wps[i-1, 1]
                X0[j-count_local +len(T):count+len(T)]= (Wps[i, 1]-Wps[i-1,1])\
                /(Wps[i, 0] - Wps[i-1, 0])

                Y0[j-count_local:count]= np.ones(len(Y0[j-count_local:count]))\
                *Wps[i-1, 2]
                Y0[j-count_local +len(T):count +len(T)]=(Wps[i, 2]-Wps[i-1,2])\
                /(Wps[i, 0] - Wps[i-1, 0])

                Z0[j-count_local:count] =np.ones(len(Z0[j-count_local:count]))\
                *Wps[i-1, 3]
                Z0[j-count_local +len(T):count+len(T)]=(Wps[i, 3]-Wps[i-1, 3])\
                /(Wps[i, 0] - Wps[i-1, 0])
                count_local = 0
            i = i+1
        count_local = count_local+1
        count = count + 1

#-------------------------------Constraints-----------------------------------#
#Define constraints (Velocity and acceleration in each axis)
def cons_vx (Bsolx): #X
    X = Bsolx[0:len(T)]
    vx = Bsolx[len(T):2*len(T)]
    ax = Bsolx[2*len(T):3*len(T)]

    new_T, dx, k = splrep(T, X)
    X_dot = BSpline(new_T,dx,k).derivative(1)(T)

    return ( vx - X_dot )

def cons_ax (Bsolx): #X
    X = Bsolx[0:len(T)]
    vx = Bsolx[len(T):2*len(T)]
    ax = Bsolx[2*len(T):3*len(T)]

    new_T, dx, k = splrep(T, X)
```

```
    X_dot2 = BSpline(new_T,dx,k).derivative(2)(T)

    return ( ax - X_dot2 )

def cons_vy (Bsoly): #Y
    Y = Bsoly[0:len(T)]
    vy = Bsoly[len(T):2*len(T)]
    ay = Bsoly[2*len(T):3*len(T)]

    new_T, dy, k = splrep(T, Y)
    Y_dot = BSpline(new_T,dy,k).derivative(1)(T)

    return ( vy - Y_dot )

def cons_ay (Bsoly): #Y
    Y = Bsoly[0:len(T)]
    vy = Bsoly[len(T):2*len(T)]
    ay = Bsoly[2*len(T):3*len(T)]

    new_T, dy, k = splrep(T, Y)
    Y_dot2 = BSpline(new_T,dy,k).derivative(2)(T)

    return ( ay - Y_dot2 )

def cons_vz (Bsolz): #Z
    Z = Bsolz[0:len(T)]
    vz = Bsolz[len(T):2*len(T)]
    az = Bsolz[2*len(T):3*len(T)]

    new_T, dz, k = splrep(T, Z)
    Z_dot = BSpline(new_T,dz,k).derivative(1)(T)

    return ( vz - Z_dot )

def cons_az (Bsolz): #Z
    Z = Bsolz[0:len(T)]
    vz = Bsolz[len(T):2*len(T)]
    az = Bsolz[2*len(T):3*len(T)]

    new_T, dz, k = splrep(T, Z)
    Z_dot2 = BSpline(new_T,dz,k).derivative(2)(T)

    return ( az - Z_dot2 )

cons_bound = np.zeros(len(T), float)
consvx = NonlinearConstraint(cons_vx, cons_bound, cons_bound)
consax = NonlinearConstraint(cons_ax, cons_bound, cons_bound)
consvy = NonlinearConstraint(cons_vy, cons_bound, cons_bound)
consay = NonlinearConstraint(cons_ay, cons_bound, cons_bound)
consvz = NonlinearConstraint(cons_vz, cons_bound, cons_bound)
consaz = NonlinearConstraint(cons_az, cons_bound, cons_bound)

consx = [consvx, consax]
consy = [consvy, consay]
consz = [consvz, consaz]

#---------------------------------Solve-----------------------------------------#
print("\nPerforming optimization on x axis ...")
print("x axis optimization results:")
solx = minimize(objectivex , X0, bounds=bndsx, constraints = consx, \
                method='trust-constr', \
            options={'disp': True, 'gtol':gtol,'maxiter':maxiter, 'verbose':1})

print("\nPerforming optimization on y axis ...")
print("y axis optimization results:")
```

```
soly = minimize(objectivey , Y0, bounds=bndsy, constraints = consy,\
                method='trust-constr', \
            options={'disp': True, 'gtol':gtol,'maxiter':maxiter, 'verbose':1})

print("\nPerforming optimization on z axis ...")
print("z axis optimization results:")
solz = minimize(objectivez , Z0, bounds=bndsz, constraints = consz, \
                method='trust-constr', \
            options={'disp': True, 'gtol':gtol, 'maxiter':maxiter, 'verbose':1})

#-------------------------------Post processing--------------------------------#
T_plot = np.linspace(ti, tf, 1000)

new_T, dx, k = splrep(T, solx.x[0:len(T)])
Sol_x = BSpline(new_T,dx,k)(T_plot)
Sol_vx = BSpline(new_T,dx,k).derivative(1)(T_plot)
Sol_ax = BSpline(new_T,dx,k).derivative(2)(T_plot)

new_T, dy, k = splrep(T, soly.x[0:len(T)])
Sol_y = BSpline(new_T,dy,k)(T_plot)
Sol_vy = BSpline(new_T,dy,k).derivative(1)(T_plot)
Sol_ay = BSpline(new_T,dy,k).derivative(2)(T_plot)

new_T, dz, k = splrep(T, solz.x[0:len(T)])
Sol_z = BSpline(new_T,dz,k)(T_plot)
Sol_vz = BSpline(new_T,dz,k).derivative(1)(T_plot)
Sol_az = BSpline(new_T,dz,k).derivative(2)(T_plot)

Sol_Phi = []
Sol_Lam = []
Sol_h = []

#Convert results from XYZ to Phi, Lam, h
for i in range (len(T_plot)):
    phi, lam, h = ecef2geodetic(Sol_x[i], Sol_y[i], Sol_z[i], deg=True)
    Sol_Phi.append( phi )
    Sol_Lam.append( lam )
    Sol_h.append( h )

#-----------------------------------Plot---------------------------------------#
import matplotlib.pyplot as plt
print("\nPlot...")
#3D
fig = plt.figure(1)
ax = fig.add_subplot(111, projection='3d')
ax.plot(Sol_x,Sol_y,Sol_z,'k-')
ax.set_xlim(min(Sol_x),max(Sol_x))
ax.set_ylim(min(Sol_y),max(Sol_y))
ax.set_zlim(min(Sol_z),max(Sol_z))
ax.set_xlabel('x [m]')
ax.set_ylabel('y [m]')
ax.set_zlabel('z [m]')


fig = plt.figure(2)
ax = fig.add_subplot(111, projection='3d')
ax.plot(Sol_Phi,Sol_Lam,Sol_h,'k-')
ax.set_xlim(min(Sol_Phi),max(Sol_Phi))
ax.set_ylim(min(Sol_Lam),max(Sol_Lam))
ax.set_zlim(min(Sol_h),max(Sol_h))
ax.set_xlabel('φ [°]')
ax.set_ylabel('λ [°]')
ax.set_zlabel('h [m]')
```

```
#2D
#X, Y, Z Position
plt.figure(3) #X
plt.plot(T_plot,Sol_x,'k-',lw=2,label=r'$x$')
plt.legend(loc='best')
plt.xlabel('Time [s]')
plt.ylabel('X coordinate [m]')

plt.figure(4) #Y
plt.plot(T_plot,Sol_y,'k-',lw=2,label=r'$y$')
plt.legend(loc='best')
plt.xlabel('Time [s]')
plt.ylabel('Y coordinate [m]')

plt.figure(5) #Z
plt.plot(T_plot,Sol_z,'k-',lw=2,label=r'$z$')
plt.legend(loc='best')
plt.xlabel('Time [s]')
plt.ylabel('Z coordinate [m]')



#Pi, Lam, h Position
plt.figure(6) #Phi
plt.plot(T_plot,Sol_Phi,'k-',lw=2,label=r'$φ$')
plt.legend(loc='best')
plt.xlabel('Time [s]')
plt.ylabel('φ [°]')

plt.figure(7) #Lam
plt.plot(T_plot,Sol_Lam,'k-',lw=2,label=r'$λ$')
plt.legend(loc='best')
plt.xlabel('Time [s]')
plt.ylabel('λ [°]')

plt.figure(8) #h
plt.plot(T_plot,Sol_h,'k-',lw=2,label=r'$h$')
plt.legend(loc='best')
plt.xlabel('Time [s]')
plt.ylabel('h [m]')
plt.show()
print("\nEnd of program execution. Please run again to choose another example.")
```