Flight optimization with obstacle avoidance for the collaborative Mini-Bee project

Cédric Belmant Ilinka Clerc

January 16, 2020

Abstract

Mini-Bee project is a TRL3 collaborative project, launched in 2015, centered on the eponymous innovative vehicle. The Mini-Bee is a hybrid Vertical Take-Off and Landing (VTOL) aircraft which aims to become a flying ambulance. The different aspects of the project have been dispatched over time between different French schools. A 3D collaborative GPS is an important part of the project which still needs to be implemented. The goal of this GPS is two-fold: facilitate flight management workload for the pilot and optimize features like flight time or fuel consumption. In our project, we aimed to build the bases of a 3D GPS program, in a simplified model with basic constraints.

To construct the core algorithm of this GPS, we first worked on describing our physical model with a system of equations. We obtained a nonlinear control problem with constraints on the final state, which represented a complex problem to solve. In continuous time, this was an infinite-dimensional optimization problem, which could be very difficult to solve, so we chose to approximate it by a finite-dimensional problem with a polynomial basis. The Legendre Pseudospectral (PS) resolution method is an adequate technique for this type of problem. We adapted this method to our model and used it to implement a search algorithm to get the optimal trajectory.

Finally, we implemented an algorithm with adequate libraries with the aim of solving our optimization problem. Nevertheless, we were not able to obtain results due to a lack of time. Our algorithm is not giving tangible results at the moment and different parts of our model could be improved. However, we have found a method able to obtain great results in similar problems and our research remains a solid base which would allow the completion of our initial objective in a near future.

Keywords: 3D GPS – Optimal trajectory – Nonlinear control problem – Legendre Pseudospectral resolution method

Contents

1 Introduction

1.1 Project overview

Mini-Bee project is a collaborative project in TRL3 (Technology readiness level 3). The Mini-Bee is a hybrid Vertical Take-Off and Landing (VTOL) aircraft, a mix between a plane, a helicopter, a drone and a car. This innovative vehicle could have multiple use cases such as a taxi, however a medical purpose is mainly studied and the Mini-Bee could particularly be useful as an ambulance. A TRL3 representation of the Mini-Bee is shown below:

Figure 1: Concept of the TRL3 Mini-Bee

The project is developed through a Lesser Open Source Licence. Since its launch in January 2015, multiple schools have participated in different fields (on aviation, mechanical or computing aspects). In particular, EISTI, ESTACA and Centrale Supelec defined the bases of a 3D collaborative GPS. The goal of this GPS was to facilitate the flight plan management workload for the pilot and reduce flight time or fuel consumption.

However, the resulting algorithm was simple and not efficient. The previous system was discrete, with a step by step evaluation, that was not effective in terms of computation performance and obstacles avoidance. Our objective is to redefine this 3D GPS system to generate a good base which could be further developed in future projects.

To generate a flight solution that conducts the Mini-Bee from a point A to a point B, we have an infinite number of possibilities. It contrasts with existing 2D path optimization tools such as Google Maps that considers a finite number of routes. The number of possible route combinations can be very large, but stays finite. Our problem falls under the category of continuous optimization, that require a complete different approach compared to that of traditional 2D GPS tools.

The evolution of the aircraft position follows physical rules, described with ordinary differential equations. However, these equations involve also other variables, such as the mass of the aircraft and its velocity. Usually we do not seek to minimize the distance travelled. Instead, we want to find a path that minimizes the fuel consumption or flight time. The solution is expressed in terms of aircraft control, e.g. thrust, and direction changes. Therefore, it is important to integrate the aircraft dynamics into the core of the problem.

1.2 State of Art

Optimal control problems with nonlinear systems can be very complex. Advanced developments in optimization lead to two main families of methods: direct and indirect methods.

1.2.1 Indirect methods

Indirect methods involve the differential equation of the problem directly into their formulation. They decline the problem into optimality conditions, associated to the differential equation and to the cost (flight time of fuel consumption) of the resulting trajectory. They yield several highly nonlinear equations, known as Hamilton-Jacobi-Bell (HJB) equations. We will not detail these equations because they require setting a lot of different functions, and we will not use them for this project. The solution to these equations is known as the *value* function, which attributes a performance for all given states (positions) in time. Then the control can be retrieved from it with other optimal equations. The solution is very hard to obtain, so a neural network is often used to approximate it with a sophisticated training called reinforcement learning.

Many algorithms were developed to solve these equations. For instance, an improved Newton algorithm was studied to resolve an optimal control problem of a constrained nonlinear system [Wu and Zhang, 2016]. This method aimed to be used with complex constraints that did not allow the application of standard optimization methods. This method integrates inequality constraints but no equality constraints. For example, states and controls can be explicitly bounded, but their value cannot be explicitly set at certain points. Also, the algorithm needs a specified final time. In our case, the final time is part of the optimization process and is thus flexible. This algorithm cannot be used for our problem.

Terminal control problems are defined as finite-time horizon problems with constraints on the final state, in the sense that the desired state has to be reach within a finite time. Many trajectory planning problems like that of the Mini-Bee belong to terminal control problems [Heydari and Balakrishnan, 2013]. Several methods can be used to solve these kinds of problems, based on neurocontrollers, that are neural networks which output the optimal control. However, these techniques mostly rely on neural networks and reinforcement learning (a complex training logic) to solve the optimization problem. Furthermore, convergence proofs have not been established yet to ensure that the given problem can be solved. Additionally, a 3D trajectory optimization includes many constraints that make the convergence a lot harder. One area where neurocontroller methods shine particularly would be flight automation: most methods reformulate the problem as the minimization of an error (for example, the error with respect to a given reference trajectory), that we cannot apply to our problem. For example, an online adaptive critic flight control was proposed [Ferrari and Stengel, 2004], [Kim and Calise, 1997]. This approach develops action and critic networks in a two-stage logic. First, they are trained offline with a linear reference model, for which an optimal solution can be extracted; second,

they are embedded in the operating control loop, on board an aircraft for example. Then, they learn online from sensor data to improve their results and update their parameters to take into account the real (nonlinear) behavior of the aircraft dynamics.

Exclusive offline training methods that include the nonlinear problem with free final time are also developed [Han and Balakrishnan, 2002]. However, these methods have mostly been tested for cases where evolution of the state variables can be reformulated with respect to a variable that has a fixed final value. In our case, this would mean reformulating the problem with the position as driving variable, instead of the time. Such reformulation is then non applicable here. Although those methods do not exclude our case, neural network convergence is a lot harder to obtain without this change of variable.

1.2.2 Direct methods

Direct methods aim at solving the problem in continuous-time using approximations, e.g. polynomial approximations of the states and controls of the problem. The differential equation dictating the evolution of the physical system is enforced as an equality constraint at certain points, along with initial and final conditions on the state. The polynomial ("spectral") representation of the states and controls possess great properties in terms of smooth functions approximation with a small amount of parameters [Gourgoulhon, 2005]. In the case where discontinuities (such as switchs or impulses) are present in the optimal solution, refinement methods can split the approximation interval in several phases [Gong et al., 2008]. However, the approximated optimal states and controls need to be sufficiently smooth most of the time to get the best results.

For trajectory planning problems, a robust family of direct methods using polynomial approximations were developed, with convergence proofs and excellent results. For example, a Zero-Propellant Maneuver to orient the International Space Station successfully used the Legendre Pseudospectral (PS) method in 2006 [Ross and Karpenko, 2012]. They proved very efficient for a wide range of trajectory planning problems, such as boat, car and drone maneuvers [Gong et al., 2009]. PS methods output state and control coefficients in a polynomial basis, e.g. with Lagrange polynomials. They also output dual variables that can be used to check the optimality of the given solution.

The Legendre PS method best fits our problem as a direct method with given initial and terminal states and a free final time [Gong et al., 2009], [Ross and Karpenko, 2012].

1.3 Problem definition

To construct a useful 3D GPS, many aspects have to be considered. The Mini-Bee has to reach a given point through an optimal trajectory according to a given criterion while avoiding obstacles and possibly satisfying other constraints. Through this optimization, we try to minimize two principal factors: fuel consumption and flight time.

In our project, we consider only obstacles which are static during the simulation (for instance: cities, mountains or protected airspace). In further developments of this work, the Mini-Bee should also avoid moving objects and take meteorological phenomena (wind, rain, storms...) into account (avoiding dangerous zones or integrating meteorological effects into the dynamic model). The global project also includes air corridor that the Mini-Bee should prefer and use as roads with related constraints. The final specification also plans to potentially add

intermediary points on the trajectory (that could potentially be removed during the flight). Finally, the GPS must contain a collaborative aspect. During the conception of our algorithm, we need to think about these future features to facilitate their development and incorporation.

2 Physical model

We need to represent the Mini-Bee by a physical model on which we could work. The Mini-Bee can fly as a plane, as a helicopter or as both. To simplify the problem, we consider a plane configuration within the frame of this project.

2.1 Coordinate frames

For the sake of simplicity, we will write the model equations in a local coordinate system also named body coordinate system, centered on the aircraft and rotating with it. It is then a rotating and translating frame with respect to the frame of reference, that named inertial frame.The body orientation, or attitude, will be represented by the three base vectors of the orthonormal body coordinate system, namely i, j, k. The local coordinate system (i, j, k) considered is as shown below:

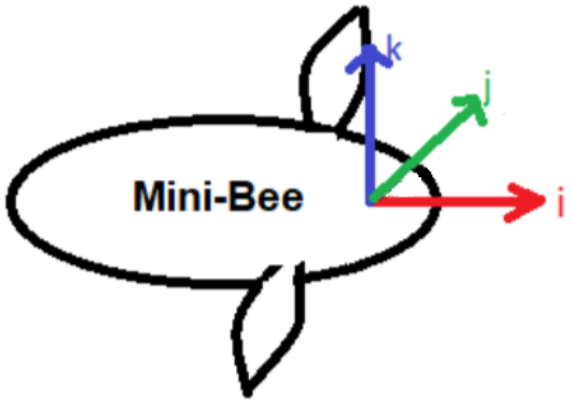

Figure 2: Body coordinates for the Mini-Bee.

2.2 Commands

The commands of an aircraft are usually comprised of:

- A thrust lever, impacting the gas consumption and throttling the aircraft forward,
- A stick, directing the aircraft along its pitch and roll axes,
- A rubber, directing the aircraft along its yaw axis.

We will then have one direct control over the fuel consumption of the aircraft, propelling the aircraft only in one axis (i). The other commands will orient the aircraft without any fuel consumption by exerting a torque (due to a resulting aerodynamic force away from the center of mass) along any of its local axes (i, j, k).

The dynamics can be decomposed in two main parts:

1. Those describing the evolution of the aircraft orientation, its angular velocity and its angular acceleration,
2. The translational dynamics of the aircraft, involving the position, speed and acceleration of the aircraft.

Additionally, we include the mass as part of the system, with a simple model linking the derivative of the mass proportionally to the thrust exerted :

$$\dot{m} = -k_t U_1 \tag{1}$$

with $k_t > 0$ a proportionality factor that depends on the engine type and parameters, as well as atmospheric values (pressure, temperate, air intake...). In our case, we will keep this proportionality factor constant through time.

The control torque that is exerted typically makes use of deflective control surfaces δ that will generate an aerodynamic torque. However, in order to simplify and generalize the problem to a wider range of aircrafts, we can consider that the control is exerted directly on the angular velocity, without considering any aerodynamic torque and neglecting the inertial acceleration (considering a small angle ω). We will then consider that the angular velocity is directly controlled, i.e. that $\omega = U_\omega := \begin{pmatrix} U_2 & U_3 & U_4 \end{pmatrix}^\top$. The aircraft attitude is represented with quaternions in our model, known as more stable than Euler angles, but of more abstract nature. More information about these mathematical beings can be found in Appendix A.

In the local coordinate system, the control commands directly a force (thrust) $U_1 \in \mathbb{R}^+$ along the i axis, as well as an angular velocity $U_\omega \in \mathbb{R}^3$ exerted in all three axes of the body frame. The ordinary differential equation (ODE) reads:

$$\begin{cases} \dot{q} = \frac{1}{2} M_s(U_\omega) q \\ \dot{x} = v \\ \dot{v} = \frac{F_a}{m} + g + \frac{U_1}{m}(k_t v + i) \\ \dot{m} = -k_t U_1 \end{cases} \tag{2}$$

With:

- $q(t) = \begin{pmatrix} q_0(t) & q_1(t) & q_2(t) & q_3(t) \end{pmatrix}^\top$ denotes the attitude quaternion defining the orientation of the aircraft,
- $i(t) = \begin{pmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1 q_2 + q_0 q_3) & 2(q_1 q_3 - q_2 q_0) \end{pmatrix}^\top$ is the forward axis in the body frame,
- $x(t) = \begin{pmatrix} x_1(t) & x_2(t) & x_3(t) \end{pmatrix}^\top$ is the position of the Mini-Bee in the global coordinate system,
- $v = \begin{pmatrix} v_1(t) & v_2(t) & v_3(t) \end{pmatrix}^\top$ is the speed of the Mini-Bee in the global coordinate system,
- $U := \begin{pmatrix} U1 & U_2 & U_3 & U_4 \end{pmatrix}^\top$ the control,
- m the Mini-Bee mass,

- $k_t > 0$ a positive constant proportionality factor depending on the engine type and atmospheric values,
- g the gravitational force,
- F_a the aerodynamic force, which we will detail further.

This system defines the problem $\dot{X}(t) = f(X(t), U(t))$ with the state $X(t)$ as shown below:

$$X(t) = \begin{pmatrix} q(t) & x(t) & v(t) & m(t) \end{pmatrix}^\top$$

2.3 Objective function

The objective function, or cost, is the quantity that will be minimized in the optimization problem. Considering here that the final time and the fuel consumption are to be minimized, a cost of the following form can be chosen:

$$J(X, U, t_f) = \gamma(t_f - t_0) + (1-\gamma)(m_f - m_0) = \gamma(t_f - t_0) + (1-\gamma) \int_{t_0}^{t_f} \underbrace{(k_t U_1(t))}_{=\dot{m}} \, dt \tag{3}$$

with:

- t_0 and t_f respectively the initial and final time,
- m_0 and m_f respectively the initial and final mass,
- $\gamma \in [0, 1]$ a weight that can be chosen arbitrarily to focus on flight time, fuel consumption or both.

2.4 Trajectory requirements

The control needs to be bounded, i.e. we want that the absolute value of each component of the control does not go beyond a specified threshold. Also, the thrust U_1 needs to be positive. Mathematically, the control reads the following set of constraints:

$$\|U_i\|_\infty = \max_t |U_i(t)| < U_i^{\max} \quad \forall i \in [\![1, 4]\!] \tag{4}$$

$$U_1(t) \geq 0 \quad \forall t \in [t_0, t_f] \tag{5}$$

The final position is fixed:

$$x(t_f) = x_f \tag{6}$$

The initial state is given:

$$X(t_0) = X_0 \tag{7}$$

The state needs to satisfy obstacle-avoidance constraints, that can be modelled by the following inequality constraints. Setting a set of N_c obstacle objects $\mathcal{C} = \{O_1, O_2, ..., O_{N_c}\}$, a obstacle avoidance criterion can be defined as a set of inequalities of the form

$$\left\| \frac{x_1 - x_c^{1,i}}{a^i} \right\|^{p_1^i} + \left\| \frac{x_2 - x_c^{2,i}}{b^i} \right\|^{p_2^i} + \left\| \frac{x_3 - x_c^{3,i}}{c^i} \right\|^{p_3^i} \geq 1, \quad \forall i \in [\![1, N_c]\!] \tag{8}$$

In other terms, the position $(x_1, x_2, x_3)^\top$ must be outside of N_c regions defined by p-norms. p-norms define regions that can have a shape between that of a cube ($p = +\infty$) and a sphere ($p = 2$). The scaling factors a^i define the shape elongation, i.e. how much the shape is stretched in a particular direction. Finally, the position of each obstacle object is expressed through $(x_c^{1,i}, x_c^{2,i}, x_c^{3,i})^\top$.

2.5 Aerodynamic model

The aerodynamic model describes the forces and torques exerted on the plane by the surrounding fluid (air, for planes). The ODE system defined in (2) does not need the aerodynamic torque, so we will skip the torque model here. The model for the aerodynamic force will be explicited and the underlying assumptions clearly stated.

2.5.1 Aerodynamic forces

The aerodynamic force F_a can be expressed in the wind coordinate system (x_a, y_a, z_a) through aerodynamic coefficients [Kai, 2018]:

$$F_a = F_D + F_L + F_C \tag{9}$$
$$F_D = -\eta_a \|v_a\|^2 C_D x_a \tag{10}$$
$$F_C = -\eta_a \|v_a\|^2 C_C y_a \tag{11}$$
$$F_L = \eta_a \|v_a\|^2 C_L z_a \tag{12}$$

Where:

- C_D, C_L and C_C are respectively the drag coefficient, the lift coefficient and the crosswind coefficient,
- v_a is the air velocity (relative to the aircraft),
- $\eta_a = \frac{1}{2}\rho_a S$ with ρ_a the air density and S a characteristic surface of the aircraft.

The wind coordinate system is obtained through the attack angle α and the sideslip angle β. These angles are calculated from the air velocity and the orientation of the aircraft. The attack angle is the pitch angle between the velocity vector projected onto the plane (i, k) and the axis i. The sideslip angle is the yaw angle between the velocity vector projected onto the plane (i, j) and the axis i. In the absence of wind, the air velocity in the inertial frame is negligible compared to the velocity of the plane, so the air velocity is assumed to be the opposite of the aircraft velocity.

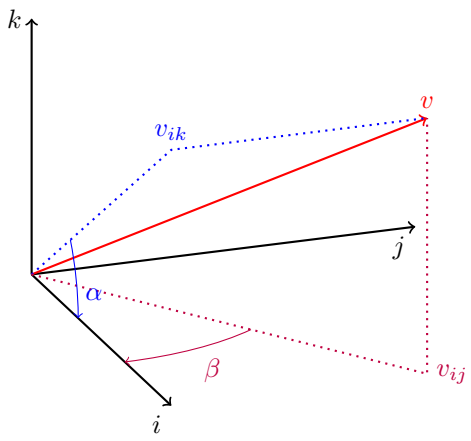

Figure 3: Graphic representation of the attack angle α and of the sideslip angle β.

Given a velocity $v = (v_i, v_j, v_k)$ in the body coordinate system, the attack and sideslip angles can be retrieved as:

$$\alpha = \arcsin\left(\frac{-v_k}{\sqrt{v_i^2 + v_j^2 + v_k^2}}\right), \beta = \arcsin\left(\frac{-v_j}{\sqrt{v_i^2 + v_j^2 + v_k^2}}\right) \tag{13}$$

Once these angles are determined, a change of coordinate system can be done through the transition matrix $P_{B\to W}$. This rotation matrix is the composition of a rotation of angle β around the axis k and a rotation of angle α around the new axis j':

$$P_{B\to W} = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix} \begin{pmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{14}$$

$$\implies P_{B\to W} = \begin{pmatrix} \cos\alpha\cos\beta & -\cos\alpha\sin\beta & \sin\alpha \\ \sin\beta & \cos\beta & 0 \\ -\sin\alpha\cos\beta & \sin\alpha\sin\beta & \cos\alpha \end{pmatrix} \tag{15}$$

Through a change of frame, we can obtain aerodynamic coefficients expressed in the body frame:

$$\begin{pmatrix} C_X \\ C_Y \\ C_Z \end{pmatrix} = P_{B\to W} \begin{pmatrix} -C_D \\ -C_C \\ C_L \end{pmatrix} \tag{16}$$

from which we can express the aerodynamic force in the body coordinate system (i, j, k):

$$F_a = F_X + F_Y + F_Z \tag{17}$$

$$F_X = \eta_a \|v\|^2 C_X i \tag{18}$$

$$F_Y = \eta_a \|v\|^2 C_Y j \tag{19}$$

$$F_Z = \eta_a \|v\|^2 C_Z k \tag{20}$$

The aerodynamic coefficients are variable through time and depend on the orientation of the aircraft with respect to the relative air velocity and on the aircraft speed. For low speed aicrafts as ours (less than 300km/h) this last effect can be neglected as a first approach [Kai, 2018].

2.5.2 Determination of drag, lift and crosswind coefficients in a steady case.

For small attack and sideslip angles ($|\alpha|, |\beta| \leq 20°$) and for an aircraft with vertical symmetry, we can consider a simple model for the variation of the aerodynamic coefficients C_D, C_L and C_C in a steady case. This model holds when control surfaces are not deflecting the air ($\delta = 0$) :

$$C_D(\alpha, \beta) \simeq C_{D0} + C_{D,\alpha^2}\alpha^2 \text{ with } C_{D,\alpha^2} = \frac{C_{L,\alpha}^2}{\pi e_{os} AR} \tag{21}$$

$$C_L(\alpha, \beta) \simeq C_{L,\alpha}\alpha \tag{22}$$

$$C_C(\alpha, \beta) \simeq C_{C,\beta}\beta \tag{23}$$

$$\tag{24}$$

where $C_{L,\alpha}$, C_{D0} and $C_{C,\beta}$ are experimentally determined constants, and with AR and e_{os} the aspect ratio and the Oswald efficiency factor, respectively. More information about the meaning of these constants and about the exact hypothesis under which these relations hold can be accessed in Kai's article [Kai, 2018].

In an unsteady case, changes of the aerodynamic coefficients are generally modelled as linear perturbations on the steady coefficients:

$$C(.) \simeq C_{st}(\alpha, \beta) + \Delta_C(\dot{\alpha}, \dot{\beta}, \omega, \dot{\omega}, \delta, \dot{\delta}) \tag{25}$$

The unsteady effects for these coefficients will be neglected in a first approach. The final expression of the aerodynamic force reads, in the global coordinate system:

$$F_a = \eta_a(v_x^2 + v_y^2 + v_z^2) \begin{pmatrix} i_1 & j_1 & k_1 \\ i_2 & j_2 & k_2 \\ i_3 & j_3 & k_3 \end{pmatrix} \begin{pmatrix} \cos\alpha\cos\beta & -\cos\alpha\sin\beta & \sin\alpha \\ \sin\beta & \cos\beta & 0 \\ -\sin\alpha\cos\beta & \sin\alpha\sin\beta & \cos\alpha \end{pmatrix} \begin{pmatrix} -C_D \\ -C_C \\ C_L \end{pmatrix} \tag{26}$$

2.6 Construction of the optimization problem

In this section, we have defined a model $\dot{X}(t) = f(X(t), U(t))$ (2) to respect, four constraints (4)-(5)-(6)-(7)-(8) and a cost $J(X, U, t_f)$ (3) that we want to minimize. Thus, our problem reads:

$$(B) \begin{cases} \text{Find the solution to } \min\limits_{X,U,t_f} J(X, U, t_f) \\ \qquad \text{Subject to } \dot{X}(t) = f(X(t), U(t)) \\ \qquad\qquad \|U_i\|_\infty < U_i^{\max} \quad \forall i \in [\![1, 4]\!] \\ \qquad\qquad U_1(t) \geq 0 \quad \forall t \in [t_0, t_f] \\ \qquad\qquad x(t_f) = x_f \\ \qquad\qquad X(t_0) = X_0 \\ \qquad\qquad \left\| \dfrac{x_1 - x_c^{1,i}}{a^i} \right\|^{p_1^i} + \left\| \dfrac{x_2 - x_c^{2,i}}{b^i} \right\|^{p_2^i} + \left\| \dfrac{x_3 - x_c^{3,i}}{c^i} \right\|^{p_3^i} \geq 1, \quad \forall i \in [\![1, N_c]\!] \end{cases} \tag{27}$$

3 Optimizing flight time and fuel consumption

We need to find a trajectory that conducts the Mini-Bee from an initial location x_0 to a final location x_f that is physically acceptable, i.e. that is a solution of the flight dynamics equation (2). We also want it to be optimal in terms of time travelled and fuel consumed. In continuous-time, this is an infinite-dimensional optimisation problem, which is very difficult to solve. Analytically, we cannot find any solution to this problem because of the complexity of the dynamics ($N_{states} = 11$ state variables and $N_{controls} = 4$ control variables in our model) and the complexity of the constraints (obstacle avoidance, limitations on attack and sideslip angles for model accuracy...). First, the infinite-dimensional problem is approximated by a finite-dimensional one with a polynomial basis, then the Legendre Pseudospectral (PS) resolution method is described, to arrive to the final optimisation problem to be solved numerically.

3.1 Approximation using polynomial series

An approach in continuous-time dynamics consists in approximating each component of the state and control with a polynomial series:

$$X_i(t) = \sum_{n=0}^{+\infty} X_{i,n} P_n(t), \quad U_i(t) = \sum_{n=0}^{+\infty} U_{i,n} P_n(t) \tag{28}$$

Assuming that the state and control are smooth enough, a good approximation can be obtained by truncating this polynomial series to the N^{th} degree for all components, obtaining their *spectral* representation:

$$\tilde{X}_i(t) = \sum_{n=0}^{N} \tilde{X}_{i,n} P_n(t), \quad \tilde{U}_i(t) = \sum_{n=0}^{N} \tilde{U}_{i,n} P_n(t) \tag{29}$$

We end up with approximate state and control, that we can express with a finite number of coefficients in the basis used for the polynomial approximation. We have now an optimization problem in finite dimension, that is easier to solve. We denote the number of control and states variables by $m = N_{states} + N_{controls} = 4 + 11 = 15$ in our model. Therefore we have $m(N+1)+1$ variables to find, without forgetting the final time t_f that is part of the optimization process.

Many polynomial bases (Laguerre, Jacobi, Chebychev, Hermite, Lagrange...) can be used to approximate a continuous function. In the case where a function is to be approximated over a bounded interval $[t_0, t_f]$, Lagrange and Jacobi bases are good choices that are very commonly used. It can be noted that Legendre and Chebychev polynomials are a specific case of Jacobi polynomials. The PS method provides the series coefficients that can be used in any polynomial basis suitable for the representation of the states and controls (Legendre, Lagrange, Chebychev...).

3.1.1 Jacobi and Legendre polynomials

Legendre polynomials are a particular case of Jacobi polynomials, among which are the Chebychev polynomials, also widely used in numerical modelling. They are mostly used for the quadrature rules they provide, because they offer excellent properties. Let us consider the functional space:

$$L^2_w([-1,1]) = \left\{ f : [-1,1] \to \mathbb{R}, \int_{-1}^{1} f(\tau)^2 w(\tau)\, d\tau < \infty \right\} \tag{30}$$

Let $\mathbb{P} \subset L^2_w([-1,1])$ be a subspace of polynomials, we define a family of orthogonal polynomials as a set $(p_i)_{i\in\mathbb{N}}$ such that:

$$p_i \in \mathbb{P}, \quad deg(p_i) = i, \text{ and } \quad i \neq j \implies < p_i, p_j >_w = 0$$

with $< p_i, p_j >_w = \int_{-1}^{1} p_i(\tau)p_j(\tau)w(\tau)\, d\tau$ the scalar product associated to the space $L^2_w([-1,1])$. $(p_i)_{i\in\mathbb{N}}$ is a Hilbert basis of the vector space $\mathbb{P}$.

Jacobi polynomials are orthogonal polynomials with respect to the weight $w(x) = (1-x)^\alpha(1+x)^\beta$, with α and β two real parameters. Legendre polynomials $(L_n)_{n\in\mathbb{N}}$ are obtained in the particular case where $\alpha = \beta = 0$, i.e. in the case $w(\tau) = 1$. They are defined on $[-1,1]$ by the recursion formula:

$$L_0(\tau) = 1, \quad L_1(\tau) = \tau, \quad (n+1)L_{n+1}(\tau) = (2n+1)\tau L_n(\tau) - nL_{n-1}(\tau) \quad \forall n \geq 2 \tag{31}$$

The problem being generally defined on a window $[t_0, t_f]$ rather than $[-1,1]$, a simple transformation is used to work on the domain $[t_0, t_f]$. The time $t \in [t_0, t_f]$ can be found from τ as $t = (\tau+1)\frac{t_f - t_0}{2} + t_0$ and reciprocally $\tau = 2\frac{t-t_0}{t_f-t_0} - 1$.

3.1.2 Discrete coefficients and Legendre-Gauss-Lobatto (LGL) nodes

In order to find a good estimation of a target function f, we need to project it onto the subspace $\mathbb{P}_N$ spanned by a polynomial orthogonal basis $(P_n)_{0\leq n\leq N}$ of N^{th}-degree polynomials. For example, we can decide to project f on the Lagrange basis $(\phi_n)_{0\leq n\leq N}$, setting $P_n = \phi_n$ or on the Legendre basis by setting $P_n = L_n$ with $(L_n)_{n\in\mathbb{N}}$ defined in the section 3.1.1. In other words, we seek the coefficients $\tilde{f}_i$ in the following decomposition:

$$\forall \tau \in [-1,1], \quad f(\tau) \simeq \tilde{f}(\tau) := \sum_{n=0}^{N} \tilde{f}_n P_n(\tau) \tag{32}$$

For an approximation using any orthogonal polynomial basis, the coefficients of the approximated function in this basis are given by:

$$\tilde{f}_n = \frac{1}{\|P_n\|^2} \int_{-1}^{1} f(\tau)P_n(\tau)w(\tau)dt \tag{33}$$

However, this integral cannot be computed exactly, which motivates the use of a Gaussian quadrature to get a good (and cheap) approximation. We choose to use a Legendre-Gauss-Lobatto (LGL) quadrature, this choice is motivated by the remarkable properties that derive from the Legendre polynomials orthogonality [Ross and Karpenko, 2012].

In effect, this quadrature allows us to include the bounds -1 and 1 as quadrature points (Figure 4). Indeed, including the boundaries will be important when formulating the optimization problem to enforce initial states and a final position. Moreover, the LGL quadrature points avoid the Runge phenomenon that can happen when approximating a function over a uniform grid with Lagrange polynomials.

The LGL quadrature points $(\tau_i^L)_{0\leq i\leq N}$ are defined as:

$$\tau_0^L = -1, \quad \tau_N^L = 1, \quad \tau_i^L \in \,]-1,1[\text{ such that } \dot{L}(\tau_i^L) = 0 \quad \forall i \in [\![1, N-1]\!] \tag{34}$$

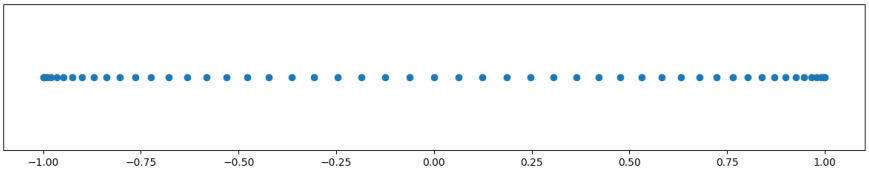

Figure 4: Legendre-Gauss-Lobato nodes for $N = 50$ (51 points). The points are sparser towards the center, and clumped on the sides.

Then, the scalar product $< f, P_n >= \int_{-1}^{1} f(\tau)P_n(\tau)w(\tau)\ d\tau$ for any $f \in L^2([-1,1])$ can be approximated with the LGL nodes by:

$$< f, P_n > \simeq \sum_{i=0}^{N} f(\tau_i^L)P_n(\tau_i^L)w_i \tag{35}$$

with $(w_i)_{0\leq i\leq N} \in \mathbb{R}$ the quadrature weights associated to the LGL nodes defined by:

$$w_i = \frac{2}{N(N+1)} \frac{1}{[L_N(\tau_i)]^2}, i \in [\![0, N]\!] \tag{36}$$

with L_N the Legendre polynomial of degree N.

This approximation is exact for polynomials of degree $2N-1$. We can then calculate the coefficients of f from (33) with a LGL quadrature, denoted here as $(\hat{f}_n)_{0\leq n\leq N}$:

$$\hat{f}_n = \frac{1}{\|P_n\|^2} \sum_{i=0}^{N} f(\tau_i^L)P_n(\tau_i^L)w_i \tag{37}$$

3.1.3 Discrete coefficients and interpolating polynomial

Let us consider a polynomial that interpolates f at the LGL nodes (e.g. a Lagrange interpolant). Expressing it in the Legendre polynomial basis and using the orthogonality of Legendre polynomials, it can be shown that the coefficients of the interpolant at the LGL nodes coincide exactly with the discrete coefficients $(\hat{f}_i)_{0\leq i\leq N}$ approximated from (37) [Gourgoulhon, 2005]. Another important note can be made regarding an approximation with Lagrange polynomials: the coefficients of the Lagrange interpolant equals the value of the function at each interpolated point, $\hat{f}_i = \hat{f}(\tau_i^L)$. However, as they are approximated with a Gaussian quadrature, so-called aliasing errors may be introduced. Therefore, an approximation of the truncated series expansion f reads, for any polynomial basis $(P_n)_{0\leq n\leq N}$ of $\mathbb{P}_N$:

$$\hat{f}(\tau) = \sum_{n=0}^{N} \hat{f}_n P_n(\tau) \simeq \tilde{f}(\tau) = \sum_{n=0}^{N} \tilde{f}_n P_n(\tau) \quad \forall \tau \in [-1,1] \tag{38}$$

In summary, for a given temporal function f defined on $[t_0, t_f]$, we achieve the following approximation:

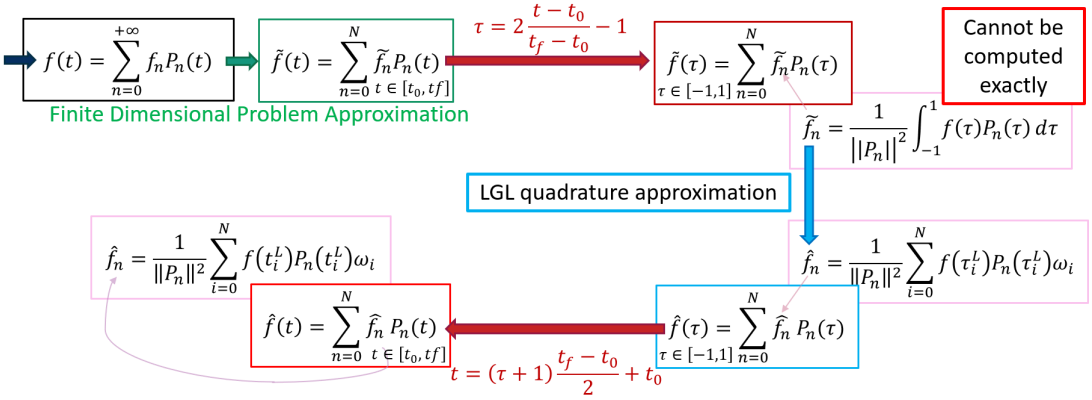

Figure 5: Summary of polynomial series approximation

3.2 Legendre Pseudospectral nonlinear problem

Now that the properties of Legendre polynomials and LGL nodes have been described, the nonlinear optimization problem can be formulated using polynomials. In our case, we use the Lagrange basis $(\phi_n)_{0\leq n\leq N}$, setting $P_n = \phi_n$. This is a common choice which allows for a simpler formulation of the problem, thanks to the properties of Lagrange polynomials (Section 3.1.3).

3.2.1 Formulation

The state and control are parameterized according to (29). Let us state the constraints that the model needs to satisfy.

First, given a certain control the parameterized state $\tilde{X}$ needs to be physically acceptable, which means that its derivative must be equal to the derivative given by the ordinary differential equation (ODE) function $f(t, \tilde{X}, \tilde{U})$ (2). In our finite-dimensional representation, this ODE constraint can be reasonably satisfied by having the derivative of the approximated state $\hat{X}$ interpolate this ODE function at the LGL nodes. For the approximation $\tilde{X}_i$ of the i$^{\text{th}}$ state component X_i, we have at the shifted (mapped to $[t_0, t_f]$) LGL nodes $(t_k^L)_{k\in[\![0,N]\!]}$ the following relation [Gong et al., 2009]:

$$\dot{X}_i(t_k^L) \simeq \dot{\tilde{X}}_i(t_k^L) \simeq \dot{\hat{X}}_i(t_k^L) = \sum_{n=0}^{N} \hat{X}_{i,n}\dot{\phi}_n(t_k^L) = \sum_{n=0}^{N} D_{kn}\hat{X}_{i,n} = \sum_{n=0}^{N} D_{kn}\hat{X}_i(t_n^L) \tag{39}$$

using $\hat{X}_{i,n} = \hat{X}_i(t_n^L)$ from section 3.1.3 and where $D = \left(\dot{\phi}_n(t_k^L)\right)_{0\leq k,n\leq N}$ is the $(N+1)\times(N+1)$ differentiation matrix, which reads:

$$D_{kn} = \frac{2}{t_f - t_0}\begin{cases} \dfrac{L_N(\tau_k)}{L_N(\tau_n)}\dfrac{1}{\tau_k - \tau_n}, & \text{if } k \neq n, \\ -\dfrac{N(N+1)}{4}, & \text{if } k = n = 0, \\ \dfrac{N(N+1)}{4}, & \text{if } k = n = N, \\ 0, & \text{otherwise} \end{cases} \tag{40}$$

To have the closest approximation of X with $\hat{X}$, all components $\hat{X}_i$ must satisfy the algebraic equality constraint:

$$\sum_{n=0}^{N} D_{kn}\hat{X}_i(t_n^L) = f\left(\hat{X}_i(t_k^L), \hat{U}_i(t_k^L)\right) \tag{41}$$

In practice, this equality constraint should be satisfied only within an acceptable margin δ, in order to facilitate the convergence of optimization methods [Gong et al., 2008]. This relaxation yields an inequality constraint:

$$\left\| \sum_{n=0}^{N} D_{kn}\hat{X}_i(t_n^L) - f\left(\hat{X}_i(t_k^L), \hat{U}_i(t_k^L)\right) \right\|_\infty \leq \delta \tag{42}$$

Second, the constraints defined in the model (4)-(5)-(6)-(7)-(8) need to be applied at every LGL node. The optimization problem is then the following:

$$(B^N) \begin{cases} \text{Find the solution to } \min\limits_{\hat{X},\hat{U},t_f} J(\hat{X},\hat{U},t_f) \\ \qquad \text{Subject to } \left\| \sum\limits_{n=0}^{N} D_{kn}\hat{X}_i(t_n^L) - f\left(\hat{X}_i(t_k^L), \hat{U}_i(t_k^L)\right) \right\|_\infty - \delta \leq 0 \\ \qquad \left\|\hat{U}_i\right\|_\infty - \hat{U}_i^{\max} < 0 \quad \forall i \in [\![1,4]\!] \\ \qquad -\hat{U}_1(t_n^L) \leq 0 \quad \forall n \in [\![0,N]\!] \\ \qquad \left\|\hat{x}_i(t_f) - x_f^i\right\|_\infty - \delta \leq 0 \\ \qquad \left\|\hat{X}_i(t_0) - X_i^0\right\|_\infty - \delta \leq 0 \\ \qquad 1 - \left\| \frac{\hat{x}_1 - x_c^{1,i}}{a^i} \right\|^{p_1^i} + \left\| \frac{\hat{x}_2 - x_c^{2,i}}{b^i} \right\|^{p_2^i} + \left\| \frac{\hat{x}_3 - x_c^{3,i}}{c^i} \right\|^{p_3^i} \leq 0, \quad \forall i \in [\![1, N_c]\!] \end{cases} \tag{43}$$

We end up with a Nonlinear Problem (NLP) that can be solved with dedicated algorithms, for example with the Interior Point Optimizer (IPOPT). Other methods such as SQP (Sequential Quadratic Programming) can be used, among which we can find the proprietary solver SNOPT (Sparse Nonlinear Optimizer) library implemented in FORTRAN.

3.2.2 Refined approximation for nonsmooth functions

Because the objects in obstacle avoidance may not be regular (i.e. with a cubic or diamond-like shape), we can expect to have an optimal control that is not smooth in some cases, particularly when the solution sought is a minimum-time trajectory. However, the non-smoothness can be localized in only a few areas (on the edge of an obstacle for example), and one can expect to have a piecewise smooth solution. In other terms, we can subdivide the entire problem in subintervals $[t_0^i, t_f^i]$ over which the solution is smooth, except at the borders $[t_0^i, t_f^i]$. In this case, instead of having a unique LGL segment from $[t_0, t_f]$ (Section 3.1.2) that forces us to have only few points in the middle where nonsmooth behaviors could potentially appear, we can subdivide the problem into N_s segments $[t_0^i, t_f^i]$, whose borders are centered on areas of interest:

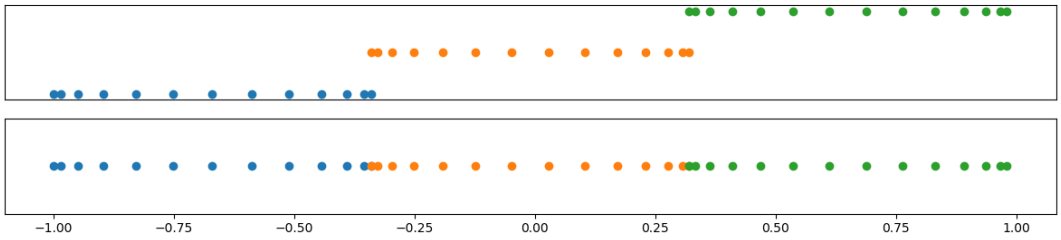

Figure 6: Legendre-Gauss-Lobato nodes for $N = 50$ (51 points) and 3 segments. The points are clumped towards points of interest (potentially non-smooth areas), and sparser in the middle of each segment (smooth area).

This refinement is more accurately described in Gong's article [Gong et al., 2008] and was not implemented in this project.

4 Results

We describe here our efforts and results to solve the optimization problem described in Section 2.4. First, we will present some early results we obtained previous to learning about the Legendre Pseudospectral (PS) method. In a second time, we will describe the algorithmic implementation of the Legendre Pseudospectral method presented in Section 3.

Unfortunately, we did not manage to get results in due time, partially due to the late discovery of this technique. Only after half of the project had passed, we discovered the PS method. Early on, we focused on indirect methods, which were the most prominent methods we knew of at the time. After a lot of attempts to find a suitable indirect method that matches our problem, we still did not find anything promising. We tried a brute-force method, but with limited results. Then, when we discovered the PS method, we had to change our approach of the problem that we initially thought to solve with neural networks. The complexity of the PS algorithm made its implementation quite hard, but still easier than most methods we had investigated so far. As we have no concrete results to show, we will instead describe our work on the algorithmic implementation of the PS technique which will have to be developped in order to obtain the core of the 3D GPS. We also present other previous methods, in particularly the brute-force technique.

Throughout this project, several approaches were attempted to solve our optimization problem. The following methods were investigated:

1. Brute-force global optimization using NOMAD,
2. Local optimization with the Legendre PS method, using:
   (a) A dedicated C++ library, PSOPT, adapted to more general use cases,
   (b) A self-written Python code interfaced with high performance C++ librairies.

The first method will be briefly described, to give some insight on its advantages and disadvantages. Then, the reasons for discarding it will be presented. The second (PS) method will be described more in-depth. In particular, we will focus on our motivations for rewriting ourselves the PS algorithm instead of using the existing C++ version, PSOPT.

4.1 Brute-force global optimization with NOMAD

NOMAD is an open-source blackbox optimization tool that uses the Mesh-Adaptive Direct Search algorithm (MADS) [Le Digabel, 2011]. It is a derivative-free algorithm, which means that no derivative information (on the cost nor on the constraints) is required in order to conduct the optimization process. It performs a global search for all parameters, based on a trial and error logic. It searches through the optimized variables using the results of the preceding calculations. NOMAD is quite easy to use, it only needs:

1. An executable that outputs the cost related to the optimization problem,
2. A configuration file that specifies the problem.

Similarly to the Legendre PS method but prior to its discovery, we parameterize the control with polynomials; however, the states are not parameterized. In order to evaluate the performance (cost) of the trajectory, we simulate a trajectory with the flight dynamics defined in (2) using a fourth-order Runge-Kutta (RK) scheme, at every optimization step.

This approach has a few differences with respect to the Pseudospectral method:

- Only the control is parameterized, through cubic splines,
- The states are calculated by performing a simulation, using the parametrized control,
- The cost derives directly from the simulated states, which respects the model differential equation.

The case we considered was a formulation of our trajectory optimization problem, without obstacle avoidance constraints.

The major issue with this approach is the computational cost associated to performing a full simulation to retrieve the state variables. The first results we got from this technique sometimes needed more than 10 hours to find a solution, and we only took a few node points. Typically, we parameterize each component of the control with a cubic spline defined by 5 points between $[t_0, t_f]$. Additionally, no information on the optimality of the solution is given, except the fact that as a global optimization algorithm the results would have high chances of being close to optimal for the chosen optimization problem. However, the optimization problem is always kept very simple (5 node points is very small), and typical results from PS approaches show that at least 15 or 20 nodes should be taken when approximating a smooth function - and this on a LGL grid which greatly improves convergence.

Because of very long runtimes and uncertainties regarding the optimality of the solution, we decided that global optimization with NOMAD does not fit well within the frame of this project.

4.2 Implementation of the Legendre Pseudospectral method

The Legendre Pseudospectral approach has already known major algorithmic developments [Gong et al., 2008], but very few open-source software implementing the method can be found. Most PS algorithms are proprietary. For example, a well-known PS algorithm developped in MATLAB called DIDO was often found to be used when looking at articles on the PS method [Xu et al., 2012]. However, we do not have access to any of those proprietary software, and the Mini-Bee being an open-source project it is preferable to use open-source tools.

4.2.1 Nonlinear solver

In order to solve the nonlinear problem formulated in (43), we use a C++ version of IPOPT, a free and open-source Interior Point algorithm. This optimizer is known to be one of the best open-source tools for nonlinear optimization. IPOPT is written in a few languages, including C++. In our project, we use a C++ to Python interface for IPOPT written by Eric Xu (Washington University). The repository is available at https://github.com/xuy/pyipopt. The interface to IPOPT has been tested and worked perfectly.

4.2.2 Automatic differentiation

Conjunctively, we use an automatic differentiation tool, named ADOL-C (Automatic Differentiation by Overloading), also written in C++, interfaced using SWIG (Simple Wrapper and Interface Generator), whose files are available in the *swig* branch of the GitLab ADOL-C repository https://gitlab.com/adol-c/adol-c/tree/swig. This module allows us to compute the derivative of any function, e.g. the cost or the constraints, with little effort and great performance. The

motivation behind the use of automatic differentiation instead of a classical finite-differences approach is a significant performance improvement and a greater accuracy.

4.2.3 PSOPT

We have found an implementation of the PS method called PSOPT written in C++ by Victor M. Becerra. It is a general PS library which allows for the definition of more sophisticated problems. It relies on ADOL-C for automatic differentiation and on IPOPT for solving the resulting nonlinear problem. For example, multi-phase problems with different controls or states from one phase to another can be specified. However, grid refinement procedures discussed in Section 3.2.2 are not implemented. This can be potentially a problem when considering sharp obstacles that would make the optimal control solution non-smooth. We compiled PSOPT from source with all its dependencies, and managed to get it working on the Windows Subsystem for Linux (WSL) - Ubuntu 18.04. However, we do not manage to get PSOPT working on version 19.10 of Ubuntu directly, because of compilation errors present in the source code. Troubleshooting on Internet suggests that the involved lines of code need to be corrected, but we do not want to touch the source code for obvious reasons. It remains a mystery as to why PSOPT worked on Ubuntu from Windows and not on a pure Ubuntu environment.

Our optimization problem is quite easy to transcript in PSOPT using examples provided by the author. We manage to get a result on a simple case without obstacle avoidance constraints, but we cannot really interpret it as it represents a very simple trajectory. With a given weight between the time cost and the fuel cost in our objective function (3), everything works fine, but when starting tweaking them or when providing different target positions, the algorithm does not converge or throws us a segmentation fault. This problem reveals to be quite frustrating, as we do not know of any way to debug it. It might be an error caused by a C++ poorly written code from our end, or a bug in PSOPT.

After investigating the problem for some time, we looked for alternatives. We did some research and found Python interfaces to those two core C++ modules, ADOL-C and IPOPT. As we are a lot more experienced in Python than in C++, going ahead and implementing the PS algorithm ourselves looked like a good idea. Furthermore, it will be a lot easier to interface with the rest of the project. With the theoretical knowledge of the Legendre PS method implementation, this should not have been a big problem once the high performance modules were interfaced and usable from Python. We will discuss some of the key challenges encountered during software development and those who remain yet unsolved in the next part.

To sum up, we decide to rewrite a PS algorithm in Python for the following reasons:

1. We have major difficulties in tweaking the C++ library PSOPT,
2. We can not determine whether the bug was in our code or in the source code,
3. Python offers a cleaner interface with other programs, particularly for an embedding within a GPS software,
4. The same level of performance can be achieved using C++ modules,
5. We have the theoretical knowledge to implement the PS method,
6. Writing code in Python allows us to focus on high level mathematical aspects rather than low-level coding aspects such as memory management.

4.3 Challenges

During the development of our PS algorithm, we faced some difficulties. Performance was at the core of the algorithmic development. We wanted to get a reasonably quick running algorithm to avoid long waiting times between results. As mentionned in Sections 4.2.1 - 4.2.2, we used existing interfaces to ADOL-C and IPOPT from C++ to Python. While IPOPT revealed very simple to interface by running a standard installation script with Python, it proved to be more difficult to get ADOL-C working. This was mainly due to parameters that we had to change in the installation script, that we found after several hours of work. Past that, both modules worked perfectly fine.

Additionally, we had to work on our knowledge of the PS method in parallel of the coding process, because of timing constraints. Thus, we had to learn about the method itself and its algorithmic implementation, which was not always explicited clearly. This led to a few misunderstandings and bugs within the program.

Even after a lot of effort, the optimization still did not converge. The algorithm seems to have a lot of difficulties in finding a solution within the bounds of our constraints. Sometimes, it converges towards an infeasible point. Therefore, we decided to test it on a smaller problem with two state variables and one control variable, but we still did not have any luck. The project ended before we had the time to debug it and make it converge. Organizing the code with a clear structure has been ultimately the focus of the project, to help detect any bugs and provide a software architecture easier to test, extend and interface with.

5 Conclusion

5.1 Project progress

We have worked on the construction of a 3D GPS for the Mini-Bee. This task is huge and complex, and the entire development could be shared with multiple research projects. In this first project, we could only focus on a short part of the advancement and lay the foundation of the future GPS through a simple case.

We have considered the Mini-Bee in a plane configuration and used plane documentation to write a physical model adapted to our problem. We have defined the different parts of the optimization problem as the control, the constraints and the cost. We were seeking to reach a final position from an initial state in a classic configuration with static obstacles, while minimizing a given feature (either flight time or fuel consumption). Other problem constraints, like moving objects, meteorological data or roads, were omitted.

Next, we have searched various methods to numerically solve the optimization problem. We had a nonlinear control problem with constraints on the final position, which have constrained the choice of resolution methods. Indirect methods were first considered but they were not applicable because of their gap with our problem or their complexity. We have finally chosen a direct method: the Legendre Pseudospectral method. This type of method had already given good results for similar problems and our approach was very promising for the Mini-Bee. Unfortunately, due to a lack of time and a difficult implementation, we have not obtained functional results within the time limit.

5.2 Remaining work

The next task of the 3D GPS construction is to complete the program that we have started. Once our programming problem is solved, we could consider developing our work to integrate all the specificities of the initial goal. First, the physical model needs to be elaborated to be adapted to an innovative aircraft like the Mini-Bee. Indeed, the Mini-Bee can fly as a helicopter or a plane, so a classic plane model is too restrictive. Moreover, the model should take into account physical phenomena, like rain.

Meteorological phenomena can also represent an avoidance constraint variable during the simulation, like moving objects do. The future program should integrate these space constraints. Furthermore, a real time aspect should be considered through the collaborative dimension desired. Finally, the specification also includes roads which can be added by different ways according to the importance given to the respect of these roads. The addition of intermediary points (possibly removable) could be considered, to offer more functionalities for the GPS application.

5.3 Project continuity

Even if we have not obtained satisfying results our work represents a good progress from the initial problem. Indeed, we have found a method adapted to the problem, optimization and differentiation libraries and a fast algorithm prototype. We have laid the foundation of a future program which could allow for a resolution in real time. With a program like that, the collision avoidance constraints and the collaborative aspect could be easily added. Furthermore, our code architecture can easily include modifications of the model, thus, once the model is elaborated, the algorithm will stay functional. Broadly speaking, we have worked while thinking about the

continuity of the project to facilitate the transition. We hope that our work will serve to the next teams and, one day, allow the Mini-Bee to flight toward multiple destinations.

A Attitude representation

There are many attitude representations, each with their own set of advantages and inconvenients. For example, we have Euler angles, that are easy to interpret and represent but that yield discontinuities and, for particular values (commonly 0 or 90 degrees) of the first or second euler angle in a chosen sequence, a singularity arises (called gimbal lock), which drops one dimension of the image of the parametrization. Quaternions, on the other hand, are numerically stable and thus usually used in computations but are more complex to represent than Euler angles.

A.1 Quaternions

Quaternions are dimension 4 elements (also called hypercomplex numbers), that are written as the following [Robert and Rolland, 2011]:

$$Q = q_0 e + q_1 i + q_2 j + q_3 k \tag{44}$$

with i, j, k such that :

$$ij = jk = ki = -ji = -kj = -ik \; ; \quad i^2 = j^2 = k^2 = -e$$

Quaternions algebra allows us to represent the cross product as defined in $\mathbb{R}^3$, characterized by the anti-commutativity described above. $q_0 e$ is called the real part of Q, while $q_1 i + q_2 j + q_3 k$ is its pure part. The pure part of a quaternion is identifiable to a vector in $\mathbb{R}^3$. By analogy with the complex numbers, we denote the conjugate of a quaternion Q by

$$\overline{Q} = q_0 e - q_1 i - q_2 j - q_3 k$$

To represent $\mathbb{R}^3$ rotations, only a certain subset of quaternions is used: the set S of norm 1 quaternions (i.e. quaternions $Q \in \mathbb{H}$ such that $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$). Those are called **attitude quaternions**. For any element Q of S, there exists $\theta \in \mathbb{R}$, $\Delta \in \mathbb{R}^3$ such that:

$$Q = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)\Delta \tag{45}$$

There is a group homomorphism between S and the set of $\mathbb{R}^3$ rotations, in other words any rotation in $\mathbb{R}^3$ can be represented by an attitude quaternion and reciprocally. An attitude quaternion that has the same kind of expression as above is associated to a rotation of axis Δ and angle θ.
This analogy between quaternions and rotations relies on the Euler theorem, that states [Avanzini, 2009] that any rotation from a given basis to another can be described as a single rotation along a unique axis common to both bases.

Quaternion derivative and angular velocity. The angular velocity of a movement is an intrisic characteristic, and consequently it does not change from one frame to another. The only difference is that its expression in different frames is usually stated via different basis. We will denote by I and B respectively a given inertial (reference) frame and a body (rotating) frame, of basis (XYZ) and (xyz).
The quaternion derivative w.r.t. time is expressed as follows [Robert and Rolland, 2011]:

$$\dot{Q} = \frac{1}{2}\overline{Q}\omega_{B/I} \quad ; \quad \omega_{B/I} = pi + qj + rk \tag{46}$$

where $\omega_{B/I}$ is expressed as a pure quaternion, that yields in the rotating basis (as $\mathbb{R}^3$ vector):

$$\omega_{B/I} = px + qy + rz \tag{47}$$

This allows us to calculate the derivative of a quaternion with respect to time by knowing its angular velocity, and vice-versa. Let us note that the equation above involves quaternions products. However we can calculate $\dot{Q} = \frac{1}{2}\overline{Q}\omega_{B/I}$ in matrix form by setting:

$$\dot{Q} = \begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2}\overline{Q}\omega_{R/I} = \frac{1}{2}M_S Q = \frac{1}{2}\begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix}\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{48}$$

Attitude quaternion and rotation matrix. Once that we determined the attitude quaternion, the transition from the body frame to the inertial frame is done via the following $P_{I\to B}$ transition matrix [Robert and Rolland, 2011]:

$$P_{I\to B} = \begin{bmatrix} 2(q_0^2+q_1^2)-1 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_2q_0) \\ 2(q_1q_2+q_0q_3) & 2(q_0^2+q_2^2)-1 & 2(q_2q_3-q_1q_0) \\ 2(q_1q_3-q_2q_0) & 2(q_2q_3+q_1q_0) & 2(q_0^2+q_3^2)-1 \end{bmatrix}$$

B Convergence rates for Legendre series approximation

The approximations $\tilde{f}$ and $\hat{f}$ using a N^{th}-order Legendre series expansion possess the following convergence properties [Gourgoulhon, 2005]:

$$\left\| f - \tilde{f} \right\|_2 \leq \frac{C_1}{N^m} \|f\|_{H^m}$$
$$\left\| f - \hat{f} \right\|_2 \leq \frac{C_2}{N^{m-\frac{1}{2}}} \|f\|_{H^m}$$

with $\|f\|_{H^m}$ the Sobolev norm of f on the interval $[-1, 1]$ defined as

$$\|f\|_{H^m} = \left(\sum_{k=0}^{m} \left\| f^{(k)} \right\|^2_{L^2([-1,1])} \right)^{\frac{1}{2}}$$

References

[Avanzini, 2009] Avanzini, G. (2009). Spacecraft attitude dynamics and control. Politecnico di Torino.

[Ferrari and Stengel, 2004] Ferrari, S. and Stengel, R. F. (2004). Online adaptive critic flight control. *Journal of Guidance, Control, and Dynamics*, 27(5):777–786.

[Gong et al., 2008] Gong, Q., Fahroo, F., and Ross, I. M. (2008). Spectral algorithm for pseudospectral methods in optimal control. *Journal of Guidance, Control, and Dynamics*, 31(3):460–471.

[Gong et al., 2009] Gong, Q., Lewis, R., and Ross, M. (2009). Pseudospectral motion planning for autonomous vehicles. *Journal of guidance, control, and dynamics*, 32(3):1039–1045.

[Gourgoulhon, 2005] Gourgoulhon, E. (2005). An introduction to polynomial interpolation.

[Han and Balakrishnan, 2002] Han, D. and Balakrishnan, S. (2002). State-constrained agile missile control with adaptive-critic-based neural networks. *IEEE Transactions on Control Systems Technology*, 10(4):481–489.

[Heydari and Balakrishnan, 2013] Heydari, A. and Balakrishnan, S. N. (2013). Fixed-final-time optimal control of nonlinear systems with terminal constraints. *Neural networks : the official journal of the International Neural Network Society*, 48:61–71.

[Kai, 2018] Kai, J.-M. (2018). Nonlinear automatic control of fixed-wing aerial vehicles. Université Côte d'Azur.

[Kim and Calise, 1997] Kim, B. S. and Calise, A. J. (1997). Nonlinear flight control using neural networks. *Journal of Guidance, Control, and Dynamics*, 20(1):26–33.

[Le Digabel, 2011] Le Digabel, S. (2011). Algorithm 909: Nomad: Nonlinear optimization with the mads algorithm. *ACM Trans. Math. Softw.*, 37(4).

[Robert and Rolland, 2011] Robert, G. and Rolland, C. (2011). Dynamique d'un satellite et quaternion d'attitude.

[Ross and Karpenko, 2012] Ross, I. M. and Karpenko, M. (2012). A review of pseudospectral optimal control: From theory to flight. *Annual Reviews in Control*, 36(2):182–197.

[Wu and Zhang, 2016] Wu, X. and Zhang, K. (2016). Constrained optimal control problems of nonlinear systems based on improved newton algorithms. In *2016 3rd International Conference on Informative and Cybernetics for Computational Social Systems (ICCSS)*, pages 229–234. IEEE.

[Xu et al., 2012] Xu, N., Kang, W., Cai, G., and Chen, B. M. (2012). Minimum-time trajectory planning for helicopter uavs using computational dynamic optimization. In *2012 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 2732–2737. IEEE.