Politecnico di Milano
School of Industrial and Information Engineering
Master of Science in Aeronautical Engineering
Department of Aerospace Science and Technology

POLITECNICO
MILANO 1863

Flatness-based trajectory generator for VTOL transition aircraft

Supervisor: Prof. Marco Lovera
Co-supervisors: Prof. Florian Holzapfel
Pranav Bhardwaj, M.Sc.

Graduation Thesis of:
Marco Capodiferro 905333
Giovanni Castelli Dezza 905678

Academic Year 2019-2020

POLITECNICO
MILANO 1863

Marco Capodiferro, Giovanni Castelli Dezza:
Flatness-based trajectory generator for VTOL transition aircraft.
|Master of Science in Aeronautical Engineering, Politecnico di Milano.
|Research in collaboration with the institute of Flight System Dynamics, Technical University of Munich.
Supervisor: Prof. Marco Lovera
Co-supervisors: Prof. Florian Holzapfel
Pranav Bhardwaj, M.Sc.

To our families.

Acknowledgements

"Talent wins games, but teamwork and intelligence wins championships."
Michael Jordan

"The greater the struggle, the more glorious the triumph."
Nick Vujicic

Firstly, we would like to thank professor Marco Lovera because he has given us the possibility to do a part of this experience abroad and has followed our work with his expertise despite the distance. We are also very grateful to professor Florian Holzapfel for the warm welcome at the FSD institute and for the opportunity to work on such an interesting topic.

A special thanks goes to our co-supervisor Pranav Bhardwaj because he has shared with us his knowledge and he has demonstrated constant availability always replying to our questions and doubts. Without him, it would have been impossible to finish our work and conclude our experience at university.

Moreover, we would like to thank our families because they have given us the possibility to attend university always being ready to support us in every situation and choice.

Last but not least, we are very thankful to all our friends because their fellowship has been too important to face all the challenges along the road.

Abstract

The research and development of Vertical Take-Off and Landing (VTOL) transition Unmanned Aerial Vehicle (UAV) configurations is one of the latest target in the aeronautical field. Aircrafts which vertically take-off and land or hover at a desired altitude and which perform a "wingborne flight" can unify the advantages of the multirotor configuration and the fixed wing one, e.g. no need of a long runway and efficiency during a flight at high speed and with long duration. Aeronautical engineers have to deal with the trajectory generation and control in the development of UAV to enable a complete autonomous flight. An approach based on flatness, which is a property of some non-linear system, can simplify the problem since the existence of flat outputs gives the possibility to write all the states and control inputs using only algebraic relationships. The aim of the thesis is to build an off-line trajectory generation algorithm based on flatness theory for the position loop of a VTOL transition aircraft. Normally, the two flight phases of this kind of aircraft need different trajectory generators and controllers while the definition of the same flat outputs for both phases allow to generate physically feasible trajectories with one unified algorithm. Four flight plans are defined to verify whether the implemented flatness-based trajectory generator fulfill all the requirements. Finally, the control problem, which is analyzed only considering the wingborne phase, is introduced in order to demonstrate the advantages that the built trajectory generator brings to the control scheme in terms of feedforward.

Sommario

La ricerca e lo sviluppo di velivoli a pilotaggio remoto con configurazione a decollo e atterraggio verticale e transizione è uno degli ultimi obiettivi nel campo aeronautico. Velivoli che decollano e atterrano verticalmente o volano a punto fisso ad una quota desiderata e che completano un volo ad alta velocità possono unificare i vantaggi della configurazione multirotore e di quella ad ala fissa, per esempio l'eliminazione della necessità di piste di decollo e atterraggio lunghe e l'efficienza durante un volo ad alta velocità e lunga durata. Gli ingegneri aeronautici devono occuparsi della generazione e del controllo della traiettoria nello sviluppo di un velivolo a pilotaggio remoto al fine di permettere un volo completamente autonomo. Un approccio basato sulla flatness, che è una proprietà di alcuni sistemi non lineari, può semplificare il problema dal momento che l'esistenza dei flat outputs fornisce la possibilità di scrivere tutti gli stati e gli input di controllo usando soltanto relazioni algebriche. Lo scopo di questa tesi è di costruire un algoritmo di generazione di traiettoria off-line basato sulla flatness per l'anello di posizione di un velivolo a decollo e atterraggio verticale con transizione. Normalmente, le due fasi di volo di questo tipo di velivolo necessitano di generatori e controllori di traiettoria diversi mentre la definizione dei medesimi flat outputs per entrambe le fasi di volo permette di generare delle traiettorie fisicamente fattibili con un unico algoritmo. Vengono definiti quattro piani di volo al fine di verificare se il generatore di traiettoria basato sulla flatness soddisfa tutti i requisiti prestabiliti. Infine, il problema del controllo, che è analizzato considerando solo la fase di volo ad alta velocità, viene introdotto per dimostrare i vantaggi che il generatore di traiettoria, implementato in questa tesi, fornisce all'intero schema di controllo, in particolare in termini di controllo in anello aperto.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The combination of the multirotors and the fixed wing configurations is the latest target of the research and development in UAV field. The classical multicopters are not the best solution for many applications, such as the inspection of linear infrastructures (highways, railways, pipelines, etc...) and the deliveries, because of their physical and technological constraints. An aircraft characterized by the possibility to vertically take-off and land or hover at a desired altitude and also to have a "wingborne flight" can unify the advantages of the two configurations, in particular:

- no need of a long runway;
- the efficiency during a flight at high speed and with long duration.

In the history of aeronautics, these ideas have been already followed in the development of manned aircraft and not only UAVs. For example, the first generation of Harrier, realized by the British manufacturing company Hawker Siddeley and reported in Fig. 1.1, is the first truly successful aircraft capable of Vertical/Short Take-Off and Landing (V/STOL) which used vectored thrust provided by a turbofan with rotating nozzle.

Figure 1.1: Hawker Siddeley Harrier GR1. Photo by [36]

The AgustaWestland AW609 and Bell Boeing V-22 Osprey, represented in Fig. 1.2a and Fig. 1.2b respectively, are two examples which achieve the transition between the hover flight and the forward one using tilt-rotors.

(a) AgustaWestland AW609. Photo by [37]

(b) Bell Boeing V-22 Osprey. Photo by [38]

Figure 1.2: Tilt-rotor aircrafts

In general, there are different technical solutions to accomplish the transition task:

- in the engine with rotating nozzle, the thrust can be directed;
- in the tilt-rotor, only the propeller-engine turns;
- in the tilt-wing, the propeller-engine and the wing turn together.

It is worth to notice that more complicated configurations, which mix the previously mentioned concepts to the standard multirotor and fixed wing, can be developed.

The Flight System Dynamics (FSD) institute of Technical University of Munich (TUM), where the thesis work was carried out, is doing research about these topics; in particular, the aircraft considered during the thesis is an electric Vertical Take-Off and Landing (eVTOL) transition aircraft, whose simple scheme is shown in Fig. 1.3.

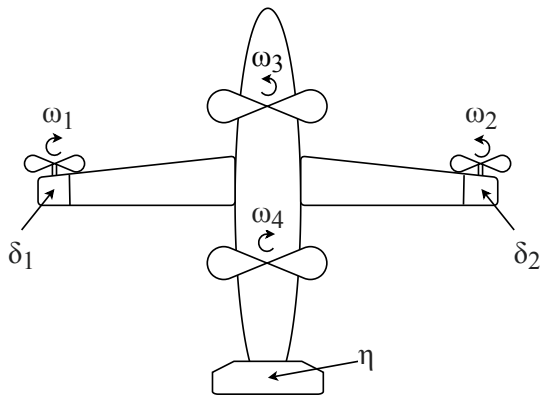

Figure 1.3: Scheme of the FSD VTOL transition configuration

It is characterized by four propellers where two of them, called main propellers, are fixed on the fuselage and the other two are tilt-rotors fixed on two moving aerodynamic surfaces, named as tilt servos. All the control effectors are listed in Table 1.1.

Control effectors		Activation	
		Wingborne	Hover
η	Elevator deflection	✓	
δ_1	Deflection tilt servo left	✓	✓
δ_2	Deflection tilt servo right	✓	✓
ω_1	Rotational rate tilt propeller left	✓	✓
ω_2	Rotational rate tilt propeller right	✓	✓
ω_3	Rotational rate main propeller front		✓
ω_4	Rotational rate main propeller rear		✓

Table 1.1: Control effectors

The names wingborne and hover are used to define the two different flight phases of a VTOL transition aircraft, namely the forward flight and the hover one respectively.

The control problem of this kind of aircraft is challenging due to the presence of the two phases and the transition between them. It is usually treated considering different controllers for wingborne and hover.

The thesis work is inserted in the already started project of the FSD institute regarding a unified control strategy based on an Incremental Nonlinear Dynamic Inversion (INDI).

1.1 Aim of the thesis

The aim of the thesis is to build an off-line trajectory generation algorithm based on the flatness theory for VTOL transition aircrafts.

The idea consists in generating physically feasible trajectories for the position loop of the hover flight phase and the wingborne flight phase with one unified algorithm which provides the entire flight path before the flight starts. In particular, the trajectory generation module shall have kinematic speed and waypoint list (with all related properties) as inputs. The trajectory generation module must produce outputs which can be followed by the controller in all flight phases.

The approach based on flatness, which is a property of some non-linear system, allows to build a trajectory generator which provides only the flat outputs. If it is possible to find the same flat outputs for both phases of VTOL transition aircraft, an unique algorithm can generate trajectories for any kind of maneuver. In addition, the study of all the available types of trajectories is fundamental in the construction of the algorithm so that the trajectory generator can fulfill all the requirements. Specifically, the flight path has to be 3D and continuous in

position, speed and acceleration; moreover, the absolute value of speed must be constant and equal to 25 m/s during the wingborne phase. Finally, the maximum value of the track rate $\dot{\chi}$ and the tangential acceleration acc_{tang} must be lower or equal to 10 deg/s and 2 m/s^2 respectively.

The advantage of flatness property is related also to the complete control scheme because the flatness relationships between the flat outputs and the states and control inputs can be used in several way. For example, the feedforward control can be implemented without the need to solve any differential equations but only algebraic ones.

The software used to perform all the practical work is `MATLAB & Simulink`.

1.2 Thesis overview

A brief outline of the thesis is reported below.

Chapter 2 provides the literature review concerning the topic of the thesis, which is divided in four paragraphs regarding the flatness theory, the VTOL transition dynamics, the trajectory generation and the flatness-based trajectory control in order to be as clear as possible.

Chapter 3 introduces the flatness theory and its application to simple systems. Then, the mathematical model for both phases of the considered aircraft is presented and finally, the flatness property of the system is proved with the choice of the proper flat outputs.

Chapter 4 explains the trajectory generation problem both from a general point of view and from the aeronautical one. Subsequently, the chosen approach between all the possibilities is motivated and lastly, the `Simulink` model of the trajectory generator is described.

Chapter 5 focuses on the results and their analysis of the trajectory generator for four different flight plans. The latter ones are defined in order to verify whether all the requirements can be satisfied.

Chapter 6 examines the simulation of the complete control scheme composed by the trajectory generator, the trajectory controller and the simple VTOL model only for the wingborne phase. It is important to underline that the model is really simple because it has not been possible to use the high fidelity one already existing at the FSD institute because of Covid-19.

Chapter 7 shows all the reached conclusions and the possible further developments.

Chapter 2

State of the art

The VTOL transition aircraft is a very innovative machine and research about all its aspects, such as the dynamic modeling, the control system, etc ..., is growing fast as mentioned in Chapter 1. The literature still provides a few number of papers regarding the VTOL aircraft and, all the more reason, the application of flatness theory to solve its control problem is not already taken into account in a deep way.

Considering the lack of specific material, the authors decide to face the problem step-by-step which means that all the aspects of the thesis are studied and analyzed in order to have a complete overview of the problem. Firstly, an in-depth knowledge of flatness theory is necessary, then the second step is to study the VTOL dynamics. The third step consists in going through the problem of trajectory generation and its resolution thanks to flatness. Since the advantages of flatness can be seen not only by the trajectory generation point of view but also by the control one, it is useful to understand which are the pros in a complete control scheme and in particular for what concerns the feedforward control part, despite the main aim of this thesis is to build a flatness-based trajectory generator.

The literature is divided in paragraphs considering which is the most interesting aspect treated in any paper although some of them regard two or more topics.

Flatness theory

The definition of flatness property for a nonlinear system, all the mathematical formalism and the related properties are presented and discussed in [1]. Among them, the interesting concept of defect δ is used in the case of a not-completely flat system to mark the distance to flatness. As already mentioned, the request is to demonstrate if the VTOL satisfies the flatness condition and which are the flat outputs; if it is not possible, the solution is to consider the VTOL as a system with a certain defect from flatness. In Chapter 3, it will be proved that the chosen VTOL mathematical model is completely flat and so, the defect is useless in this thesis.

In [1], there are also some practical examples, both for flat and non-flat systems, but the most useful ones to understand the theory with an engineering point of view are shown in [2]. Considering a DC motor, two different control laws based on a classical PID controller, where the first one has a step speed reference and the other has a flatness-based reference trajectory, are tested to demonstrate the advantages of the theory during the transient phase. For what concerns the linear motor with oscillating masses, it is evident that a flatness-based control scheme provides better results than a standard one in terms of oscillations after the displacement of the motor. These examples prove that it is possible to generate and follow fast trajectories with high accuracy positioning, using poor actuators and sensors.

The example of a generic aircraft is provided in [2] and [3]. Firstly, the flatness property of the system is proved and then the application in the design of the autopilot is presented; in particular, considering the case of a steady turn, it is demonstrated how the flatness approach solves the problem of discontinuity at the junction point of the straight line and the curve.

VTOL transition aircraft dynamics

The dynamic modeling and flatness analysis of a convertible aircraft is studied in [4]. First of all, a description of the VTOL is carried out specifying eleven control inputs and the two different flight phases, namely the take-off/landing/hover and the fast forward flight; under the hypothesis of fault-free conditions, only five control inputs are identified for each phase while the other six ones are considered redundant. Subsequently, the nonlinear model of the aircraft and the formulae of the forces and moments are presented. The standard set of equations for a conventional airplane is shown; in addition, vertical Euler angles, explained in Section 7.6.2 of [5], are introduced for the hover phase in order to avoid a gimbal lock problem for γ close to 90 *deg*. The consequence is that there are two different models for the two different phases. The last passage of the paper is the choice of flat outputs and their mathematical demonstration; since there are five control inputs, the flat outputs are five: $x(t)$, $y(t)$, $z(t)$, $\alpha(t)$ and $\beta(t)$.

A novel model of the so-called Tilt-Rotor VTOL is presented in [6]; the structure, based on a blended wing equipped with a tilt-rotor, gives the possibility to vertically take-off/land and to have the transition to a forward flight at high speed. A good characterization of the three operational modes of a VTOL aircraft is provided by [7].

1. Hover Flight (HF): *"the 3D vehicle's motion relies only on the rotors. Within this phase the vehicle features VTOL flight profile. The controller for this regime disregard the aerodynamic terms due to the negligible translational speed".*

2. Slow-Forward Flight (SFF): *"it is possible to distinguish an intermediate operation mode, the SFF, which links the two flight conditions, HF and FFF. This is probably the most complex dynamics".*

3. Fast-Forward Flight (FFF): *"FFF regime mode (Aft position), at this flight mode the aircraft has gained enough speed to generate aerodynamic forces to lift and control the vehicle motion"*.

A representation is reported in Fig. 2.1.

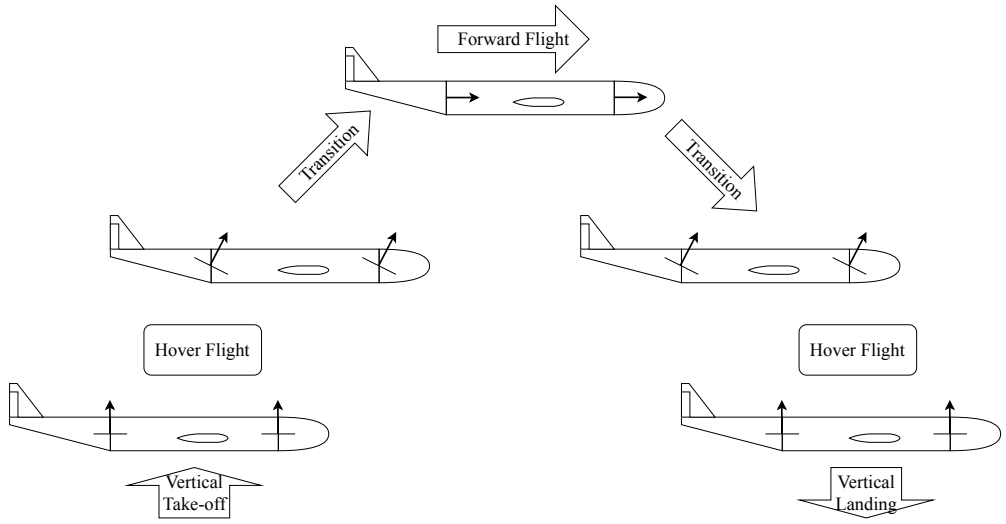

Figure 2.1: Operational modes of a VTOL aircraft

Regarding the first mode, a complete dynamical model is provided by the paper [8]. It is worth to notice that the dynamics of the hover phase is similar to the one of a quadcopter while, for what concerns the wingborne phase, the VTOL can be seen as a conventional airplane. For these reasons, the wingborne phase is described by the equations of forward flight reported in [3] and in [4] while, since the use of vertical Euler angles in the latter one is not so advantageous, the decision is to face the hover phase taking into consideration the quadcopter dynamics.

A lot of research on the quadcopter has been done in the past years. A mathematical model under a precise set of hypotheses is proposed by [9]; the equations of motions are found thanks to a Newton-Euler method and listed in the cited paper. A similar set of equations, relying on the reported assumptions, is presented also in [10] where a flatness-based approach is used to deal with the position control block. The choice and the demonstration of the flat outputs, namely $\mathbf{w} = [x, y, z]^T$, is inherent with the needs of this thesis.

Trajectory generation

A characterization of the flat outputs for a linear time-invariant controllable system is reported in [11] and the application on the trajectory planning is provided; knowing the relationships between flat outputs and the states and inputs, *"it suffices to generate a polynomial trajectory for y with respect to time to generate displacements of the stage from one steady state to another one"*. A list of

flatness-based trajectory generators applied to the aeronautical field is reported below.

- The quadrotor desired path used in the control scheme of [10] is provided by a real-time trajectory generator explained in [12].

- A similar work is carried out in [13]; an optimization function on the integral of the norm square of the snap (fourth derivative of the position) is added to find an optimal trajectory connecting a sequence of points in the 3D space. Other constraints, such as on velocities or to remain in a specified corridor, can be imposed.

- A possible application to military field consists in the differential flatness-based optimal maneuver generation algorithm for one-to-one aerial combat games, treated in [14]. It is worth to notice that the flat outputs of the fixed wing aircraft 2D model are x and y and that B-spline functions are used to parameterize them.

- A trackable reference trajectory for the quadrotor aerial vehicle is presented in [15]. In this case, the flat outputs are x, y, z and ψ; it recalls the choice of [10]. The trajectory generator, providing only the function of flat outputs in time, can determine all the other states, namely the components of speed, the Euler angles and rates, and the control inputs.

Another contribution to the potentialities of the trajectory generator is given by the couple of papers [16] and [17] despite they do not involve flatness theory. This method is *"not only able to compute offline trajectories, but is also capable of controlling position, velocity, and acceleration of the Micro Air Vehicle (MAV) in real-time"*; for example, the latter feature allows to generate trajectories from an inadmissible state to a permissible one.

In order to discover more aspects on flatness-based trajectory generation, also examples, which do not regard the aeronautical field, are taken into account and listed below.

- The ballbot, presented in [18], is *"an omnidirectional, dynamically stable mobile robot. It is a human-sized robot that balances on a single spherical wheel"*. The mathematical model is based on two degrees of freedom, namely the lean angle ϕ and the ball angle θ, and one control input, the torque between ball and body. The flat output has a physical meaning because it is a linear combination of the lean and ball angles despite it is not part of the state vector. Since it is required a C^4 continuity, ten conditions need to be imposed and, as a consequence, the trajectory has to be generated with a ninth degree polynomial. Differently from the quadcopter in [13], the minimization function is defined on the fifth derivative of the flat output, named crackle. The last interesting aspect is the possibility to generate recovery trajectories and not only rest-to-rest motions.

- Flatness applied to the trajectory generator of the battery state of charge for Hybrid Electric Vehicle (HEV) is presented in [19]. The first interesting part of the paper consists in the choice and the demonstration of the flat output y_f, which is battery State of Charge (SoC). The second one regards the use of B-spline functions of third degree for the parameterization of y_f.
- An overview of a complete control scheme for a pumped storage power station using flatness is shown in [20]. In particular, the trajectory planning problem is developed choosing appropriate flat outputs and smooth trajectories as spline ansatz. Moreover, a flatness-based trajectory generator allows to build a dynamic feedforward control which has greater advantages than the static one, which is normally used.
- In [21], an analytical offline multi-point trajectory generation scheme for differentially flat systems is explained in a general way. A 3-DoF gantry crane, whose flat outputs are the payload positions x_p, y_p and z_p, is considered as example.

In order to have a complete overview on all the possible smooth trajectories used in robotics, the authors refer to the book [22].

Flatness-based trajectory control

Considering the 2-DoF control scheme shown in Fig. 2.2, [23] proposes a new approach to the feedforward Σ_{FF} based on flatness property for the transition between stationary setpoints of nonlinear SISO systems. The most interesting aspect is the remark on the differences between a flat approach and the standard inversion-based ones. In particular, the first one brings to a purely algebraic design of the feedforward control while the other ones are more challenging and need to solve partial differential equations.

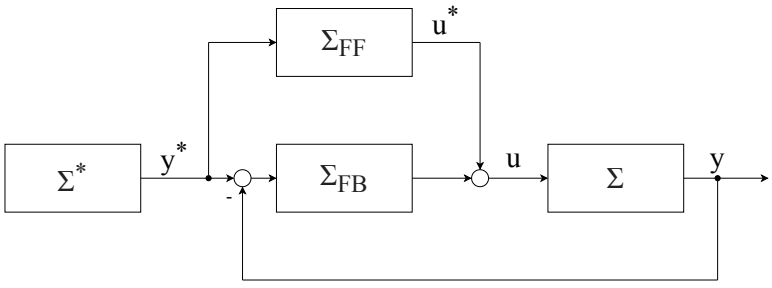

Figure 2.2: Structure of the 2-DoF control scheme

The problem of output tracking based on flatness theory for a Planar Vertical Take-Off and Landing (PVTOL) aircraft is treated in a complete way in [24]. After having presented its mathematical model based on the three degrees of freedom (x, z and θ), the procedure to get the flat outputs is explained; it is worth to notice that the latter ones are not part of the state but they have a physical

meaning. After that, the problem of trajectory generation and control is analyzed, simulated and compared with a conventional design. In addition, three different types of inversion-based feedforward for nonminimum-phase PVTOL aircraft, are examined and compared in [25].

Recalling the quadcopter example in [10], the exact feedforward linearization, presented in [26], is used in order to design the controller. According to [26], *"the overall control law consists of two parts: the feedforward signal based on differential flatness, which steers the system when being on the desired trajectory, and a feedback control part, which forces the system to converge to the desired trajectory"*. The expression of the new control input to be used in the flatness relationships is

$$v_i = \dot{\xi}^*_{i,k_i} + \Lambda_i(\mathbf{e}) = v_i^* + \Lambda_i(\mathbf{e}), \qquad i \in \{1, \ldots, m\}, \tag{2.1}$$

where $\mathbf{e} = \xi - \xi^*$, v_i^* is the desired trajectory and $\Lambda_i(\mathbf{e})$ is the feedback part which can be any type of control. In [10] and in [26] the latter one is an extended PID controller. The SISO case is deeply discussed in [27]. Since only stability results are reported in the previous papers, the robustness with respect to parametric uncertainty is analyzed in [28]. Together with exact feedforward linearization, other possible control schemes based on flatness are presented in Section 5.2 of [29].

Considering the conventional airplane of [3], a single universal controller can be designed, i.e. *"a controller able to make the aircraft track any trajectory without reconfiguration of the variables on which we close the loop and without modification of the gains and other parameters involved in the feedback loop"*. The outer position loop of the helicopter in [30] satisfies the flatness property and so, after having defined the flat outputs x, y, z and ψ, the outer controller, built under flatness considerations, generates a desired trajectory for the inner system.

All the previously mentioned works have been really useful to the authors in order to understand the different aspects of the thesis topic. In particular, some of them have a greater importance to develop each specific part and so, they will be recalled in the next chapters.

Chapter 3

Flatness theory and mathematical model

The flatness theory can be a very useful approach to the control problem of a nonlinear system. It was developed by Jean Levine but a lot of researchers summarized in their works and papers its main aspects and possible applications to different systems, as shown in Chapter 2. The purpose of this chapter is to explain which is the concept of flatness, seen by an engineering point of view, and which are the most important advantages of this theory for trajectory generation and control [Section 3.1]. An overview of the used reference frames is reported in Section 3.2. Moreover, a mathematical model of the specific problem, namely the VTOL aircraft and so both wingborne and hover phase, is presented in Section 3.3, leading to the flat outputs choice and the demonstration of the considered system flatness, which are shown in Section 3.4.

3.1 Brief explanation of flatness

A nonlinear system $\dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t))$ with time $t \in \mathbb{R}$, state $\mathbf{x}(t) \in \mathbb{R}^n$ and input $\mathbf{u}(t) \in \mathbb{R}^m$, is said to be (*differentially*) *flat* if there exists a set of m differentially independent variables $\mathbf{w} = [w_1, ..., w_m]^T$, called *flat outputs*, such that:

$$\begin{aligned} \mathbf{w} &= G(\mathbf{x}, \mathbf{u}, \dot{\mathbf{u}}, ..., \mathbf{u}^{(\delta)}) \\ \mathbf{x} &= f_x(\mathbf{w}, \dot{\mathbf{w}}, ..., \mathbf{w}^{(\rho)}) \\ \mathbf{u} &= f_u(\mathbf{w}, \dot{\mathbf{w}}, ..., \mathbf{w}^{(\rho+1)}) \end{aligned} \tag{3.1}$$

where G, f_x and f_u are smooth functions of their arguments, at least in an open subset of their domain, and δ, ρ are the maximum orders of derivatives of u and w needed to describe the system ([1] and [10]).

The usefulness of this property is provided by the fact that it is possible to write all the variables in dependence of flat outputs and finite number of their derivatives; an interesting remark would be that there are no more distinctions

between states, inputs and outputs. The application to trajectory generation and control problem is evident for the following reasons:

- the outputs of trajectory generation block are only the flat ones, which can be defined as different kind of splines (polynomial, trigonometric, exponential, etc);

- the feedforward control is realized only knowing algebraic relationships between control inputs and flat outputs. In the feedback part, the states are constructed knowing the measurements and so, having previously defined the relationships between flat outputs and states, it is simple to obtain the actual flat outputs and compute the error with respect to the reference values. A consequence is that there are no differential equations to be solved.

3.1.1 Examples of flat systems

The following examples, taken from [2], are reported to show the flatness property of two simple systems in order to clarify the theoretical concepts previously explained.

Mass-spring system: considering the system in Fig. 3.1:

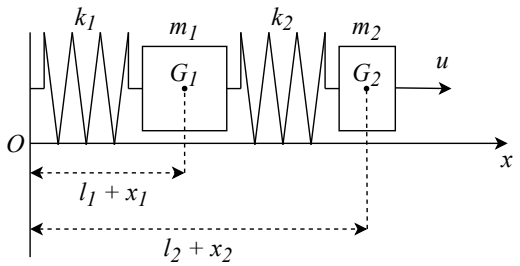

Figure 3.1: Mass-spring system

- m_1 and m_2 are the masses of the two bodies;

- k_1 and k_2 are the springs stiffness;

- G_1 and G_2 are the gravity centres;

- l_1 and l_2 are the equilibrium positions of G_1 and G_2;

- $\gamma_1(\dot{x}_1)$ and $\gamma_2(\dot{x}_2)$ are the viscous frictions and they are non negative, twice continuously differentiable and $\gamma_1(0) = \gamma_2(0) = 0$;

- u is the force applied to G_2.

The dynamics equations are

$$\begin{aligned} m_1\ddot{x}_1 + k_1 x_1 + \gamma_1(\dot{x}_1) &= k_2(x_2 - x_1) \\ m_2\ddot{x}_2 + k_2(x_2 - x_1) + \gamma_2(\dot{x}_2) &= u. \end{aligned} \tag{3.2}$$

In order to prove that the flat output is x_1, it is necessary to demonstrate that x_2 and u can be expressed only through x_1 and its derivatives. Starting from the first equation of Eq. (3.2), the expression of x_2 is found:

$$x_2 = \frac{1}{k_2}\left(m_1\ddot{x}_1 + (k_1 + k_2)\, x_1 + \gamma_1(\dot{x}_1)\right). \tag{3.3}$$

Then, differentiating the latter one, its derivative is

$$\dot{x}_2 = \frac{1}{k_2}\left(m_1 x_1^{(3)} + (k_1 + k_2)\,\dot{x}_1 + \gamma_1'(\dot{x}_1)\,\ddot{x}_1\right). \tag{3.4}$$

and, using the second equation of Eq. (3.2) and the expressions of x_2 and $\dot{x}_2$, u is written as

$$\begin{aligned} u =& \frac{m_1 m_2}{k_2} x_1^{(4)} + \left(\frac{m_2 k_1}{k_2} + m_2 + m_1\right)\ddot{x}_1 + k_1 x_1 + \gamma_1(\dot{x}_1) + \dots \\ &+ \frac{m_2}{k_2}\left(\gamma_1''(\dot{x}_1)\,(\ddot{x}_1)^2 + \gamma_1'(\dot{x}_1)\, x_1^{(3)}\right) + \dots \\ &+ \gamma_2\left(\frac{m_1}{k_2} x_1^{(3)} + \frac{1}{k_2}\left((k_1 + k_2)\,\dot{x}_1 + \gamma_1'(\dot{x}_1)\,\ddot{x}_1\right)\right). \end{aligned} \tag{3.5}$$

Eq. (3.3) and Eq. (3.5) prove that the states and the control inputs can be expressed as functions of x_1 and its derivatives until the fourth order. In conclusion, the system is flat with x_1 as flat output.

Non holonomic vehicle: considering the vehicle with four wheels which roll without slipping on the horizontal plane in Fig. 3.2:

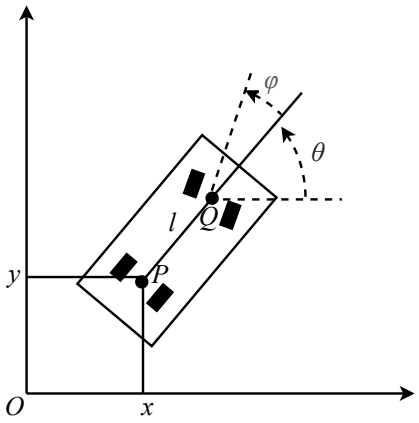

Figure 3.2: Non holonomic vehicle

- (x, y) are the coordinates of the point P;
- P and Q are the middle points of the rear axle and the front one respectively;
- l is the distance between P and Q;
- θ is the angle between the longitudinal axis of the vehicle and the x-axis;
- φ is the angle of the front wheels.

This system is called *non holonomic vehicle* because the rolling without slipping condition is a *non holonomic constraint*.

The dynamics equations are

$$\begin{aligned} \dot{x} &= u\cos\theta \\ \dot{y} &= u\sin\theta \\ \dot{\theta} &= \frac{u}{l}\tan\varphi, \end{aligned} \tag{3.6}$$

where the modulus of the car speed u and the angle φ are the control variables.

The system is flat with x and y as flat outputs. The demonstration is the following one; considering the first and the second equation of Eq. (3.6), Eq. (3.7) is obtained dividing them as

$$\tan\theta = \frac{\dot{y}}{\dot{x}}, \tag{3.7}$$

while Eq. (3.8) is reached summing the squares of them:

$$u^2 = \dot{x}^2 + \dot{y}^2. \tag{3.8}$$

Differentiating Eq. (3.7), the expression of $\dot{\theta}$ is

$$\dot{\theta}\left(1+\tan^2\theta\right) = \frac{\ddot{y}\dot{x} - \dot{y}\ddot{x}}{\dot{x}^2} \qquad \rightarrow \qquad \dot{\theta} = \frac{\ddot{y}\dot{x} - \dot{y}\ddot{x}}{\dot{x}^2 + \dot{y}^2}. \tag{3.9}$$

Inserting the square root of Eq. (3.8) and Eq. (3.9) in the third equation of Eq. (3.6), it is deduced that:

$$\tan\varphi = \frac{l\dot{\theta}}{u} = l\frac{\ddot{y}\dot{x} - \dot{y}\ddot{x}}{\left(\dot{x}^2 + \dot{y}^2\right)^{3/2}}. \tag{3.10}$$

In conclusion, all the states and control inputs can be expressed as functions of x and y and their derivatives until the second order.

3.2 Reference frames

Before proceeding with the definition of the VTOL mathematical model and the flatness analysis of the problem, a brief recap of all the reference frames used in the thesis is reported in the next paragraphs. The information are basic knowledge of aerospace engineering but, to be as complete as possible, the authors refer to [31] and [32].

WGS84 coordinates: the WGS84 is a geodetic system based on a reference ellipsoid and defined in 1984. Its features are listed below:

- the origin coincides with the Earth center of mass;
- the x-axis lies in the equatorial plane and it is directed to the Greenwich meridian;
- the y-axis lies in the equatorial plane and it is obtained in order to form a right-hand system with the x-axis and z-axis;
- the z-axis is the rotation axis of the Earth and it points to the North Pole.

The geographical coordinates of a point are the following ones.

- **Latitude** (lat)**:** angle, measured in the meridian plane, between the equatorial plane and the parallel passing through the point.
- **Longitude** ($long$)**:** angle, measured in the equatorial plane, between Greenwich meridian and the meridian plane of the point.
- **Altitude** (h)**:** height above the ellipsoid.

ECEF frame: the characterization of the ECEF frame is identical to the one previously explained for the WGS84 ellipsoid. The position of an object in the space is defined thanks to the x, y and z coordinates; it is obvious that the center of the Earth corresponds to $(0, 0, 0)$ coordinates. It is possible to pass from the WGS84 coordinates to the ones in ECEF frame with the formulas

$$\begin{aligned} x &= (N(lat) + h)\cos(lat)\cos(long) \\ y &= (N(lat) + h)\cos(lat)\sin(long) \\ z &= \left(\frac{b^2}{a^2}N(lat) + h\right)\sin(lat), \end{aligned} \tag{3.11}$$

where $N(lat) = \frac{a}{\sqrt{1-e^2\sin(lat)^2}}$, a is the semi-major axis, b is the semi-minor axis and e is the eccentricity of the ellipsoid.

Local NED frame - O frame: the local NED frame (named O frame) is characterized by the following features.

- The origin is a reference point. In this thesis, it is fixed as the position of the FSD Institute since all the flight plans are set in the proximity of this place;
- the x-axis is parallel to the local geoid surface and it is directed to the north direction.
- The y-axis is parallel to the local geoid surface and it is directed to the east direction.

- The z-axis is perpendicular to the local geoid surface and it is directed downwards.

The local NED and ECEF coordinates are related by

$$\mathbf{r_{NED}} = R^T \left(\mathbf{r_{ECEF}} - \mathbf{r_{ref}} \right), \tag{3.12}$$

where

$$R = \begin{bmatrix} -\sin(lat)\cos(long) & -\sin(long) & -\cos(lat)\cos(long) \\ -\sin(lat)\sin(long) & \cos(long) & -\cos(lat)\sin(long) \\ \cos(lat) & 0 & -\sin(lat) \end{bmatrix}. \tag{3.13}$$

Kinematic frame - K frame: the kinematic frame (named K frame) is defined in the following way:

- the origin is the aircraft reference point;
- the x-axis is aligned with the kinematic speed and it points to its direction;
- the y-axis is directed to the right and it forms a right-hand system with the x-axis and z-axis;
- the z-axis is parallel to the projection of local surface normal of the WGS84 ellipsoid into a plane perpendicular to the x-axis and it points downwards.

The rotation matrix to pass from the K frame to the O frame is

$$M_{OK} = \begin{bmatrix} \cos\chi\cos\gamma & -\sin\chi & \cos\chi\sin\gamma \\ \sin\chi\cos\gamma & \cos\chi & \sin\chi\sin\gamma \\ -\sin\gamma & 0 & \cos\gamma \end{bmatrix}, \tag{3.14}$$

where χ and γ are the track and climb angles respectively.

Rotated kinematic frame - $\bar{\mathbf{K}}$ frame: rotating the K frame by the bank angle, the rotated kinematic frame (named $\bar{K}$ frame) is obtained. The rotation matrix is defined as

$$\mathbf{M}_{\bar{K}K} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & \sin\mu \\ 0 & -\sin\mu & \cos\mu \end{bmatrix}, \tag{3.15}$$

where μ is the bank angle.

Body-fixed frame - B frame: the body-fixed frame (named B frame) is characterized by the fact that:

- the origin is the aircraft reference point;
- the x-axis lies in the aircraft symmetry plane and points towards the aircraft nose;

- the y-axis is directed towards the right wing and it forms a right-hand system with the x-axis and the z-axis;
- the z-axis lies in the aircraft symmetry plane; it points downwards and it is perpendicular to the x-axis and the y-axis.

Aerodynamic frame - A frame: the aerodynamic frame (named A frame) is defined in the following way:

- the origin is the aircraft reference point;
- the x-axis is aligned with the aerodynamic speed and points to its direction;
- the y-axis is directed towards the right and it forms a right-hand system with the x-axis and the z-axis;
- the z-axis points downwards; it is parallel to the projection of the local surface normal of the WGS84 ellipsoid into a plane perpendicular to the x-axis.

3.3 Mathematical model

The flatness analysis is performed only on the position loop because the attitude one has already been treated in a different way. In general, the procedure explained below can be carried out for a complete dynamic model of a VTOL aircraft.

The mathematical models, regarding only position information and not attitude ones, are obtained both for the wingborne phase and hover one in Section 3.3.1 and Section 3.3.2 respectively. The hypothesis for the VTOL aircraft modeling are:

- constant mass equal to m;
- rigid body and so no aerolastic effects are present;
- constant gravity acceleration, equal to $g = 9.81\ m/s^2$ and so the Center of Mass (CoM) coincides with the Center of Gravity (CG);
- zero wind condition.

3.3.1 Wingborne phase

Firstly, the complete set of equations for a generic aircraft is presented using as reference Eq. (3) of [3] and Eq. (28.1 - 28.10) of [4]. It is worth to notice that the translational motion is expressed in the aerodynamic frame A while the rotational one is written in the body frame B.

$$\dot{x} = V_A^a \cos\chi \cos\gamma$$

$$
\begin{aligned}
\dot{y} &= V_A^a \sin\chi\cos\gamma \\
\dot{z} &= -V_A^a \sin\gamma \\
\dot{V}_A^a &= \frac{X_A}{m} - g\sin\gamma \\
\dot{\beta} &= p\sin\alpha - r\cos\alpha + \frac{mg\cos\gamma\sin\mu + Y_A}{mV_A^a} \\
\dot{\alpha} &= q - (p\cos\alpha + r\sin\alpha)\tan\beta + \frac{g}{V_A^a}\frac{\cos\gamma\cos\mu}{\cos\beta} + \frac{Z_A}{mV_A^a\cos\beta} \\
\dot{\chi} &= \frac{-Z_A\sin\mu + Y_A\cos\mu}{mV_A^a\cos\gamma} \\
\dot{\gamma} &= \frac{-mg\cos\gamma - Y_A\sin\mu - Z_A\cos\mu}{mV_A^a} \\
\dot{\mu} &= \frac{-g\cos\mu\cos\gamma\sin\beta}{V_A^a\cos\beta} + \frac{p\cos\alpha + r\sin\alpha}{\cos\beta} - \frac{Z_A\sin\beta}{mV_A^a\cos\beta} + \dots \\
&\quad + \frac{(Y_A\cos\mu - Z_A\sin\mu)\sin\gamma}{mV_A^a\cos\gamma} \\
\dot{p} &= \frac{(I_{xz}(I_{xx} - I_{yy} + I_{zz})\,p - (I_{xz}^2 - I_{zz}(I_{yy} - I_{zz}))\,r)\,q + I_{xz}N_B + I_{zz}L_B}{I_{xx}I_{zz} - I_{xz}^2} \\
\dot{q} &= \frac{-I_{xz}p^2 - r(I_{xx} - I_{zz})\,p + I_{xz}r^2 + M_B}{I_{yy}} \\
\dot{r} &= \frac{((I_{xz}^2 + I_{xx}(I_{xx} - I_{yy}))\,p - I_{xz}(I_{xx} - I_{yy} + I_{zz})\,r)\,q + I_{xx}N_B + I_{xz}L_B}{I_{xx}I_{zz} - I_{xz}^2}
\end{aligned}
\tag{3.16}
$$

where:

- x, y and z are the position coordinates of the CoM in the O frame;
- V_A^a is the aerodynamic speed;
- α and β are the aerodynamic angles, namely the angle of attack and the side-slip angle respectively;
- χ, γ and μ are the track, the climb and the bank angles;
- p, q and r are the roll, the pitch and the yaw rates;
- X_A, Y_A and Z_A are the forces in the A frame while L_B, M_B and N_B are the moments in the B frame;
- I_{ij} are the inertial moments.

As mentioned before, only a sub-set of Eq. (3.16) is selected to describe the position with respect to time. The useful set of equations is

$$
\begin{aligned}
\dot{x} &= V_A^a\cos\chi\cos\gamma \\
\dot{y} &= V_A^a\sin\chi\cos\gamma
\end{aligned}
$$

$$\begin{aligned}
\dot{z} &= -V_A^a \sin\gamma \\
\dot{V}_A^a &= \frac{X_A}{m} - g\sin\gamma \\
\dot{\chi} &= \frac{-Z_A \sin\mu + Y_A \cos\mu}{mV_A^a \cos\gamma} \\
\dot{\gamma} &= \frac{-mg\cos\gamma - Y_A \sin\mu - Z_A \cos\mu}{mV_A^a},
\end{aligned} \tag{3.17}$$

where $\mathbf{x} = [x, y, z, V_A^a, \chi, \gamma]^T$ is the states vector and $\mathbf{u} = [X_A, Y_A, Z_A]^T$ is the control inputs vector.

In order to have a well-posed problem, an expression of μ, taken from Eq. 4.8 of [31], needs to be added as

$$\mu = \arctan\frac{V_A^a \dot{\chi}}{g}. \tag{3.18}$$

3.3.2 Hover phase

During the hover phase, the γ angle is close to 90 *deg* and Eq. (3.16) can not be used because a gimbal lock problem is introduced. For this reason, another set of equations must be used. Since the hover phase is a typical dynamics of a quadcopter, its complete set of equations, taken from [10], is considered. These equations are derived with respect to the O and B reference frames.

$$\begin{aligned}
\ddot{x} &= U_1 \frac{(\cos\psi\cos\phi\sin\theta + \sin\psi\sin\phi)}{m} \\
\ddot{y} &= U_1 \frac{(\sin\psi\cos\phi\sin\theta - \sin\phi\cos\psi)}{m} \\
\ddot{z} &= U_1 \frac{(\cos\theta\cos\phi)}{m} - g \\
\dot{p} &= \frac{(I_y - I_z)}{I_x} qr + \frac{1}{I_x} U_2 - \frac{J_m}{I_x} q\Omega_R \\
\dot{q} &= \frac{(I_z - I_x)}{I_y} pr + \frac{1}{I_y} U_3 + \frac{J_m}{I_y} p\Omega_R \\
\dot{r} &= \frac{(I_x - I_y)}{I_z} pq + \frac{d}{I_z} U_4 \\
\dot{\phi} &= p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \\
\dot{\theta} &= q\cos\phi - r\sin\phi \\
\dot{\psi} &= \frac{\sin\phi}{\cos\theta} q + \frac{\cos\phi}{\cos\theta} r
\end{aligned} \tag{3.19}$$

where:

- x, y and z are the position coordinates of the CoM in the O frame;
- ϕ, θ and ψ are the roll, the pitch and the yaw Euler angles;

- p, q and r are the attitude rates;
- U_1, U_2, U_3 and U_4 are the inputs and are defined in terms of motor angular rates Ω_i;
- I_i are the body principal moments of inertia;
- J_m is the inertia motor;
- $\Omega_R = -\Omega_1 - \Omega_3 + \Omega_2 + \Omega_4$.

Since it is required to describe only the position, the first three equations are selected.

In the following approach, the aircraft during the hover phase is treated as a point mass, simplifying the equations. As a matter of fact, θ, ϕ, ψ and U_1 loose their meaning and, as done for the wingborne phase, the control inputs are the three components of the force.

Considering the first three equation of Eq. (3.19) and taking into account all the above considerations, the useful equations are

$$\begin{aligned} \dot{x} &= v_x \\ \dot{v}_x &= \frac{X_O}{m} \\ \dot{y} &= v_y \\ \dot{v}_y &= \frac{Y_O}{m} \\ \dot{z} &= v_z \\ \dot{v}_z &= \frac{Z_O}{m} + g, \end{aligned} \tag{3.20}$$

where $\mathbf{x} = [x, v_x, y, v_y, z, v_z]^T$ is the states vector, $\mathbf{u} = [X_O, Y_O, Z_O]^T$ is the control inputs vector and it is important to notice that also the identity equations are added for the seek of completeness.

The unique problem could be that there are no information about the yaw angle in this kind of system but the attitude controller is able to deal with it during the hover phase and, in particular, during the initial tracking of the first waypoint.

3.4 Flat outputs choice

The following remarks need to be taken into account during the choice of flat outputs:

- the number of flat outputs has to be equal to the one of control inputs;
- it is not necessary that they have a physical meaning;

- the simplest procedure to find the flat outputs is to start from states or quantities which have a particular physical meaning and, after having chosen them, provide the mathematical demonstration.

The number of control inputs is equal to three for both phases; as explained before, the number of flat outputs has to be the same.

The decision is taken following these considerations. It is already known that χ and γ are available flat outputs for a conventional airplane and so they could be chosen also for the wingborne phase of a VTOL aircraft. The problem is that these angles are useless to describe the hover phase since it is already shown that they are not present in its mathematical model of Eq. (3.20). In [10], x, y and z are the flat outputs for the quadcopter position controller while in [4], they are part of flat outputs vector. Taking into account these suggestions and referring to Eq. (3.20), it is simple to notice that the same flat outputs can be chosen. For what concerns Eq. (3.17), the same quantities appear in the formulation and they could be valid flat outputs.

For the reasons listed above, the idea of this thesis is to choose the flat outputs vector as

$$\mathbf{w} = [w_1, w_2, w_3] = [x, y, z]^T . \tag{3.21}$$

It is very simple to demonstrate that $\mathbf{w}$ is a function of the states vector for both phases:

- wingborne phase

$$\mathbf{w} = G(\mathbf{x}) = G\mathbf{x}, \qquad G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}; \tag{3.22}$$

- hover phase

$$\mathbf{w} = G(\mathbf{x}) = G\mathbf{x}, \qquad G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{3.23}$$

3.4.1 Mathematical demonstration

The mathematical demonstration is reported below for both phases.

3.4.1.1 Wingborne phase

The first step consists in substituting the expression of μ, using Eq. (3.18) in Eq. (3.17). By dividing the second and the first equations of Eq. (3.17) as

$$\frac{\dot{y}}{\dot{x}} = \tan\chi \quad \rightarrow \quad \chi = \arctan\frac{\dot{y}}{\dot{x}} = \arctan\frac{\dot{w}_2}{\dot{w}_1}. \tag{3.24}$$

Then, the division between the third and the second equations of Eq. (3.17) provides

$$\frac{\dot{z}}{\dot{y}} = -\frac{\tan\gamma}{\sin\chi}. \tag{3.25}$$

Inserting Eq. (3.24) in Eq. (3.25),

$$\frac{\dot{z}}{\dot{y}} = -\frac{\tan\gamma}{\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)} \tag{3.26}$$

and the expression of γ is

$$\gamma = \arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right) = \arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right). \tag{3.27}$$

Now, considering the first equation of Eq. (3.17) and substituting the relationships previously found, it is possible to write

$$\begin{aligned} V_A^a &= \frac{\dot{x}}{\cos\chi\cos\gamma} \\ &= \frac{\dot{x}}{\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right)} \\ &= \frac{\dot{w}_1}{\cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\cos\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right)}. \end{aligned} \tag{3.28}$$

Having demonstrated that the states χ, γ and V_A^a can be written in dependence of flat outputs and their derivatives, the next passage is to find the relationships for the control inputs X_A, Y_A and Z_A. The X_A force is found using the fourth equation of Eq. (3.17):

$$X_A = m\left(\dot{V}_A^a + g\sin\gamma\right) \tag{3.29}$$

where $\dot{V_A}^a$ can be computed following the normal procedure for the derivative of a composed function. In order to simplify the notation, the following expressions dependent on flat outputs are re-called as

$$\begin{aligned} A &= f_A(\mathbf{w}) = \sin\left(\arctan\frac{V_A^a\dot{\chi}}{g}\right), \\ B &= f_B(\mathbf{w}) = \cos\left(\arctan\frac{V_A^a\dot{\chi}}{g}\right), \\ C &= f_C(\mathbf{w}) = mV_A^a\dot{\gamma} + mg\cos\gamma, \\ D &= f_D(\mathbf{w}) = mV_A^a\dot{\chi}\cos\gamma, \end{aligned} \tag{3.30}$$

so that the fifth and the sixth equations of Eq. (3.17) become

$$\begin{aligned} D &= BY_A - AZ_A, \\ C &= -AY_A - BZ_A. \end{aligned} \tag{3.31}$$

They are manipulated to obtain the expression of Y_A and Z_A. From the first equation, Y_A in dependence of Z_A is found as

$$Y_A = \frac{D + AZ_A}{B} \tag{3.32}$$

and then it is inserted in the second one remembering that $A^2 + B^2 = \sin^2\mu + \cos^2\mu = 1$:

$$C = -A\frac{D + AZ_A}{B} - BZ_A \quad \rightarrow \quad Z_A = -\frac{DA + BC}{A^2 + B^2} = -DA - BC. \tag{3.33}$$

Finally, the expression of Z_A is substituted in Eq. (3.32) of Y_A:

$$\begin{aligned} Y_A &= \frac{D - DA^2 - BCA}{B} = \frac{D\left(1 - A^2\right) - BCA}{B} = \frac{DB^2 - BCA}{B} \\ &= DB - CA. \end{aligned} \tag{3.34}$$

To write the entire mathematical expression of control input forces in dependence of the flat outputs and their derivatives, it is necessary to compute the derivatives of χ, γ and V_A^a as

$$\begin{aligned} \dot{\chi} &= \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2 + \dot{y}^2}, \\ \dot{\gamma} &= \frac{\dot{z}\ddot{y}\sin\chi - \dot{y}\left(\ddot{z}\sin\chi + \dot{z}\dot{\chi}\cos\chi\right)}{\dot{y}^2 + \dot{z}^2\sin^2\chi}, \\ \dot{V}_A^a &= \frac{\ddot{x}\cos\chi\cos\gamma + \dot{x}\left(\dot{\chi}\sin\chi\cos\gamma + \dot{\gamma}\cos\chi\sin\gamma\right)}{\cos^2\chi\cos^2\gamma}. \end{aligned} \tag{3.35}$$

Substituting the expressions of A, B, C and D (reported in Eq. (3.30)) in Eq. (3.29), Eq. (3.33) and Eq. (3.34), it is possible to find the relationships of the forces in dependence of χ, γ, V_A^a and their derivatives. Knowing the formulae Eq. (3.24), Eq. (3.27), Eq. (3.28) and Eq. (3.35), the forces written using only the flat outputs and their derivatives are obtained. The complete expressions are shown below:

$$\begin{aligned} X_A =& m\Bigg\{ \frac{\ddot{x}\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right) + \dots}{\cos^2\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos^2\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right)} \\ &+\dot{x}\Bigg[\left(\frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2}\right)\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right) + \dots \\ &+\left(\frac{\dot{z}\ddot{y}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)-\dot{y}\left(\ddot{z}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)+\dot{z}\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\left(\frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2}\right)\right)}{\dot{y}^2+\dot{z}^2\sin^2\left(\arctan\frac{\dot{y}}{\dot{x}}\right)}\right)\cdot\dots \\ &\cdot\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\sin\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right)\Bigg] + \dots \end{aligned}$$

$$
\begin{aligned}
&+ g \sin\left[\arctan\left(-\frac{\dot{z}}{\dot{y}} \sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right] \Bigg\} = \\
=&m\Bigg\{ \frac{\ddot{w}_1 \cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right) \cos\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right) + \dots}{\cos^2\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\cos^2\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right)} \\
&+\dot{w}_1 \Bigg[\left(\tfrac{\dot{w}_1\ddot{w}_2 - \dot{w}_2\ddot{w}_1}{\dot{w}_1^2+\dot{w}_2^2}\right) \sin\left(\arctan\tfrac{\dot{w}_2}{\dot{w}_1}\right) \cos\left(\arctan\left(-\tfrac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\tfrac{\dot{w}_2}{\dot{w}_1}\right)\right)\right) + \dots \\
&+ \left(\frac{\dot{w}_3\ddot{w}_2 \sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right) - \dot{w}_2\left(\ddot{w}_3 \sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right) + \dot{w}_3 \cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\left(\frac{\dot{w}_1\ddot{w}_2-\dot{w}_2\ddot{w}_1}{\dot{w}_1^2+\dot{w}_2^2}\right)\right)}{\dot{w}_2^2+\dot{w}_3^2\sin^2\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)} \right) \cdot \dots \\
&\cdot \cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right) \sin\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right) \Bigg] + \dots \\
&+ g \sin\left[\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right] \Bigg\};
\end{aligned}
\tag{3.36}
$$

$$
\begin{aligned}
Y_A =&m\Bigg\{ \frac{\dot{x}}{\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right)} \frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2} \cdot \dots \\
&\cdot \cos\left[\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right] \cos\left[\arctan\left(\dots \right.\right. \\
&\left.\left. \frac{\dot{x}}{g\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right)} \frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2} \right)\right] + \dots \\
&- \Bigg[\frac{\dot{x}}{\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right)} \cdot \dots \\
&\cdot \frac{\dot{z}\ddot{y}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right) - \dot{y}\left(\ddot{z}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right) + \dot{z}\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\left(\frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2}\right)\right)}{\dot{y}^2+\dot{z}^2\sin^2\left(\arctan\frac{\dot{y}}{\dot{x}}\right)} + \dots \\
&+ g\cos\left[\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right] \Bigg] \sin\left[\arctan\left(\dots \right.\right. \\
&\left.\left. \frac{\dot{x}}{g\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right)} \frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2} \right)\right] \Bigg\} = \\
=&m\Bigg\{ \frac{\dot{w}_1}{\cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\cos\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right)} \frac{\dot{w}_1\ddot{w}_2-\dot{w}_2\ddot{w}_1}{\dot{w}_1^2+\dot{w}_2^2} \cdot \dots \\
&\cdot \cos\left[\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right] \cos\left[\arctan\left(\dots\right.\right.
\end{aligned}
\tag{3.37}
$$

$$
\begin{aligned}
&\frac{\dot{w}_1}{g\cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\cos\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right)}\frac{\dot{w}_1\ddot{w}_2-\dot{w}_2\ddot{w}_1}{\dot{w}_1^2+\dot{w}_2^2}\Bigg)\Bigg]+\dots\\
&-\left[\frac{\dot{w}_1}{\cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\cos\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right)}\cdot\dots\right.\\
&\cdot\frac{\dot{w}_3\ddot{w}_2\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)+\dots}{\dot{w}_2^2+\dot{w}_3^2\sin^2\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)}\\
&\frac{-\dot{w}_2\left(\ddot{w}_3\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)+\dot{w}_3\cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\left(\frac{\dot{w}_1\ddot{w}_2-\dot{w}_2\ddot{w}_1}{\dot{w}_1^2+\dot{w}_2^2}\right)\right)}{+g\cos\left[\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right]}+\dots\Bigg]\sin\left[\arctan\left(\dots\right.\right.\\
&\left.\left.\frac{\dot{w}_1}{g\cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\cos\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right)}\frac{\dot{w}_1\ddot{w}_2-\dot{w}_2\ddot{w}_1}{\dot{w}_1^2+\dot{w}_2^2}\right)\right]\Bigg\};\\
Z_A=&-m\Bigg\{\frac{\dot{x}}{\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right)}\frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2}\cdot\dots\\
&\cdot\cos\left[\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right]\sin\left[\arctan\left(\dots\right.\right.\\
&\left.\left.\frac{\dot{x}}{g\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right)}\frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2}\right)\right]+\dots\\
&+\left[\frac{\dot{x}}{\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right)}\cdot\dots\right.\\
&\cdot\frac{\dot{z}\ddot{y}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)-\dot{y}\left(\ddot{z}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)+\dot{z}\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\left(\frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2}\right)\right)}{\dot{y}^2+\dot{z}^2\sin^2\left(\arctan\frac{\dot{y}}{\dot{x}}\right)}+\dots\\
&+g\cos\left[\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right]\Bigg]\cos\left[\arctan\left(\dots\right.\right.\\
&\left.\left.\frac{\dot{x}}{g\cos\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\cos\left(\arctan\left(-\frac{\dot{z}}{\dot{y}}\sin\left(\arctan\frac{\dot{y}}{\dot{x}}\right)\right)\right)}\frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2}\right)\right]\Bigg\}=\\
=&-m\Bigg\{\frac{\dot{w}_1}{\cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\cos\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right)}\frac{\dot{w}_1\ddot{w}_2-\dot{w}_2\ddot{w}_1}{\dot{w}_1^2+\dot{w}_2^2}\cdot\dots\\
&\cdot\cos\left[\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right]\sin\left[\arctan\left(\dots\right.\right.\\
&\left.\left.\frac{\dot{w}_1}{g\cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\cos\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right)}\frac{\dot{w}_1\ddot{w}_2-\dot{w}_2\ddot{w}_1}{\dot{w}_1^2+\dot{w}_2^2}\right)\right]+\dots
\end{aligned}
\tag{3.38}
$$

$$
+\left[\frac{\dot{w}_1}{\cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\cos\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right)}\cdot\ldots\right.
$$

$$
\cdot\frac{\dot{w}_3\ddot{w}_2\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)+\ldots}{\dot{w}_2^2+\dot{w}_3^2\sin^2\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)}
$$

$$
\frac{-\dot{w}_2\left(\ddot{w}_3\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)+\dot{w}_3\cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\left(\frac{\dot{w}_1\ddot{w}_2-\dot{w}_2\ddot{w}_1}{\dot{w}_1^2+\dot{w}_2^2}\right)\right)}{}+\ldots
$$

$$
\left.+g\cos\left[\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right]\right]\cos\left[\arctan\left(\ldots\right.\right.
$$

$$
\left.\left.\frac{\dot{w}_1}{g\cos\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\cos\left(\arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\left(\arctan\frac{\dot{w}_2}{\dot{w}_1}\right)\right)\right)}\frac{\dot{w}_1\ddot{w}_2-\dot{w}_2\ddot{w}_1}{\dot{w}_1^2+\dot{w}_2^2}\right)\right]\right\}.
$$

These equations are reported only for the sake of completeness but they are not used during the implementation because the forces can be found through all the other states.

In conclusion, this section has demonstrated that all the states and control inputs satisfy the property of flatness and that the derivatives maximum order is two.

3.4.1.2 Hover phase

Starting from Eq. (3.20), the demonstration is immediate and really simple with respect to the case treated above:

$$
\begin{aligned}
&v_x = \dot{x} = \dot{w}_1,\\
&v_y = \dot{y} = \dot{w}_2,\\
&v_z = \dot{z} = \dot{w}_3,\\
&X_O = m\dot{v}_x = m\ddot{w}_1,\\
&Y_O = m\dot{v}_y = m\ddot{w}_2,\\
&Z_O = m\left(\dot{v}_z - g\right) = m\left(\ddot{w}_3 - g\right).
\end{aligned}
\tag{3.39}
$$

It is therefore evident that the states vector $\mathbf{x}$ and the control inputs vector $\mathbf{u}$ can be expressed only through the flat outputs vector $\mathbf{w}$ and its derivatives up to the second order.

A final remark is that the use of flatness for the hover phase may seem silly because the equations are very easy to solve; despite that, the interesting aspect is the possibility to use the same flat outputs for both phases; the advantage is that the trajectory generation block, providing only $\mathbf{w} = [x, y, z]^T$, could work for all VTOL dynamics and, as a consequence, the controller is simpler to realize.

3.4.2 Final relationships

The control inputs are needed in the kinematic frame K.

For what concerns the wingborne phase, the equations provide control inputs in the aerodynamic frame. In zero wind condition, the A frame coincides with the rotated kinematic frame $\bar{K}$ so that the results must be rotated of the μ angle with the matrix of Eq. (3.15).

Regarding the hover phase, the results are in the O frame and the rotation into the K frame is not treated in this thesis because it will be done in the inner loop.

The following scheme collects the useful expressions of each quantity which are used in the control system and, after that, the rotations of the control inputs are shown.

3.4.2.1 Wingborne phase

- States:

$$\begin{aligned} x &= w_1 \\ y &= w_2 \\ z &= w_3 \\ V_A^a &= \frac{\dot{w}_1}{\cos\chi\cos\gamma} \\ \chi &= \arctan\frac{\dot{w}_2}{\dot{w}_1} \\ \gamma &= \arctan\left(-\frac{\dot{w}_3}{\dot{w}_2}\sin\chi\right); \end{aligned} \tag{3.40}$$

- Control inputs:

$$\begin{aligned} X_{\bar{K}} &= X_A = m\left(\dot{V}_A^a + g\sin\gamma\right) \\ Y_{\bar{K}} &= Y_A = m\left(V_A^a\dot{\chi}\cos\gamma\cos\mu - (V_A^a\dot{\gamma} + g\cos\gamma)\sin\mu\right) \\ Z_{\bar{K}} &= Z_A = -m\left(V_A^a\dot{\chi}\cos\gamma\sin\mu + (V_A^a\dot{\gamma} + g\cos\gamma)\cos\mu\right); \end{aligned} \tag{3.41}$$

- Rotation from $\bar{K}$ to K frame: it is a 2D rotation around the x-axis. In matrix form, the expression is

$$\mathbf{F}_K = \mathbf{M}_{K\bar{K}}\mathbf{F}_{\bar{K}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\mu & -\sin\mu \\ 0 & \sin\mu & \cos\mu \end{bmatrix}\mathbf{F}_{\bar{K}}. \tag{3.42}$$

The relationships of the forces in K frame with respect to the ones in $\bar{K}$ frame are

$$\begin{aligned} X_K &= X_{\bar{K}} \\ Y_K &= Y_{\bar{K}}\cos\mu - Z_{\bar{K}}\sin\mu \\ Z_K &= Y_{\bar{K}}\sin\mu + Z_{\bar{K}}\cos\mu. \end{aligned} \tag{3.43}$$

3.4.2.2 Hover phase

- States:

$$\begin{aligned} x &= w_1 \\ v_x &= \dot{w}_1 \\ y &= w_2 \\ v_y &= \dot{w}_2 \\ z &= w_3 \\ v_z &= \dot{w}_3; \end{aligned} \tag{3.44}$$

- Control inputs:

$$\begin{aligned} X_O &= m\ddot{w}_1 \\ Y_O &= m\ddot{w}_2 \\ Z_O &= m\left(\ddot{w}_3 - g\right). \end{aligned} \tag{3.45}$$

Chapter 4

Trajectory generation

This chapter deals with the trajectory generation problem. Firstly, Section 4.1 provides an overview for a generic control system; a categorization of the already existing kinds of trajectories, used in many environments of robotics, is proposed. Section 4.2 analyzes the topic from an aeronautical point of view; all the fundamentals of flight path generation are listed, starting from the concept of fixes and leg types and arriving to the necessity of transitions to have the correct smoothness of the path. Considering all the types of function that can be used and are presented in Section 4.1 and taking into account the aeronautical requirements, the followed approach is motivated in Section 4.3. Finally, Section 4.4 describes in a deeper way all the model realized by the authors thanks to the `MATLAB & Simulink` software.

4.1 Outline of trajectory generation problem

Generally, in a control system the trajectory generator provides the setpoint, namely the reference trajectory which has to be followed by the aircraft and is used by the feedback controller to compute the error with respect to the measurements. This control system block has the aim to find the relationships between the time and the quantities chosen as outputs. In this case, it is realized under a flatness-based approach, so the problem consists only in finding the relationships between the time and the flat outputs because all the variables of the system, namely the states and the control inputs, can be written in dependence of flat outputs and a finite number of their derivatives. A simple scheme of the trajectory generation block is reported in Fig. 4.1 where it is possible to notice that the flat outputs are x, y and z, coordinates of the 3D position.

All the information explained in this section are mainly taken from the book *"Trajectory Planning for Automatic Machines and Robots"* of L. Biagiotti and C. Melchiorri [22]; the useful topics for this thesis are then elaborated. The part related to the combination of straight-lines and circular-arcs is explained in the PhD dissertation of Volker Schneider [31].

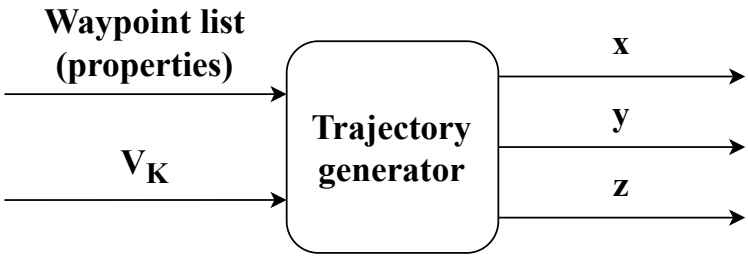

Figure 4.1: Trajectory generation block

4.1.1 Trajectory categories

As shown in Fig. 4.2, the main distinction among all the trajectory categories is between one- and multi-dimensional; the first one is related to a system characterized by only one DoF while the second one concerns a system in a multi-dimensional space.

Another classification is based on the fact that a trajectory can be defined from the initial and final points only, called point-to-point, or with also a list of intermediate points between the initial and final ones; in the latter case, known as multi-point trajectory, all the points have to be interpolated or approximated. In the interpolation, the curve crosses the given points for all the values of time while in the approximation the curve does not pass exactly through the points but an error with a prescribed tolerance is present.

Both point-to-point and multi-point trajectories are used to obtain a complex motion. With the former ones, the path is calculated by joining several point-to-point trajectories which are individually optimized, considering the initial and final boundary conditions on velocity, acceleration, etc... and the constraints on their maximum values. Conversely, in the case of multi-point trajectories, by specifying the intermediate points, the path is found as the solution of a global optimization problem which depends on the conditions imposed on each via-point and on the overall profile.

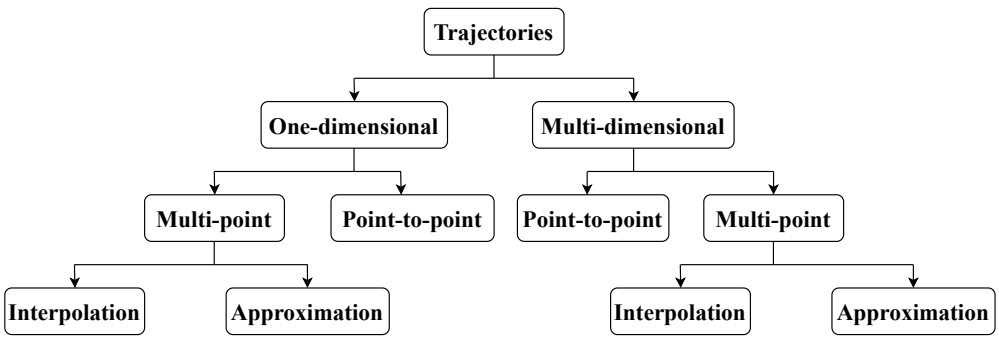

Figure 4.2: Main trajectory categories

4.1.2 Point-to-point trajectories

In Section 4.1.1, the use of several point-to-point trajectories to generate a complex path is introduced. The problem is to find a function $q(t)$, $t \in [t_0, t_1]$ which satisfies conditions on position, velocity, acceleration and also their successive derivatives at t_0, t_1. Since the constraints can be of any kind in a wide spectrum, it is fundamental to fix them in order to make a choice of the most suitable trajectory. Contemporaneously, a description of the characteristics of each type of trajectory, from the most elementary to the more complex one, is needed.

4.1.2.1 Polynomial trajectories

Polynomial trajectories are defined as

$$q(t) = a_0 + a_1 t + a_2 t^2 + ... + a_n t^n, \tag{4.1}$$

where the $n+1$ coefficients a_i are determined imposing the initial and final conditions, not only on the trajectory but also on its time derivatives (velocity, acceleration, jerk, snap, etc...). For this reason, the degree n of the polynomial is related to the number of conditions that have to be imposed and to the smoothness of the motion. The number of constraints is usually even and consequently the degree n is odd, i.e three, five, seven and so on.

A brief description of different polynomial functions is presented below.

Linear trajectories:

$$q(t) = a_0 + a_1 (t - t_0) \tag{4.2}$$

where coefficients a_0 and a_1 are computed imposing positions q_0 and q_1. A characteristic of this trajectory is the constant velocity.

Parabolic trajectories: realized as the composition of two second degree polynomials, one in the interval $[t_0, t_f]$ and the other one in $[t_f, t_1]$ where t_f is the flex point. The main characteristic is that the absolute value of the acceleration is constant with opposite sign in the two segments.

$$\begin{aligned} q_a(t) &= a_0 + a_1 (t - t_0) + a_2 (t - t_0)^2, \qquad t \in [t_0, t_f] \\ q_d(t) &= a_3 + a_4 (t - t_f) + a_5 (t - t_f)^2, \qquad t \in [t_f, t_1]. \end{aligned} \tag{4.3}$$

In order to obtain the coefficients a_i, the conditions are the positions q_0, q_f and the velocity v_0 for the function $q_a(t)$ while for the function $q_d(t)$ the conditions are the positions q_f, q_1 and the velocity v_1.

Cubic trajectories:

$$q(t) = a_0 + a_1(t - t_0) + a_2(t - t_0)^2 + a_3(t - t_0)^3, \quad t_0 \le t \le t_1 \tag{4.4}$$

where coefficients a_0, a_1, a_2 and a_3 are determined imposing position and velocity at t_0 and t_1.

Fifth degree polynomials: the necessity of having not only continuous position and velocity profiles but also acceleration leads to

$$q(t) = a_0 + a_1(t-t_0) + a_2(t-t_0)^2 a_3(t-t_0)^3 + a_4(t-t_0)^4 + a_5(t-t_0)^5, \quad (4.5)$$

where coefficients a_0, a_1, a_2, a_3, a_4 and a_5 are obtained imposing six conditions on initial and final position, velocity and acceleration. The result is a smoother profile than the cubic one which is already a smooth one.

Seventh degree polynomials: in particular cases, the request of smooth profile can lead to the definition of a high degree polynomial, such as seven or higher, if necessary.

$$\begin{aligned} q(t) = &a_0 + a_1(t-t_0) + a_2(t-t_0)^2 a_3(t-t_0)^3 + a_4(t-t_0)^4 + \dots \\ &+ a_5(t-t_0)^5 + a_6(t-t_0)^6 + a_7(t-t_0)^7 \end{aligned} \quad (4.6)$$

where coefficients a_0, a_1, a_2, a_3, a_4, a_5, a_6 and a_7 are found using eight conditions.

Section 4.1.3 deals with polynomial functions in the case of multi-point trajectories and so, more details are provided.

4.1.2.2 Trigonometric trajectories

Trigonometric trajectories are characterized by non-null continuous derivatives for any order of derivation in the interval $[t_0, t_1]$; in t_0 and t_1, the derivatives may be discontinuous. There are three different analytical expressions, namely harmonic, cycloidal and elliptic motion, which have specific characteristics in terms of curve profiles, such as acceleration and jerk.

In this section, they are not taken into account in a deeper way since they are not useful in aeronautics.

4.1.2.3 Exponential trajectories

Since discontinuities in the trajectory may cause vibrations and oscillations in the machine, the use of exponential trajectories may be convenient in order to adjust smoothness. The mathematical formulation of exponential function for velocity is

$$\dot{q}(\tau) = v_c e^{\sigma f(\tau,\lambda)}, \quad (4.7)$$

where σ and λ are free parameters and the possible choices for the function $f(\tau, \lambda)$ are

$$\begin{aligned} f_a(\tau,\lambda) &= \frac{(2\tau)^2}{\left|1-(2\tau)^2\right|^\lambda}, \\ f_b(\tau,\lambda) &= \frac{\sin \pi\tau^2}{\left|\cos \pi\tau\right|^\lambda}. \end{aligned} \quad (4.8)$$

The most important decision concerns σ and λ since their values can be assigned in order to minimize the maximum amplitude of the high frequency components of the acceleration profile, which are related to the vibrations induced in the machine. Conversely, the choice of f_a and f_b has not so great importance on the motion profile.

4.1.2.4 Combination of elementary trajectories

The functions explained in the pages before can be combined together in order to satisfy particular requests; for example, it could be useful to define a continuous function not only with continuous derivatives but also with other features, such as minimum values for the maximum acceleration and jerk. The most common ones are the trapezoidal velocity and the double S trajectories.

A detailed explanation is provided only for the last one which is used to build the trajectory generator.

Trajectories with double S velocity profile: this kind of trajectory is constructed adopting a continuous, linear piece-wise, acceleration profile. It is assumed that:

$$j_{min} = -j_{max}, \qquad a_{min} = -a_{max}, \qquad v_{min} = -v_{max}, \tag{4.9}$$

where j_{min}, j_{max}, a_{min}, a_{max}, v_{min} and v_{max} are respectively the minimum and maximum values of jerk, acceleration and velocity.

The aim is to plan a trajectory connecting q_0 and q_1 that, when it is possible, reaches the maximum or minimum values for jerk, acceleration and velocity, so that the total duration T is minimized.

For the sake of simplicity, only the case with $q_1 > q_0$ is considered. The boundary conditions are:

- initial and final velocities v_0 and v_1,
- initial and final accelerations a_0 and a_1, both set to zero.

Three phases can be distinguished.

1. Acceleration phase ($t \in [0, T_a]$): the acceleration has a linear profile from the zero initial value to the maximum one, then it keeps a constant value and finally it goes back to zero.

2. Maximum velocity phase ($t \in [T_a, T_a + T_v]$): the acceleration is equal to zero and a constant velocity is maintained.

3. Deceleration phase ($t \in [T_a + T_v, T]$): remembering that $T = T_a + T_v + T_d$, the profiles are opposite with respect to the acceleration phase.

First of all, it is important to verify whether a trajectory can be performed satisfying the constraints of Eq. (4.9). For example, the distance between q_0 and q_1 may be too small in order to change velocity between initial and final values, considering the constraints on maximum acceleration and jerk.

The conditions to be satisfied in order to have a feasible trajectory are

$$q_1 - q_0 > \begin{cases} T_j{}^* \left(v_0 + v_1\right), & if \quad T_j{}^* < \frac{a_{max}}{j_{max}} \\ \frac{1}{2}\left(v_0 + v_1\right)\left[T_j{}^* + \frac{|v_1 - v_0|}{a_{max}}\right], & if \quad T_j{}^* < \frac{a_{max}}{j_{max}} \end{cases}, \tag{4.10}$$

where

$$T_j{}^* = min\left\{\sqrt{\frac{|v_1 - v_0|}{j_{max}}}, \quad \frac{a_{max}}{j_{max}}\right\}. \tag{4.11}$$

After the check shown above, it is important to compute every time interval. There are many trajectories with only acceleration or deceleration part, other ones which reach the maximum velocity or vice versa; for each case, the time intervals have to be calculated in different ways which are reported in [22]. Every possibility is not analyzed in detail; only a list of all possible time interval types is shown below.

- T_{j1}: time-interval in which the jerk is constant (j_{max} or j_{min}) during the acceleration phase;
- T_{j2}: time-interval in which the jerk is constant (j_{max} or j_{min}) during the deceleration phase;
- T_a: acceleration period;
- T_v: constant velocity period;
- T_d: deceleration period;
- $T = T_a + T_v + T_d$: total duration of the trajectory.

Once the time intervals are calculated the trajectory can be computed as follows. The case with $q_1 > q_0$ and $t_0 = 0$ is considered.

1. **Acceleration phase:**

   a) $t \in [0, T_{j1}]$

$$\begin{cases} q\left(t\right) = q_0 + v_0 t + j_{max}\dfrac{t^3}{6} \\ \dot{q}\left(t\right) = v_0 + j_{max}\dfrac{t^2}{2} \\ \ddot{q}\left(t\right) = j_{max} t \\ q^{(3)}\left(t\right) = j_{max}; \end{cases} \tag{4.12}$$

b) $t \in [T_{j1}, T_a - T_{j1}]$

$$\begin{cases} q(t) = q_0 + v_0 t + j_{max}\dfrac{t^3}{6} \\ \dot{q}(t) = v_0 + j_{max}\dfrac{t^2}{2} \\ \ddot{q}(t) = j_{max} t \\ q^{(3)}(t) = j_{max}; \end{cases} \tag{4.13}$$

c) $t \in [T_a - T_{j1}, T_a]$

$$\begin{cases} q(t) = q_0 + (v_{lim} + v_0)\dfrac{T_a}{2} - v_{lim}(T_a - t) - j_{min}\dfrac{(T_a - t)^3}{6} \\ \dot{q}(t) = v_{lim} + j_{min}\dfrac{(T_a - t)^2}{2} \\ \ddot{q}(t) =, -j_{min}(T_a - t) \\ q^{(3)}(t) = j_{min} = -j_{max}. \end{cases} \tag{4.14}$$

2. **Constant velocity phase:**

a) $t \in [T_a, T_a - T_v]$

$$\begin{cases} q(t) = q_0 + (v_{lim} + v_0)\dfrac{T_a}{2} + v_{lim}(t - T_a) \\ \dot{q}(t) = v_{lim} \\ \ddot{q}(t) = 0 \\ q^{(3)}(t) = 0. \end{cases} \tag{4.15}$$

3. **Deceleration phase:**

a) $t \in [T - T_d, T - T_d + T_{j2}]$

$$\begin{cases} q(t) = q_1 + (v_{lim} + v_1)\dfrac{T_d}{2} + v_{lim}(t - -T + T_d) + \dots \\ \qquad j_{max}\dfrac{(t - T + T_d)^3}{6} \\ \dot{q}(t) = v_{lim} - j_{max}\dfrac{(t - T + T_d)^2}{2} \\ \ddot{q}(t) = -j_{max}(t - T + T_d) \\ q^{(3)}(t) = j_{min} = -j_{max}; \end{cases} \tag{4.16}$$

b) $t \in [T - T_d + T_{j2}, T - T_{j2}]$

$$\begin{cases} q(t) = q_1 + (v_{lim} + v_1)\dfrac{T_d}{2} + v_{lim}(t - T + T_d) + \dots \\ \qquad + \dfrac{a_{lim_d}}{6}\left(3(t - T + T_d)^2\right) + \dots \\ \qquad - \dfrac{a_{lim_d}}{6}\left(3T_{j2}(t - T + T_d) + {T_{j2}}^2\right) \\ \dot{q}(t) = v_{lim} + a_{lim_d}\left(t - T + T_d - \dfrac{T_{j2}}{2}\right) \\ \ddot{q}(t) = -j_{max}T_{j2} = a_{lim_d} \\ q^{(3)}(t) = 0; \end{cases} \tag{4.17}$$

c) $t \in [T - T_{j2}, T]$

$$\begin{cases} q(t) = q_1 + v_1(T - t) - j_{max}\dfrac{(T - t)^3}{6} \\ \dot{q}(t) = v_1 + j_{max}\dfrac{(T - t)^2}{2} \\ \ddot{q}(t) = -j_{max}(T - t) \\ q^{(3)}(t) = j_{max}. \end{cases} \tag{4.18}$$

4.1.2.5 Combination of straight-line and circular-arc segments

The approach, explained in this paragraph, has been used by Volker Schneider for the trajectory generation system of the Automatic Flight Guidance and Control System of the TUM-FSD [31]. The main features of this method are listed below:

1. it is based on dividing the flight path in segments instead of describing the full path as a whole;

2. a trajectory is composed by three different kind of segments, namely straight-line, circular-arc and transitions. The first and the second one are steady-state while the third one depends on the aircraft dynamics.

A brief explanation of each segment is reported below.

Straight-line segments: the mathematical formula is

$$\mathbf{x}(k) = k\begin{Bmatrix} \cos\chi \\ \sin\chi \end{Bmatrix}, \tag{4.19}$$

where χ is the angle representing the line direction and k is the running parameter. The issue is that the curve is not continuous differentiable at each waypoint (named r^{Ci} in Fig. 4.3) and so the circular-arcs are introduced.

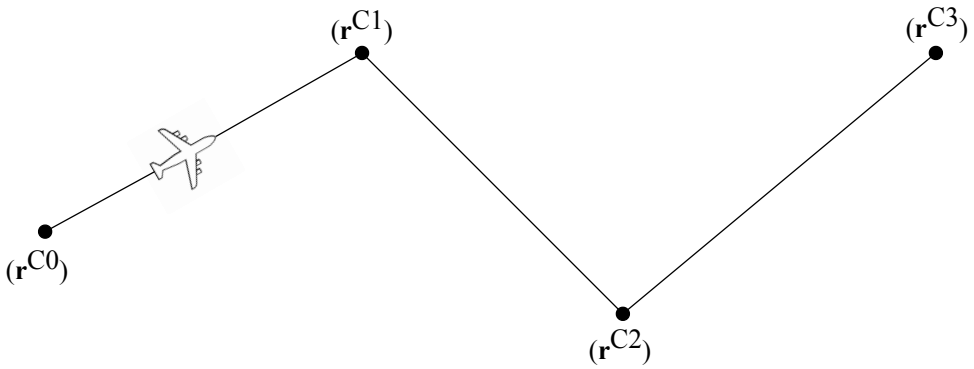

Figure 4.3: Straight-lines trajectory

Circular-arc segments: the parameterization of a circle is provided by

$$\left(\mathbf{x}\left(\varphi_c\right)\right)_c = r_c \begin{Bmatrix} \sin\varphi_c \\ 1-\cos\varphi_c \end{Bmatrix}_c, \tag{4.20}$$

where $r_c > 0$ and φ_c is the angle defined in Fig. 4.4.

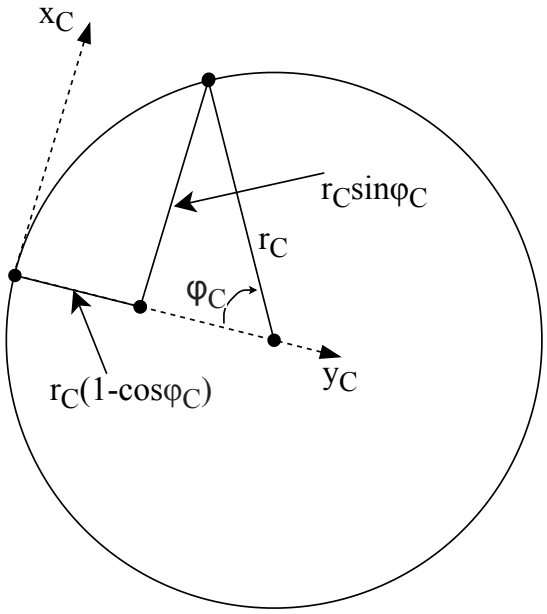

Figure 4.4: Circular-arc segment

Eq. (4.20) represents a right hand turn circle; if the y_c component is multiplied by -1, it represents a left turn. The arc length is equal to $s = r_c\varphi_c$ and, as a consequence, the curvature κ is defined by

$$\kappa\left(\varphi_c\right) = \frac{d\varphi_c}{ds} = \frac{1}{r_c}. \tag{4.21}$$

In the case of an horizontal circular flight path with a constant trajectory speed V_T, the turn radius is expressed by

$$r_c = \frac{V_T}{\dot{\chi}_T}, \tag{4.22}$$

where $\dot{\chi}_T$ is the turn rate.

Considering the following relationship between the turn rate and the kinematic bank angle μ_K,

$$\tan \mu_K = \frac{V_T \dot{\chi}_T}{g_0}, \tag{4.23}$$

assuming that the aircraft is flying in wind-free condition and without any sideslip angle, namely $\mu_K \approx \phi$ where ϕ is the Euler bank angle, the turn radius can be written in dependence of the bank angle as

$$r_c = \frac{V_T^2}{g_0 \tan \phi}. \tag{4.24}$$

The equations presented in the previous lines characterize the circular-arc segments. It is evident not only the computational simplicity but also the critical aspect of this kind of curve. Analyzing Eq. (4.21), it is possible to notice that the curvature is a constant non-zero value. Combining straight-line and circular-arc segments, there is a curvature step in the points r^{P1} and r^{P4}, shown in Fig. 4.5.

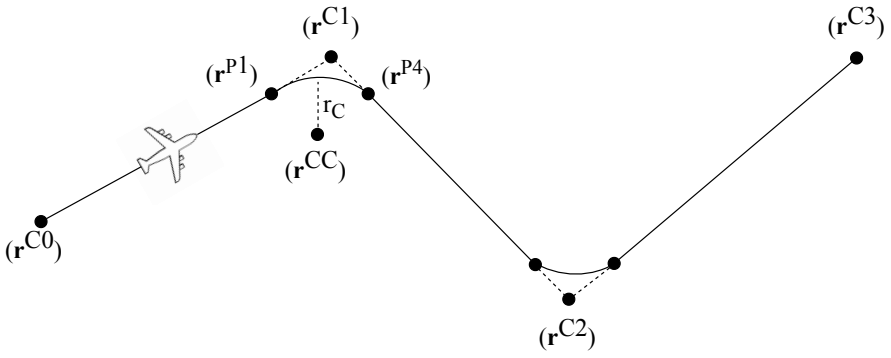

Figure 4.5: Trajectory with a combination of straight-lines and circular-arcs

Transitions: these segments, characterized by a smoother change in the curvature, can be used to solve the problem mentioned in the previous paragraphs regarding the step in the curvature function.

It is necessary to introduce two points, r^{P2} and r^{P3}; the first one links the turn-in and the circular-arc while the second one links the circular-arc and the turn-out. In Fig. 4.6, transition segments are used to link two straight-lines and one circular-arc. The clothoid curve is one of the possible solutions for the transition

problem. It is characterized by a constantly increasing curvature, proportional to the curve length. The specific features, such as the mathematical formulas, are not presented in this document because they are not useful for this thesis.

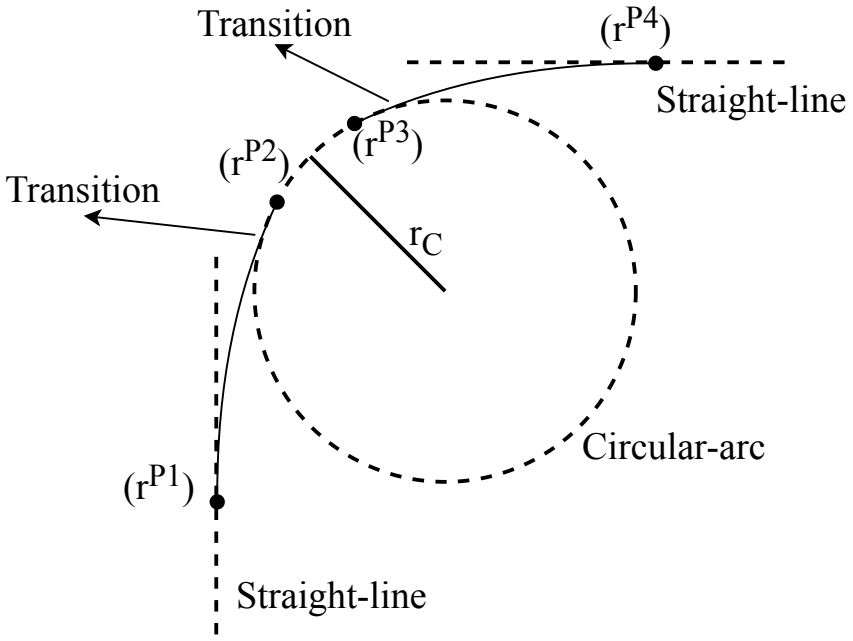

Figure 4.6: Transition maneuver between straight-line and circular arc segments

4.1.3 Multi-point trajectories

Methods to build a trajectory using all waypoints at once are considered in this section. Differently from point-to-point trajectories, which take into account only the initial and final points, also intermediate points are used.

4.1.3.1 Polynomial functions

A polynomial function of degree n is able to interpolate $n+1$ points:

$$q(t) = a_0 + a_1 t + \cdots + a_n t^n. \tag{4.25}$$

Given the points (t_k, q_k) and $k = 0, \ldots, n$ a unique polynomial of degree n exists. The interpolation problem can be seen as the solution of a linear system of $n+1$ equations in $n+1$ unknowns (the coefficients a_k of the interpolating polynomial). A vector representation of the problem, using *Vandermonde* matrix $\mathbf{T}$, can be expressed as

$$\mathbf{q} = \begin{bmatrix} q_0 \\ q_1 \\ \vdots \\ q_{n-1} \\ q_n \end{bmatrix} = \begin{bmatrix} 1 & t_0 & \ldots & t_0^n \\ 1 & t_1 & \ldots & t_1^n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & t_{n-1} & \ldots & t_{n-1}^n \\ 1 & t_n & \ldots & t_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} = \mathbf{T}\mathbf{a}. \tag{4.26}$$

If $t_{k+1} > t_k$, $k = 0, \ldots, n-1$, the matrix $\mathbf{T}$ is invertible and the coefficients a_k can be computed as

$$\mathbf{a} = \mathbf{T}^{-1}\mathbf{q}. \tag{4.27}$$

The following aspects are the main advantages of using polynomial functions.

- This technique guarantees the exact crossing of the given points.
- The function is described only by the $n+1$ coefficients.
- The derivatives of the function $q(t)$ are continuous in the range $[t_0, t_n]$. The n-th derivative is constant and all the higher order derivatives are null.
- $q(t)$ is unique.

The disadvantages are listed below.

- It is characterized by inefficiency from the computational point of view and by the possibility of numerical errors for large values of n since the condition number of $\mathbf{T}$ increases with n. This problem can be avoided using Lagrange formula or recursive formulations such as the Neville algorithm.
- The number of calculations may be heavy because it depends on n.
- The change of one point (t_k, q_k) brings to the re-computation of all the coefficients of the polynomial;
- The insertion of an additional point (t_{n+1}, q_{n+1}) implies an higher degree polynomial and the calculation of all the coefficients.
- The obtained trajectories usually show pronounced oscillations.
- In order to add conditions on initial, final or intermediate velocities and accelerations, it is necessary to use high order polynomial functions and consider other constraints on the coefficients. The consequences are the problems related to an high value of n which have already been explained above.

4.1.3.2 Orthogonal polynomials

An alternative to the standard polynomials is the use of orthogonal polynomials. Their formulation of degree m is

$$q(t) = a_0 p_0(t) + a_1 p_1(t) + \cdots + a_m p_m(t), \tag{4.28}$$

where $a_0, a_1, \ldots, a_m$ are constant parameters and $p_0(t), p_1(t), \ldots, p_m(t)$ are polynomials of proper degree. The last ones are called *orthogonal* because they have the following properties:

$$\begin{aligned} \gamma_{ji} &= \sum_{k=0}^{n} p_j(t_k)\, p_i(t_k) = 0, \qquad \forall j, i : j \neq i \\ \gamma_{ii} &= \sum_{k=0}^{n} \left[p_i(t_k)\right]^2 \neq 0, \end{aligned} \tag{4.29}$$

where $t_0, t_1, \ldots, t_n$ are the instants in which the polynomials are orthogonal. It is possible to interpolate, or approximate with a tolerance, $n+1$ given points. The error ϵ_k for the least squares method is defined as

$$\epsilon_k = \left(q_k - \sum_{j=0}^{m} a_j p_j \left(t_k \right) \right), \qquad k = 0, \ldots, n \tag{4.30}$$

and the total square error is equal to

$$\varepsilon^2 = \sum_{k=0}^{n} \epsilon_k^2. \tag{4.31}$$

If $\varepsilon^2 = 0$, the interpolation is exact. In general, a_j are calculated in order to minimize ε^2. Defining

$$\begin{aligned} \delta_i &= \sum_{k=0}^{n} q_k p_i \left(t_k \right), \qquad i \in [0, \ldots, m] \\ \gamma_{ji} &= \sum_{k=0}^{n} p_j \left(t_k \right) p_i \left(t_k \right), \qquad j, i \in [0, ..., m] \end{aligned} \tag{4.32}$$

and omitting some mathematical passages, the expression of the parameters is given by

$$a_j = \frac{\delta_j}{\gamma_{jj}}. \tag{4.33}$$

The next issue consists in finding the orthogonal polynomials $p_j(t)$ and there are a lot of different methods to find them (e.g. the *Chebyshev polynomials*), but it is not treated in detail in this explanation. The advantages of orthogonal polynomials are:

- great flexibility for the definition of a trajectory interpolating a sequence of point;
- simplicity to use standard formulation once a_i and $p_j(t)$ are found.

On the contrary, a disadvantage is that the calculation of the polynomials is not very efficient from a computational point of view.

4.1.3.3 Trigonometric polynomials

Trigonometric polynomials are convenient when the trajectory is characterized by a periodic motion, namely a value T exists such that $q(t) = q(t+T)$. The mathematical formula of a trigonometric polynomial is

$$q(t) = a_0 + \sum_{k=1}^{m} a_k \cos\left(k \frac{2\pi t}{T} \right) + \sum_{k=1}^{m} b_k \sin\left(k \frac{2\pi t}{T} \right), \tag{4.34}$$

whose coefficients are computed imposing the interpolation conditions on the points. Given a set of points q_k, $k = 0, \ldots, n$ with $q_0 = q_n$ (condition of periodic motion) to be interpolated at time instants t_k, $k = 0, \ldots, n$, the degree m of

the trigonometric polynomial is found such that $2m + 1 = n$. Assuming $t_0 = 0$, the result is that $t_n = T$. The set of $2m + 1$ system equations in the unknowns a_k, b_k is

$$\begin{bmatrix} q_0 \\ q_1 \\ \vdots \\ q_{n-2} \\ q_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & c_1(t_0) & s_1(t_0) & \dots & c_m(t_0) & s_m(t_0) \\ 1 & c_1(t_1) & s_1(t_1) & \dots & c_m(t_1) & s_m(t_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & c_1(t_{n-2}) & s_1(t_{n-2}) & \dots & c_m(t_{n-2}) & s_m(t_{n-2}) \\ 1 & c_1(t_{n-1}) & s_1(t_{n-1}) & \dots & c_m(t_{n-1}) & s_m(t_{n-1}) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ b_1 \\ \vdots \\ a_m \\ b_m \end{bmatrix}, \tag{4.35}$$

where $c_k(t) = \cos\left(k\frac{2\pi t}{T}\right)$ and $s_k(t) = \sin\left(k\frac{2\pi t}{T}\right)$. The values of coefficients a_k and b_k are computed by solving Eq. (4.35). The advantages of trigonometric polynomials are that:

- the trajectory is periodic by construction;
- the continuity of the trajectory is guaranteed for any order of derivatives (in other words, the trajectory is C^∞ continuous). Fixing the periodic conditions, the derivatives are also continuous between the initial and the final points.

On the other hand, the negative aspects are that:

- the inversion of the matrix could require a high computational cost. The problem can be solved using a similar form to the Lagrange formula for polynomial interpolation, namely

$$q(t) = \sum_{k=0}^{n} \left(q_k \prod_{j=0, j\neq k}^{n} \frac{\sin\left(\frac{\pi}{T}(t - t_j)\right)}{\sin\left(\frac{\pi}{T}(t_k - t_j)\right)} \right). \tag{4.36}$$

- The trigonometric curves show larger oscillations than the algebraic ones and, as a consequence, larger speeds and accelerations.

4.1.3.4 Cubic splines

Given $n + 1$ points, it is possible to use n polynomials of degree p, usually lower than n, for each segment instead of a unique polynomial of degree n. The function $s(t)$ is called *spline* of degree p, where the value of p is related to the number of derivatives for which the continuity is guaranteed. The most famous kind of *spline* is the cubic one which guarantees the continuity of the velocity and acceleration at the time instants t_k:

$$\begin{aligned} &s(t) = \{q_k(t), \quad t \in [t_k, t_{k+1}], \quad k = 0, \dots, n-1\}, \\ &q_k(t) = a_{k0} + a_{k1}(t - t_k) + a_{k2}(t - t_k)^2 + a_{k3}(t - t_k)^3. \end{aligned} \tag{4.37}$$

Since n polynomials are necessary for the definition of a trajectory through $n + 1$ points and each polynomial is characterized by four coefficients, the total number of unknowns is equal to $4n$. As a consequence, it is necessary to find $4n$ conditions:

- each cubic polynomial has to cross the points at the extremities and so $2n$ conditions for the interpolation of the given points are provided;
- $n-1$ conditions for the continuity of the velocities at the transition points;
- $n-1$ conditions for the continuity of the accelerations at the transition points.

The result is a total amount of $2n + 2(n-1) = 4n - 2$ conditions. Since the unknowns are $4n$, other two conditions have to be found to set a well-posed problem. Among all the possibilities, there are different constraints that can be assigned:

- the initial and final velocities $\dot{s}(t_0) = v_0$ and $\dot{s}(t_n) = v_n$;
- the initial and final accelerations $\ddot{s}(t_0)$ and $\ddot{s}(t_n)$;
- the cyclic conditions for a periodic spline, with period $T = t_n - t_0$: $\dot{s}(t_0) = \dot{s}(t_n)$ and $\ddot{s}(t_0) = \ddot{s}(t_n)$;
- the continuity of the jerk at time instants t_1 and t_{n-1}:

$$\left.\frac{d^3 s(t)}{dt^3}\right|_{t=t_1^-} = \left.\frac{d^3 s(t)}{dt^3}\right|_{t=t_1^+} \quad , \quad \left.\frac{d^3 s(t)}{dt^3}\right|_{t=t_{n-1}^-} = \left.\frac{d^3 s(t)}{dt^3}\right|_{t=t_{n-1}^+} . \tag{4.38}$$

In general, the features of a spline $s(t)$ are:

1. the sufficient parameters for the definition of a trajectory $s(t)$ of degree p, interpolating the points (t_k, q_k), $k = 0, \ldots, n$, are $n(p+1)$;
2. given $n+1$ points and the boundary conditions, the interpolating spline $s(t)$ of degree p is univocally determined;
3. there is no relationship between the degree p of the polynomials and the number of data points;
4. $s(t)$ has continuous derivatives up to the $p-1$ order;
5. assuming that $\ddot{s}(t_0) = \ddot{s}(t_n) = 0$, the cubic spline is, among all the functions $f(t)$ interpolating the given points and with continuous first and second derivatives, the function minimizing the functional:

$$J = \int_{t_0}^{t_n} \left(\frac{d^2 f(t)}{dt^2}\right)^2 dt. \tag{4.39}$$

It can be interpreted as a sort of deformation energy, proportional to the curvature of $f(t)$. With these conditions, the spline is called *natural spline*.

This list provides a general overview of splines and so how to decide whether this kind of trajectory fit in a correct way the necessities; in particular, the most important advantage is the guarantee of continuous derivatives until $p-1$ order.

On the other hand, considering the case of a cubic spline, it is not possible to assign the initial and final velocities and accelerations at the same time and, as a consequence, at the extremities, the spline is discontinuous either on the velocity or on the acceleration. The solutions to this problem can be the following ones.

- Use of a fifth degree polynomial for the first and the last segments. The cons would be to allow larger overshoots in these segments and to increase the computational cost.
- Addition of two free extra points in the first and the last segments. Their values are computed by imposing the desired initial and final velocities and accelerations.

4.1.3.5 B-spline functions for trajectories with high degree of continuity

Some applications require trajectories with continuous derivatives up to an higher order than two. In these cases, it is very useful to use splines in the B-form (called B-splines).

$$\mathbf{s}(u) = \sum_{j=0}^{m} \mathbf{p}_j B_j^p(u), \qquad u_{min} \leq u \leq u_{max} \tag{4.40}$$

where u is the independent variable, $B_j^p(u)$ are the basis functions of degree p, defined for knot vector $\mathbf{u} = [u_0, \ldots, u_{n_{knot}}]$, and $\mathbf{p}_j$ are the control points. This approach is really useful for multi-dimensional problems but it can be used also for the one-dimensional ones. In the latter case, the independent variable is $u = t$ and the general expression can be rewritten as

$$s(t) = \sum_{j=0}^{m} p_j B_j^p(t), \qquad t_{min} \leq t \leq t_{max}, \tag{4.41}$$

where p_j are scalar parameters. The degree p of the B-spline represents also the degree of continuity. Knowing p and q_k, $k = 0, \ldots, n$ to be interpolated at t_k, the problem consists in finding p_j, $j = 0, \ldots, m$, which makes true the expression

$$s(t_k) = q_k, \qquad k = 0, \ldots, n. \tag{4.42}$$

The first passage to follow is the definition of the knot vector $\mathbf{u}$. To this aim, there are many possibilities which are not treated in a deep way in this thesis. After that, the interpolation condition of each q_k at time t_k is given by

$$q_k = \left[B_0^p(t_k), B_1^p(t_k), \ldots, B_{m-1}^p(t_k), B_m^p(t_k)\right] \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{m-1} \\ p_m \end{bmatrix}, \qquad k = 0, \ldots, n. \tag{4.43}$$

In order to have a unique solution, other conditions must be added (e.g. on the velocity, acceleration, jerk, etc...). Since the parameters p_m are constant, the constraints are written using the above expression and deriving only B_j^p. A generic formulation for all conditions may be

$$s^{(i)}(t_k) = \sum_{j=0}^{m} p_j B_j^{p(i)}(t_k). \tag{4.44}$$

The continuity condition of the curve and its derivatives, at the initial and final time instants, can be also added as

$$s^{(i)}(t_0) = s^{(i)}(t_n). \tag{4.45}$$

All these conditions can be mixed to obtain the desired profiles. The final vector generic formulation is the following one:

$$\mathbf{Ap} = \mathbf{c} \tag{4.46}$$

where $\mathbf{A}$ is the matrix containing all $B_j^{p(i)}$, $\mathbf{c}$ is the vector containing all the values of conditions to be satisfied in every time instant (e.g position, velocity, etc...) and $\mathbf{p}$ are the parameters. Finally, the B-splines own a lot of advantages:

- they are very useful in multi-variable trajectories because of their clear geometric meaning;
- they are $p - k$ continuous differentiable at a knot of multiplicity k;
- B-splines can be used also in one-dimensional problems since they simply provide continuity between contiguous segments.

4.1.4 Multi-dimensional case

Multi-dimensional trajectories concern a system in a multi-dimensional space. In this section, methods which consider the problem as a vector one, are presented.

First of all, it is useful to consider a parametric representation of a curve in the space:

$$\mathbf{p} = \mathbf{p}(u), \qquad u \in [u_{min}, u_{max}] \tag{4.47}$$

where $\mathbf{p}$ is a continuous vector function whose independent variable u ranges over some interval of the domain space. The planning of a trajectory consists in defining:

- the function $\mathbf{p}(u)$, which interpolates a set of points;
- the motion law $u = u(t)$ describing how the object should move along the path.

Before starting to analyze different methods, it is important to notice that, in the 3D space, two different kinds of continuity exist, namely the geometric and the parametric ones (velocity and acceleration), and both of them must be guaranteed.

The following list explains the main different criteria for multi-variable trajectories.

- Interpolation: exact fitting of the points.
- Approximation: passage near the points within a prescribed tolerance.
- Global: based on the whole set of points, it involves an optimization problem. If one point is modified, the entire curve must be changed.
- Local: based on local data for each pair of points, it causes less computational issues but, at the same time, the continuity at the joints is more difficult to achieve. It allows to construct corners, straight-line segments and other shapes in a simpler way. If one point is modified, only the two adjacent segments must be changed.

The main methods for multi-dimensional trajectories are described in the following lines.

4.1.4.1 Definition of the geometric path through motion primitives

As in the one-dimensional space, the simplest kinds of trajectory are available also for the multi-dimensional case.

Straight lines: between two points $\mathbf{p}_0$ and $\mathbf{p}_1$, the curve is

$$\mathbf{p}(u) = \mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)\, u, \qquad with \quad 0 \leq u \leq 1. \tag{4.48}$$

Circular arc: given a starting point $\mathbf{p}_0$, the center located on a desired axis defined by a unitary vector $\mathbf{z}_1$, a generic point $\mathbf{d}$ and having defined $\mathbf{r} = \mathbf{p}_0 - \mathbf{d}$, the procedure brings to the definition of the geometric path.

The radius and the position of the circle center with respect to the reference point $\mathbf{o}_0$, from which also the position $\mathbf{p}_0$ is computed, are equal to

$$\begin{aligned} \rho &= |\mathbf{p}_0 - \mathbf{o}_1| \\ \mathbf{o}_1 &= \mathbf{d} + \left(\mathbf{r}^T \mathbf{z}_1\right) \mathbf{z}_1. \end{aligned} \tag{4.49}$$

The parametric representation of the circular arc in the reference frame centered in $\mathbf{o}_1$ is

$$\mathbf{p}_1(u) = \begin{bmatrix} \rho\cos(u) \\ \rho\sin(u) \\ 0 \end{bmatrix}, \qquad with \quad 0 \leq u \leq \theta, \tag{4.50}$$

while the one in the reference frame centered in $\mathbf{o}_0$ is

$$\mathbf{p}(u) = \mathbf{o}_1 + \mathbf{R}\mathbf{p}_1(u) \tag{4.51}$$

where $\mathbf{R}$ is the rotation matrix between the two reference frames.

A remarkable aspect is that the trajectories computed with these techniques show continuity but have discontinuous derivatives at the intermediate points.

4.1.4.2 Global interpolation using B-splines

Firstly, u_k for each $\mathbf{q}_k$ and a knot vector $\mathbf{u} = [u_0, \ldots, u_{n_{knot}}]$ have to be defined. After that, the problem can be written in the following forms:

$$\mathbf{q}_k = \mathbf{s}\left(\bar{u}_k\right) = \sum_{j=0}^{m} \mathbf{p}_j B_j^p\left(\bar{u}_k\right), \tag{4.52}$$

$$\mathbf{q}_k^T = \left[B_0^p\left(\bar{u}_k\right), B_1^p\left(\bar{u}_k\right), \ldots, B_{m-1}^p\left(\bar{u}_k\right), B_m^p\left(\bar{u}_k\right)\right] \begin{bmatrix} \mathbf{p}_0^T \\ \mathbf{p}_1^T \\ \vdots \\ \mathbf{p}_{m-1}^T \\ \mathbf{p}_m^T \end{bmatrix}. \tag{4.53}$$

The cubic B-splines with $p = 3$ are often used.

4.1.4.3 Global approximation using B-splines

Simpler curves are used when there is the need of reducing the computational issues through the decrease of the data points number, at the expense of lower precision. These trajectories approximate the via-points within a prescribed tolerance δ.

Starting from the points $\mathbf{q}_0, \ldots, \mathbf{q}_n$ to be approximated, considering $p \geq 1$ and providing that $n > m > p$, the trajectory using B-splines is expressed as

$$\mathbf{s}\left(u\right) = \sum_{j=0}^{m} \mathbf{p}_j B_j^p\left(u\right), \qquad u_{min} \leq u \leq u_{max}. \tag{4.54}$$

The end points are exactly interpolated, i.e. $\mathbf{q}_0 = \mathbf{s}_k\left(0\right)$ and $\mathbf{q}_n = \mathbf{s}\left(1\right)$ while the internal points $\mathbf{q}_k$ are approximated in the least square sense, i.e. by minimizing the functional

$$\sum_{k=1}^{n-1} w_k |\mathbf{q}_k - \mathbf{s}\left(\bar{u}_k\right)|^2. \tag{4.55}$$

The coefficients w_k can be freely chosen to weight the error at different points.

4.1.4.4 Smoothing cubic B-splines

The formulation is the same of B-splines of order $p = 3$ whose control points are computed considering the goals to find a good fit of the points and to obtain a trajectory as smooth as possible.

In order to obtain the control points, the functional to minimize is

$$L := \mu \sum_{k=0}^{n} w_k |\mathbf{s}\left(\bar{u}_k\right) - \mathbf{q}_k|^2 + (1-\mu) \int_{\bar{u}_0}^{\bar{u}_n} \left| \frac{d^2 \mathbf{s}\left(u\right)}{du^2} \right|^2 du, \tag{4.56}$$

where $\mu \in [0,1]$ expresses the importance given to the two conflicting goals and w_k can be arbitrarily chosen to modify the weights of different quadratic errors on the global estimation.

4.1.4.5 B-spline functions for trajectories with high degree of continuity

The use of B-splines with $p > 3$ can be useful when it is required to have also continuous jerk, snap or even higher order derivatives. This approach can be used for interpolation and also for approximation problems.

The basis is the same of the B-splines already explained and the following considerations can be done for the interpolation problem. To build a trajectory r times differentiable, which interpolates $n+1$ points, it is necessary to choose a B-spline of degree $p = r+1$. The choice of the knots distribution u_k depends on p, which could be odd or even. If p is odd, $p-1$ additional constraints are necessary to have a unique solution while, if p is even, p additional conditions are needed. Adding new knots and increasing the number of p_j, it is possible to consider new constraints, for example on the value of the tangent vectors.

4.1.4.6 Use of NURBS for trajectory generation

The Non-Uniform Rational B-Spline (NURBS) can represent a wide variety of curves like circles, parabolas, ellipses, lines and hyperbolas. They can be written as

$$\mathbf{n}(u) = \frac{\sum_{j=0}^{m} \mathbf{p}_j w_j \mathbf{B}_j^p(u)}{\sum_{j=0}^{m} w_j \mathbf{B}_j^p(u)}, \qquad u_{min} \leq u \leq u_{max}, \tag{4.57}$$

where $\mathbf{B}_j^p(u)$ are standard B-spline basis functions, $\mathbf{p}_j$ are the control points and w_j are the weights associated to each control point. If $w_j = 1, j = 0, ..., m$ the NURBS curves are standard B-splines. They are really useful in CAD/CAM environments.

4.1.4.7 Local interpolation with Bézier curves

Bézier curves are used when the trajectory has to be constructed in an interactive manner or when it is necessary to find a trajectory based only on local data. Nevertheless, these techniques require to know not only the points to be interpolated but also the derivatives at these points in order to guarantee the necessary smoothness. It is possible that the tangent and the curvature directions have to be computed; for this purpose, many methods exist. Firstly, the cubic Bézier curves are presented:

$$\mathbf{b}(u) = (1-u)^3 \mathbf{p}_0 + 3u(1-u)^2 \mathbf{p}_1 + 3u^2(1-u)\mathbf{p}_2 + u^3 \mathbf{p}_3, \quad u \in [0,1]. \tag{4.58}$$

The most important steps to obtain a trajectory interpolating $\mathbf{q}_k$, $k = 0, \ldots, n$ are the following ones:

- if necessary, compute the tangent vectors $\mathbf{t}_k$;
- given $\mathbf{q}_k$ and $\mathbf{q}_{k+1}$, the Bézier curve $\mathbf{b}_k(u)$, $u \in [0,1]$, can be computed by assuming that:

$$\begin{array}{ll} \mathbf{p}_{0,k} = \mathbf{q}_k & \mathbf{p}_{3,k} = \mathbf{q}_{k+1} \\ \mathbf{t}_{0,k} = \mathbf{t}_k & \mathbf{t}_{3,k} = \mathbf{t}_{k+1} \end{array} \tag{4.59}$$

  and computing the position of the internal control points $\mathbf{p}_{1,k}$ and $\mathbf{p}_{2,k}$. To do this, these expressions are used:

$$\left|\mathbf{b}^{(1)}(0)\right| = \left|\mathbf{b}^{(1)}(1)\right| = \left|\mathbf{b}^{(1)}(1/2)\right| = \alpha,$$

$$\mathbf{p}_1 = \mathbf{p}_0 + \frac{1}{3}\alpha\mathbf{t}_0, \qquad \mathbf{p}_2 = \mathbf{p}_3 - \frac{1}{3}\alpha\mathbf{t}_3, \tag{4.60}$$

$$a\alpha^2 + b\alpha + c = 0,$$

  where

$$\begin{aligned} a &= 16 - |\mathbf{t}_0 + \mathbf{t}_3|^2, \\ b &= 12\,(\mathbf{p}_3 - \mathbf{p}_0)^T(\mathbf{t}_0 + \mathbf{t}_3), \\ c &= -36\,|\mathbf{p}_3 - \mathbf{p}_0|^2. \end{aligned} \tag{4.61}$$

The fifth degree Beziér curves, characterized by six parameters, can be useful to define a G^2 continuous trajectory. Its expression is

$$\mathbf{b}_k(u) = \sum_{j=0}^{5} B_j^5(u)\,\mathbf{p}_j, \qquad 0 \le u \le 1. \tag{4.62}$$

Both tangent and curvature vectors $\mathbf{t}_k$ and $\mathbf{n}_k$ at each point $\mathbf{q}_k$ are necessary. For each pair $(\mathbf{q}_k, \mathbf{q}_{k+1})$ the control points are computed by imposing the conditions on initial and final tangents $(\mathbf{t}_{0,k} = \mathbf{t}_k, \mathbf{t}_{5,k} = \mathbf{t}_{k+1})$ and curvature vectors $(\mathbf{n}_{0,k} = \mathbf{n}_k, \mathbf{n}_{5,k} = \mathbf{n}_{k+1})$ assumed of unit length:

$$\begin{cases} \mathbf{b}_k(0) = \mathbf{p}_{0,k} = \mathbf{q}_k \\ \mathbf{b}_k(1) = \mathbf{p}_{5,k} = \mathbf{q}_{k+1} \\ \mathbf{b}_k^{(1)}(0) = 5\,(\mathbf{p}_{1,k} - \mathbf{p}_{0,k}) = \alpha_k\mathbf{t}_k \\ \mathbf{b}_k^{(1)}(1) = 5\,(\mathbf{p}_{5,k} - \mathbf{p}_{4,k}) = \alpha_k\mathbf{t}_{k+1} \\ \mathbf{b}_k^{(2)}(0) = 20\,(\mathbf{p}_{0,k} - 2\mathbf{p}_{1,k} + \mathbf{p}_{2,k}) = \beta_k\mathbf{n}_k \\ \mathbf{b}_k^{(2)}(1) = 20\,(\mathbf{p}_{5,k} - 2\mathbf{p}_{4,k} + \mathbf{p}_{3,k}) = \beta_k\mathbf{n}_{k+1} \end{cases} \tag{4.63}$$

where α_k and β_k are parameters which can be freely chosen. For example, α_k can be computed by assuming that $\left|\mathbf{b}_k^{(1)}\right|$ is equal at the endpoints and at the midpoint in addition to the fact that $\beta_k = \bar{\beta}\alpha_k^2$. Following this procedure, the equations, already shown and used in the cubic expression, can be formulated also for this case.

4.1.4.8 Linear interpolation with polynomial blends

The trajectory, which approximates the points $\mathbf{q}_k$, $k = 0, \ldots, n$ with a tolerance δ, is constructed following these steps:

- for each $\mathbf{q}_k$, excluding the first and the last ones ($\mathbf{q}''_0 = \mathbf{q}_0$, $\mathbf{q}'_n = \mathbf{q}_n$), two additional points $\mathbf{q}'_k$ and $\mathbf{q}''_k$ are obtained finding the intersections between the lines $\overline{\mathbf{q}_{k-1}\mathbf{q}_k}$, $\overline{\mathbf{q}_k\mathbf{q}_{k+1}}$ and a ball of radius δ centered on $\mathbf{q}_k$;
- a straight line is used to join the pair $\left(\mathbf{q}''_k, \mathbf{q}'_{k+1}\right)$;
- a Bézier curve of fourth (or fifth) degree is used to interpolate the pair $(\mathbf{q}'_k, \mathbf{q}''_k)$.

This method guarantees a G^2 continuous trajectory which is a sequence of linear segments $\mathbf{l}_k(u)$ and the Bézier blends $\mathbf{b}_k(u)$, each one defined for $u \in [0, 1]$:

$$\begin{aligned}\mathbf{p} = \{&\mathbf{l}_0(u), \mathbf{b}_1(u), \mathbf{l}_1(u), \ldots, \mathbf{b}_k(u), \mathbf{l}_k(u), \mathbf{b}_{k+1}(u), \ldots, \\ &\mathbf{l}_{n-2}(u), \mathbf{b}_{n-1}(u), \mathbf{l}_{n-1}(u)\}.\end{aligned} \tag{4.64}$$

The generic i-th trajectory segment can be named $\mathbf{p}_i(u)$, $i = 0, \ldots, 2n-2$.

4.2 Fundamentals on flight path generation

This section presents the basic aspects of the flight path generation providing a deep analysis to the ones that regard this thesis. In Section 4.2.1 the concept of fixes and a brief description of their different categories are exposed. Section 4.2.2 presents the definition of leg and all the types used in the generation of the trajectory for the VTOL aircraft. Finally, Section 4.2.3 provides an outline of the possible transitions between legs, which are distinguished by International Civil Aviation Organization (ICAO) [33].

4.2.1 Fixes

A fix is described as a specific location on the face of Earth [34]. There are different kinds of fixes, such as Navigational Aids (NAVAIDs), waypoints, intersections and airports. The NAVAIDs are ground stations of several types, namely Non Directional Beacon (NDB), Very High Frequency Omnidirectional Radio Range (VOR) and Distance Measurement Equipment (DME) [31].

Initially, NAVAIDs are the basis for the Conventional Design Procedure not related to the area navigation. This type of flight is not optimal because it is constrained to the location of the NAVAIDs. During the years, the development of the technology and the use of on-board systems gives the possibility to define a new type of navigation, called Area Navigation (RNAV). In this case, the flight path is constructed through a list of navigation fixes, which is defined by WGS84 latitude/longitude position [31]. The advantage is that it is no more necessary to

use a physical station, located on the ground but it is sufficient to define a list of waypoints in the space thanks to their coordinates.

In this thesis, the list of waypoints to develop and test the trajectory generation algorithm is built by using the WGS84 coordinates taken from Google Maps around the FSD institute at TUM.

4.2.2 Leg types

ARINC 424 is the air transport standard for the definition of the airborne system navigation databases. The concepts of path and terminator are the basis for the different leg types. The path describes how the aircraft reaches the terminator by flying direct; it can be a heading, a track, a course, etc... . The terminator is the condition for which the aircraft switches to the following leg and it can be a fix, an altitude, an intercept, etc...[34].

The ARINC 424 standard defines 23 different leg types that can be combined by RNAV systems to generate complex paths; none of the existing database equipment is capable of using all of them. A description of the features of all the possible leg types is reported in the Instrument Procedures Handbook of the Federal Aviation Administration (FAA) [34].

In this thesis, only the ones useful to describe all the paths for the wingborne phase are reported, namely initial fix, track to fix and radius to fix. The reason is that, through these kind of legs with the add of transitions between them, it is possible to generate complex paths between a sequence of waypoints. For what concerns the hover phase, it is sufficient to develop similar concepts in a proper way in order to take into account also this VTOL aircraft dynamics.

4.2.2.1 Initial Fix (IF) leg

The IF defines a fix as a point in space which is used to set the beginning of a route [Fig. 4.7].

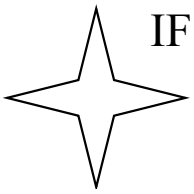

Figure 4.7: Initial Fix

4.2.2.2 Track to a Fix (TF) leg

The TF leg defines a generic track over the ground between two known database fixes. The preferable method to connect the waypoints is the straight leg [Fig. 4.8].

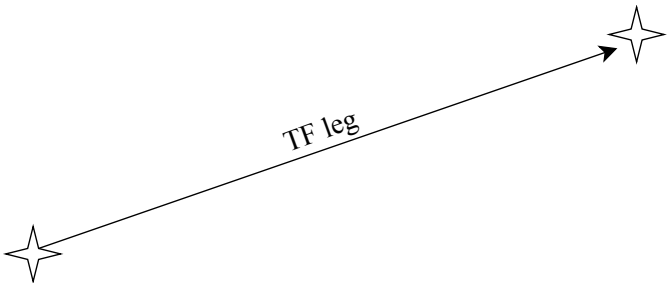

Figure 4.8: Track to a Fix leg

4.2.2.3 Radius to a Fix (RF) leg

The RF leg defines a constant radius turn between two database fixes with the definition of a specific center fix. It is also characterized by the fact that the previous and the next lines are tangent to the arc [Fig. 4.9].

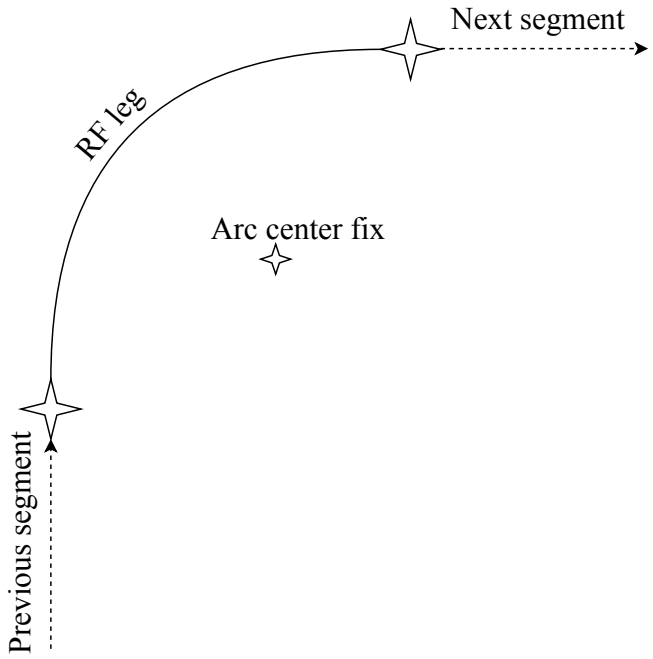

Figure 4.9: Radius to a Fix leg

The RF leg can be confused with the fixed-radius transition which will be briefly exposed in Section 4.2.3. Actually, [33] sets the difference between them; the fixed-radius transition is used during an en-route procedure while the RF leg is useful when there is a requirement for a constant radius path during a terminal or an approach procedure.

Since the application of the VTOL transition aircraft differs from the one of the civil airplanes to which the Performance-based Navigation (PBN) manual is referred, it is decided to enlarge the use of the RF leg also to en-route procedures in order to get some results and analyze them.

4.2.3 Transitions

The last fundamental aspect of the path construction is the characterization of the transitions between flight legs. ICAO distinguishes between three different types of transition, namely fly-by, fly-over and fixed-radius [33]. The most important information, useful to the development of this thesis, are reported below.

4.2.3.1 Fly-by transition

The fly-by transition is defined as a turn which is anticipated with respect to the prescribed waypoint in order to connect the two legs without any overshoot. Knowing the turn radius, it is necessary to compute two new waypoints that substitute the original one and are the beginning and the end of the turn. The RTCA DO-236B limits the path within a theoretical transition area defined by the connection of the previous and next waypoints through a circular arc segment; if the aircraft remains in this area, the trajectory is considered valid [31].

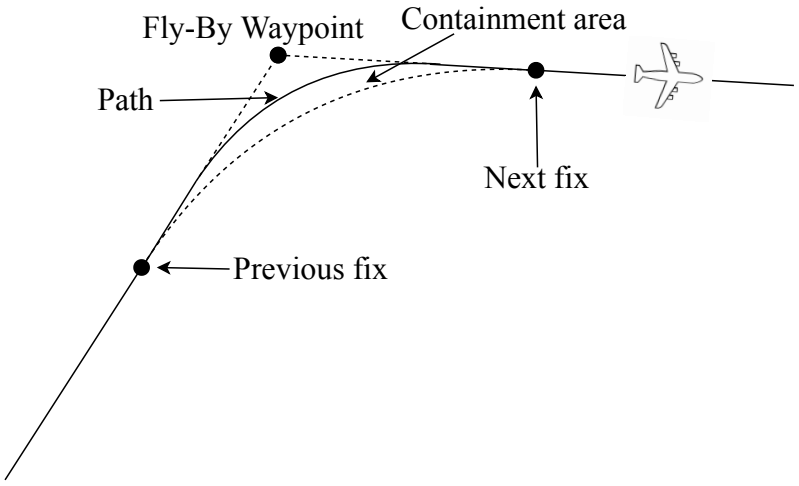

Figure 4.10: Fly-by transition

In Fig. 4.10, the solid line is the flight path while the region restricted by the dashed lines is the containment area.

4.2.3.2 Fly-over transition

The fly-over transition is characterized by the fact that the aircraft starts the turn to a new course only after being passed over the fix. The disadvantages with respect to the fly-by transition are the presence of the overshoot and the impossibility to set a containment area [Fig. 4.11].

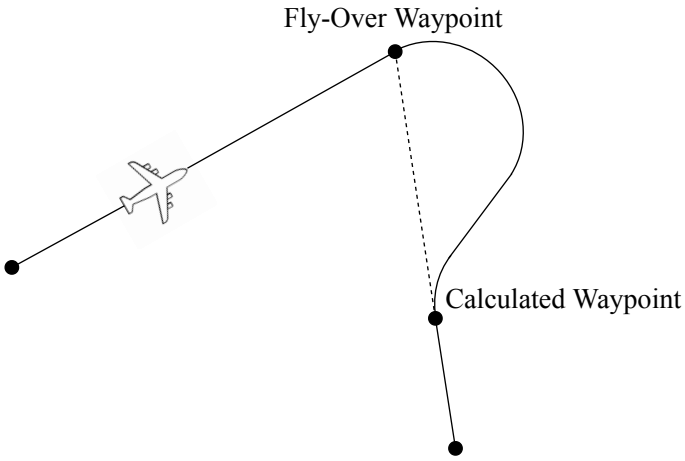

Figure 4.11: Fly-over transition

4.2.3.3 Fixed-radius transition

The fixed-radius transition is a fixed radius turn between two legs of a route. It is used when there is the necessity to have a predictable path since there is a predefined radius.

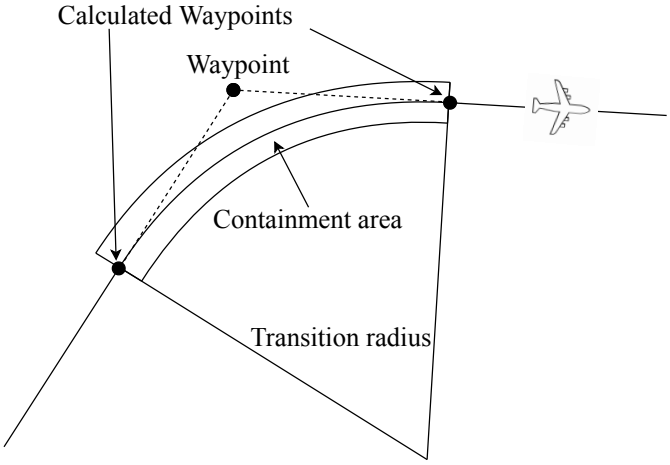

Figure 4.12: Fixed-radius transition

The containment area can be determined in relation to the kind of operation that is considered, for example when it is necessary an exact maneuver or the airspace does not allow a big transition area [Fig. 4.12]. Since there are no particular constraints on the airspace, the thesis does not use this type of transition.

4.3 Motivation for the chosen approach

Once the problem of trajectory generation is introduced by providing a brief dissertation of all kinds of trajectories, used not only in aeronautics but also in robotics,

the aim of this section is to explain what are the features and the constraints of the trajectory that the aircraft has to follow and what is the motivation of the chosen approach.

In general, the trajectory of an aircraft in the space is a three-dimensional problem and, also in this thesis, the trajectory generation block has to satisfy the same requirement since the flat outputs are the components of position, x, y and z. To treat the multi-dimensional problem, two approaches are available.

- The first one is based on the use of multi-dimensional approach introduced in Section 4.1.4. The advantage is that there are specific methods to deal with it which already return the solution as a vector.

- The second one consists in splitting the entire problem in three scalar ones. The advantage is that the one-dimensional strategies, simpler than the multi-dimensional ones, can be used. The con is that the synchronization of time is needed to guarantee that each scalar component can be unified to the other ones in a unique vector maintaining a physical meaning.

The decision is to solve a set of scalar problems; the simplicity of the trajectory formulation, despite the necessity of synchronization, is preferred to the advantages of the multi-dimensional approach.

Between the two sub-categories, namely point-to-point [Section 4.1.2] and multi-point trajectories [Section 4.1.3], it is decided to use the first one for the following aspects:

- in aeronautics, different types of trajectories may be used between each couple of points to take into account all the flight phases;

- using the multi-point trajectory, an overshoot can occur more frequently;

- another negative aspect of multi-point trajectory is that it is necessary to re-calculate the coefficients for the entire path each time a point is added, removed or changed.

These reasons lead to build the entire trajectory as a sequence of point-to-point segments.

A fundamental constraint, which influences the kind of chosen trajectory, is the continuity not only in the position but also in the first and second derivatives, namely speed and acceleration. The reason is that having a discontinuity in them is unfeasible in a real problem. For what concerns speed, the continuity does not concern only the modulus but also the direction. Considering the combination of straight lines and circular arcs of Section 4.1.2.5, it is not sufficient to impose that the speed on each straight line is equal to the same value because the change of direction causes a step in its three components. Similarly, the same consideration on modulus and direction can be done for the acceleration; referring to the same example, the constant value and the continuity of speed does not guarantee the continuity of acceleration. As a matter of fact, the connection of a straight line

and a circular arc causes a sudden change in the curvature, which is related to the centripetal acceleration and, as a consequence, a discontinuity in the profile of total acceleration.

Between all the available trajectories, the decision is to combine first and fifth degree polynomials in order to describe the basic maneuvers of wingborne and hover phase. The motivations are reported below.

- First degree polynomial gives the possibility to set two conditions and so, it is used when two points have to be connected by a straight line with a constant speed [par. 1 of Section 4.1.2.1].

- Fifth degree polynomial is constructed imposing six conditions. For this characteristic, it can be used whenever a curve along the entire path is required without loosing the continuity; the reason is that it is possible to set not only the initial and final positions but also a specific value of initial and final speeds and accelerations [par. 4 of Section 4.1.2.1]. This type of function is useful also in the case of vertical displacements for which more information are added in Section 4.4.4.

For the horizontal acceleration and deceleration phases, a double S trajectory is chosen. It is worth to notice that, for the former phase, only the first part of double S, namely the acceleration and constant speed, is considered while for the latter one, the portion of constant speed and deceleration is used. The great advantage is also that the maximum absolute value of acceleration is not overcome [Section 4.1.2.4].

Since the saturation of the track rate can be required, also circular arc is implemented because it is the only trajectory that, thanks to a constant angular rate, can provide a constant track rate value [par. 2 of Section 4.1.2.5].

4.4 Description of the simulink model

In the previous sections, all the fundamentals of the trajectory generation problem in the aeronautical field are introduced and the reasons of how the algorithm is constructed to satisfy all the requests, are explained.

This section contains the description of the model which is realized on `MATLAB & Simulink` software. As explained in Section 4.1, the trajectory generation block receives the waypoint list and the speed as inputs while provides the flat outputs and their derivatives as outputs.

The model is built through a step by step procedure. It is worth to remember the hypotheses of constant gravity and no wind. Firstly, only the wingborne phase is taken into account under the hypothesis of 2D trajectory, so that the γ angle is fixed to zero. The initial approach provides a profile of speed modulus variable in time, in particular during the curves; since another request is to maintain a constant speed during the wingborne phase, a scaling in time is introduced to reach this goal. A further requirement, caused by a physical constraint, consists

in keeping $\dot{\chi}$ under a prescribed value. After that, the model is generalized eliminating the hypothesis of $\gamma = 0\ deg$ in order to consider also climb path and, as a consequence, the trajectory always results as a 3D one. Finally, the last passage is to add the hover phase; it means introducing not only all the dynamics of a quadcopter but also the transition between the two phases.

Each subsection regards a part of the final complete model. In Section 4.4.1, the waypoint list is treated in a deep way. Section 4.4.2 and Section 4.4.3 tell respectively how the transformation of the waypoint list in the local NED frame is performed and how the geometrical computations are done to generate the new waypoint list. This one is used in the algorithm, presented in Section 4.4.4, to find the flat outputs. Finally, Section 4.4.5 shows the last necessary manipulations to provide the outputs as vectors.

4.4.1 Waypoint list

The waypoint list is one of the two inputs of the trajectory generation block. It is a sequence of points in the space which are defined in the `main` code as vectors of dimension four; in Table 4.1, each element of the vector is associated to a feature of the waypoint. The ID number varies between 0 and 9 and each of these

Element	Feature	Measurement unit
id	ID Number	-
lat	Latitude	deg
$long$	Longitude	deg
h	Altitude	m

Table 4.1: Waypoint features

values corresponds to a specific kind of leg. The procedure for the definition of the waypoint list follows this logic: considering two fixes, the information about the type of segment that connects them are contained in the second fix of each couple. This decision is a precise request also justified by what is reported in the handbook of FAA [34], used as reference for the classification of the fixes and legs; as a matter of fact, the latter one cites the ARINC 424 specification which defines a leg through the concept of path and terminator and so the choice to insert all the features in the second fix is reasonable.

The motivation to set this wide categorization comes from the necessity to describe all the flight phases of the VTOL aircraft. The first passage is to take into account the wingborne phase and the hover one in a separate way since their dynamics are completely different. Secondarily, it is necessary to consider the transition between the two phases and to define legs in which this procedure is completed.

The initial fix represents the beginning of a trajectory both for the wingborne and the hover phases.

For what concerns the wingborne phase, the types of legs that are developed are the track to a fix and the radius to a fix. TF is used to generate a path between two following waypoints with straight legs as preferred method [34]. Despite ARINC 424 defines specific legs for climb path, such as fix to an altitude or course to an altitude, TF developed in this thesis can be used not only for a displacement in a plane but also for a three-dimensional one in order to avoid the use of another leg and make the algorithm simpler. For the TF leg, it is also important to characterize the transition to the next leg whenever it is necessary. Specifically, the 0 ID number is referred to a TF leg which reaches the fix and no transition is required; for example, it is the case of TF leg before a RF leg because the latter one is built to guarantee the continuity. The 1 and 2 ID numbers are used in the case of TF leg with fly-by and fly-over transitions respectively. From now on, these two cases are called fly-by and fly-over waypoints for simplicity. As anticipated in Section 4.2.3, the fixed-radius transition is not taken into account. As already mentioned, radius to a fix leg connects the two waypoints with a curve and it can be used when a huge change in the track angle is required.

On the other hand, the hover phase requires that the aircraft stays in the hovering position, changes altitude by moving vertically along a straight line and completes a fly-by maneuver to reach the next leg. For the second request, a similar idea to TF along the vertical axis is defined and for the third one, the concept of vertical fly-by waypoint is introduced. The latter one is realized by the authors in order to perform a change in altitude with a smooth transition between the vertical (or horizontal) plane and the horizontal (or vertical) one before (or after) an acceleration (or deceleration) phase.

The last part is to consider the acceleration and deceleration maneuvers between two predefined speeds. The features related to each ID number are reported in Table 4.2.

ID Number	Leg	Phase
0	Track to a Fix leg	Wingborne
1	Fly-by waypoint	Wingborne
2	Fly-over waypoint	Wingborne
3	Radius to a Fix	Wingborne
4	Vertical fly-by waypoint	Hover
5	Acceleration phase	Wingborne/Hover
6	Deceleration phase	Wingborne/Hover
7	Hovering	Hover
8	Change in altitude	Hover
9	Initial Fix	Wingborne/Hover

Table 4.2: Meaning of the ID numbers

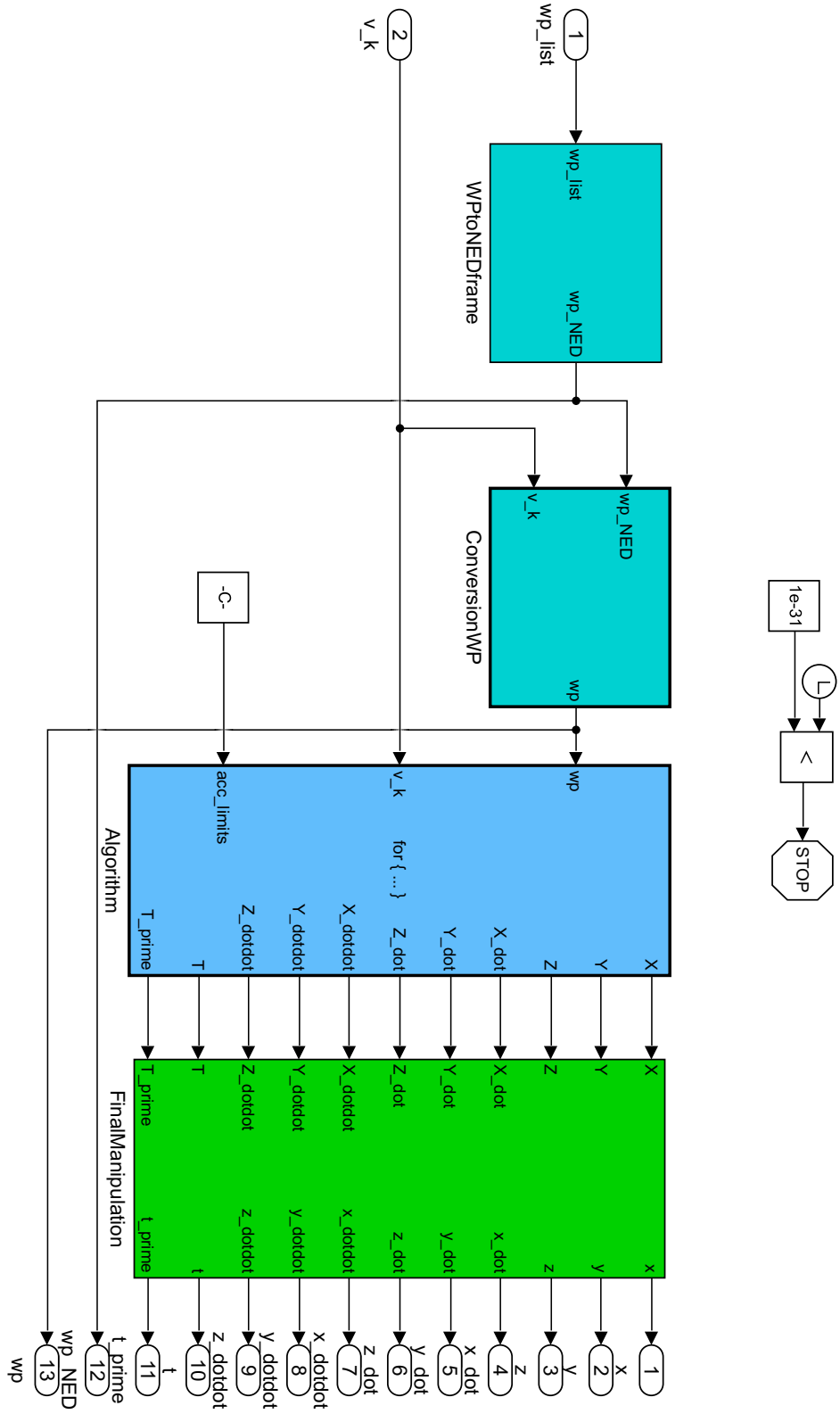

Figure 4.13: Trajectory generator

4.4.2 Transformation in the local NED frame

The first important manipulation, to be done on the waypoint list, consists in the transformation of the coordinates from the WGS84 reference system $(lat, long, h)$ to the local NED one (x, y, z) having as origin the position of the FSD institute. In addition to this change of coordinates, the new waypoint list contains the values of χ and γ, which are the track and the climb angles respectively. These quantities are calculated for the connection legs between every couple of waypoints. In the scheme of Fig. 4.13, the subsystem named `WPtoNEDframe` provides the transformation explained. Fig. 4.14 shows the internal scheme of the subsystem.

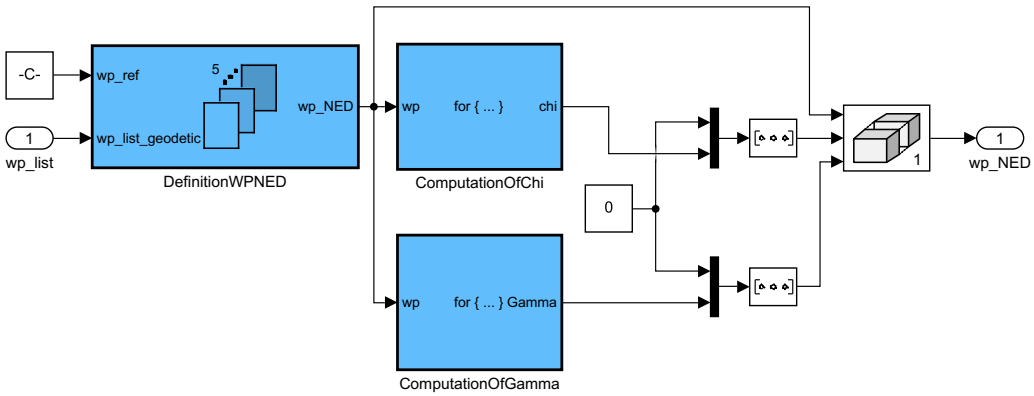

Figure 4.14: Transformation in the local NED frame and addition of χ and γ

The input is the waypoint list explained in Section 4.4.1 and the constant corresponds to the position of FSD institute. The main important blocks are `DefinitionWPNED`, `ComputationOfChi` and `ComputationOfGamma`. The output wp_{NED} contains χ and γ and it is constructed thanks to the use of `Reshape` and `Matrix Concatenate` blocks. For the first waypoint, χ and γ angles are set to zero while for all the other waypoints they are referred to the leg which comes before them, as explained in Section 4.4.1.

4.4.2.1 Definition of waypoint list in the local NED frame

`DefinitionWPNED` is a `For Each` block, which always uses the same values for the reference waypoint and let flow all the other ones [Fig. 4.15].

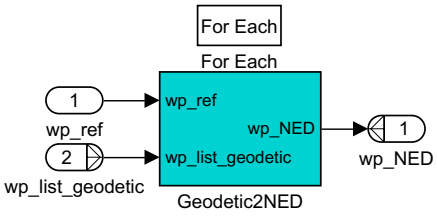

Figure 4.15: For each block

For every waypoint, the `Simulink` block `LLAtoECEF`, contained both in `Geodetic2ECEF_ref` and `Geodetic2ECEF`, transforms the coordinates from WGS84 system to ECEF reference frame, as shown in Fig. 4.16. Finally, the subtraction and the matrix multiplication are used to pass into the local NED frame. The first one centers the origin of the new reference frame while the second one aligns the axes in the direction of the NED frame.

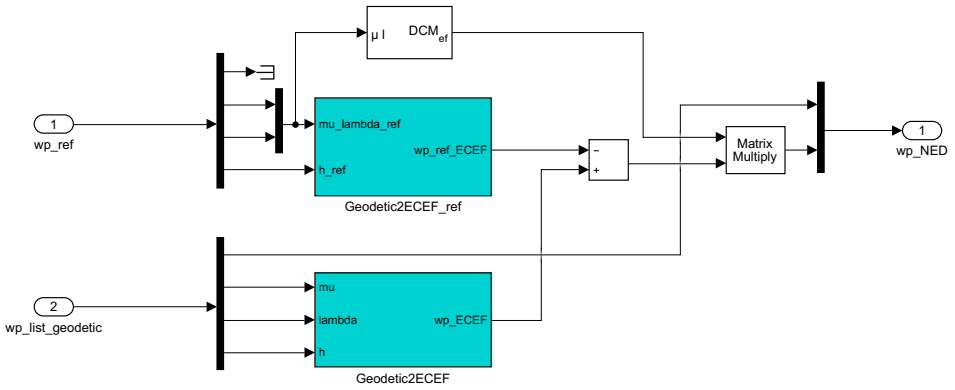

Figure 4.16: Transformation in the local NED frame

4.4.2.2 Computation of χ angle

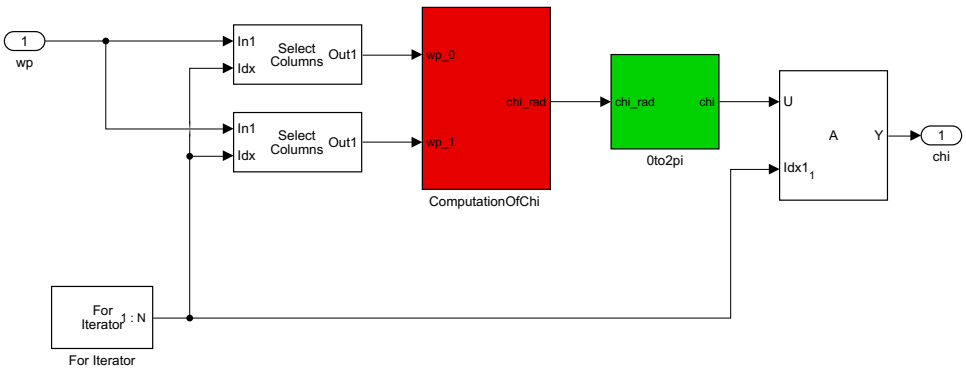

Figure 4.17: `For Iterator` block of χ computation

The value of χ for every couple of waypoints is calculated as shown in Fig. 4.18 using

$$\chi = \arctan \frac{y_1 - y_0}{x_1 - x_0}. \tag{4.65}$$

Starting from x and y coordinates and thanks to the use of `atan2` block, the value of χ is reached. In order to pick every couple of waypoints one by one, a for iterator is used [Fig. 4.17].

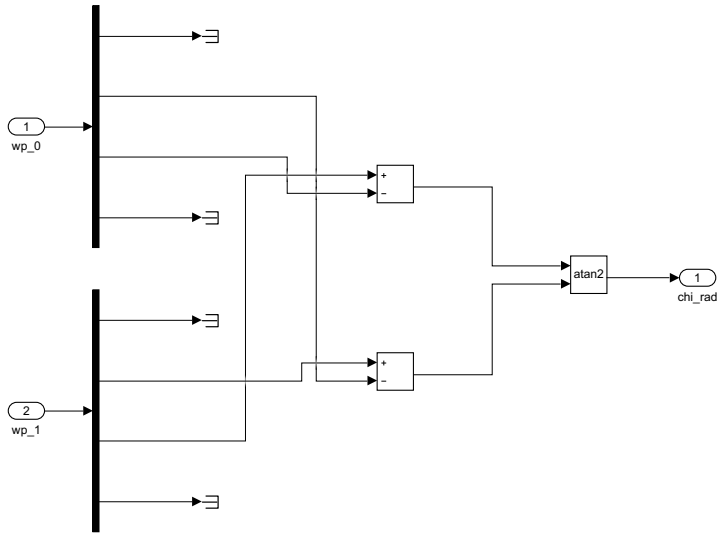

Figure 4.18: Computation of χ

The `0to2pi` block, shown in Fig. 4.19 guarantees that the value of χ is the correct one. The problem is that `atan2` block in Fig. 4.18 provides angles only between $-\pi$ and π; this output enters the scheme in Fig. 4.19 which returns a value of the track angle in the proper quadrant, namely with $\chi \in [0, 2\pi]$.

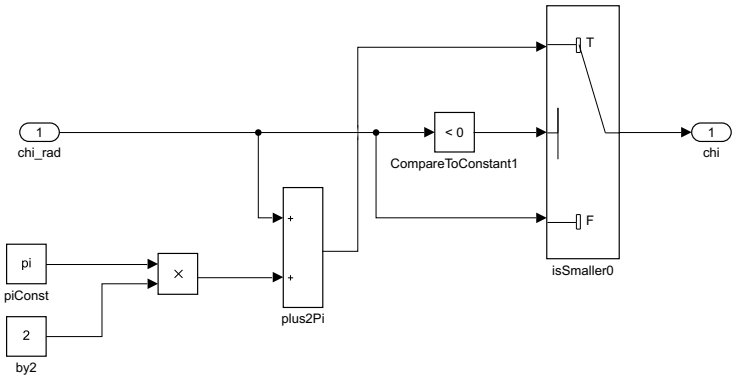

Figure 4.19: Conversion from $\chi \in [-\pi, \pi]$ to $\chi \in [0, 2\pi]$

4.4.2.3 Computation of γ angle

A `For Iterator` block is used with the same purpose of the one explained before for χ [Fig. 4.20].

Once the coordinates in O frame of the two waypoints are picked, the value of the climb angle is reached using

$$\gamma = \arctan \frac{z_0 - z_1}{\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}}. \tag{4.66}$$

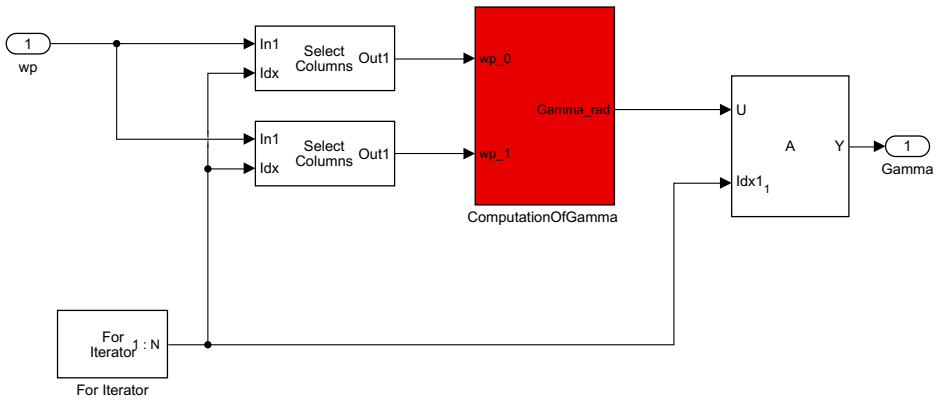

Figure 4.20: `For Iterator` block of γ computation

The block `atan2` provides $\gamma \in [-\pi, \pi]$. It is fine because the climb angle physically remains included between $-\pi/2$ and $\pi/2$ along the entire trajectory. Differently from the procedure used to compute the track angle, γ does not need further manipulations [Fig. 4.21].

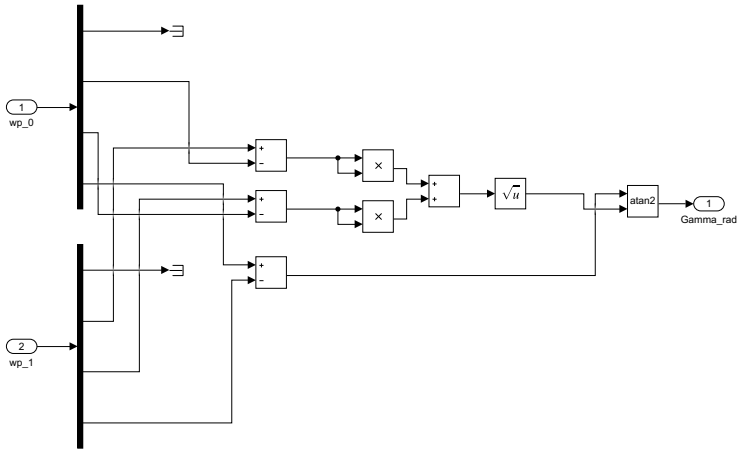

Figure 4.21: Computation of γ

4.4.3 Geometry calculation and definition of new waypoint list

Once the waypoint list is transformed from the WGS84 coordinates to the local NED frame, the next step is to make the geometry calculation when it is needed or to change only the ID number with the ones used in the algorithm; so, this passage brings to the definition of a new waypoint list. This is the aim of `ConversionWP` block of Fig. 4.13.

The inputs are the waypoint list wp_{NED} obtained in `WPtoNEDframe` block and the vector of speeds, named v_k and defined in the `main` code. Fig. 4.22 shows that this block is composed by two sub-blocks: `ComputationOfNewWPMatrix` and `ExtrapolationNewWP`. The output (called wp) is a matrix which contains the new waypoint list. It is worth to notice that its dimension, in particular the number of columns, depends strictly on the ID numbers, as it will be explained in the following pages.

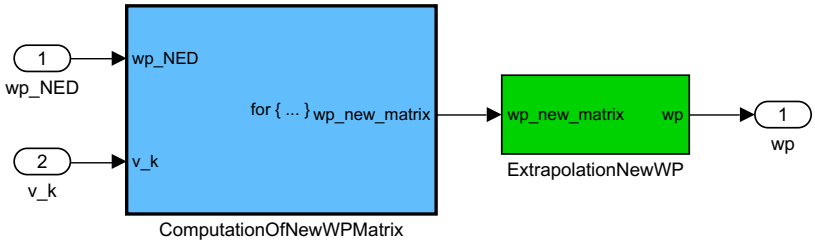

Figure 4.22: Conversion of waypoint block

4.4.3.1 Computation of the new waypoint matrix

This block is the main part of the geometry calculation since it is the one that analyzes all the waypoints and makes the necessary computations to obtain the new waypoint list. The number of `Switch Case` conditions is equal to ten because it is the same of the possible ID number, listed in Section 4.4.1.

Some `Case Action` blocks need only the current waypoint (named wp_1), other ones need also the next one (named wp_2) and finally other ones need both the next waypoint and the previous one (named wp_0) in addition to wp_1. In order to make it possible, a `For Iterator` block is used with the combination of a `Variable Selector`.

When only the change of ID number is required, simple `Simulink` schemes are sufficient while in the cases of complex geometry computations `MATLAB Function` blocks are used.

The output of all `Case Actions` must have the same dimension to ensure that the `Merge` works properly. Specifically, the maximum dimension of the output is equal to twelve because two waypoints are contained in the vector; if a `Case Action` block provides only one waypoint, a vector of zeros has to be added.

The last remark is that the first position of each new waypoint is given in relation to the kind of curve that has to be calculated in the algorithm. A specific analysis will be done in Section 4.4.4. A deep view to all the `Case Actions` is reported below.

Case 0: receiving wp_1 as input, it returns the same waypoint with the add of zeros vector.

Case 1: receiving wp_0, wp_1, wp_2, v_k and $\dot{\chi}_{des}$ as inputs, the output is the vector containing the two calculated waypoints for the fly-by maneuver during wingborne phase.

The s_{turn}, which is the distance between fly-by fix and the two calculated waypoints, is found approximating the transition as a circular-arc segment. The first passage is to calculate the α_t angle:

$$\begin{aligned} \Delta\chi &= |\chi_2 - \chi_1|, \\ \alpha_t &= |\pi - \Delta\chi|. \end{aligned} \tag{4.67}$$

Then, the radius r_c is found:

$$\begin{aligned} \dot{\chi} &= \dot{\chi}_{des}\frac{\pi}{180}, \\ r_c &= \frac{v_k}{\dot{\chi}} \end{aligned} \tag{4.68}$$

where $v_k = 25\ m/s$ and $\dot{\chi}_{des}$ is the desired track rate imposed by the user in the `main`. Finally, the formula for s_{turn} is

$$s_{turn} = \left| \frac{r_c}{\tan\frac{\alpha_t}{2}} \right|. \tag{4.69}$$

Starting from wp_1, in particular the position $\mathbf{r_1}$, the geometry calculation of the two fly-by waypoints, namely $\mathbf{r_3}$ and $\mathbf{r_4}$, is

$$\begin{aligned} \mathbf{r_3} &= \mathbf{r_1} + S_1\frac{\mathbf{r_0} - \mathbf{r_1}}{\|\mathbf{r_0} - \mathbf{r_1}\|} \\ \mathbf{r_4} &= \mathbf{r_1} + S_2\frac{\mathbf{r_2} - \mathbf{r_1}}{\|\mathbf{r_2} - \mathbf{r_1}\|}, \end{aligned} \tag{4.70}$$

where $S_2 = S_1 = s_{turn}$ and it must be less than the distances $\overline{\mathbf{r_0}\mathbf{r_1}}$ and $\overline{\mathbf{r_1}\mathbf{r_2}}$ so that $\mathbf{r_3}$ and $\mathbf{r_4}$ remain on the connection legs.

For what concerns wp_3 there is an `if` on the id_0 value so that the function is generalized and can work not only for a fly-by waypoint during wingborne but also for a "first" fly-by waypoint; this name is referred to the case in which the fly-by is the first waypoint after an hover one and so an acceleration is necessary to reach the standard wingborne speed of 25 m/s.

$$\begin{aligned} wp_3 &= [4; \mathbf{r_3}; \chi_1; \gamma_1], \qquad if \quad id_0 = 4, 6, 7 \\ wp_3 &= [0; \mathbf{r_3}; \chi_1; \gamma_1], \qquad all\ the\ other\ cases \end{aligned} \tag{4.71}$$

where the `if` is done in that way because 4, 6 and 7 are the ID number referred to the hover phase. Defining wp_4 as

$$wp_4 = [1; \mathbf{r_4}; \chi_2; \gamma_2], \tag{4.72}$$

the last passage is two build the output vector wp_{flyby} as a concatenation of wp_3 and wp_4.

The complete `MATLAB Function` is reported in Appendix A.1.

Case 2: receiving wp_1 and wp_2 as inputs, the output $wp_{flyover}$ is the vector containing wp_1 and the calculated waypoint of the end of the fly-over maneuver. The latter one is obtained with the following formula:

$$\mathbf{r_3} = \mathbf{r_1} + S\frac{\mathbf{r_2} - \mathbf{r_1}}{\|\mathbf{r_2} - \mathbf{r_1}\|} \tag{4.73}$$

where $S = 2/3\,(distance)$ and

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}. \tag{4.74}$$

As done for the previous case, wp_1 and the waypoint just calculated are concatenated in the vector $wp_{flyover}$:

$$\begin{aligned} wp_1 &= [0; \mathbf{r_1}; \chi_1; \gamma_1] \\ wp_3 &= [1; \mathbf{r_3}; \chi_2; \gamma_2]\,. \end{aligned} \tag{4.75}$$

The complete `MATLAB Function` is reported in Appendix A.1.

Case 3: before explaining the calculations, it is important to remember that the input wp_1 is the end of the RF leg while the input wp_2 is the following waypoint. The approach for which the end of the RF leg contains its ID number is a decision done by the authors of this thesis, already reported in Section 4.4.1.

The output is the vector containing the position $\mathbf{r_1}$ of waypoint wp_1 but the χ and γ angles are the ones of wp_2 since the ones contained in wp_1 represent the straight line connecting wp_0 and wp_1. A vector of zeros is added to have the correct dimension in the following way:

$$wp_{radiustofix} = [1; \mathbf{r_1}; \chi_2; \gamma_2; zeros(6, 1)]\,. \tag{4.76}$$

The complete `MATLAB Function` is reported in Appendix A.1.

Case 4: receiving wp_0, wp_1, wp_2 and the distance of vertical fly-by imposed by the user and named $vert_{flyby_{dist}}$, the output is the vector containing the two calculated waypoints for the vertical fly-by maneuver. The procedure is more or less the same of the **Case 1**; differently from this case, the distance s_{turn} is not computed with the hypothesis of circular-arc segment (and so setting the desired track rate) but it is directly fixed by the user as $vert_{flyby_{dist}}$.

Starting from the position $\mathbf{r_1}$ of waypoint wp_1, the geometry calculation of the two fly-by waypoints, namely $\mathbf{r_3}$ and $\mathbf{r_4}$, is

$$\begin{aligned} \mathbf{r_3} &= \mathbf{r_1} + S_1\frac{\mathbf{r_0} - \mathbf{r_1}}{\|\mathbf{r_0} - \mathbf{r_1}\|} \\ \mathbf{r_4} &= \mathbf{r_1} + S_2\frac{\mathbf{r_2} - \mathbf{r_1}}{\|\mathbf{r_2} - \mathbf{r_1}\|} \end{aligned} \tag{4.77}$$

where $S_1 = S_2 = vert_{flyby_{dist}}$.

For what concerns wp_3, there is an `if` on the value of the distance in the vertical axis $(z_2 - z_0)$ in the algorithm so that the `MATLAB Function` works properly both for an upwards fly-by and for a downwards fly-by:

$$\begin{aligned} wp_3 &= [2; \mathbf{r_3}; \chi_1; \gamma_1], \qquad if \quad z_2 - z_0 < 0 \\ wp_3 &= [5; \mathbf{r_3}; \chi_1; \gamma_1], \qquad if \quad z_2 - z_0 > 0. \end{aligned} \tag{4.78}$$

In addition, defining wp_4 as

$$wp_4 = [3; \mathbf{r_4}; \chi_2; \gamma_2], \tag{4.79}$$

the output vector $wp_{verticalflyby}$ is the concatenation of wp_3 and wp_4.

The complete `MATLAB Function` is reported in Appendix A.1.

Case 5: receiving wp_1 as input, the output vector $wp_{accphase}$ is equal to the input with the unique difference on the first element which is set to 4, the ID number of the acceleration straight line. For the reason exposed above, a zeros vector is added.

Case 6: receiving wp_1 as input, the only change is on the first position of the output vector $wp_{decphase}$ which has to be equal to 5, the code corresponding to the deceleration straight line. Also in this case, a zeros vector is added.

Case 7: receiving wp_1 as input, it is only changed the first position of the output vector wp_{hover} which has to be 6, the code corresponding to the hovering; also in this case, a zeros vector is added.

Case 8: receiving wp_1 as input, the manipulations done in this block are quite similar to the previous cases because it is only changed the first position of the output vector $wp_{vertdisp}$ with the correct code 2 corresponding to the vertical displacement and a zeros vector is added.

Case 9: the initial fix does not need any particular change.

4.4.3.2 Extrapolation of new waypoint list

The output of the previous block is a matrix of $(2(m+2)) \times n$ dimension where m and n are respectively the number of rows and columns of the original waypoint list, 2 is added because the track and climb angles are inserted in `WPtoNEDframe` block and finally, the multiplication by 2 is due to the fact that some `Case Action` outputs contain two waypoints.

There are two problems: the first one is that a zeros vector is added in some `Case Action` blocks so that the `Merge` one works; the second one is that the matrix $(2(m+2)) \times n$ has to be transformed into a $(m+2) \times (2n)$ since each column corresponds to a new wapoint.

The `ExtrapolationNewWP`, represented in Fig. 4.23, has this aim. The `Reshape` block let that the matrix is transformed from a $(2(m+2)) \times n$ to a $(m+2) \times (2n)$ dimension one. The `Variable Selector`, in the select columns mode, extrapolates only the useful calculated waypoints eliminating the zeros vectors. This is possible because in the `main` a variable, called ind_{vector}, is defined to select only the proper waypoints.

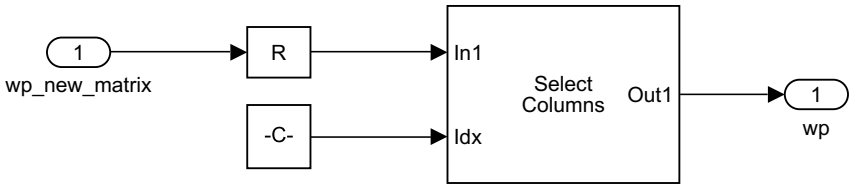

Figure 4.23: Extrapolation of new waypoint list block

4.4.4 Algorithm

The `Algorithm` block receives the new waypoint list, computed in Fig. 4.22, the vector of speed v_k and a vector containing the limits of acceleration and jerk modulus (acc_{limits}). It is a `For Iterator` and, thanks to a `Variable Selector` set in the columns mode, the current waypoint (wp_1), the next and the previous ones (respectively wp_2 and wp_0) are extrapolated.

The `TypesOfTrajectory` and `SavingDataForEachIteration` blocks, shown in Fig. 4.24, are deeply described in the next paragraphs.

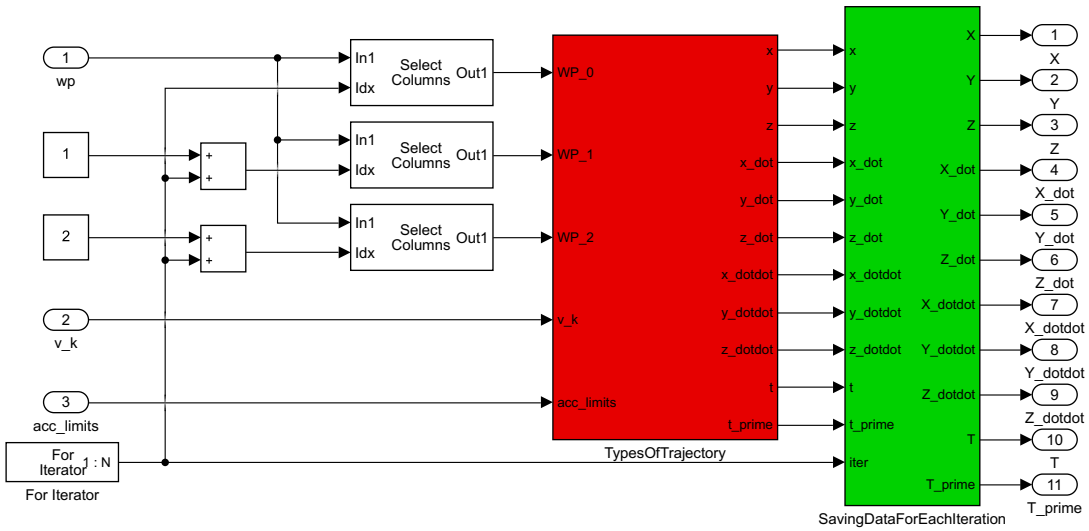

Figure 4.24: Algorithm block

The outputs are matrices of dimension equal to $(s-1) \times 1000$ where the number of rows is related to the segments generating the entire path and the

number of columns is equal to the discretization fixed by the authors. The total number of outputs is equal to eleven; nine of them are referred to the components of position vector and their first two derivatives while two of them are referred to the time. The presence of two different time matrices will be explained in the next paragraphs.

4.4.4.1 Different types of implemented trajectories

The algorithm is the core of the trajectory generator because it is the block which computes each segment of the entire path. As explained in Section 4.4.3, a new ID number is associated to the calculated waypoints in relation to the type of leg. Table 4.3 lists all kinds of trajectories.

ID Number	Trajectory
0	Horizontal straight line
1	Horizontal curve
2	Vertical acceleration and deceleration
3	Vertical curve
4	Horizontal acceleration
5	Horizontal deceleration
6	Hovering

Table 4.3: Types of trajectory used in the algorithm

As made in the `ComputationOfNewWPMatrix`, a `Switch Case` block is used to deal with all kinds of trajectories. Depending on the value of the ID number, the correct `Case Action` block is selected and, using a `MATLAB Function`, the variables $x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}, t, t'$ are computed. Finally, eleven `Merge` blocks, related to each output, choose the result of the proper `Case Action`.

A detailed dissertation of all the trajectories is reported in the next paragraphs. It is important to remember that, in all the cases in which a time scaling is not necessary, the new time vector t' is equal to the original one t.

Horizontal straight line: this trajectory is used to connect two waypoints when it is necessary to move along a straight line with a constant speed of 25 m/s during wingborne phase. Despite the name and as mentioned in Section 4.4.1, this type of leg is general because it is also possible to move with a desired climb angle γ. The mathematical formulation is a first degree polynomial and it is implemented in a `MATLAB Function` named `horizontalstraightline`.

The inputs are wp_0, wp_1 and the speed vector v_k. First of all, the extrapolation of all the waypoints information is performed. In order to compute the coefficients of the first degree polynomial, it is necessary to know the whole time frame of the straight line, named Δt. Knowing the distance in 3D space between the two

points and the constant speed, Δt is obtained using

$$\Delta t = \frac{distance}{v_k}. \tag{4.80}$$

After that, the vector of time t is built using `linspace` with one thousand values and setting Δt as the last one.

Considering the formulation of a first degree polynomial in Eq. (4.2), reported in Section 4.1.2.1, the coefficients are found using the information on position at initial and final times; evaluating Eq. (4.2) in time t, the position is easily computed.

As explained in Section 4.3, the three directions are treated separately but, using the same time vector, the synchronization is guaranteed by construction.

The complete `MATLAB Function` is reported in Appendix A.2.

Horizontal curve: this trajectory is used to connect two waypoints when a change of χ angle is requested, i.e. when a fly-by transition or a fly-over one or when a RF leg has to be performed. As in the straight line, it is possible to move with a desired climb angle γ. Fifth degree polynomial is chosen for this kind of curve as already explained in Section 4.3 and a `MATLAB Function` named `horizontalcurve` is built in order to implement the computations of this leg type.

The function inputs are the same ones of `horizontalstraightline`, namely wp_0, wp_1 and v_k. First of all, the information of the two waypoints are extrapolated. Differently from the previous case, two possibilities to compute an approximation of the time are implemented except for the fly-over maneuver. The first one is the same of Eq. (4.80) used in `horizontalstraightline` while the second one consists in approximating the curve as a perfect circular arc. The procedure is listed below:

- knowing χ of the two waypoints, the coefficients of the tangents and the perpendiculars to the trajectory are found;
- the intersection of the perpendicular lines is computed and it is the center of the circle (x_{center}, y_{center});
- knowing the radius r_c and the angle to be swept $\Delta\phi$, the circular-arc length l_{arc} is obtained using

$$\begin{aligned} r_c &= \frac{\sqrt{(x_0 - x_{center})^2 + (y_0 - y_{center})^2} + \sqrt{(x_1 - x_{center})^2 + (y_1 - y_{center})^2}}{2}, \\ \Delta\phi &= \pi - \alpha_t, \qquad \alpha_t = |\pi - \Delta\chi|, \qquad \Delta\chi = |\chi_1 - \chi_0|, \\ l_{arc} &= \Delta\phi r_c; \end{aligned} \tag{4.81}$$

- Δt is calculated with Eq. (4.80) where $distance = l_{arc}$ and $v_k = 25\ m/s$.

The vector of time is split into one thousand values thanks to `linspace`.

Once the time is constructed, the values of position, speed and acceleration, at initial and final instants, are needed to calculate the coefficients of the fifth degree polynomial. The position is simply extrapolated from the input vectors while the acceleration is set to 0 m/s^2 in the three directions, both at the initial time and at the final one. For what concerns the speed, the modulus is equal to 25 m/s and the values of the components are computed rotating the results from the K frame to the O one. The rotation matrices at the initial and final times are reported in Eq. (3.14) where χ and γ are equal to χ_0 and γ_0 at the initial time and to χ_1 and γ_1 at the final time.

Knowing the six boundary conditions for every direction, the coefficients are easily computed and so, every value of position, speed and acceleration is evaluated in the one thousand time instants using Eq. (4.5) shown in Section 4.1.2.1.

The fifth degree polynomial guarantees the continuity of the variables already explained in Section 4.3 but it does not satisfy the constraint of constant velocity. Since the latter one is a prescribed requirement, a further implementation is needed. As suggested in [22], a scaling in time is the solution to reach this goal. In particular, Section 5.2 of the book provides a general brief explanation of what does it mean. Given a trajectory

$$q = q\left(t\right), \tag{4.82}$$

it is possible to modify the profiles of velocity, acceleration, etc..., by considering a new time variable t' related to t by means of a strictly increasing function $t = \sigma\left(t'\right)$. The consequence is that:

$$\tilde{q}\left(t'\right) = \left(q \circ \sigma\right)\left(t'\right) = q\left(\sigma\left(t'\right)\right) \tag{4.83}$$

while the velocity and the acceleration are

$$\begin{aligned} \dot{\tilde{q}}\left(t'\right) &= \frac{dq\left(\sigma\right)}{d\sigma}\frac{d\sigma\left(t'\right)}{dt'} \\ \ddot{\tilde{q}}\left(t'\right) &= \frac{dq\left(\sigma\right)}{d\sigma}\frac{d^2\sigma\left(t'\right)}{dt'^2} + \frac{d^2q\left(\sigma\right)}{d\sigma^2}\left(\frac{d\sigma\left(t'\right)}{dt'}\right)^2. \end{aligned} \tag{4.84}$$

Therefore, the time derivatives of $\tilde{q}\left(t'\right)$ can be changed according to the needs by properly defining the function σ.

These equations are the starting point but useful information on how to define σ and its derivatives in order to maintain a constant speed are taken by an example reported in Section 9.4 of [22]. This part of the book analyzes the motion law of multi-dimensional trajectories and the method to have a constant velocity; the same formulae of u' and u'', the first and second derivatives of the motion law with respect to time, are used for the derivatives of σ. Their expressions are reported

below and their demonstration is in [35].

$$\begin{aligned} \frac{d\sigma\left(t'\right)}{dt'} &= \frac{v_c}{v_{mod}\left(t\right)} \\ \frac{d^2\sigma\left(t'\right)}{dt'^2} &= -v_c^2 \frac{\mathbf{v}\left(t\right)^T \cdot \mathbf{a}\left(t\right)}{v_{mod}\left(t\right)^4} \end{aligned} \tag{4.85}$$

where $v_c = 25\ m/s$ is the constant speed in the K frame, $\mathbf{v}\left(t\right)$ is the speed vector in time, v_{mod} is its modulus and $\mathbf{a}\left(t\right)$ is the acceleration vector in time. $\frac{d\sigma(t')}{dt'}$ and $\frac{d^2\sigma(t')}{dt'^2}$ are called λ_v and λ_{acc} in the `horizontalcurve`. All these quantities are computed with the fifth degree polynomial before the scaling in time. Thanks to Eq. (4.85) and using Eq. (4.84), the new values of speed and acceleration in every direction can be calculated without changing the position values. Finally, also the new time vector t' is computed thanks to

$$t' = \frac{t}{\lambda_v\left(t\right)}. \tag{4.86}$$

Another request, which has to be satisfied, is that the maximum value of $\dot{\chi}$ must remain under a value fixed by the user, namely $\dot{\chi}_{max}$. This goal can not be achieved thanks to another scaling in time, because it may ruin the results which maintain a constant speed, but there are three other possible ways.

- The first one is a good planning of the flight; if the waypoint list has proper distances and angles between the points, the overcome of the $\dot{\chi}$ limit can be avoided.
- The second one concerns only the fly-by maneuver and consists in reducing the value of $\dot{\chi}_{des}$ used to compute the s_{turn} distance.
- The third one, which is implemented in the `MATLAB Function` named `saturationchidot` and which has the maximum value of $\dot{\chi}$, the speed and the output of `horizontalcurve` as inputs, consists in replacing the fifth degree polynomial points in the instants in which the limit is exceeded with a circular arc.

The last option allows to maintain the value of $\dot{\chi}$ constant and equal to the maximum admissible value during the circular arc. As expected, hypothesizing that the first point remains in the same position, the final one changes and all the trajectory after the curve is shifted; the insertion of a further constraint plus to the already existing one on the speed generates this constraint on the geometry. If there are more than one curve along the trajectory which overcome the limit, the shifts are added one to the other. It could be seen as a problem but it can also be an automatic way of giving a new waypoint list which can be followed keeping into account the constraints on constant speed and maximum value of $\dot{\chi}$.

In order to place the circular arc in the right position and to not create discontinuities, the flatness relationships are used to compute $\dot{\chi}$, χ and γ along the fifth degree polynomial curve. First of all, making a comparison between the values of $\dot{\chi}$ and the limit, the points to be replaced are found. After that, using the formulation of Eq. (4.20) and the other related equations, the new values of position, speed and acceleration in the K frame are computed. These computations concern only the x and y directions, because the circular arc formulation has meaning only in the 2D space. Finally, a rotation of these results in the O frame and also the connection between the first part of fifth degree polynomial, the circular arc and the second part of fifth degree polynomial are necessary. For what concerns the z direction, no differences between the new trajectory and the previous one are implemented. The function is able to understand whether the limit is overcome or not thanks to an `if` condition and, as a consequence, if it is necessary to make all these calculi or maintain the results of `horizontalcurve` unchanged.

Both the `horizontalcurve` and the `saturationchidot` are reported in Appendix A.2.

Vertical acceleration and deceleration: this trajectory is a vertical straight line used for a change in altitude during the hover phase. The choice is to use a fifth degree polynomial since it is necessary to impose the initial and final conditions on the speed and acceleration to guarantee the continuity. `verticalaccdec` is the `MATLAB Function` in which the algorithm is implemented.

The required inputs are wp_0, wp_1, wp_2 and the speed vector v_k. After having extrapolated the data from the waypoints, the first passage is to define the absolute value of the initial and final speeds. Since the function has to be general, an `if` condition is used to associate the correct values to the speeds. Specifically, there are three different cases whose description is reported below. It is worth to notice that the first two ones are grouped in the same option of the `if` which checks the ID number value of the next and previous waypoints respectively. The choice between them depends on the sign of $z_1 - z_0$; remembering that the positive z-axis points downwards, the displacement is upwards if the difference $z_1 - z_0$ is negative and vice versa.

- The first one is a vertical displacement upwards before a vertical fly-by waypoint; since it is a movement along the negative z-axis, the speed has a negative sign. For these reasons, the initial speed is equal to 0 m/s while the final one is set to -2 m/s so that the continuity with the following segment is guaranteed. As a matter of fact and as it will be explained in its paragraph, it is decided that the vertical fly-by maneuver has to be completed at a constant speed modulus equal to 2 m/s.

- The second one is the opposite case of the previous one, namely a vertical displacement downwards after a vertical fly-by waypoint. Since the aircraft has to move downwards, the speed has a positive sign. In particular, the initial speed is set to 2 m/s while the final one is set to 0 m/s.

- The third one concerns a general change in altitude between two waypoints with speed equal to 0 m/s, for example two hovering waypoints. So, the initial and final speeds are set equal to 0 m/s.

Since the displacement is only vertical, the speed is in the z direction and so the x and y components have to be set to 0 m/s. For what concerns the acceleration, the initial and final values are equal to 0 m/s^2 in order to guarantee the continuity in the entire path.

It is also important to define a mean speed to compute Δt. In the first two cases, it is equal to the absolute value of the mean between the initial and final speeds while in the third one, the value has to be decided by the user because the formula used for the other cases, providing 0 m/s, is useless. The authors fix it to 1 m/s since this value allows to not overcome the maximum speed and acceleration (equal to 2 m/s and 2 m/s^2) for the performed flight plan. If there are greater vertical displacements, these limits may be violated with $v_{mean} = 1$ m/s but the user can change it to avoid the problem.

The theoretical Δt is computed in the same way of the horizontal straight line and curve. After having calculated the distance between the two waypoints, Δt is found using Eq. (4.80). Finally, the time vector t between zero and Δt is defined with a discretization of one thousand points.

The last passage is the determination of the polynomial coefficients. This procedure is realized for each direction in a similar way to the horizontal curve; imposing the initial and final conditions on position, speed and acceleration, it is possible to find the formulae for a_0, a_1, a_2, a_3, a_4 and a_5 for x, y and z directions. By using Eq. (4.5) and deriving it until to the second order, the vectors of position, speed and acceleration in all the directions are obtained.

The complete script of `verticalaccdec` is reported in Appendix A.2.

Vertical curve: this trajectory connects the two waypoints of a vertical fly-by maneuver. As in the case of horizontal curve, the chosen polynomial is the fifth degree one since the continuity of position, speed and acceleration is required. The `MATLAB Function` is named `verticalcurve`.

The inputs are wp_0, wp_1 and the speed vector v_k. Firstly, the information about the waypoints are extrapolated from the input vector. As already mentioned in the previous paragraph, the speed of this kind of curve is set equal to 2 m/s for all maneuvers.

The theoretical Δt is computed following the same method of the horizontal straight line; hypothesizing a direct connection between the two waypoints, Δt is computed with Eq. (4.80). The time vector t between zero and Δt is defined with a discretization of one thousand points.

Before computing the coefficients, it is necessary to define the values of the components of the initial and final speeds. An `if` condition on the sign of $z_1 - z_0$ is introduced to characterize a vertical curve upwards and a downwards one, respectively negative and positive signs. In the first case, the initial speed is only on the z direction and it is equal -2 m/s; the final speed has to be found

by rotating the speed into the K frame using the rotation matrix of Eq. (3.14) imposing χ and γ equal to χ_1 and γ_1 respectively. In the second case, the initial speed is found in a similar way to the final speed of the previous one; the only difference is that χ and γ are equal to χ_0 and γ_0. The final speed is on the z direction and it is equal to 2 m/s.

For what concerns the acceleration, the initial and final values are set to 0 m/s to guarantee the continuity with the previous and next segments.

The computation of the polynomial coefficients is basically the same of the horizontal curve and vertical acceleration and deceleration, already explained in their paragraphs. Evaluating Eq. (4.5) and its derivatives in t, the position, speed and acceleration are obtained.

Since the speed is not constant for the entire curve, a scaling in time is required as in the horizontal curve. The passages are the same: λ_v and λ_{acc} are computed thanks to Eq. (4.85) where $v_c = 2\ m/s$; finally, the new vectors of speed, acceleration and time are obtained by using Eq. (4.84) and Eq. (4.86).

The script `verticalcurve` is reported in Appendix A.2.

Horizontal acceleration: this leg is the segment connecting two waypoints in which the aircraft accelerates. The mathematical formulation is taken from the double S trajectory, explained in Section 4.1.2.4. The calculations are implemented in the `MATLAB Function` named `horizontalacceleration`.

The function inputs are wp_0, wp_1, v_k and acc_{limits}. The last one is a vector containing the maximum values of acceleration and jerk. First of all, the data from the waypoints are collected. The acceleration is the phase in which the aircraft increases its speed from 0 m/s or 2 m/s to the standard wingborne speed of 25 m/s. The two possible initial speeds are related to the fact that this phase is performed not only from a point with $v = 0\ m/s$ but also after a vertical fly-by. To deal with this distinction, an `if` condition on the ID number of wp_0 is inserted:

- if $id_0 = 3$, the initial speed is equal to 2 m/s;
- in the other cases, the initial speed is set to 0 m/s.

On the other hand, the final speed is always equal to 25 m/s.

It is important to remind that a minimum distance between the two waypoints is necessary to perform the acceleration with a fixed value of maximum acceleration (named acc_{max}) and with desired initial and final speeds; this issue has to be taken into account in the trajectory planning.

The first step is to define the variable q whose initial and final values are equal to 0 and *distance* between the two waypoints. Then, the reference times are found referring to the following equations:

$$\begin{aligned} T_{j1} &= \frac{acc_{max}}{j_{max}} \\ T_a &= T_{j1} + \frac{v_{k1} - v_{k0}}{acc_{max}} \\ T_v &= \frac{q_f - q_i}{v_{k1}} - \frac{T_a}{2}\left(1 + \frac{v_{k0}}{v_{k1}}\right) \end{aligned} \tag{4.87}$$

where q_i and q_f are the initial and final values of q while v_{k0} and v_{k1} are the initial and final speed modulus. If *distance* is equal to the minimum one, it is important to remember that $T_v = 0$. The time vector t is built with one thousand points sweeping from zero to the total time $T_a + T_v$. After having divided t in the following phases:

- increasing acceleration with constant jerk,
- constant acceleration with null jerk,
- decreasing acceleration with constant jerk,
- constant speed with null acceleration,

the function uses Eq. (4.12), Eq. (4.13), Eq. (4.14) and Eq. (4.15) of a double S trajectory to compute q, $\dot{q}$ and $\ddot{q}$. Concatenating the results of each phase, the complete vectors of q and its derivatives are obtained. The last passages are the rotation from the K frame to the O frame thanks to Eq. (3.14) where χ and γ are equal to χ_1 and γ_1 and the addition of the initial position $\mathbf{r_0}$ since $q_i = 0$.

The complete script of `horizontalacceleration` is reported in Appendix A.2.

Horizontal deceleration: this trajectory connects two waypoints when a deceleration is needed. As for the acceleration, the mathematical formulation is the one of double S in Section 4.1.2.4. The `MATLAB Function` is named `horizontal deceleration`.

The inputs are wp_0, wp_1, wp_2, the speed vector v_k and the vector of acceleration and jerk limits (named acc_{max} and j_{max} respectively).

Firstly, the useful data of each waypoint are extrapolated from the inputs. The deceleration is the phase in which the aircraft decreases its speed from the wingborne speed of 25 m/s to a value which can be 2 m/s or 0 m/s. The justification of the double choice is that the aircraft needs not only to reach the hovering phase (characterized by $v = 0$ m/s) but also a point with speed equal to 2 m/s in order to do the vertical fly-by maneuver. As a consequence, the initial speed is always equal to 25 m/s while for the final speed, an `if` condition on the ID number value of wp_2 is inserted:

- if $id_2 = 3$, it means that the following segment is a vertical curve of fly-by maneuver and so the final speed is imposed equal to 2 m/s;
- in the other cases, the deceleration is completed until the null speed is reached and so, the final speed is set equal to 0 m/s.

Before proceeding with the description of the algorithm, it is worth to notice that the deceleration can be completed if and only if a minimum distance between the two waypoints exists. The reason is analogous to the acceleration phase, i.e. there is a precise value of distance needed to change the speed considering the constraints on acceleration and jerk. In order to avoid errors in the simulation,

it is fundamental to plan correctly the trajectory; the deceleration starting and ending waypoints need this minimum distance.

The procedure which has to be followed to generate this kind of trajectory is similar to the acceleration one. After having computed the real distance between the two waypoints (called *distance*), it is introduced a new coordinate q whose initial and final values are equal to zero and to *distance* respectively. Then, all the reference times are calculated using

$$\begin{aligned} T_{j2} &= \frac{acc_{max}}{j_{max}} \\ T_d &= T_{j2} + \frac{v_{k0} - v_{k1}}{acc_{max}} \\ T_v &= \frac{q_f - q_i}{v_{k0}} - \frac{T_d}{2}\left(1 + \frac{v_{k1}}{v_{k0}}\right), \end{aligned} \tag{4.88}$$

remembering that, if the real distance is perfectly equal to the minimum one, there is no part of trajectory with constant speed ($T_v = 0$).

The time vector t between zero and the total time $T_d + T_v$ is determined with the discretization of one thousand points. It has to be split in each sub-segment:

- increasing deceleration with constant jerk,
- constant deceleration with null jerk,
- decreasing deceleration with constant jerk,
- constant speed with null acceleration.

After having built position, speed and acceleration vectors of each sub-segment using the equations of constant speed (Eq. (4.15)) and of deceleration (Eq. (4.16), Eq. (4.17) and Eq. (4.18)), they are concatenated to obtain the vectors for the entire phase, namely q, $\dot{q}$ and $\ddot{q}$.

The last passage is the transformation of the results from the K frame to the local NED frame; to make it possible, q, $\dot{q}$ and $\ddot{q}$ are rotated by the matrix in Eq. (3.14) where χ and γ are equal to χ_1 and γ_1. Finally, the initial position is added to the position vector since the initial value of q was previously set to zero.

The script of the `horizontaldeceleration` is reported in Appendix A.2.

Hovering: the hovering point is performed by the simplest `MATLAB Function` between all the previously explained ones, which is named `hovering`.

The function receives only wp_1 and t_{hover} as inputs, where the latter one is the duration of the hovering and can be set by the user in the `main` code. Firstly, the information of the waypoint are taken. Then, the position is built as one thousand elements vectors in x, y and z directions maintaining the values of the waypoint position. Also speed and acceleration are constructed similarly but with a constant value equal to 0 m/s and 0 m/s^2 in every direction. At the end, the corresponding time vector is built setting t_{hover} as final time and using `linspace` in order to discretize the vector into one thousand values.

The `hovering` script is reported in Appendix A.2.

4.4.4.2 Saving data block

This block is fundamental because the problem of saving data after each iteration is hard to treat in `Simulink` software. Since it is impossible to use a dynamic vector which increases its dimension after each iteration, it is decided to save the data in a matrix of a predefined dimension as mentioned in the introduction to the section. In order to do this, the `Assignment` block, which receives the initialization matrix, the current vector of results and the current iteration index as inputs, is used for each component of position, speed and acceleration and the time vectors (Fig. 4.25).

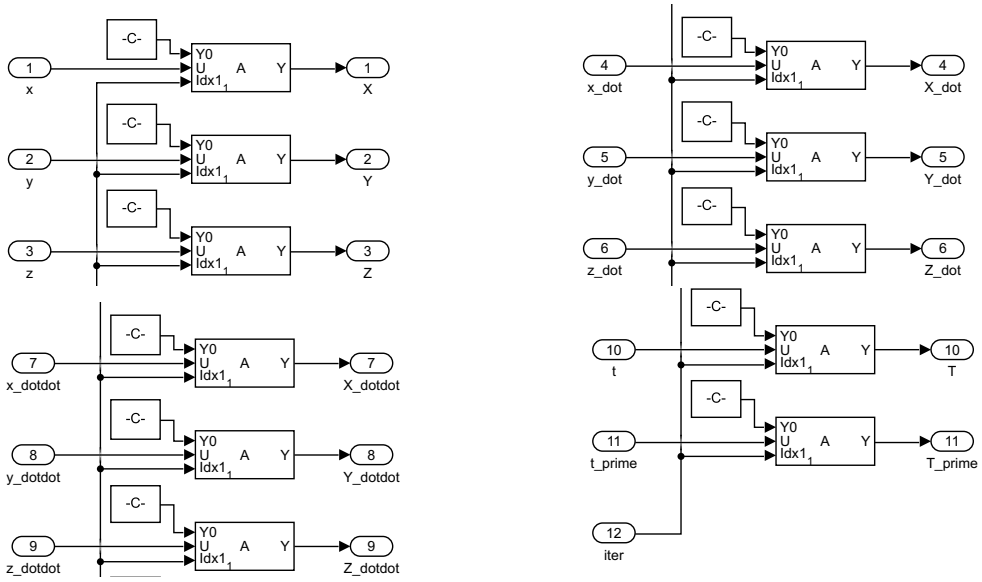

Figure 4.25: Saving data block

4.4.5 Final manipulation

The `FinalManipulation` block, shown in Fig. 4.13, performs the last passages to reach final results. Since the algorithm provides the outputs in matrix form, due to the saving issues explained in Section 4.4.4, `FromMatrixToVector` allows to pass from matrix results to vector ones; then, `ShiftInCaseOfSaturation` adjusts results with respect to saturation of $\dot{\chi}$ with circular arcs. Finally, `EliminationOf Duplicates` eliminates the duplicates inserted in the trajectory by construction.

4.4.5.1 Transformation of the results from matrix to vector form

In `FromMatrixToVector`, `Transpose` and `Reshape` blocks are used respectively to invert columns and rows, and to give a 1D array as output. For what concerns the times, the manipulation is more complicated. The time vectors do not have to restart from zero value after each leg is completed. The scheme using `Variable Selector` in select columns mode, `Cumulative Sum` in columns

mode, `Variable Selector` in select rows mode and `For Iterator` provides the right quantity to add at every time value in order to avoid this problem.

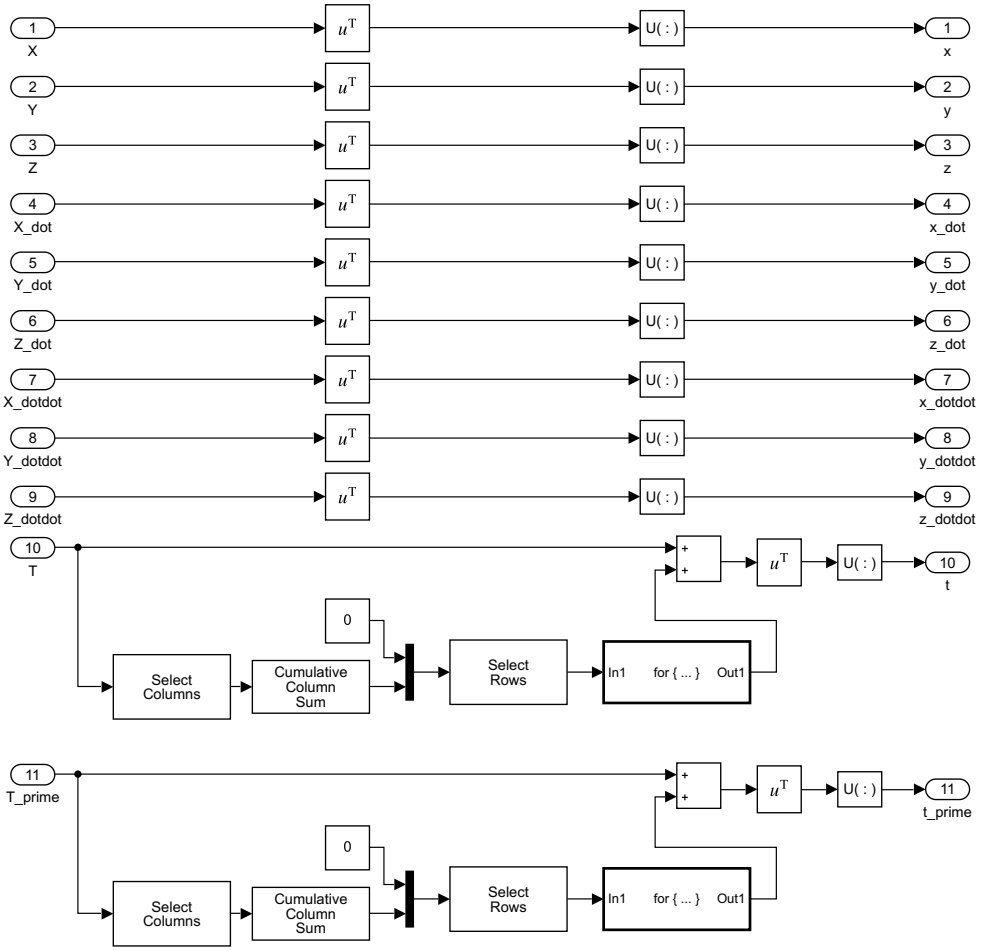

Figure 4.26: Passage from matrix results to vector ones

4.4.5.2 Shift in case of saturation

When the saturation of $\dot{\chi}$ is activated, the position is no longer continuous in time, as explained in Section 4.4.4. For this reason, a shift of x, y and z results is necessary. A `MATLAB Function` named `ShiftInCaseOfSaturation` is used to reach this purpose.

It has x, y and z as inputs and it provides their new values $x_{shifted}$, $y_{shifted}$ and $z_{shifted}$ as outputs. The shift consists in positioning the starting point of each leg exactly in the end point of the leg before. This process is performed using a `For Iterator`, starting from the first waypoint and arriving to the last one, and changing all the values in a cascade iterative way. With this method, x, y and z change only if there is the necessity, because, if the saturation of $\dot{\chi}$ is not activated, the for cycle does not change the results.

The complete `MATLAB` script is shown in Appendix A.3.

4.4.5.3 Elimination of the duplicates

The continuity between each leg is obtained by the algorithm imposing that the final point of one leg is equal to the first one of the following leg. This procedure causes the presence of duplicates at every legs connection. The block in Fig. 4.27 allows to eliminate the duplicates using `Variable Selector` in select columns mode.

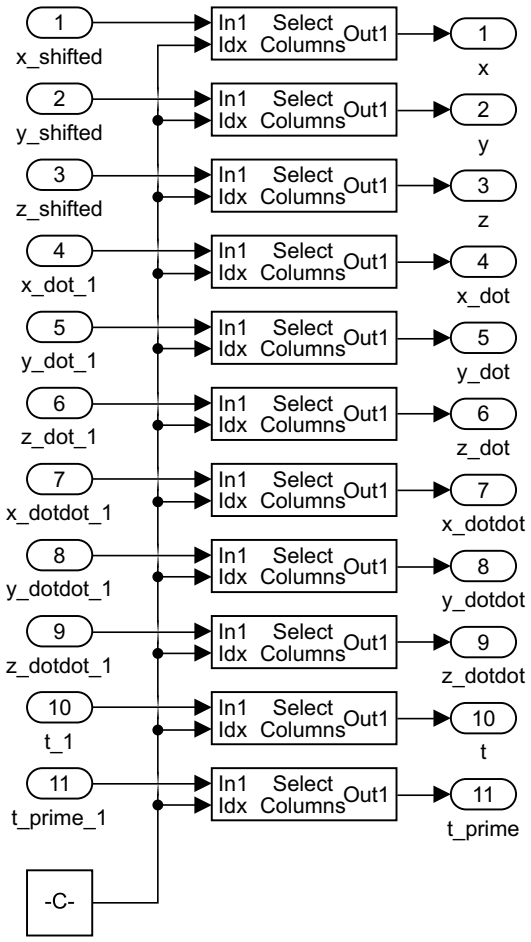

Figure 4.27: Elimination of the duplicates

Chapter 5

Results and analysis

This chapter is the core of the thesis because the results of the trajectory generator, described in Chapter 4, are presented and analyzed. Since there are a lot of requirements to be fulfilled, four flight plans are chosen in order to show all the possible maneuvers for both phases and to analyze the results in a more specific way. They are summarized in Table 5.1.

Flight plan n.	Name	Reference
1	Wingborne with All Possible Maneuvers	Section 5.1
2	Wingborne with Climb Paths	Section 5.2
3	Wingborne with Saturation of $\dot{\chi}$	Section 5.3
4	Complete Flight Plan: Hover and Wingborne	Section 5.4

Table 5.1: Flight plans

The objectives of each flight plan are listed below.

Flight plan n.1 $\rightarrow$ Verify the feasibility of TF leg with fly-by and fly-over transitions and RF leg.

Flight plan n.2 $\rightarrow$ Verify whether the request of climb path can be satisfied by the implemented algorithm.

Flight plan n.3 $\rightarrow$ Study the methods of $\dot{\chi}$ saturation.

Flight plan n.4 $\rightarrow$ Verify whether it is possible to generate a trajectory with wingborne and hover phases.

As anticipated in Section 3.2, the origin of the O frame is the FSD institute whose coordinates are reported in Table 5.2.

lat [*deg*]	*long* [*deg*]	*h* [*m*]
48.266185	11.668320	478

Table 5.2: Coordinates of the FSD institute

A general consideration on fly-by transition must be taken into account before proceeding with the analysis of the flight plans. As mentioned in Section 4.2.3.1, the trajectory has to remain into a containment area during the entire maneuver. To this aim, a fly-by transition between two straight lines with $\Delta\chi = 30\ deg$ is considered as example. The two cases of fly-by transition are shown in Fig. 5.1 and Fig. 5.2, using respectively Eq. (4.80) and Eq. (4.81) to compute the Δt.

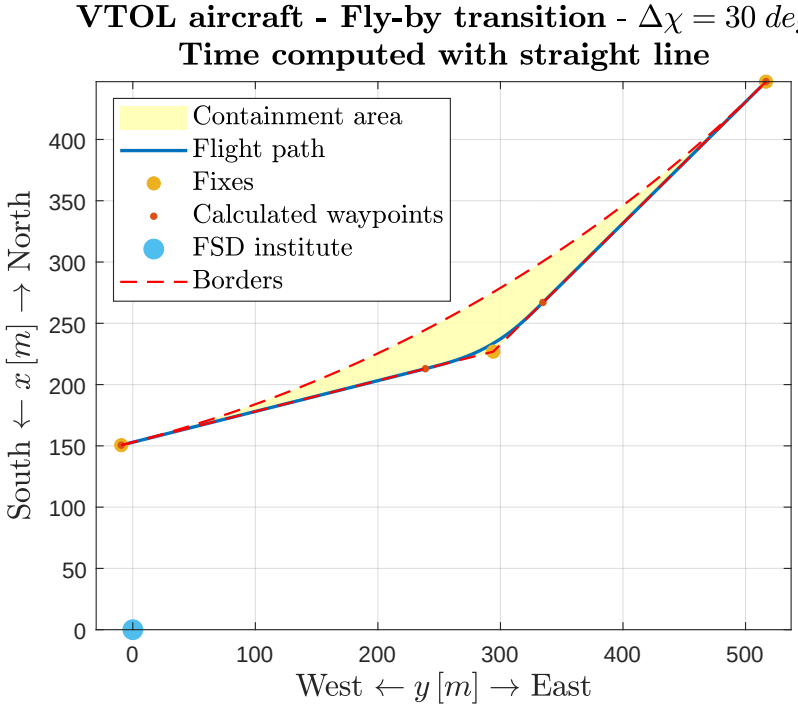

Figure 5.1: Containment area of fly-by transition using time computed with straight line

Although there are small differences in the geometry of the two cases, both trajectories lay into the containment area and so it is proved that both methods to compute Δt are valid. The results are expected by construction because the two waypoints of the beginning and the end of transition are calculated in order to remain in the connecting legs. As a consequence, the imposition of the right boundary conditions guarantee the fulfillment of the containment area request. From now on, it is decided to compute Δt with the method of straight line because it is more general and valid also for the fly-over transition. Despite the examples, presented in Fig. 5.1 and Fig. 5.2, concern maneuvers with a prescribed $\Delta\chi$, the reached conclusion can be extended to any kind of fly-by.

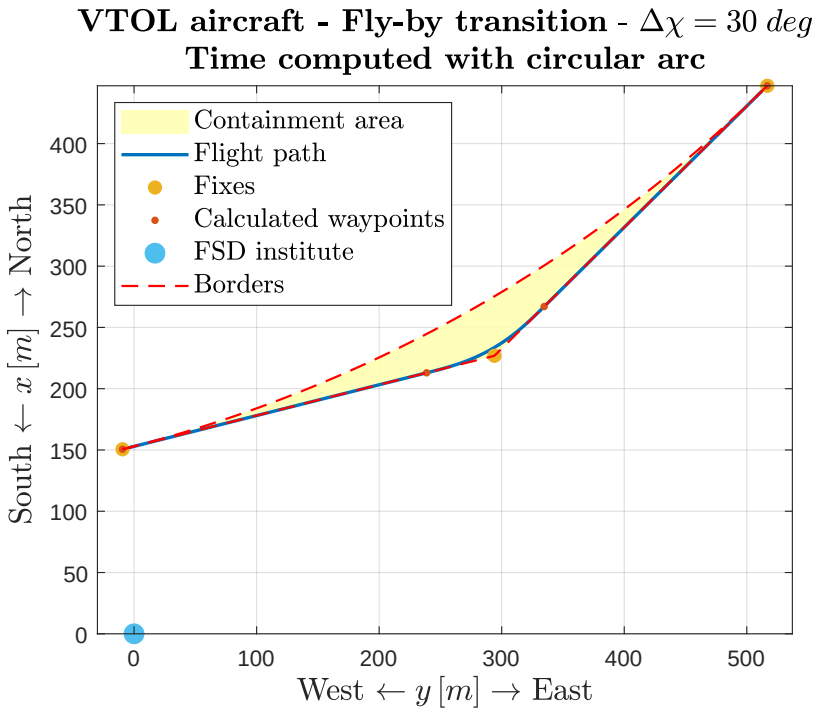

Figure 5.2: Containment area of fly-by transition using time computed with circular arc

5.1 Wingborne with all possible maneuvers

Since the objective of this flight plan is to study all the possible maneuvers of the wingborne phase, the list of waypoints, reported in Table 5.3, is defined in order to accomplish this task.

n. [−]	ID Number [−]	*lat* [*deg*]	*long* [*deg*]	*h* [*m*]
1	9	48.267539	11.668193	518
2	0	48.268225	11.672281	518
3	3	48.273412	11.673054	518
4	1	48.274658	11.668848	518
5	2	48.272274	11.664337	518
6	0	48.268236	11.667856	518

Table 5.3: Waypoint list - Flight plan n.1

First of all, the trajectory geometry is analyzed thanks to the following graphs. The representation in the 3D space is reported in Fig. 5.3.

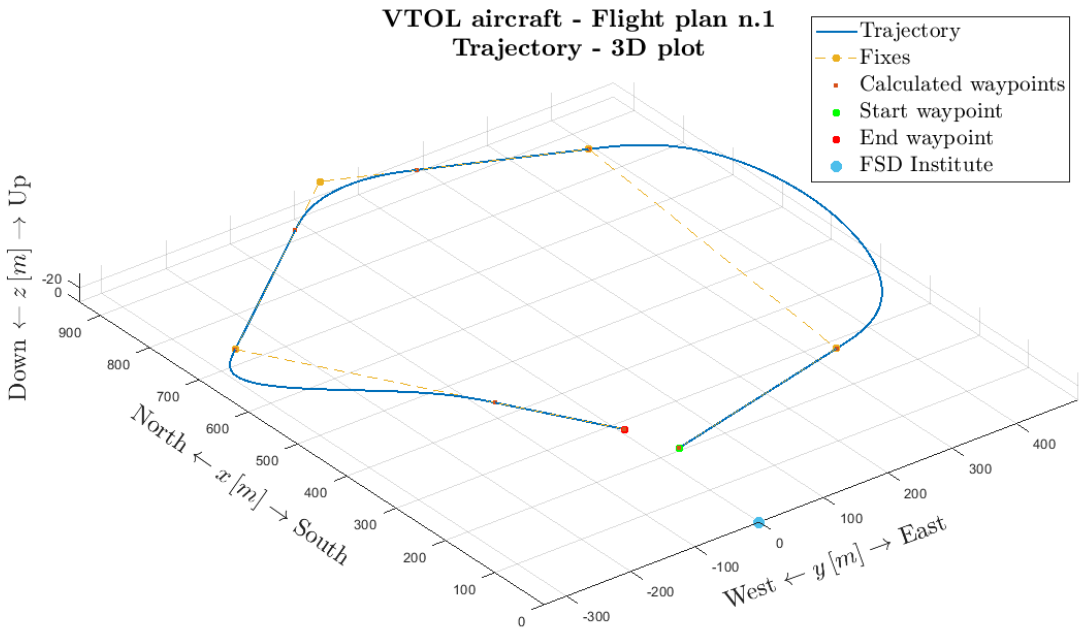

Figure 5.3: 3D trajectory - Flight plan n.1

Since all the waypoints are at the same altitude, a flat trajectory is expected. A representation in 2D planes, namely YX, XZ and YZ ones, is provided. Looking at Fig. 5.4 and Fig. 5.5, it is evident that the variation of altitude is not present.

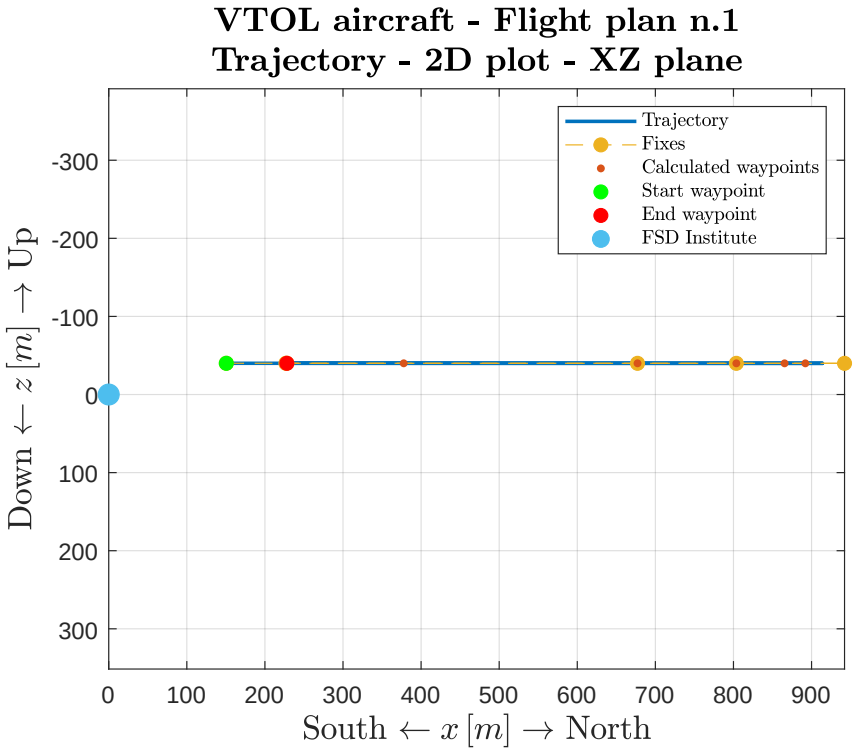

Figure 5.4: 2D trajectory in XZ plane - Flight plan n.1

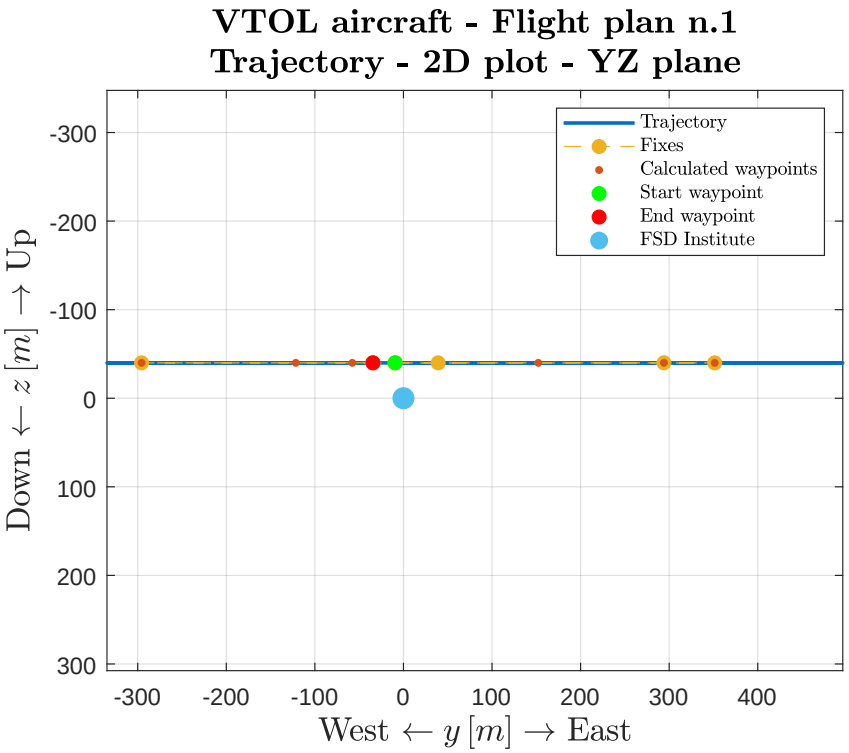

Figure 5.5: 2D trajectory in YZ plane - Flight plan n.1

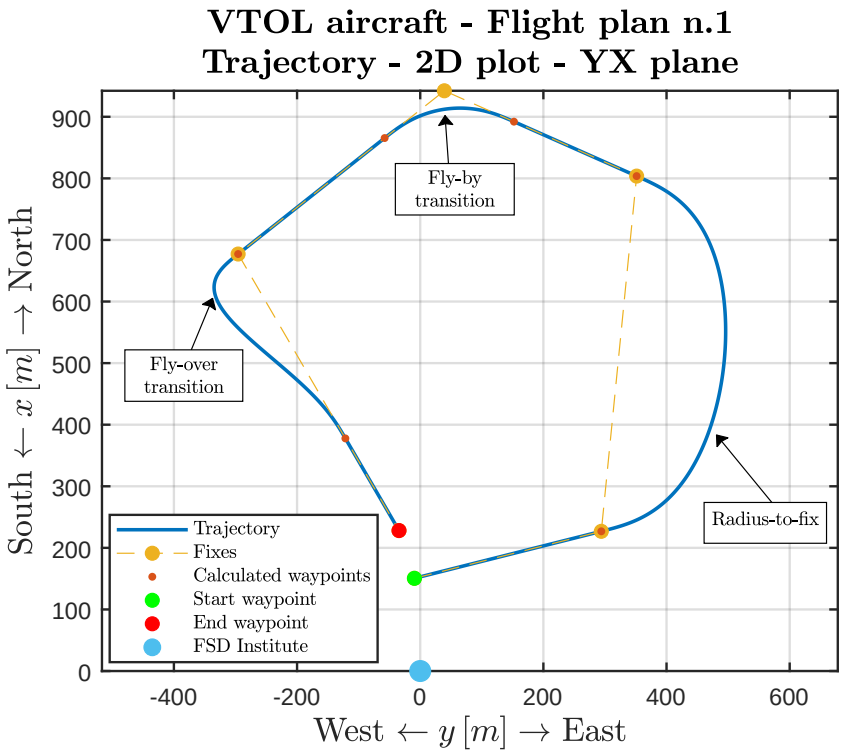

Figure 5.6: 2D trajectory in YX plane - Flight plan n.1

In Fig. 5.6, the trajectory is represented from above. It is possible to notice that all the available maneuvers in wingborne phase are performed:

- the first one is the RF leg which permits to connect two fixes with a curve;
- the second one is a fly-by turn between two straight legs which allows to have a smooth transition passing near the determined fix through the anticipation of the turn, as prescribed by [34];
- the last one is the fly-over transition for which the aircraft passes over the fix and, after that, it starts the turn to reach the next waypoint.

The information about the calculated waypoints are reported in Table 5.4.

n. [−]	ID number [−]	x $[m]$	y $[m]$	z $[m]$	χ $[deg]$	γ $[deg]$
1	9	150.5710	−9.4291	−39.9982	0	0
2	0	226.8649	294.0794	−39.9892	75.8897	−0.0017
3	1	803.6865	351.4342	−39.9396	293.9284	−0.0016
4	0	892.0771	152.2364	−39.9336	293.9284	−0.0016
5	1	865.4770	−57.7688	−39.9380	231.6339	0.0036
6	0	677.1325	−295.6894	−39.9572	231.6339	0.0036
7	1	377.7646	−121.5292	−39.9829	149.8109	0.0043
8	0	228.0807	−34.4491	−39.9958	149.8109	0.0043

Table 5.4: New waypoint list - Flight plan n.1

The values of the ID numbers reflect the fact that a wingborne trajectory can be built as a sequence of horizontal straight lines and curves and so first and fifth degree polynomials. The small variations along the z direction and in the value of γ are caused by the fact that the Earth is not flat and so, deciding a precise origin of the O frame, the altitude changes when the aircraft moves far from it although the flight is horizontal.

The second passage is the analysis of the time histories of flat outputs x, y and z and their derivatives until the second order to verify if the requirement on continuity is satisfied. It is worth to notice that the time is named t' because it is the one obtained after the scaling in time, whose usefulness will be analyzed later.

The time history of the components of position, speed and acceleration are respectively shown in Fig. 5.7, Fig. 5.8 and Fig. 5.9.

All these quantities are characterized by a continuous evolution in time and so, the combination of first and fifth degree polynomials, imposing the right boundary conditions, can fulfill this requirement for the wingborne phase.

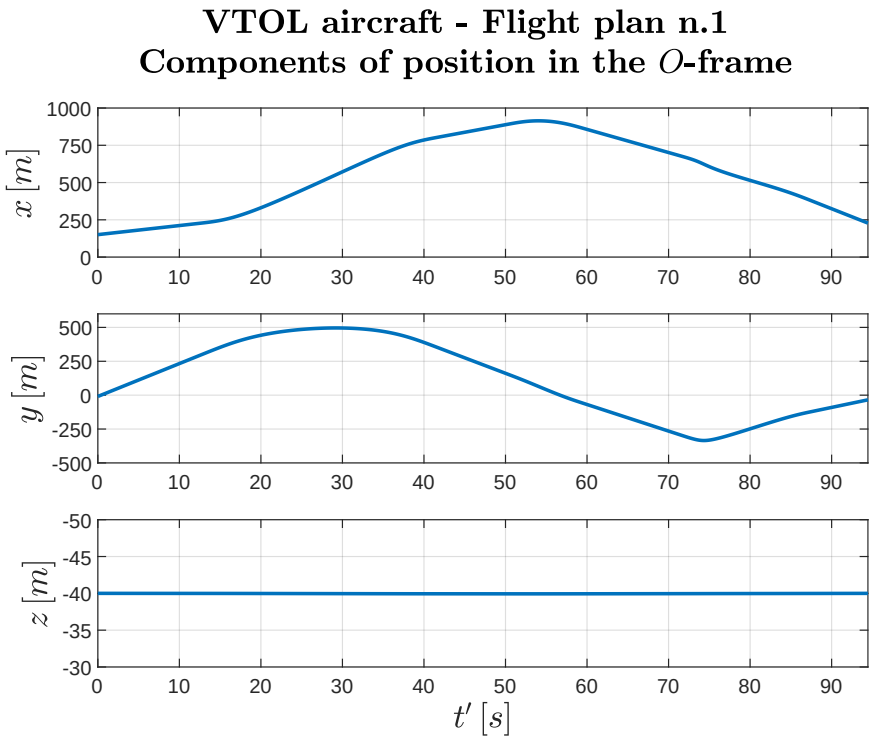

Figure 5.7: Time history of position components - Flight plan n.1

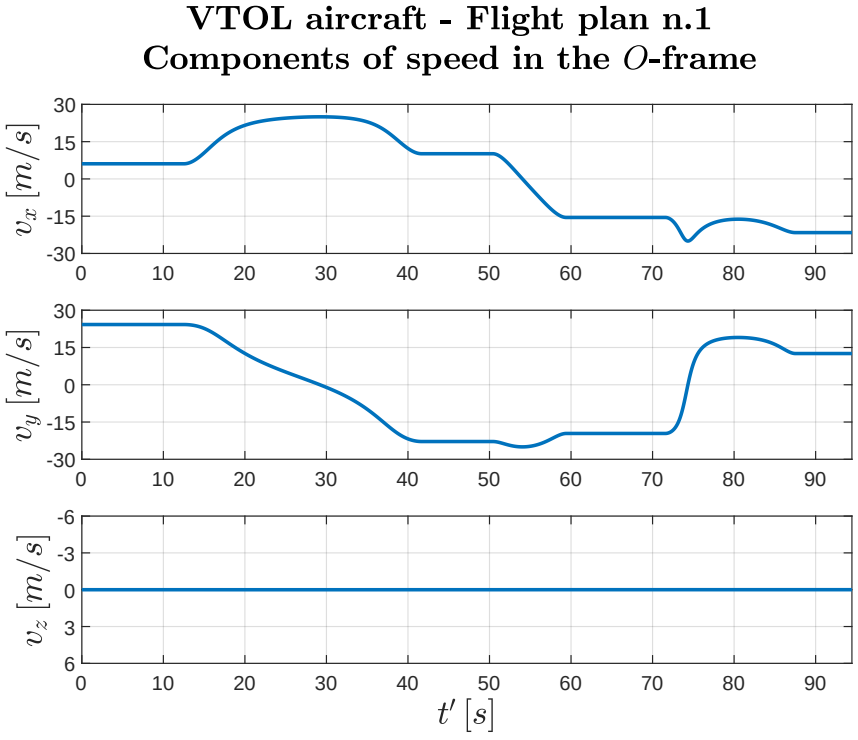

Figure 5.8: Time history of speed components - Flight plan n.1

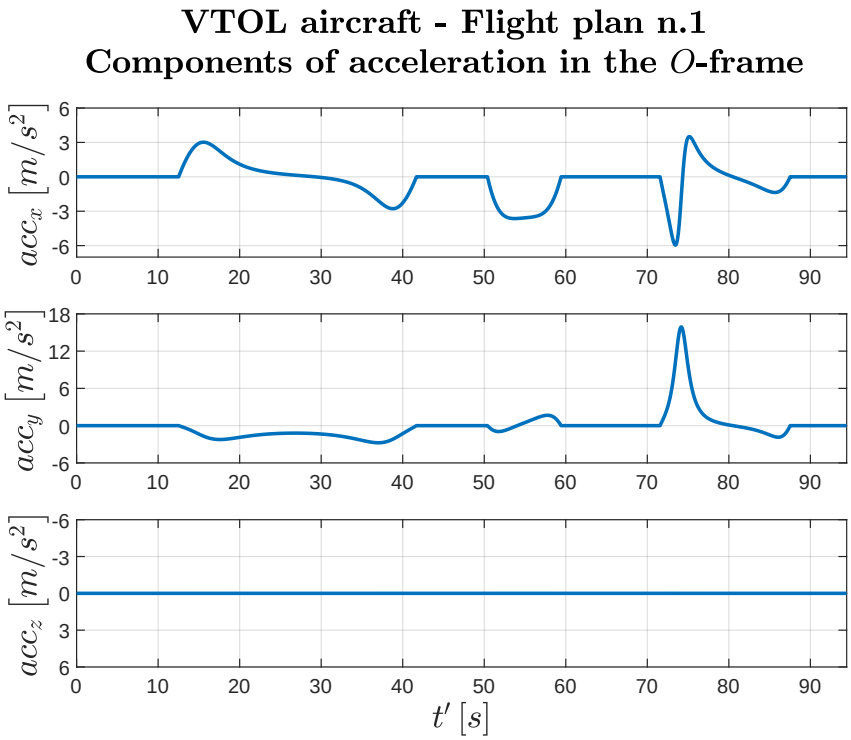

Figure 5.9: Time history of acceleration components - Flight plan n.1

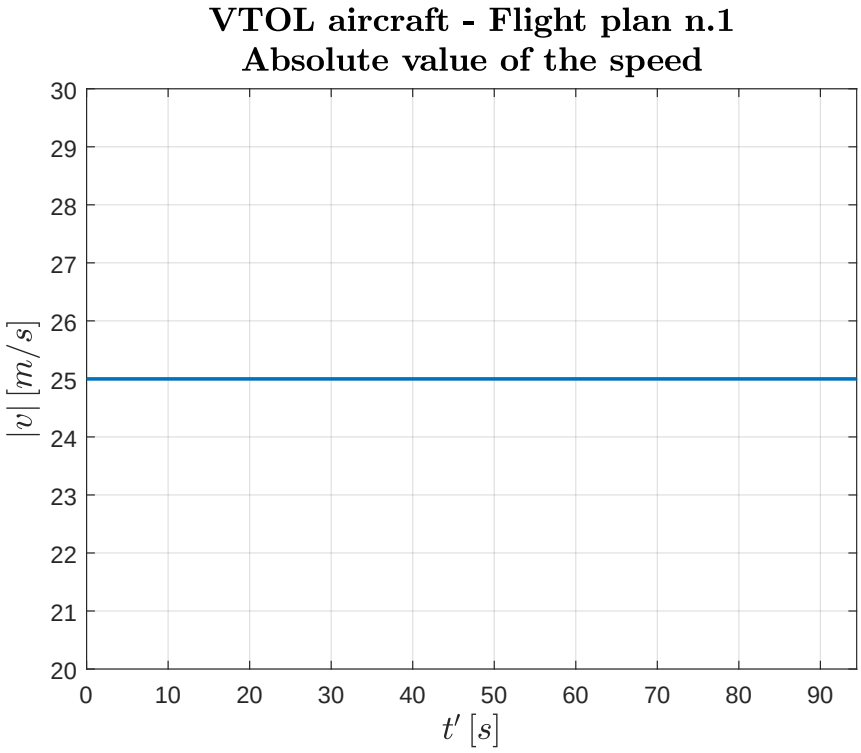

Figure 5.10: Time history of speed absolute value - Flight plan n.1

Another task consists in maintaining a constant speed equal to 25 m/s during the wingborne phase. As explained in the paragraph concerning horizontal curve of Section 4.4.4.1, a scaling in time is necessary to reach this goal. The time history of speed absolute value is shown in Fig. 5.10 and it is evident that the aircraft maintains the requested constant speed.

The scaling function σ between t and t' is represented in Fig. 5.11. It is possible to notice that the intervals contained by the red dashed circles, which correspond to the curved path, are affected by the scaling in time because the ratio between t and t' differs from one.

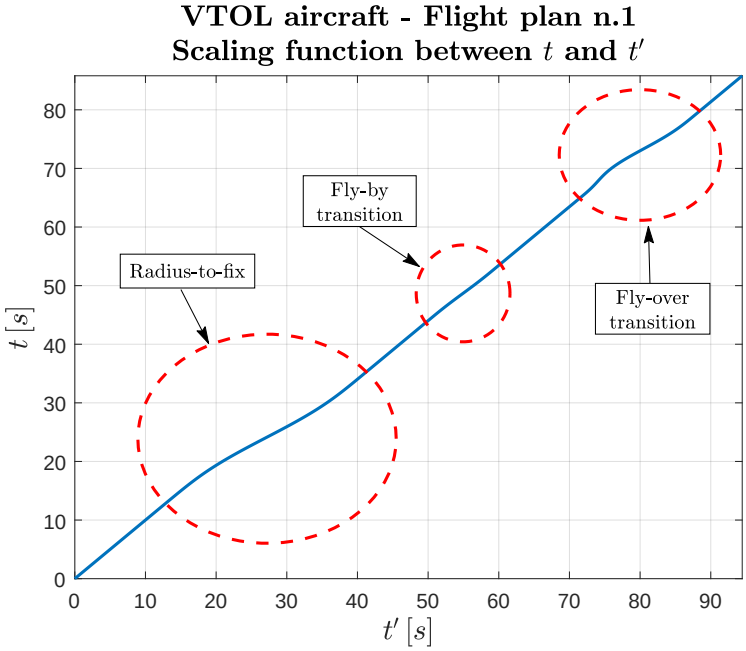

Figure 5.11: Scaling function $t = \sigma(t')$ - Flight plan n.1

Another mention can be done on the absolute value of the acceleration, reported in Fig. 5.12.

As expected, the acceleration is equal to 0 m/s^2 during the straight line but it varies during the curves although the speed is constant. It is caused by the presence of the centripetal acceleration along a curved path. The maximum value during the fly-by transition and RF leg is around 4 m/s^2 while 16 m/s^2 is reached during the fly-over maneuver. An high value of centripetal acceleration is related to an high value of $\dot{\chi}$, whose maximum value during the fly-over is equal to 37 deg/s. It causes a feasibility problem since the request is to maintain its value under 10 deg/s. This result underlines that the fifth degree polynomial is not the most proper one for this kind of maneuver. The only solution is the automatic saturation of $\dot{\chi}$ with circular arc, explained in the paragraph concerning the horizontal curve of Section 4.4.4.1 and analyzed in Section 5.3.

Despite the saturation works well, the use of fly-by and radius-to-fix is preferred; therefore, the fly-over transition will not be used in the simulations of Chapter 6.

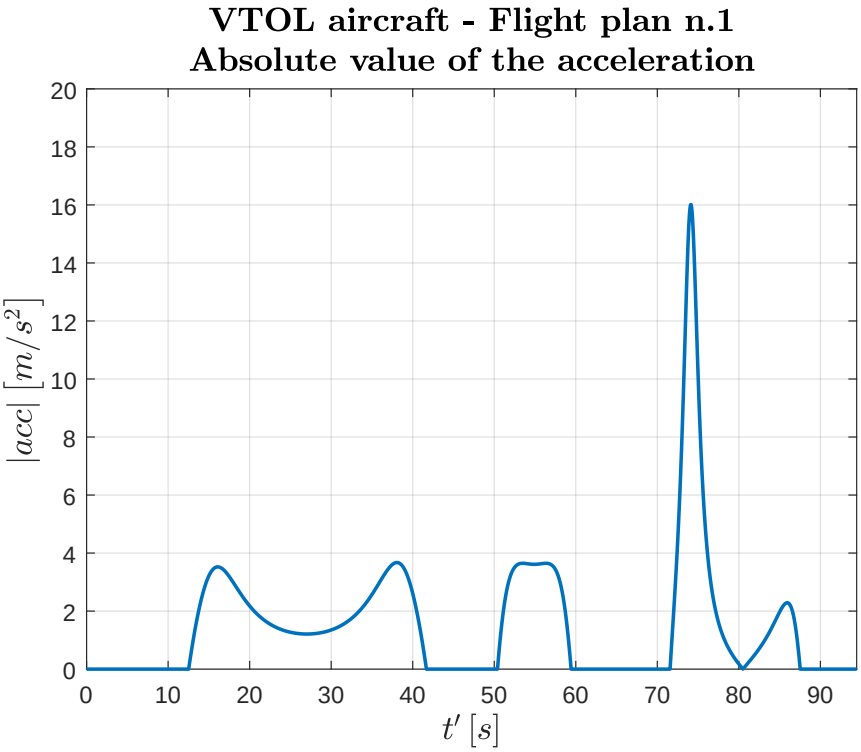

Figure 5.12: Time history of acceleration absolute value - Flight plan n.1

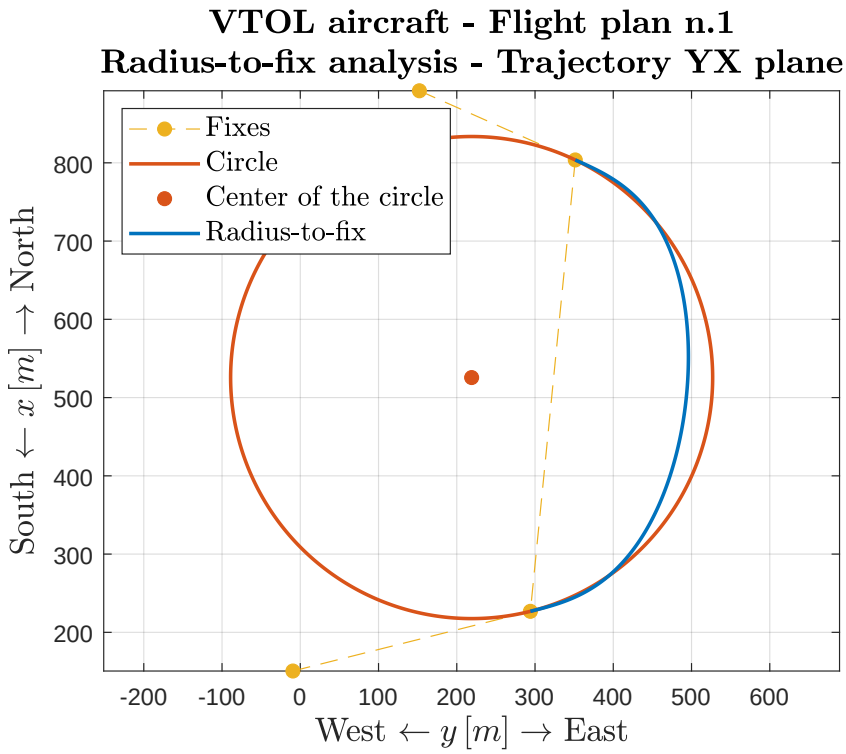

Figure 5.13: Radius-to-fix - Fifth degree polynomial vs Circular arc - Flight plan n.1

The last analysis is related to the RF leg: [34] and [33] specify that this maneuver is characterized by a constant radius but the fifth degree polynomial does not guarantee this feature. For this reason, a comparison between this curve used by the implemented algorithm and a circular arc is required. The circular arc connecting the beginning and the end of the RF leg is plotted in Fig. 5.13 and the difference with respect to the fifth degree polynomial is evident.

In addition, Fig. 5.14 shows the gap between the value of the circular arc radius and the distance from the circle center to the polynomial.

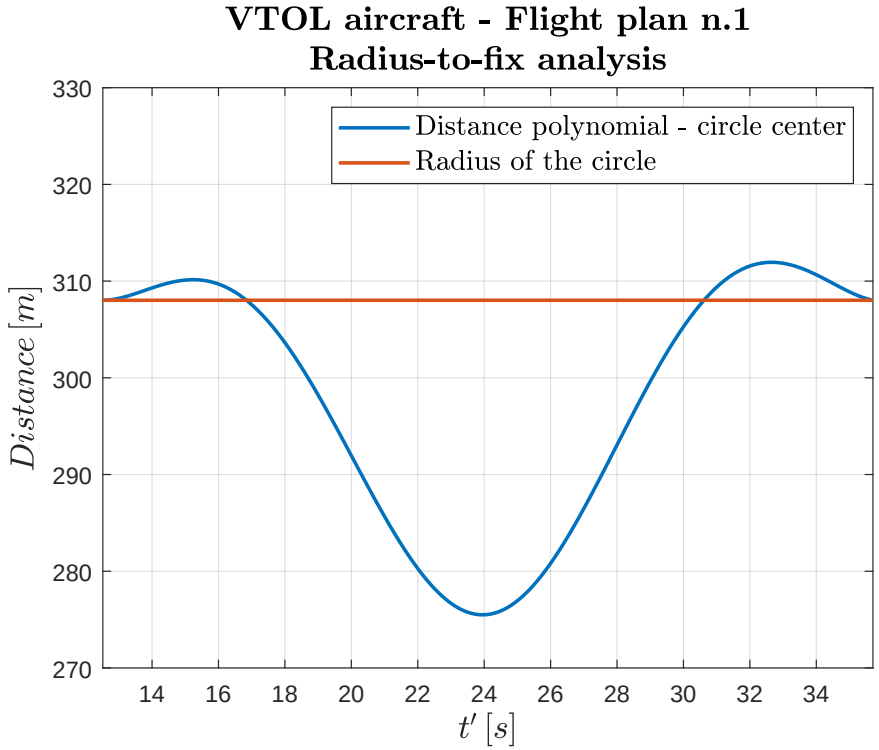

Figure 5.14: Radius-to-fix - *Distance* vs r_c - Flight plan n.1

As expected by the theory, the distance between the fifth degree polynomial and the circle center, named *Distance*, is not constant and so [34] and [33] are not strictly satisfied. On the other hand, the use of circular arc is not sufficient because of the problem on curvature discontinuity explained in Section 4.1.2.5. Taking into account these considerations, the choice of the fifth degree polynomial is a compromise of the two aspects in order to test an alternative method with respect to the one already existing in [31].

5.2 Wingborne with climb paths

The aim of this section is to show and analyze results regarding the possibility to generate climb paths during wingborne phase. The waypoints are chosen with a reasonable difference in altitude in order to accomplish this task. Additionally, the

n. [−]	ID Number [−]	*lat* [*deg*]	*long* [*deg*]	*h* [*m*]
1	9	48.267539	11.668193	518
2	1	48.264980	11.661760	568
3	0	48.263666	11.653949	598
4	3	48.262118	11.650655	598
5	1	48.258952	11.647853	598
6	0	48.258299	11.644073	558

Table 5.5: Waypoint list - Flight plan n.2

waypoint list, reported in Table 5.5, composes a trajectory in a precise direction, which is the south-west one in this case, in order to visualize in a better way the results in the three different planes, without any intersection.

First of all, Fig. 5.15 and Fig. 5.16 are different representations by two points of view of the trajectory in 3D space.

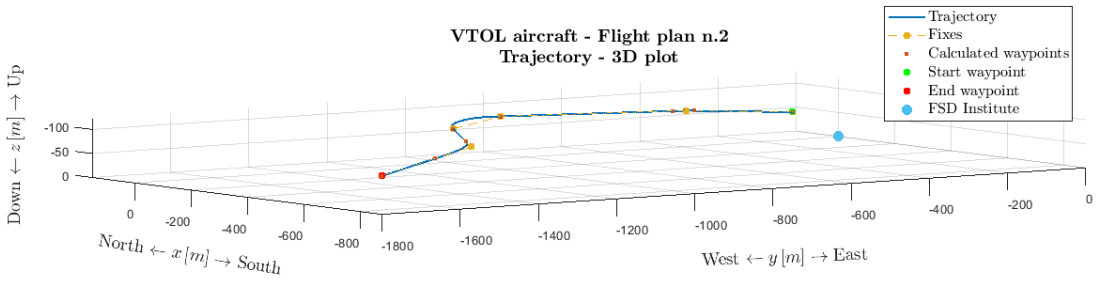

Figure 5.15: 3D trajectory - First point of view - Flight plan n.2

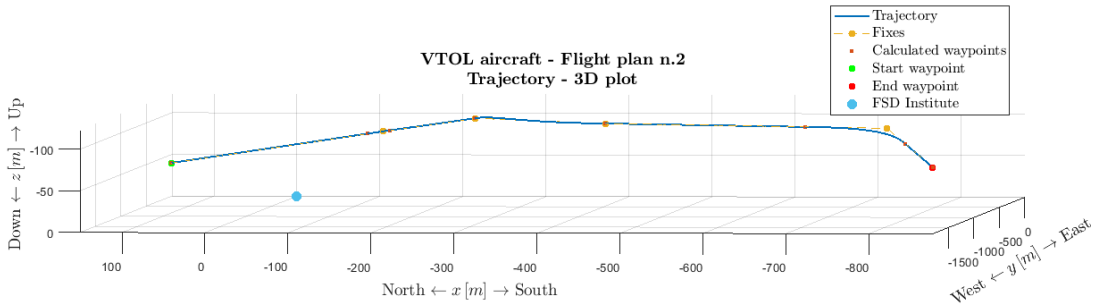

Figure 5.16: 3D trajectory - Second point of view - Flight plan n.2

In order to appreciate the altitude variations, the representations of the trajectory in XZ and YZ planes are shown in Fig. 5.17 and Fig. 5.18 respectively.

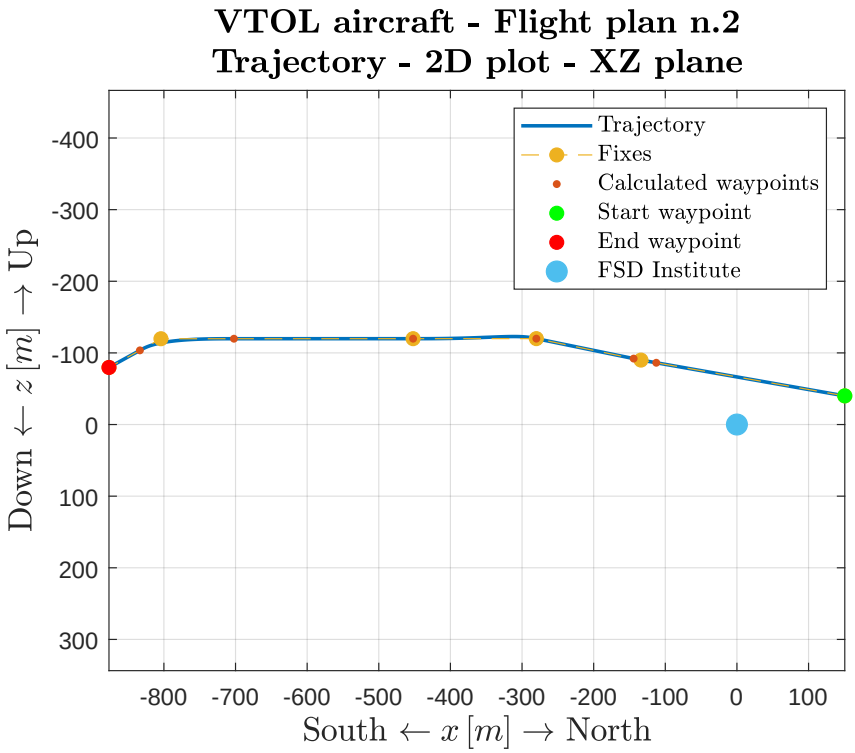

Figure 5.17: 2D trajectory in XZ plane - Flight plan n.2

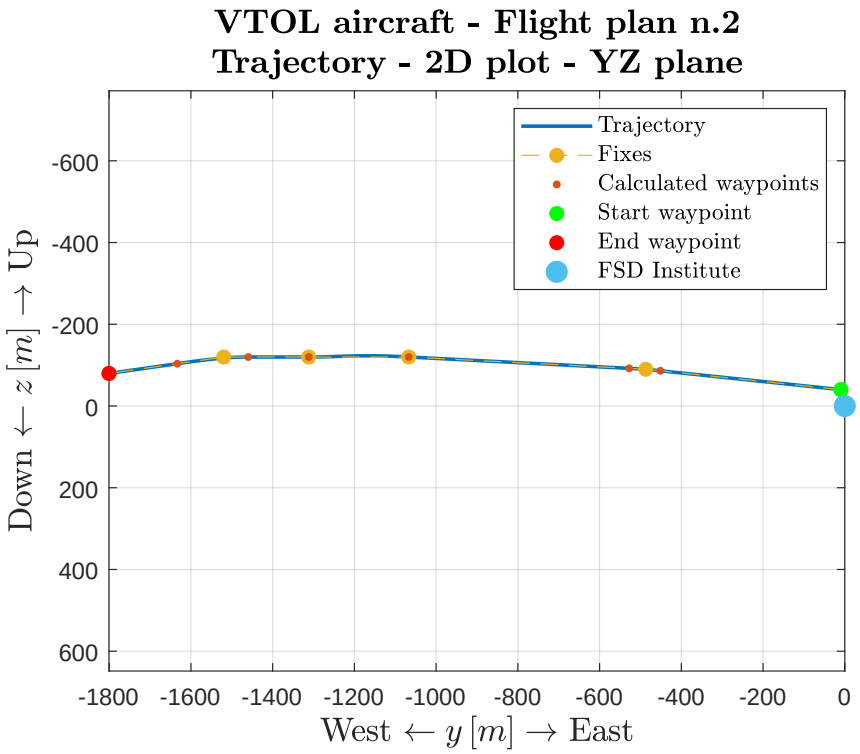

Figure 5.18: 2D trajectory in YZ plane - Flight plan n.2

As shown in Fig. 5.19, the aircraft completes a fly-by maneuver after the first climb leg, then it proceeds along a TF leg to reach a RF leg and finally another fly-by transition is performed before the last descent. The feasibility of this geometry proves what is anticipated in Section 4.4.1, namely the fact that all the implemented legs work not only for a 2D trajectory but also for a 3D one.

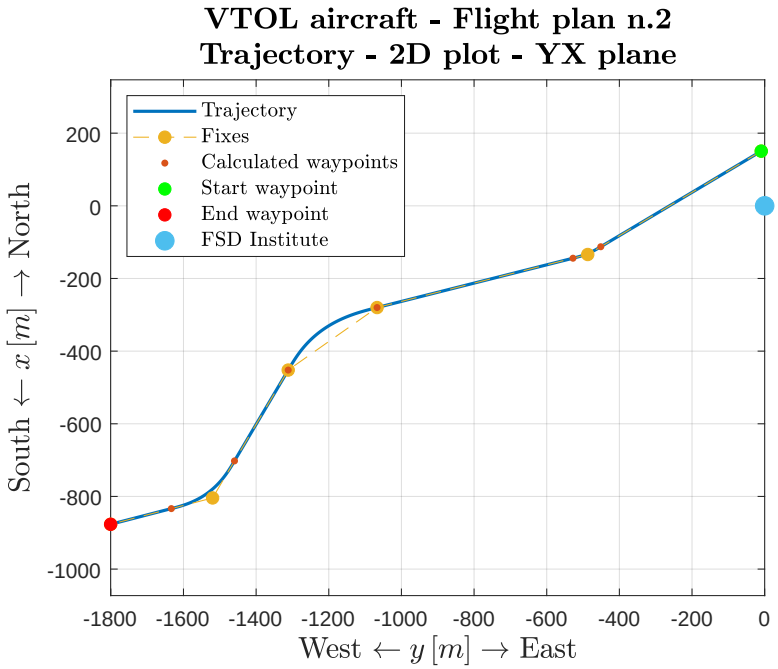

Figure 5.19: 2D trajectory in YX plane - Flight plan n.2

The information about the calculated waypoints are reported in Table 5.6.

n. [−]	ID number [−]	x $[m]$	y $[m]$	z $[m]$	χ $[deg]$	γ $[deg]$
1	9	150.5710	−9.4291	−39.9982	0	0
2	0	−112.6124	−451.2034	−86.2265	239.2160	5.1370
3	1	−144.2055	−527.6750	−92.0749	255.8663	2.8643
4	0	−280.0278	−1067.0648	−119.9048	255.8663	2.8643
5	1	−452.1235	−1311.6880	−119.8493	210.5957	−0.0113
6	0	−702.2159	−1459.5671	−119.7919	210.5957	−0.0113
7	1	−833.4952	−1633.4144	−103.5511	255.5126	−7.8713
8	0	−876.6796	−1800.5473	−79.6860	255.5126	−7.8713

Table 5.6: New waypoint list - Flight plan n.2

It is possible to notice that the value of γ reflects the variations in altitude of the waypoints; in particular the first and second climbs are characterized by $\gamma \approx 5\ deg$ and $\gamma \approx 3\ deg$ respectively while $\gamma \approx -8\ deg$ during the descent.

The change in altitude can be seen also in a clear way in the third graph of Fig. 5.20. After that, it is worth to notice that the continuity of the position, speed and acceleration in every direction remains guaranteed by the algorithm as shown also in Fig. 5.20, Fig. 5.21 and Fig. 5.22.

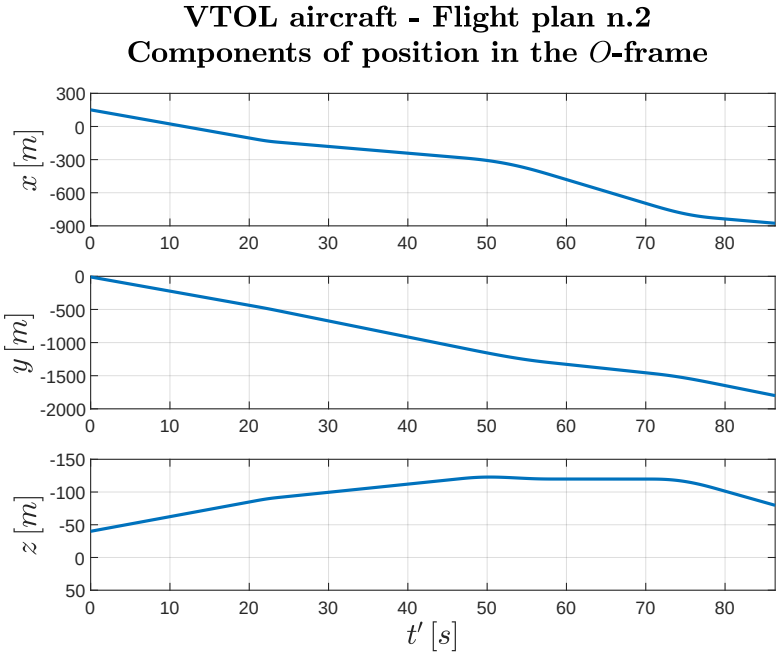

Figure 5.20: Time history of position components - Flight plan n.2

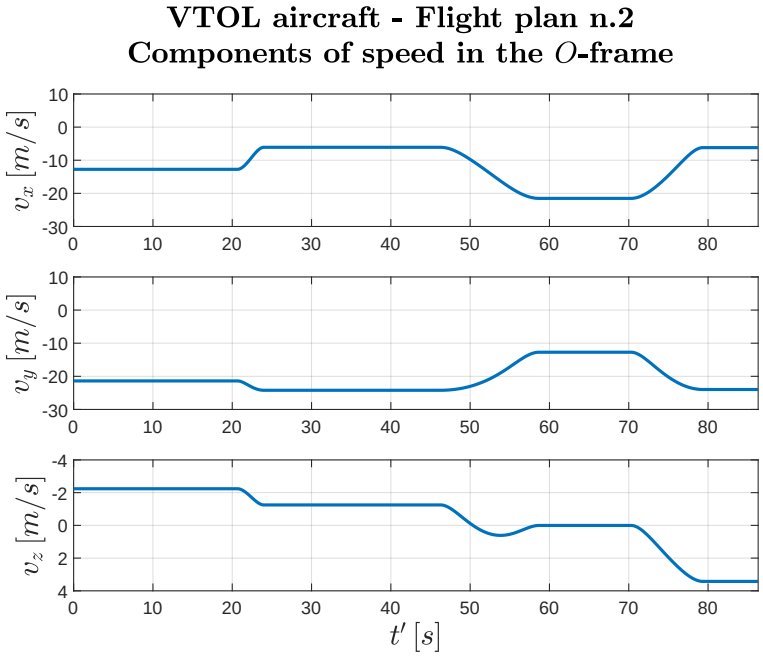

Figure 5.21: Time history of speed components - Flight plan n.2

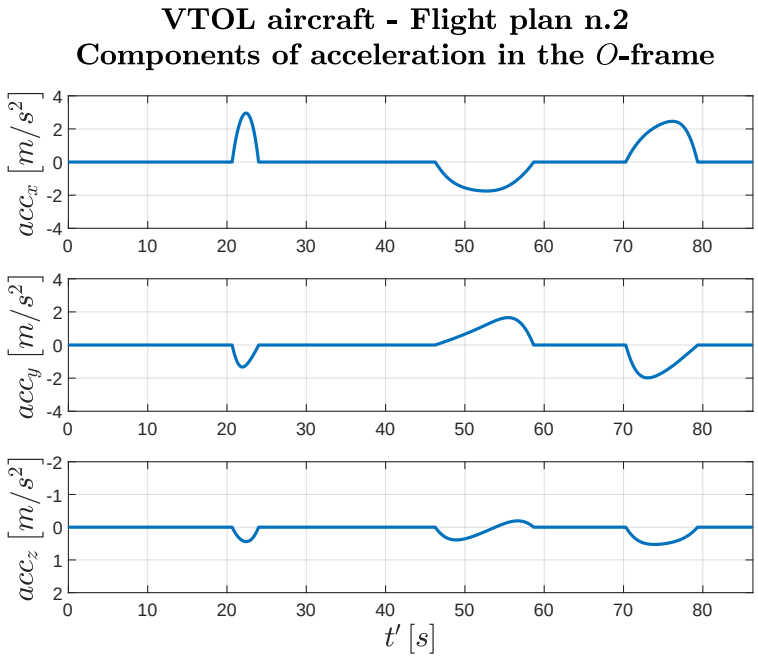

Figure 5.22: Time history of acceleration components - Flight plan n.2

Finally, the absolute value of speed and acceleration with respect to t' are reported in Fig. 5.23 and Fig. 5.24. As expected, the value of speed remains constant and equal to 25 m/s.

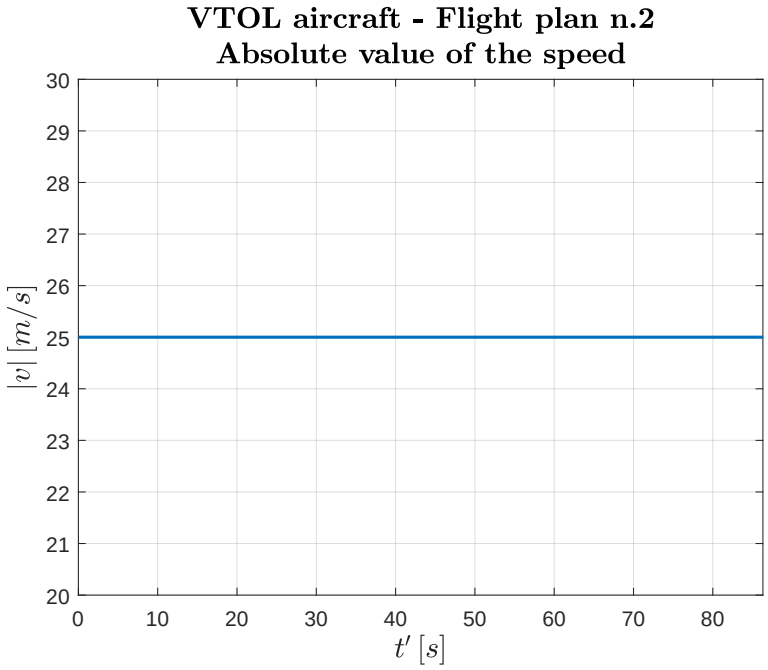

Figure 5.23: Time history of speed absolute value - Flight plan n.2

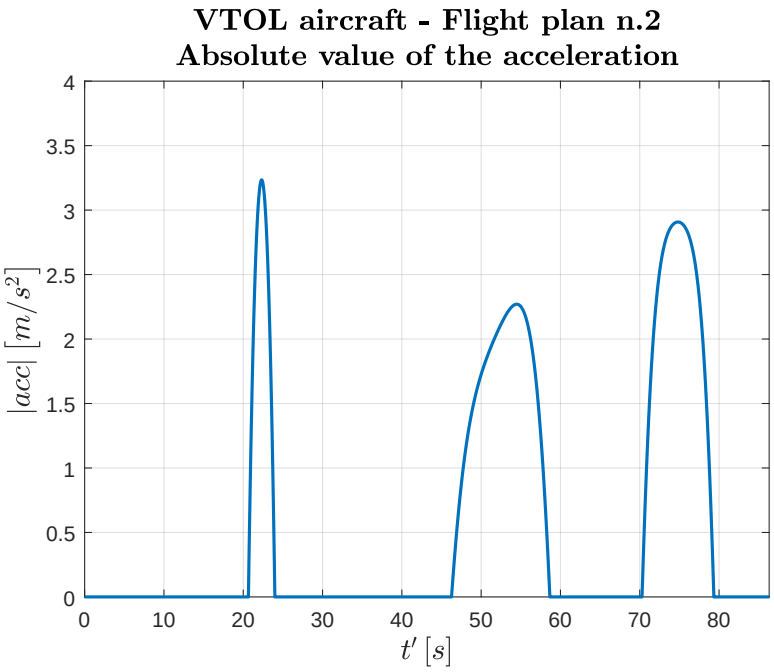

Figure 5.24: Time history of acceleration absolute value - Flight plan n.2

5.3 Wingborne with saturation of $\dot{\chi}$

Another requirement is that $\dot{\chi}$ remains under a specific value $\dot{\chi}_{max}$, in this case equal to 10 deg/s. As anticipated in the paragraph concerning the horizontal curve of Section 4.4.4.1, there are three different ways to reach this goal but only the second and the third ones are implemented in the trajectory generator block, because a better flight plan is something that has to be done before the trajectory generation.

The results reached with the variation of $\dot{\chi}_{des}$ and with the saturation using circular arc are reported in Section 5.3.1 and Section 5.3.2 respectively.

In this section the same waypoints of the flight plan n.1, reported in Table 5.3, are used but a different value of $\dot{\chi}_{des}$, equal to 20 deg/s, is chosen in order to be sure of overcoming the limits during the fly-by transition.

Looking the representation of the trajectory in YX plane, reported in Fig. 5.25, it is possible to notice that the fly-by turn is completed more rapidly than the one of flight plan n.1; a greater $\dot{\chi}_{des}$ causes a lower value of s_{turn}.

n. [−]	ID number [−]	x $[m]$	y $[m]$	z $[m]$	χ $[deg]$	γ $[deg]$
4	0	924.6809	78.7600	−39.9314	293.9284	−0.0016
5	1	915.3709	5.2582	−39.9329	231.6339	0.0036

Table 5.7: New waypoint list - Flight plan n.3 - $\dot{\chi}_{des} = 20\ deg/s$

The difference of geometry can be seen also by the coordinates of the two calculated waypoints for the fly-by, reported in Table 5.7, which differ from the ones in Table 5.4.

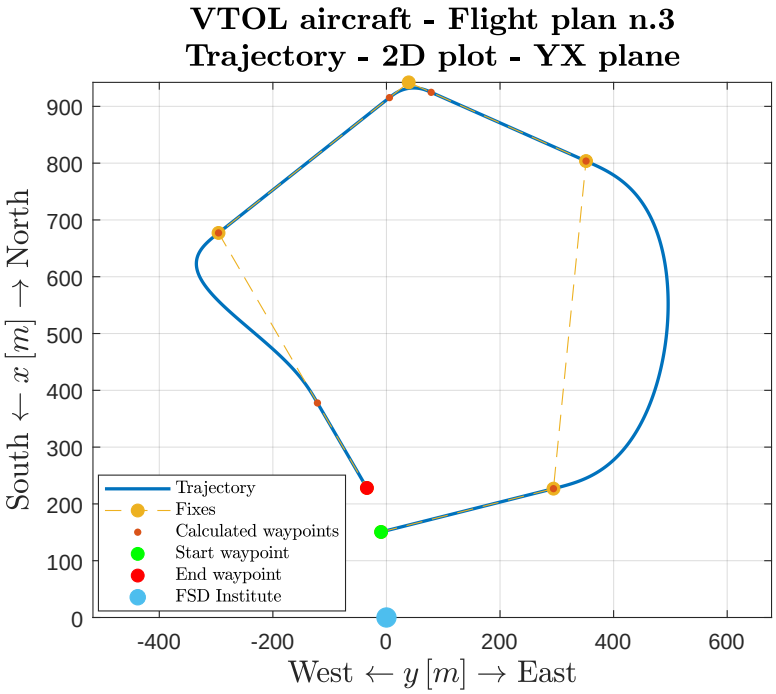

Figure 5.25: 2D trajectory in YX plane - Flight plan n.3 - $\dot{\chi}_{des} = 20\ deg/s$

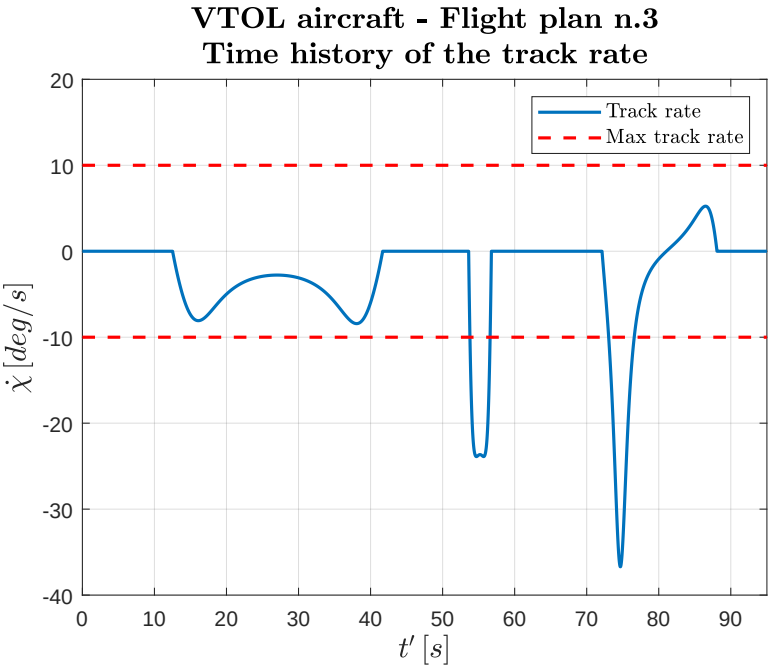

Figure 5.26: Time history of the track rate - Flight plan n.3 - $\dot{\chi}_{des} = 20\ deg/s$

It is worth to notice that the time history of $\dot{\chi}$, reported in Fig. 5.26, is computed thanks to the flatness relationships between the variables and the flat outputs, in particular the first one of Eq. (3.35). Looking Fig. 5.26, it is evident that the limit is not overcome during the RF leg while $\dot{\chi}$ is greater than $\dot{\chi}_{max}$ during the fly-by and fly-over transitions. For what concerns the fly-by turn, both methods, which will be treated in the next sub-sections, can be used to solve the problem while the fly-over can be managed using the saturation with circular arc because $\dot{\chi}_{des}$ does not influence its geometry. On the other hand, the RF leg is planned in a good way but, in case of the limit overcoming, the considerations that will be done later for the second method are valid also for this maneuver.

5.3.1 Variation of $\dot{\chi}_{des}$

Imposing the value of $\dot{\chi}_{des}$, it is possible to see that the $\dot{\chi}_{max}$ reached during a fly-by turn has a greater value; this is caused by the fact that a fifth degree polynomial is not a circular arc. A possible approach can be to decrease $\dot{\chi}_{des}$ randomly with many attempts in order to not overcome the limit of $\dot{\chi}_{max} = 10\ deg/s$ but it is not an engineering way to solve the problem. If a mathematical relationship between $\dot{\chi}_{des}$ and $\dot{\chi}_{max}$ is found, it can be used during the flight planning.

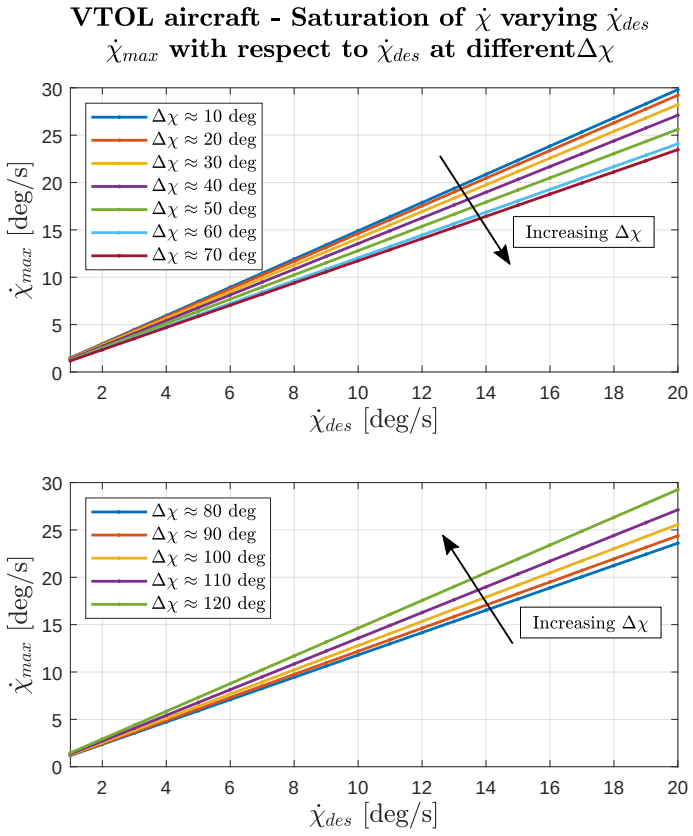

Figure 5.27: $\dot{\chi}_{max}$ with respect to $\dot{\chi}_{des}$ sweeping different values of $\Delta\chi$

To this aim, a general fly-by transition is simulated sweeping the value of $\Delta\chi$, namely the change of track angle between the two legs, from 10 deg to 120 deg and varying $\dot{\chi}_{des}$ between 1 deg/s and 20 deg/s for each case. The choice of $\Delta\chi$ maximum value for a fly-by is taken from [31]. The obtained results are reported in Fig. 5.27. It is important to underline that these results are referred to Δt calculated with straight line method and that there is a linear mathematical relation between the quantities.

The same procedure can be done also with the circular arc Δt and all the values of $\dot{\chi}_{max}/\dot{\chi}_{des}$ with respect to $\Delta\chi$ are shown in Fig. 5.28 in order to compare the two methods.

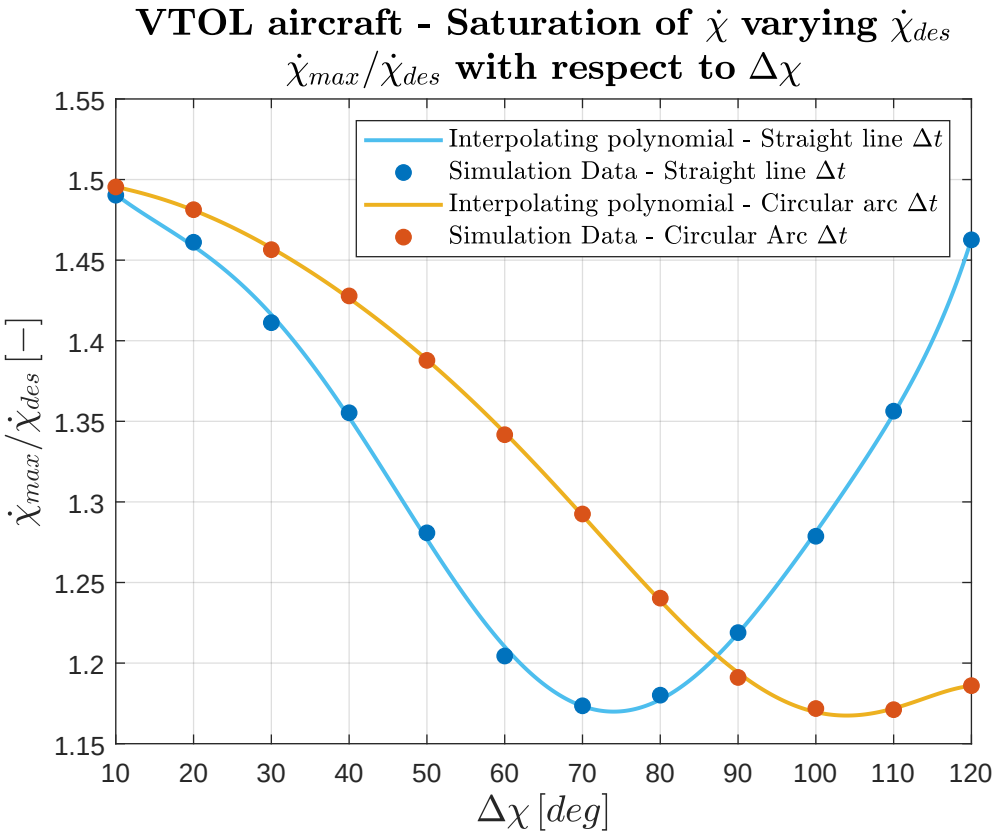

Figure 5.28: Ratio between $\dot{\chi}_{max}$ and $\dot{\chi}_{des}$ collected for $\Delta\chi \in [10, 120]$ deg

It is possible to notice that the interpolating polynomial obtained with straight line time stays under the one obtained with circular arc time for $\Delta\chi < 88$ deg. Considering the same imposed $\dot{\chi}_{max}$ and the same $\Delta\chi$, a lower value of the ratio provides a greater value of $\dot{\chi}_{des}$ and so a lower s_{turn} for a straight line Δt. This permits to design a transition which covers less space. Since the fly-by maneuver is normally executed with $\Delta\chi$ lower than 90 deg, the Δt computed with straight line is used in this thesis recalling the fact that it is more general and useful also for fly-over.

Returning to flight plan n.3 and knowing that the fly-by $\Delta\chi$ is equal to 62 deg, $\dot{\chi}_{des}$ is chosen as follows:

1. the corresponding value of $\dot{\chi}_{max}/\dot{\chi}_{des}$ is found using Fig. 5.29;

2. $\dot{\chi}_{des}$ is calculated knowing that the desired $\dot{\chi}_{max}$ is equal to 10 deg/s and using

$$\dot{\chi}_{des} = \frac{\dot{\chi}_{max}}{(\dot{\chi}_{max}/\dot{\chi}_{des}\,(\Delta\chi))}. \tag{5.1}$$

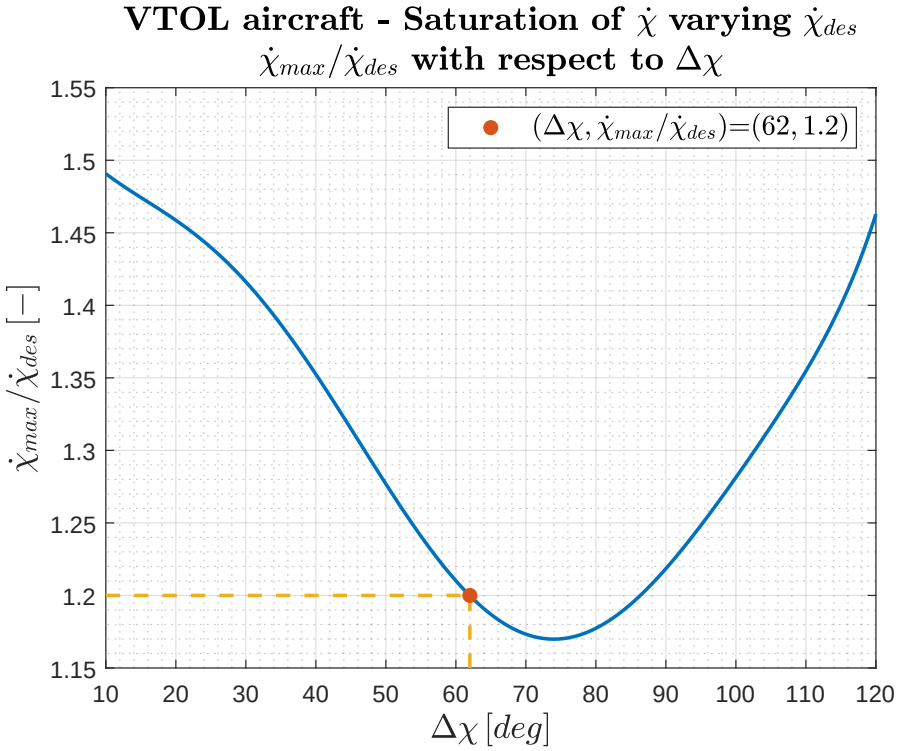

Figure 5.29: Application of the function $\dot{\chi}_{max}/\dot{\chi}_{des}\,(\Delta\chi)$ to the fly-by of flight plan n.3

In this case the value of $\dot{\chi}_{des}$ is equal to 8.33 deg/s. If there are more than one fly-by with different $\Delta\chi$, the same procedure can be followed and the more restrictive value of $\dot{\chi}_{des}$ has to be chosen.

Imposing $\dot{\chi}_{des} = 8.33$ deg/s, the geometry of the fly-by turn changes; in particular, the position of the calculated waypoints, listed in Table 5.8, is different from the ones in Table 5.7.

n. [−]	ID number [−]	x [m]	y [m]	z [m]	χ [deg]	γ [deg]
4	0	900.0858	134.1879	−39.9331	293.9284	−0.0016
5	1	877.7328	−42.2871	−39.9368	231.6339	0.0036

Table 5.8: New waypoint list - Flight plan n.3 - $\dot{\chi}_{des} = 8.33$ deg/s

Having a $\dot{\chi}_{des}$ lower than the one used in Fig. 5.25, s_{turn} is greater, as it is evident from Fig. 5.30.

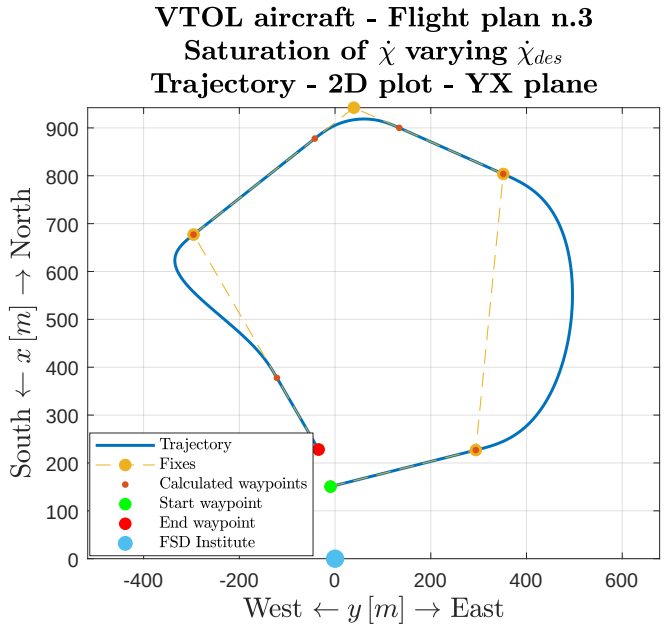

Figure 5.30: 2D trajectory in YX plane - Flight plan n.3 - $\dot{\chi}_{des} = 8.33\ deg/s$

The continuity and the constant speed are obviously guaranteed also in this case while a comparison between the results using $\dot{\chi}_{des} = 20\ deg/s$ and $\dot{\chi}_{des} = 8.33\ deg/s$ is shown in the graph of the acceleration absolute value [Fig. 5.31].

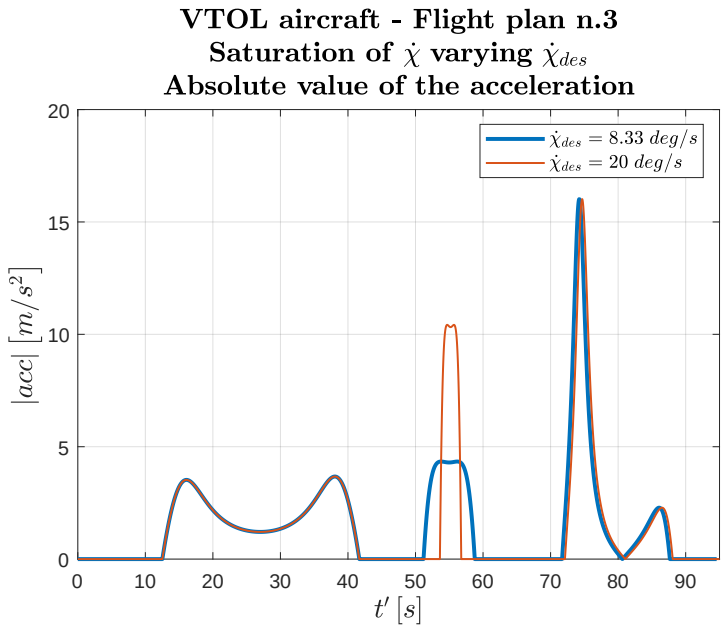

Figure 5.31: Comparison of the time history of acceleration absolute values - Flight plan n.3

During the fly-by transition, the acceleration absolute values differ significantly since the turn with $\dot{\chi}_{des} = 20\ deg/s$ is narrower and requires a greater centripetal acceleration. It is worth to notice that the red line is shifted because a different scaling in time causes a different t'.

Finally, the validity of this $\dot{\chi}$ saturation for fly-by is proved by Fig. 5.32. The maximum value, equal to 9.95 deg/s, is not precisely 10 deg/s because the function of Fig. 5.29 is found through an interpolation.

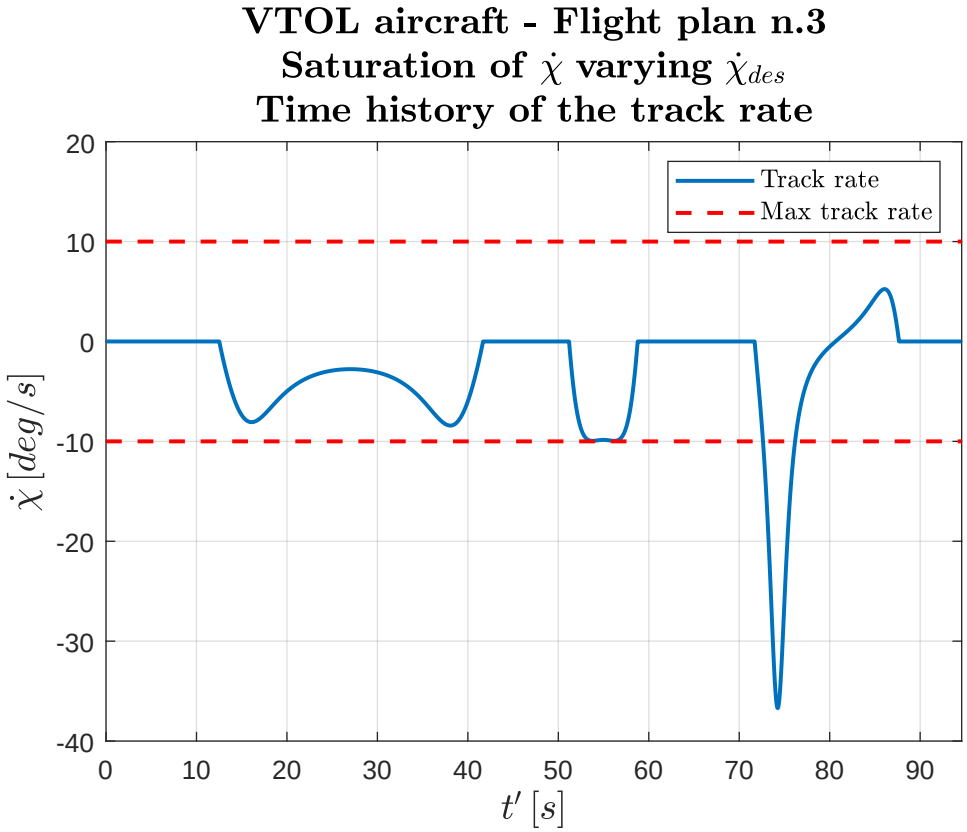

Figure 5.32: Time history of the track rate - Flight plan n.3 - $\dot{\chi}_{des} = 8.33\ deg/s$

5.3.2 Saturation with circular arc

Fixing $\dot{\chi}_{max} = 10\ deg/s$ in the `main` code, the `saturationchidot` block, explained in a deep way in the paragraph concerning horizontal curve of Section 4.4.4.1, is automatically activated.

The shift of the trajectory, caused by the substitution of fifth degree polynomial with circular arc and by the coexistence of the constraints on constant speed and maximum $\dot{\chi}$ without changing the turn starting point, can be seen by the representation in the YX plane of Fig. 5.33 and by the coordinates of the waypoints in Table 5.9.

n. [−]	ID number [−]	x $[m]$	y $[m]$	z $[m]$	χ $[deg]$	γ $[deg]$
5	1	905.6008	−71.8764	−39.9329	231.6339	0.0036
6	0	667.3623	−372.8240	−39.9572	231.6339	0.0036
7	1	257.1221	−211.5249	−39.9829	149.8109	0.0043
8	0	107.4382	−124.4448	−39.9958	149.8109	0.0043

Table 5.9: New waypoint list - Flight plan n.3 - Saturation with circular arc

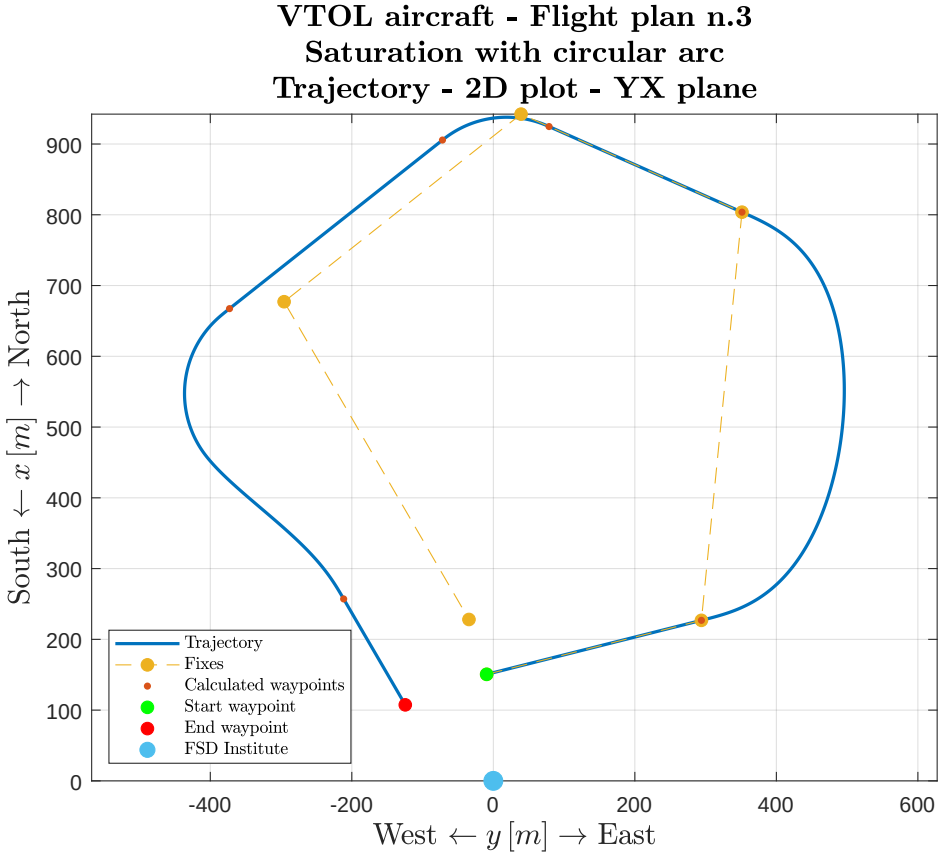

Figure 5.33: 2D trajectory in YX plane - Flight plan n.3 - Saturation with circular arc

It is worth to notice that the RF leg is not changed by the saturation because $\dot{\chi}$ does not overcome the limit during this maneuver thanks to a good planning. As already mentioned, the method can work also for a RF leg and, in this case, its activation would add another shift to the trajectory.

The inserted circular arcs do not compromise the continuity of position, speed and acceleration in every direction and the constant speed of 25 m/s as shown in Fig. 5.34, Fig. 5.35, Fig. 5.36 and Fig. 5.37.

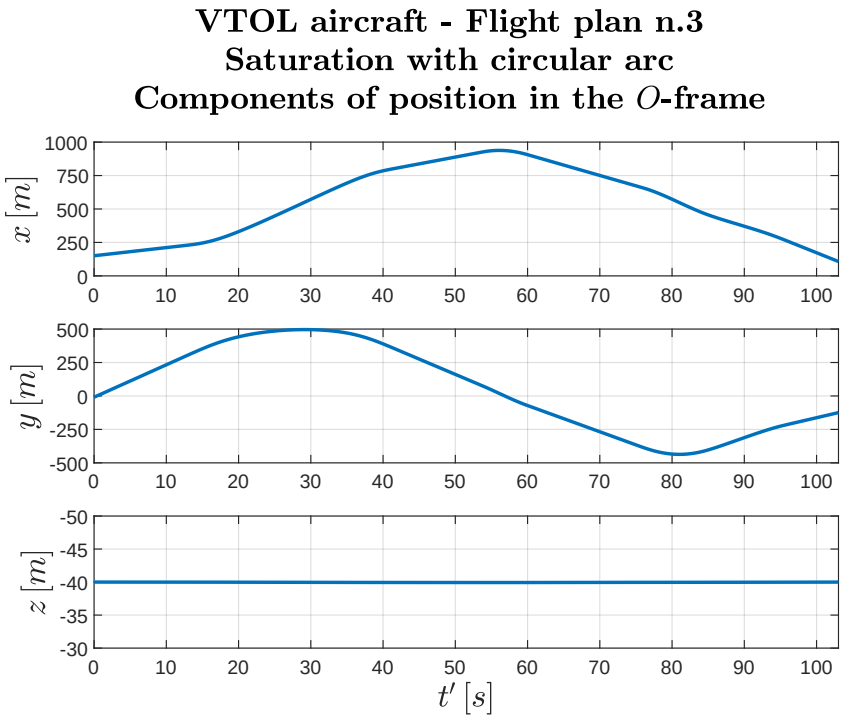

Figure 5.34: Time history of position components - Flight plan n.3 - Saturation with circular arc

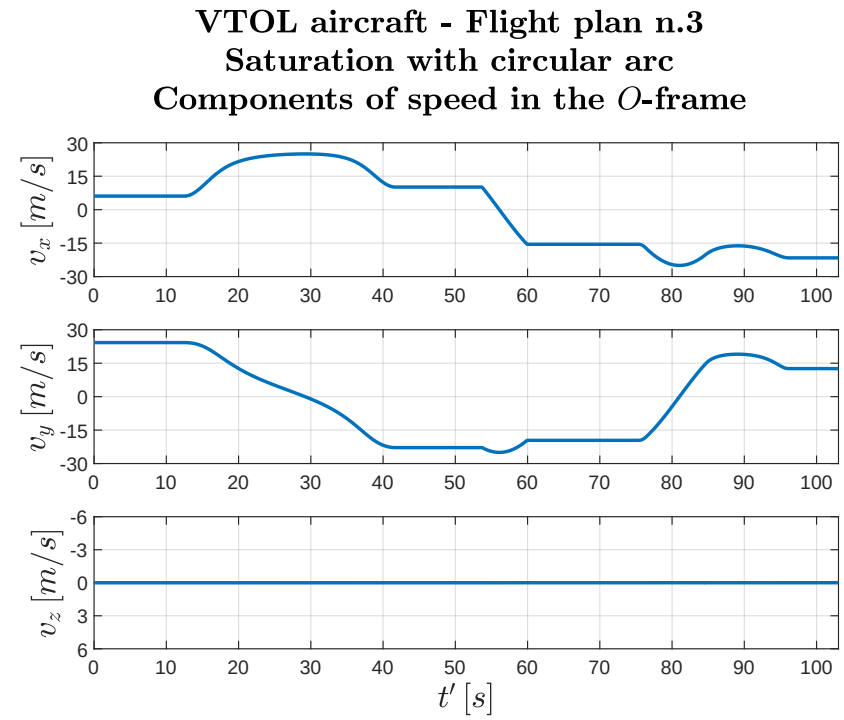

Figure 5.35: Time history of speed components - Flight plan n.3 - Saturation with circular arc

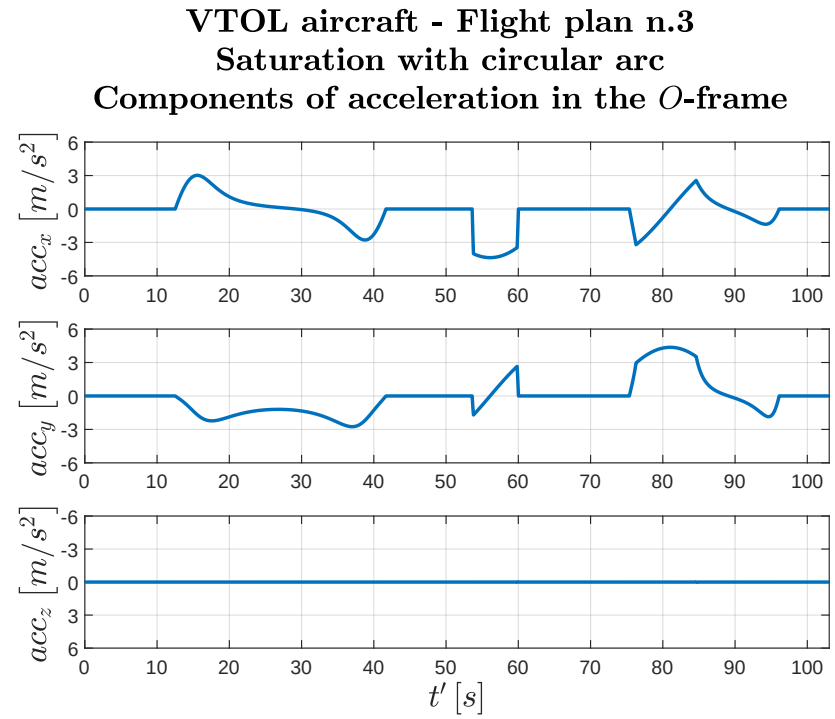

Figure 5.36: Time history of acceleration components - Flight plan n.3 - Saturation with circular arc

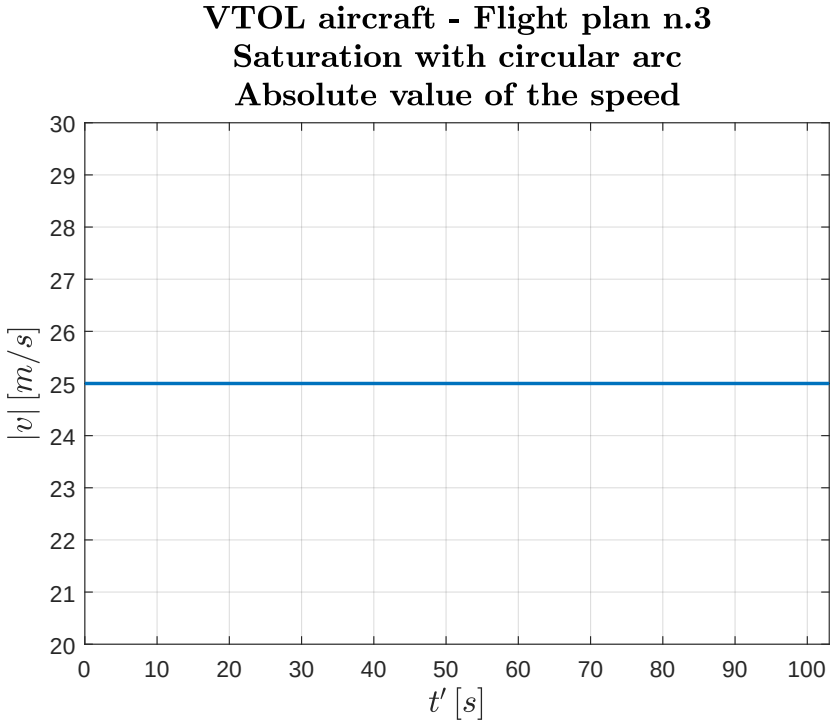

Figure 5.37: Time history of speed absolute value - Flight plan n.3 - Saturation with circular arc

Looking at Fig. 5.38, it is possible to notice the differences with Fig. 5.31 and, in particular, that the acceleration absolute value is constant during the circular arc paths since the centripetal acceleration is constant by construction differently from a fifth degree polynomial.

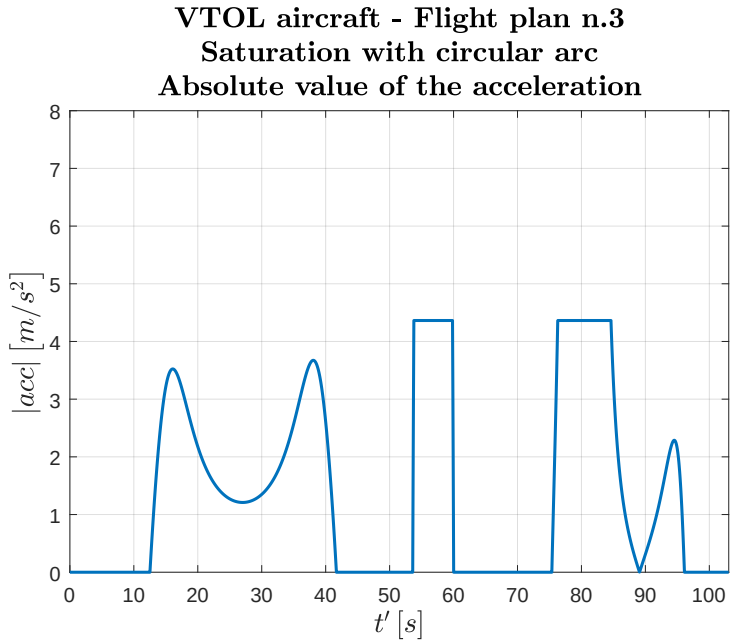

Figure 5.38: Time history of acceleration absolute value - Flight plan n.3 - Saturation with circular arc

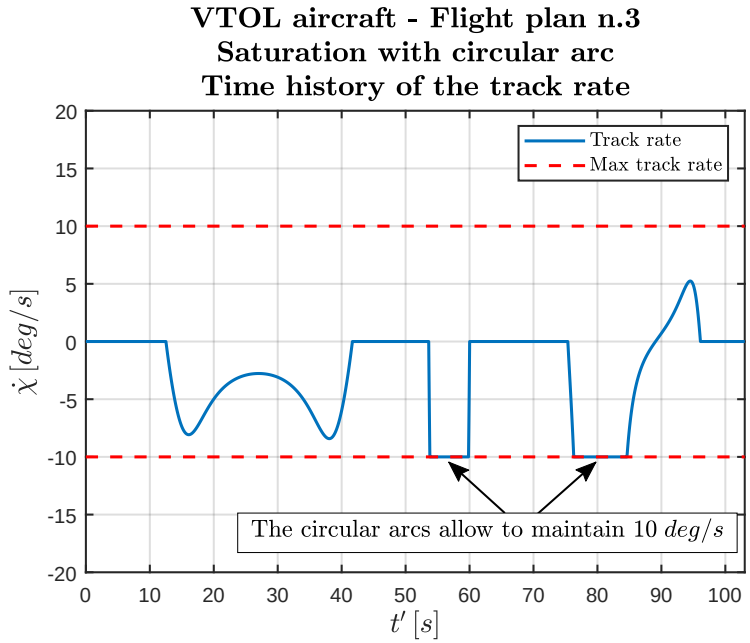

Figure 5.39: Time history of the track rate - Flight plan n.3 - Saturation with circular arc

The circular arcs maintain also a constant value of $\dot{\chi}$. If its value is set to 10 deg/s, the desired saturation is achieved as shown in Fig. 5.39.

Considering that the FSD institute is more interested in the use of RF leg and fly-by transition than the fly-over one, the approach consisting of a good planning for the RF leg and the choice of the proper $\dot{\chi}_{des}$ for fly-by is preferred to the circular arc one.

5.4 Complete flight plan: hover and wingborne

The flight plan n.4, whose waypoint list is reported in Table 5.10, is built in order to show all the possible maneuvers, both in hover phase and in wingborne one.

n. [−]	ID Number [−]	*lat* [*deg*]	*long* [*deg*]	*h* [*m*]
1	9	48.267539	11.668193	478
2	4	48.267539	11.668193	518
3	5	48.268225	11.672281	518
4	3	48.273412	11.673054	518
5	1	48.274658	11.668848	518
6	1	48.273555	11.664697	518
7	6	48.270460	11.663331	518
8	7	48.270460	11.663331	518
9	8	48.270460	11.663331	478

Table 5.10: Waypoint list - Flight plan n.4

For what concerns the fly-by turns, the analysis of $\Delta\chi$ between the legs is performed in order to find the necessary $\dot{\chi}_{des}$ and to plan a trajectory which satisfies the limit of 10 deg/s. As already explained in Section 5.3.1, the presence of two fly-by causes the choice of the more restrictive condition.

Fly-by n. [−]	$\Delta\chi$ [*deg*]	$\dot{\chi}_{max}/\dot{\chi}_{des}$ [−]	$\dot{\chi}_{des}$ [*deg*/*s*]
1	46	1.31	7.63
2	52	1.262	7.92

Table 5.11: Choice of $\dot{\chi}_{des}$ for fly-by - Flight plan n.4

The red value in Table 5.11 is fixed so that the limit is not overcome for both fly-by turns. A good planning for the RF leg together with the last consideration guarantee that the `saturationchidot` block is never activated.

A complete overview of the trajectory is reported in Fig. 5.40 where it is possible to see every maneuver.

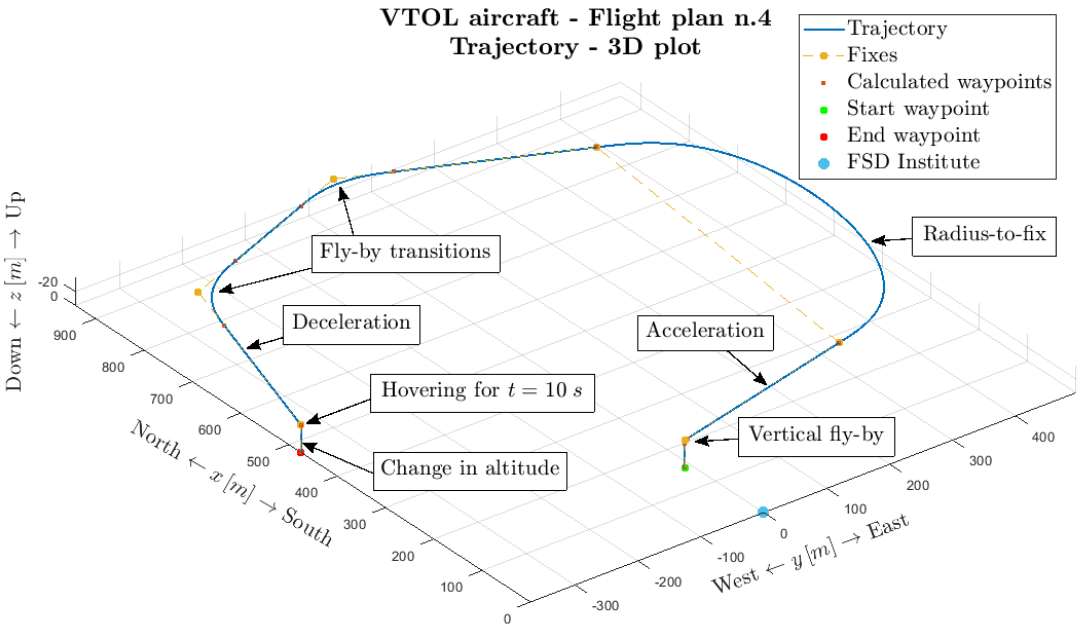

Figure 5.40: 3D trajectory - Flight plan n.4

The list of calculated waypoints is written in Table 5.12, where the sign − is referred to the fact that χ and γ angles for the waypoints which describe a leg of hover phase are meaningless.

n. [−]	ID number [−]	x $[m]$	y $[m]$	z $[m]$	χ $[deg]$	γ $[deg]$
1	9	150.5700	−9.4290	0.0018	0	0
2	2	150.5709	−9.4291	−34.9982	–	–
3	3	151.7899	−4.5799	−39.9981	75.8897	−0.0017
4	4	226.8649	294.0794	−39.9892	75.8897	−0.0017
5	1	803.6865	351.4342	−39.9396	293.9284	−0.0016
6	0	910.2041	111.3852	−39.9324	293.9284	−0.0016
7	1	913.0304	−34.1827	−39.9329	248.2962	0.0020
8	0	853.3520	−184.1190	−39.9385	248.2962	0.0020
9	1	731.9974	−294.7692	−39.9492	196.4204	0.0048
10	5	475.4116	−370.3858	−39.9715	196.4204	0.0048
11	6	475.4116	−370.3858	−39.9715	–	–
12	2	475.4086	−370.3835	0.0285	–	–

Table 5.12: New waypoint list - Flight plan n.4

The variation of the z coordinate between the couple of waypoints $(1, 2)$ or $(11, 12)$ remarks the possibility of a VTOL transition aircraft to perform vertical displacements without moving in the YX plane thanks to the combination of the hover phase and the wingborne one. The change in altitude can be appreciated also in Fig. 5.41 while the vertical fly-by maneuver is zoomed in Fig. 5.42.

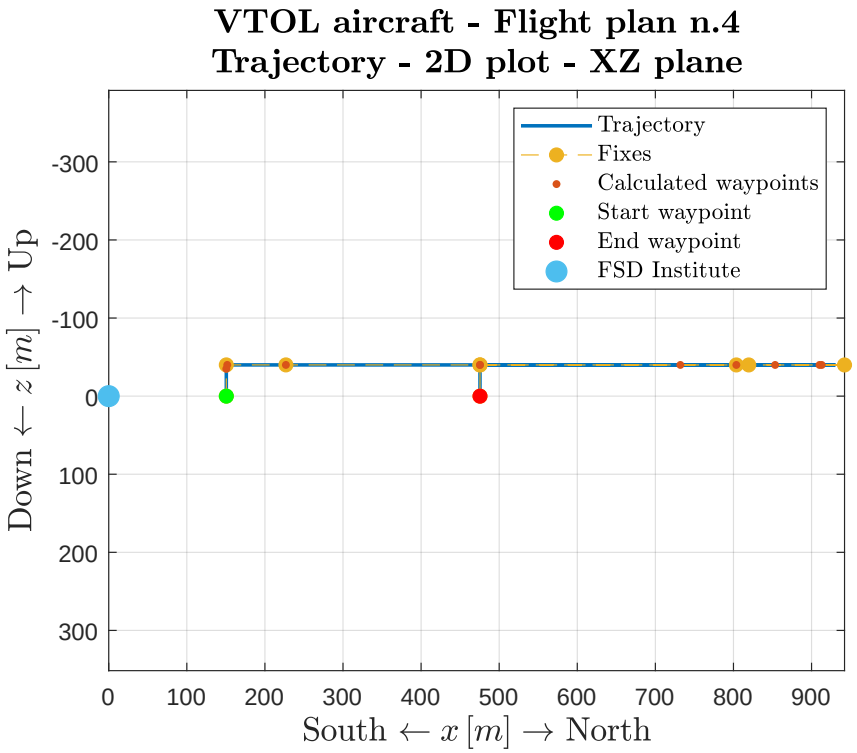

Figure 5.41: 2D trajectory in XZ plane - Flight plan n.4

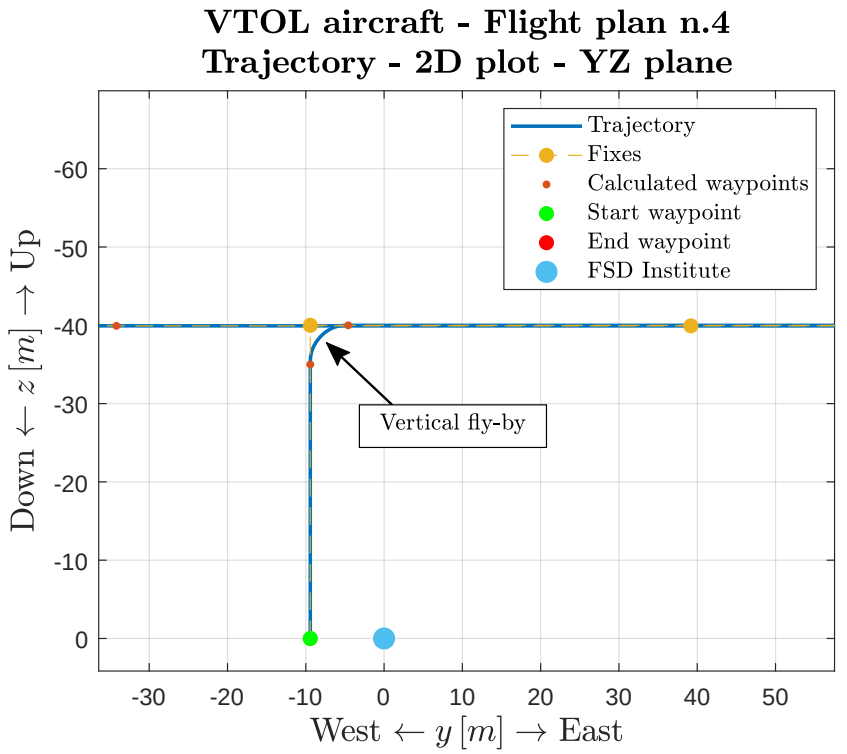

Figure 5.42: 2D trajectory in YZ plane - Flight plan n.4

All the maneuvers completed during the wingborne phase are clearly represented in Fig. 5.43.

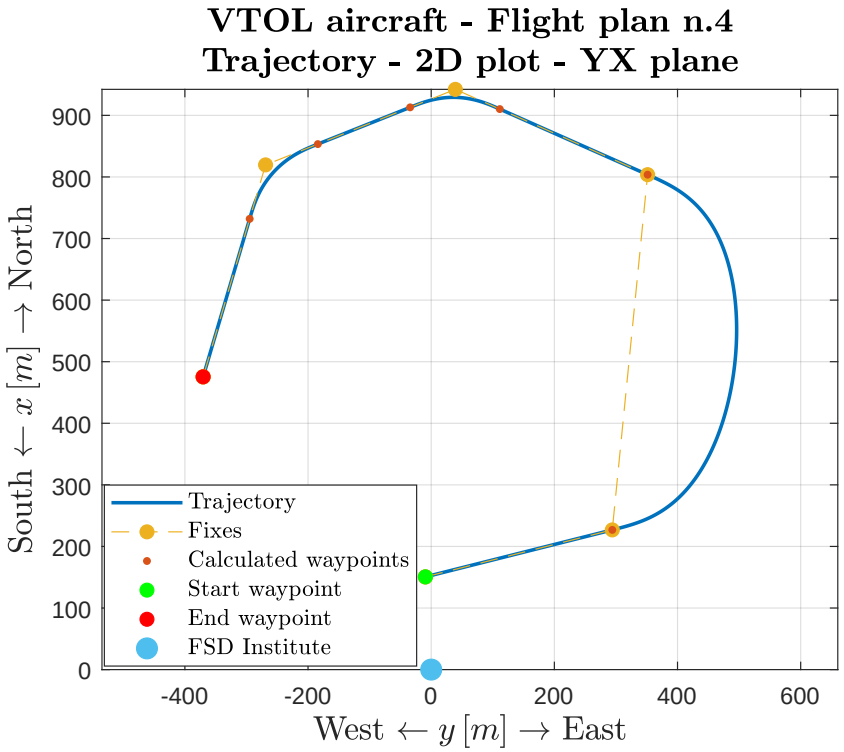

Figure 5.43: 2D trajectory in YX plane - Flight plan n.4

The continuity of the components of position, speed and acceleration is guaranteed for the entire trajectory as shown by the time histories in Fig. 5.44, Fig. 5.45 and Fig. 5.47; this result demonstrates the advantage of flatness theory because the choice of the proper flat outputs allow to generate a unique reference trajectory despite the presence of two different dynamics and different maneuvers.

In addition, it is important to verify whether all the prescribed maximum values, listed in Table 5.13, are satisfied.

$v_{wingborne}$ $[m/s]$	$v_{z_{max}}$ $[m/s]$	$acc_{tang_{max}}$ $[m/s^2]$
25	2	2

Table 5.13: Prescribed maximum values

Looking at the evolution in time of z in Fig. 5.44, it is possible to notice that the aircraft starts at an altitude of 0 m, then it goes up to 40 m and stays at the same altitude during the wingborne path, finally it returns at the initial altitude.

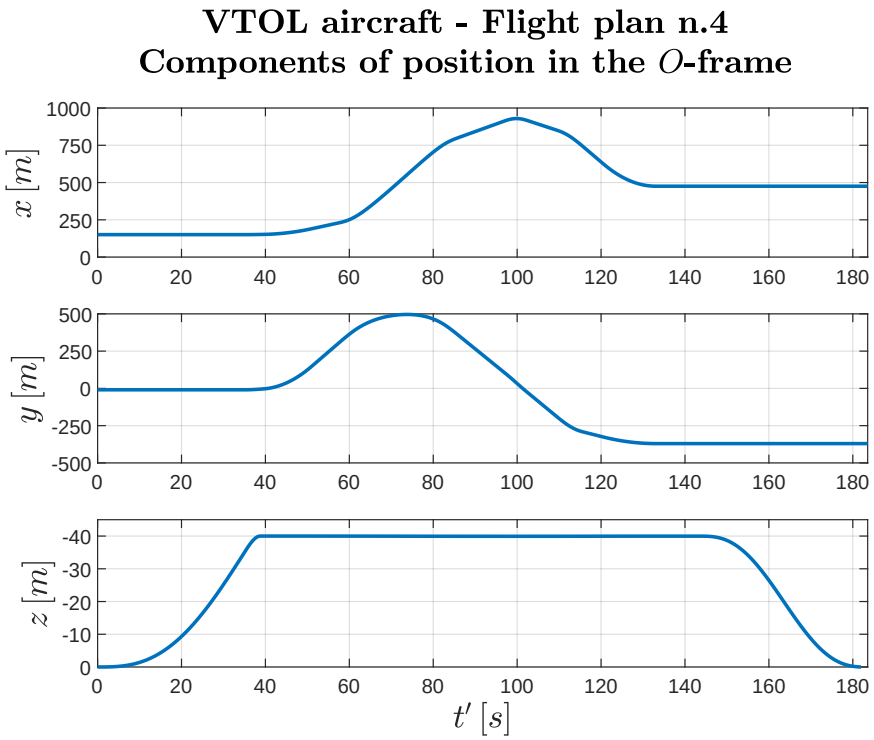

Figure 5.44: Time history of position components - Flight plan n.4

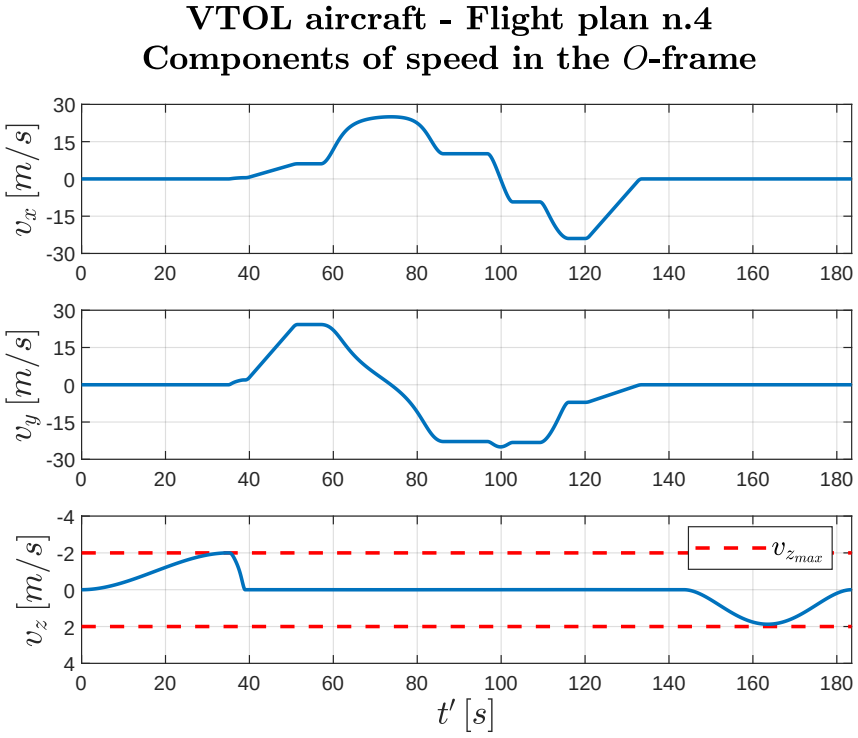

Figure 5.45: Time history of speed components - Flight plan n.4

During the hover maneuvers, namely change in altitude and vertical fly-by, the vertical speed v_z in Fig. 5.45 remains always under the maximum speed $v_{z_{max}}$.

The effectiveness of double S trajectory to describe the acceleration and deceleration, in which the transition between hover and wingborne happens, is evident by the time history of the speed absolute value, reported in Fig. 5.46. It is worth to notice that the speed remains constant and equal to 25 m/s during the wingborne maneuvers as in all the other flight plans. Between 133 s and 143 s, the absolute value of the speed is equal to 0 m/s since it corresponds to the hovering point whose duration in time is imposed to be 10 s in the `main` code.

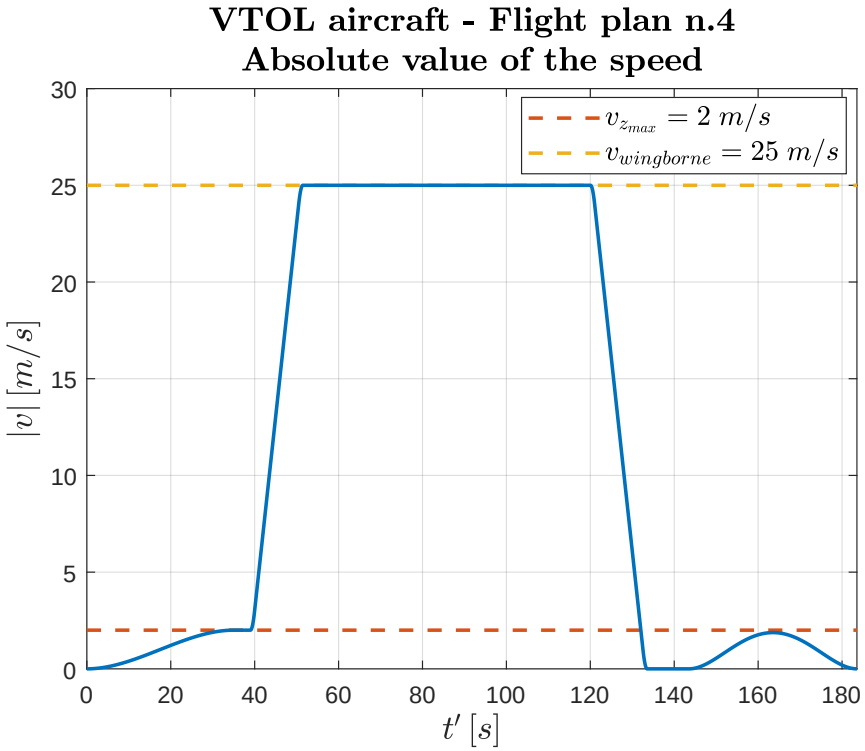

Figure 5.46: Time history of speed absolute value - Flight plan n.4

For what concerns the acceleration, the continuity is guaranteed in every direction as already mentioned before. Since its absolute value, plotted in Fig. 5.48, contains also the centripetal acceleration, it is necessary to take into account only the tangential one in order to check if the maximum value is respected. From Fig. 5.49, it is clear that:

- 2 m/s^2 is never exceeded during the vertical displacements;
- the acceleration has a trapezoidal shape during the acceleration phase, which is typical of double S trajectory. After the linear increase until 2 m/s^2, it remains constant, then decreases linearly arriving to zero value when the final speed is reached. The opposite behavior is followed during the deceleration.

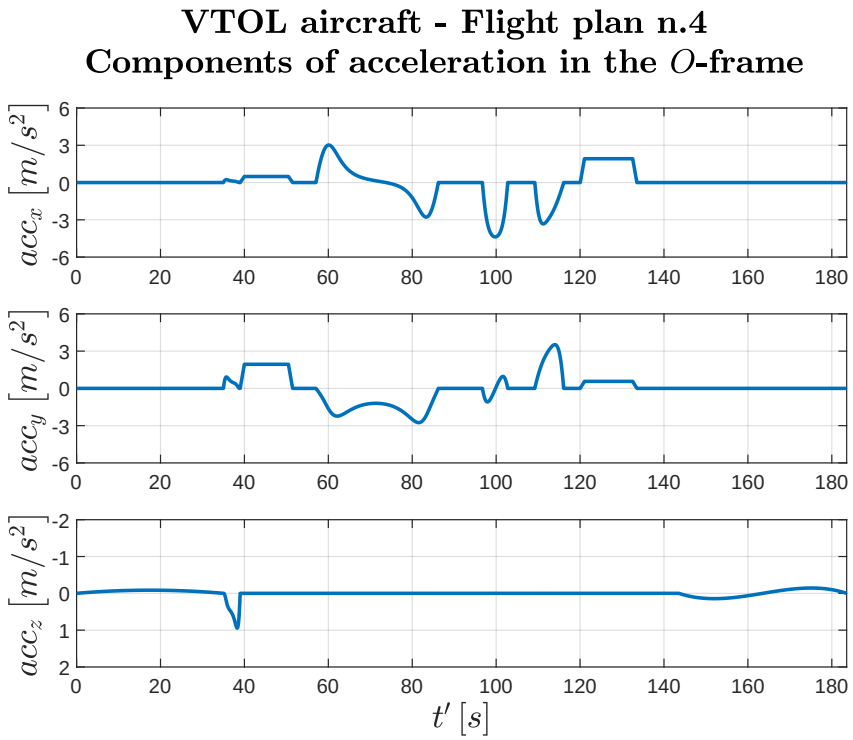

Figure 5.47: Time history of acceleration components - Flight plan n.4

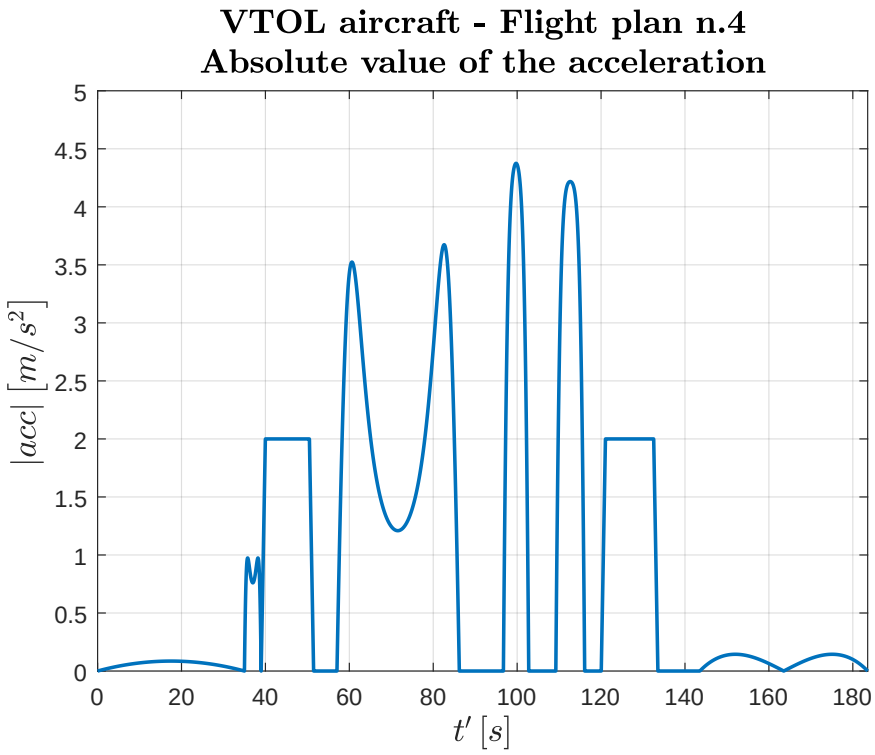

Figure 5.48: Time history of acceleration absolute value - Flight plan n.4

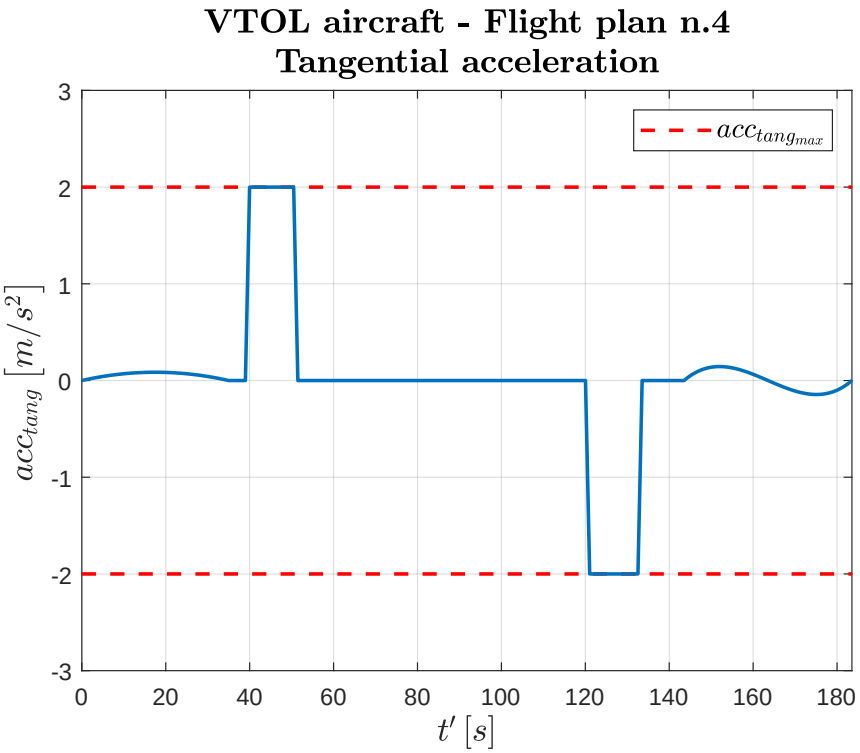

Figure 5.49: Time history of tangential acceleration - Flight plan n.4

Chapter 6

Simulation of the complete scheme for the wingborne phase

The aim of this chapter is to introduce some advantages that the flatness approach, used for the trajectory generator, brings to the complete control scheme of a nonlinear system. In particular, the possibility of having a feedforward control using only algebraic relationships, and so without resolving any differential equations, is guaranteed by Eq. (3.41) and Eq. (3.45) for wingborne and hover phase respectively. Although the flatness approach permits to generate the trajectory with a unique generator and the flatness relationships allow to write the control inputs for both phases, only wingborne phase is taken into account in this chapter because a `Simulink` scheme which considers both models requires a deeper analysis. These simulations can be seen as an introduction to the trajectory control problem of a VTOL transition aircraft based on the flatness theory.

First of all, the used simple VTOL model is explained in Section 6.1. It has a part containing the exact mathematical model of the wingborne phase and a block which simulates the presence of the actuators. It is a really simplified model but its fidelity is sufficient for the tasks of this thesis.

After that, Section 6.2 shows the `Simulink` model of the trajectory controller, which consists of two parts, namely feedforward and feedback ones.

Finally, the results and the corresponding analysis are presented in Section 6.3. In particular three different schemes are tested:

- Section 6.3.1 analyzes the results assuming an ideal VTOL model, without the actuators, and using only the feedforward control;
- Section 6.3.2 analyzes the results assuming a VTOL model considering the actuators and using only the feedforward control;
- Section 6.3.3 analyzes the results assuming a VTOL model considering the actuators and using both feedforward and feedback controls.

The entire `Simulink` scheme is represented in Fig. 6.1.

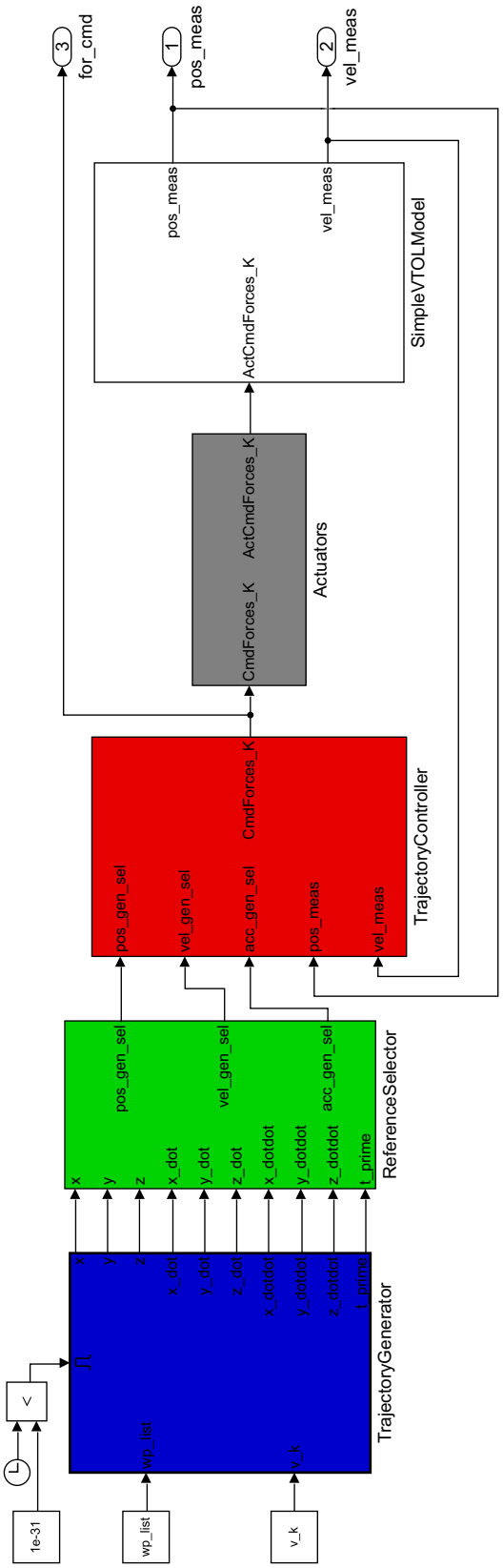

Figure 6.1: Complete control scheme

6.1 Simple VTOL model

The simple VTOL model is realized adding the effect of the actuators to the wing-borne mathematical model which corresponds to Eq. (3.17) and is implemented in Fig. 6.2.

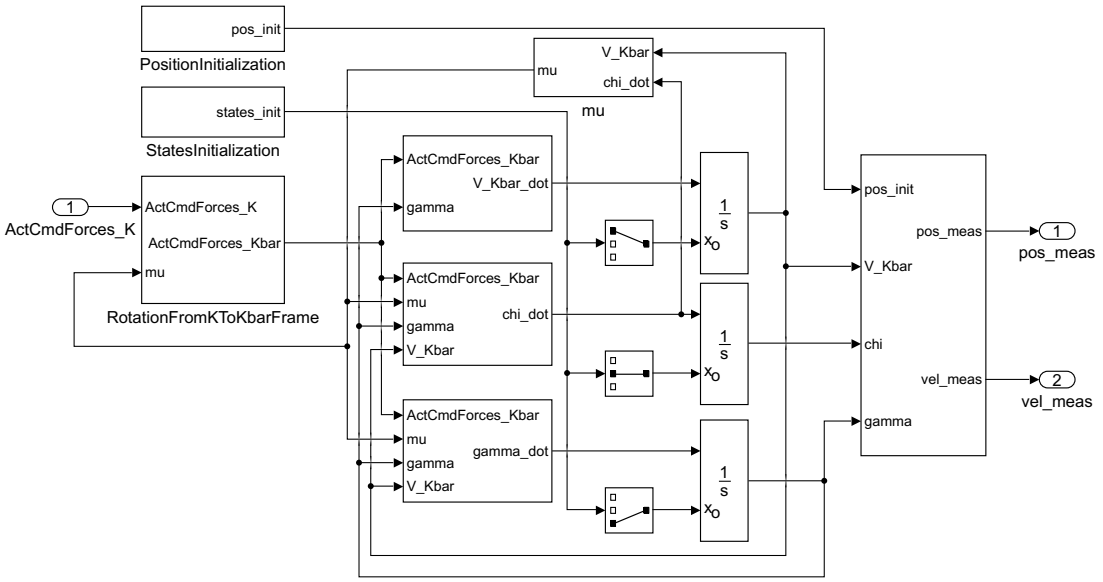

Figure 6.2: Mathematical model block

Firstly, the rotation of the command forces from the K frame to the $\bar{K}$ one is performed using the transpose of the rotation matrix in Eq. (3.42). This passage is necessary because the equations of the mathematical model need the forces in $\bar{K}$ frame.

After having computed the variables $\dot{V}_A^a$, $\dot{\chi}$ and $\dot{\gamma}$ using Eq. (3.35), they are integrated in order to obtain V_A^a, χ and γ. It is fundamental to set the proper initial values for the integrator so that the simulation works in the right way. It is worth to notice that the computation of μ angle is useful to complete the steps just mentioned.

The states V_A^a, χ and γ are used to find the derivative of position vector $\dot{x}$, $\dot{y}$ and $\dot{z}$ using the first three equations of Eq. (3.17). Finally, the latter ones are integrated to obtain the position components x, y and z. As already said for the other integrators, it is very important to set the most proper initial values to reach reliable results.

The actuators dynamics is modeled as a second order transfer function:

$$G_{actuator}(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{6.1}$$

whose natural frequency ω_n and damping ratio ζ are reported in Table 6.1.

ω_n $[1/s]$	ζ $[-]$
20	1

Table 6.1: Features of the actuators transfer function

The `Simulink` block is shown in Fig. 6.3.

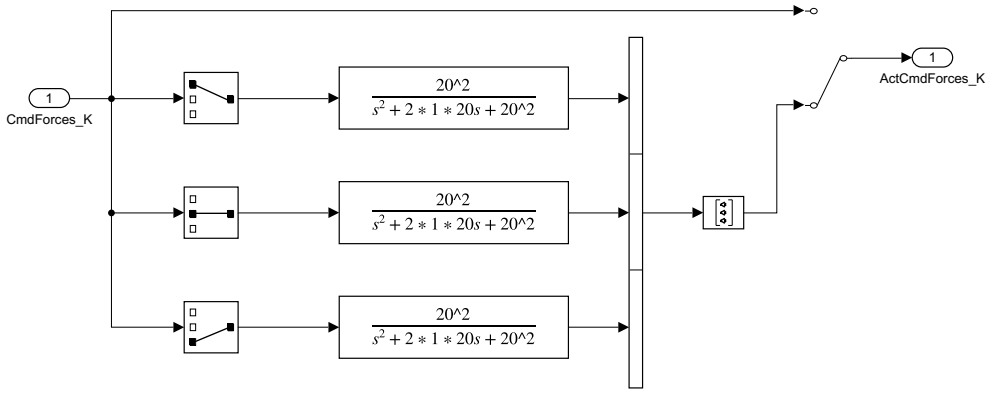

Figure 6.3: Actuators block

It is possible to notice that the `Manual Switch` is inserted in order to stop the actuators effects when the simulation of Section 6.3.1 is performed.

6.2 Trajectory controller

The control part of the scheme is composed by two blocks:

- the `ReferenceSelector` chooses the correct reference for each variable among all the ones provided by the generator at each time instant of the simulation;
- the `TrajectoryController` computes the control inputs of the simple VTOL model, explained in the previous section.

Starting from the first one, shown in Fig. 6.4, the basic idea is to select the reference comparing the actual simulation time and t' provided by the trajectory generator. The reason is that the `TrajectoryGenerator` block is off-line and so all the quantities, that describe the reference trajectory, are already existing from the beginning of the simulation.

To this aim, a counter selects the reference values of x, y, z and theirs derivatives thanks to a `Select Rows` block. The difference between the selected t' and the actual time, provided by `Clock` block, is computed; whenever it becomes

smaller than a prescribed value, a `Triggered Subsystem` is activated and the value of the counter is increased.

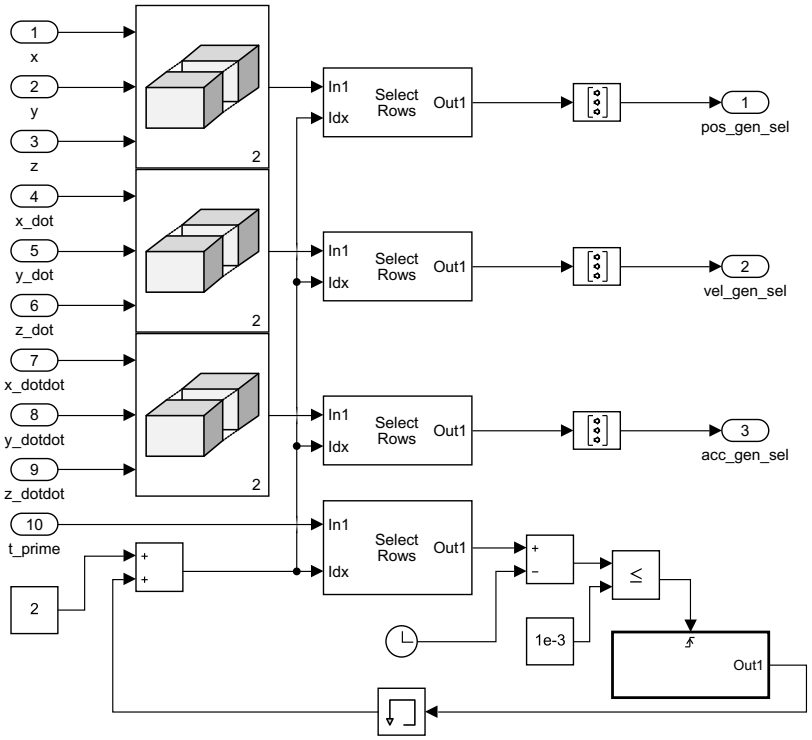

Figure 6.4: Reference selector

This method works well for this thesis but, obviously, other methods could be implemented to select the reference values; for example, a condition on the error position could be another solution.

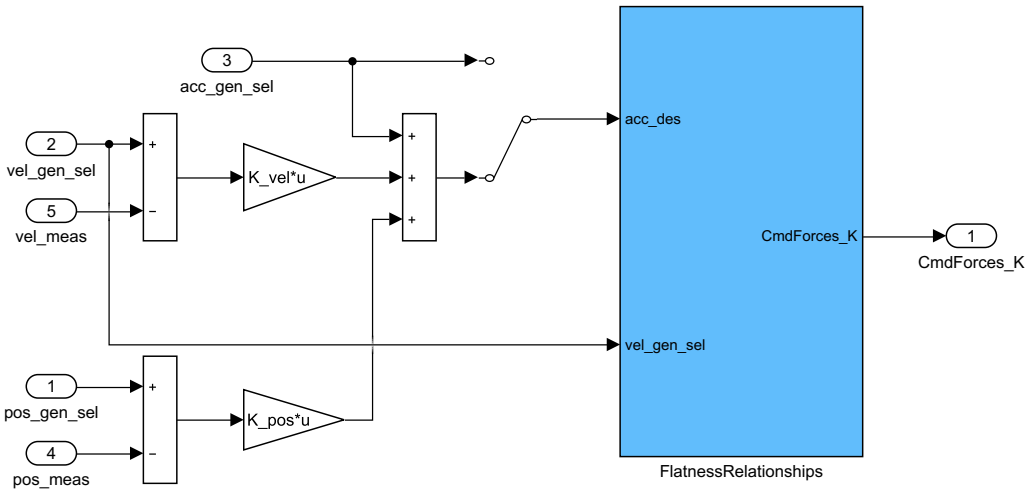

Figure 6.5: Trajectory controller

On the other hand, the `TrajectoryController`, whose `Simulink` model is reported in Fig. 6.5, is built as a combination of feedforward and feedback parts; as explained in Chapter 2, the former one steers the aircraft along the desired trajectory while the latter one breaks down the error between the desired path and the actual one.

The `FlatnessRelationships` receives the desired acceleration $\mathbf{acc_{des}}$ and the generated speed $\mathbf{vel_{gen_{sel}}}$ as inputs and provides the command forces in the K frame as outputs. This block, shown in Fig. 6.6, implements the equations of Section 3.4.1, which relate the states and control inputs to flat outputs and their derivatives. It is worth to notice that they are concatenated because the states are necessary to compute the commands.

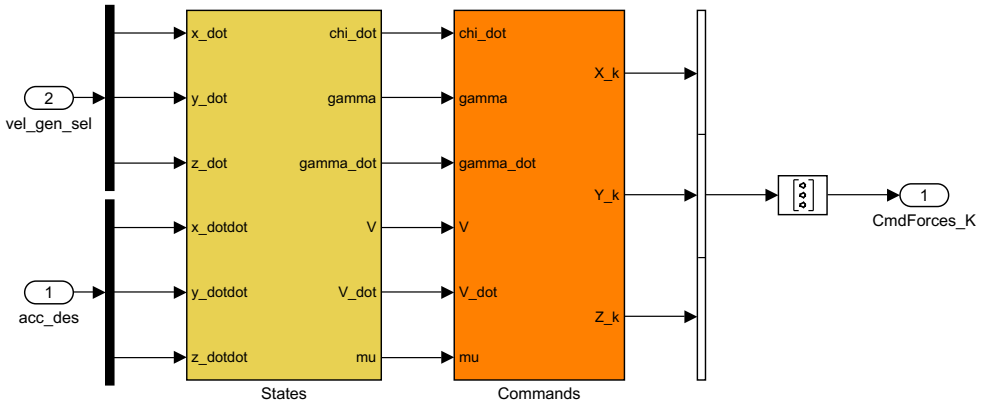

Figure 6.6: Flatness relationships block

Referring to Fig. 6.5, it is possible to understand which control law is implemented. The variable $\mathbf{vel_{gen_{sel}}}$ comes directly from the `ReferenceSelector` while $\mathbf{acc_{des}}$ is influenced by the feedback as follows:

$$\mathbf{acc_{des}} = \mathbf{acc_{gen_{sel}}} + \mathbf{K_{vel}}\left(\mathbf{vel_{gen_{sel}}} - \mathbf{vel_{meas}}\right) + \mathbf{K_{pos}}\left(\mathbf{pos_{gen_{sel}}} - \mathbf{pos_{meas}}\right) \quad (6.2)$$

where $\mathbf{acc_{des}}$ is the desired acceleration, $\mathbf{pos_{gen_{sel}}}$, $\mathbf{vel_{gen_{sel}}}$ and $\mathbf{acc_{gen_{sel}}}$ are the selected reference values of position, speed and acceleration, $\mathbf{pos_{meas}}$ and $\mathbf{vel_{meas}}$ are the measured position and speed. All these variables are vectors of dimension 3×1. Finally, $\mathbf{K_{vel}}$ and $\mathbf{K_{pos}}$ are the gain matrices of dimension 3×3 and they are diagonal.

A `Manual Switch` permits to deactivate the feedback during the simulations of Section 6.3.1 and Section 6.3.2.

6.3 Results and analysis

This section presents the results and the corresponding analysis for the simulations summarized in Table 6.2. The used waypoint list is the same of flight plan n.4 of Chapter 5 considering only the wingborne phase and hypothesizing that the speed

starts and ends at 25 m/s; the consequence is that there are no acceleration and deceleration phases.

n.	Name	Reference
1	Feedforward without Actuators	Section 6.3.1
2	Feedforward with Actuators	Section 6.3.2
3	Feedforward + Feedback	Section 6.3.3

Table 6.2: Simulations of the complete scheme

During all the simulations the mass m of the VTOL aircraft and the gravity acceleration g are set constant and equal to 5 kg and 9.81 m/s^2 respectively.

6.3.1 Feedforward without actuators

Simulating the model without the actuators and using only the feedforward action based on the flatness relationships, the trajectory is perfectly followed as expected. The motivation is that the simple VTOL model without the actuators and any other uncertainties corresponds exactly to the mathematical one used to find the flatness relationships. Despite it is an ideal case, this analysis is very useful to verify the exactness of the flatness relationships used to build the feedforward control part.

The graphs below compare the generated trajectory with the simulated one. In particular Fig. 6.7 and Fig. 6.8 show the path in the YX plane and in the XZ one respectively.

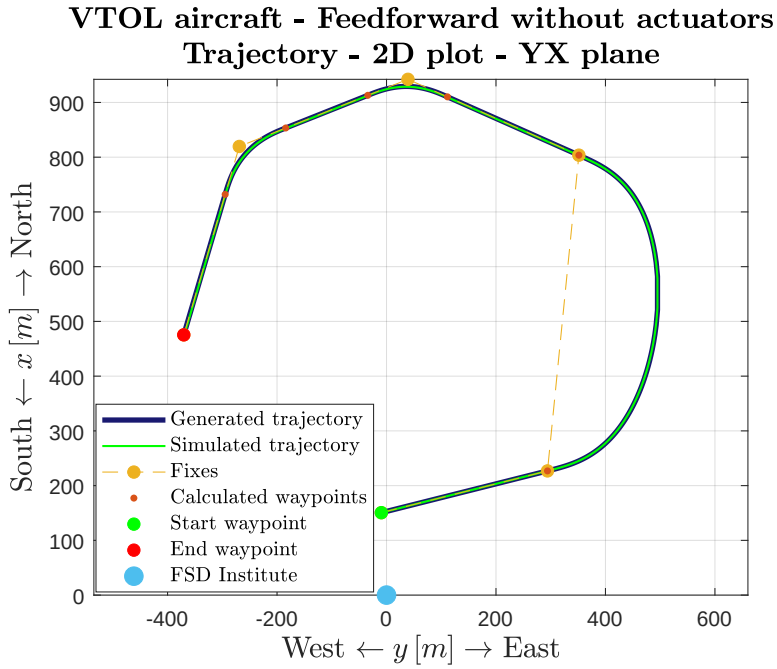

Figure 6.7: 2D trajectory in YX plane - Feedforward without actuators

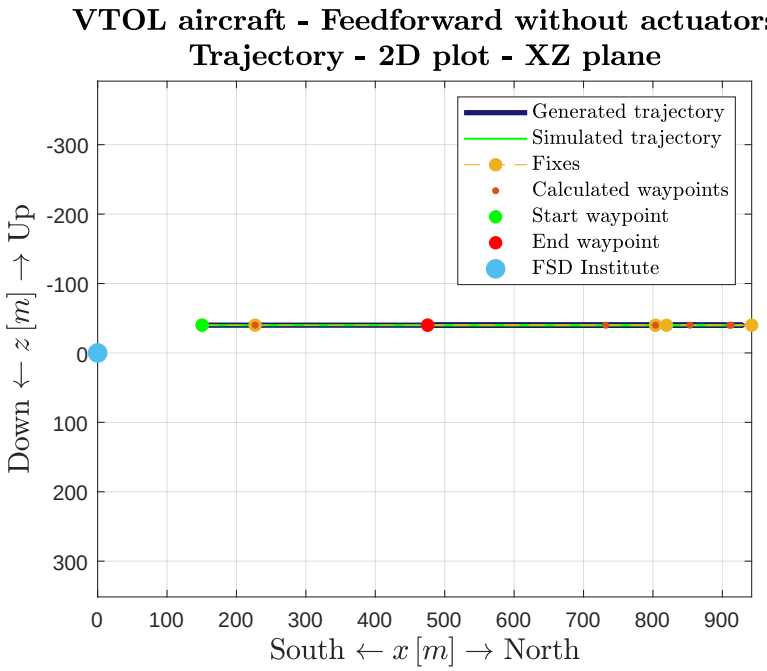

Figure 6.8: 2D trajectory in XZ plane - Feedforward without actuators

The time histories of the position and speed components and the speed absolute value are reported in Fig. 6.9, Fig. 6.10 and Fig. 6.11.

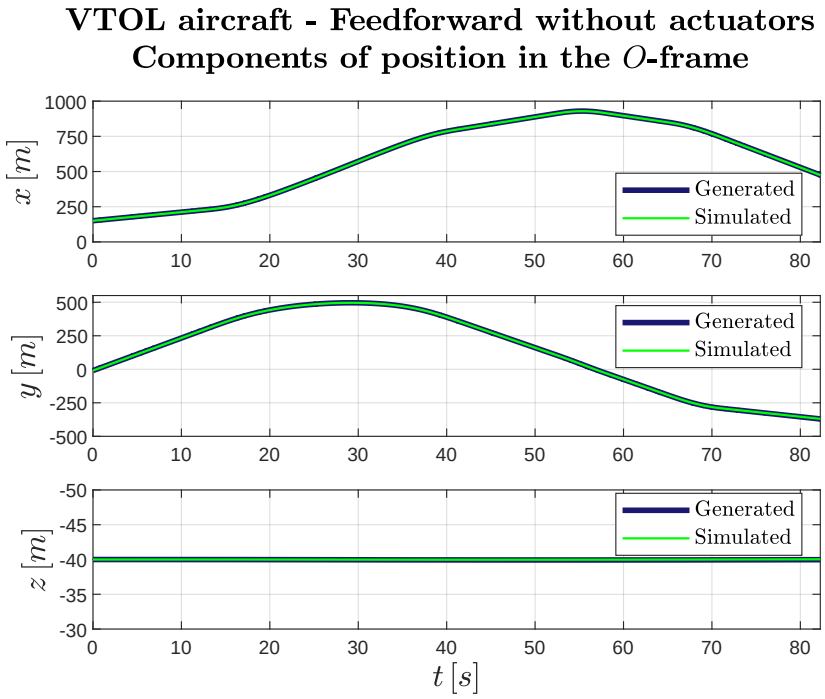

Figure 6.9: Time history of position components - Feedforward without actuators

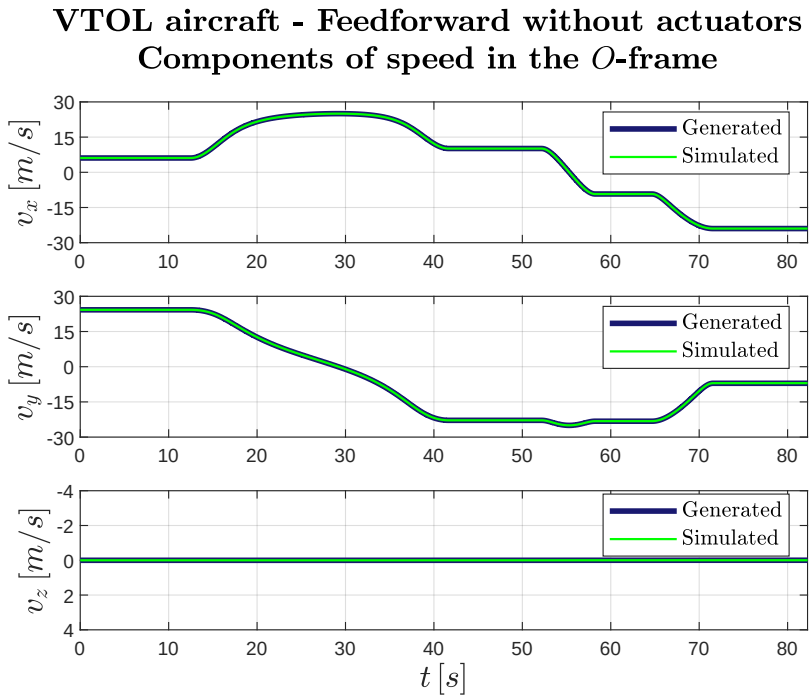

Figure 6.10: Time history of speed components - Feedforward without actuators

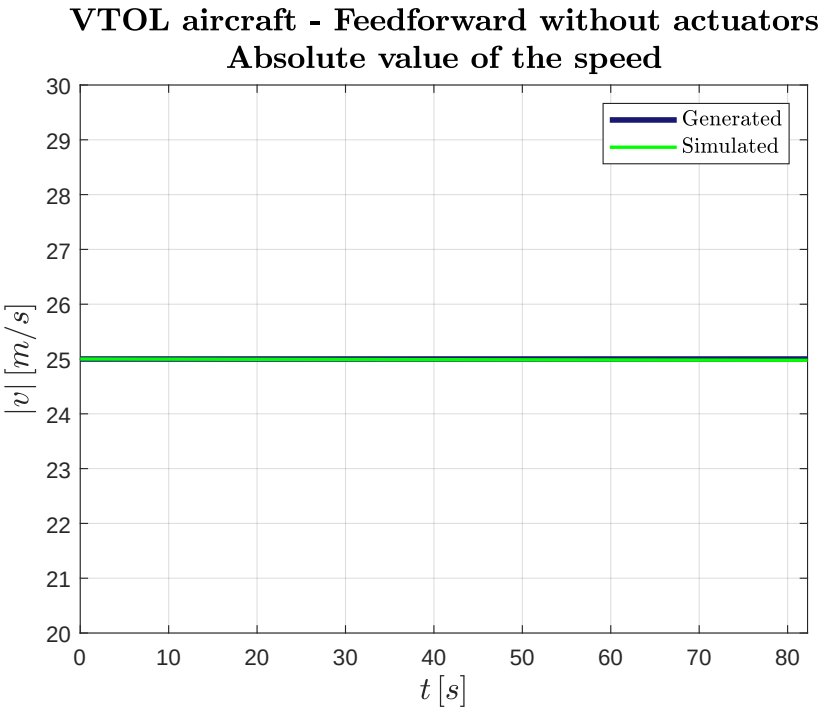

Figure 6.11: Time history of speed absolute value - Feedforward without actuators

Finally, it is possible to see the evolution in time of each component of the command force from Fig. 6.12.

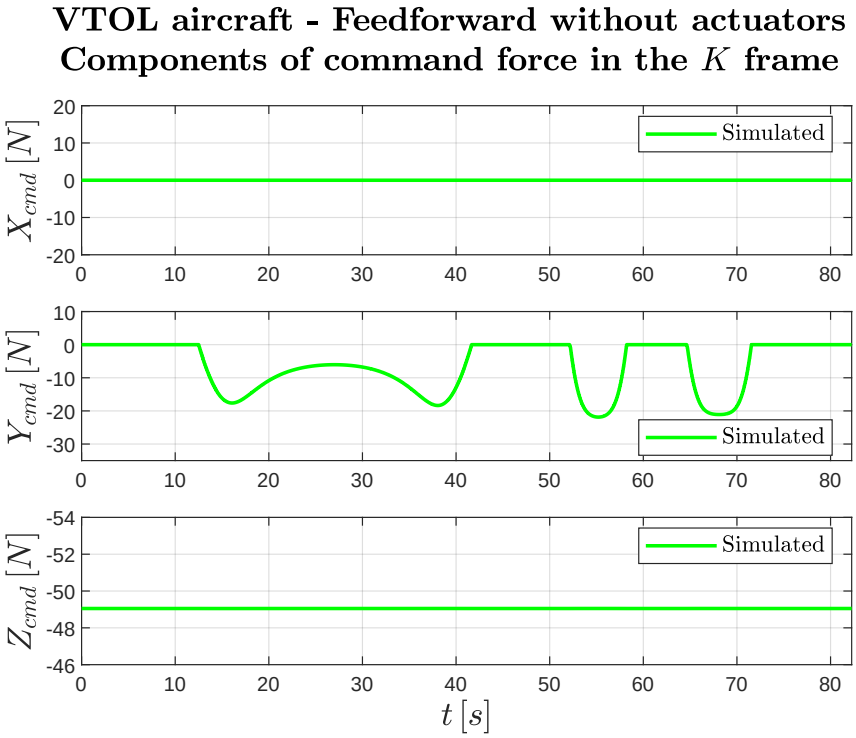

Figure 6.12: Time history of command force components in K frame - Feedforward without actuators

The following considerations can be done:

- X_K is always equal to 0 N because the absolute value of the speed is constant during the entire trajectory;
- Y_K differs from 0 N during the turns and its sign during them is always negative because they are on the left;
- Z_K is always equal to -49.05 N. This force is perfectly equal to mg during an horizontal flight.

6.3.2 Feedforward with actuators

The aim of this subsection is to demonstrate that the feedforward action is not sufficient to follow the trajectory when the actuators are inserted in the VTOL model. Looking at Fig. 6.13 it is evident that the simulated trajectory is completely different from the desired one in the YX plane. The difference between the two trajectories is not only in this plane but also in the z direction [Fig. 6.14].

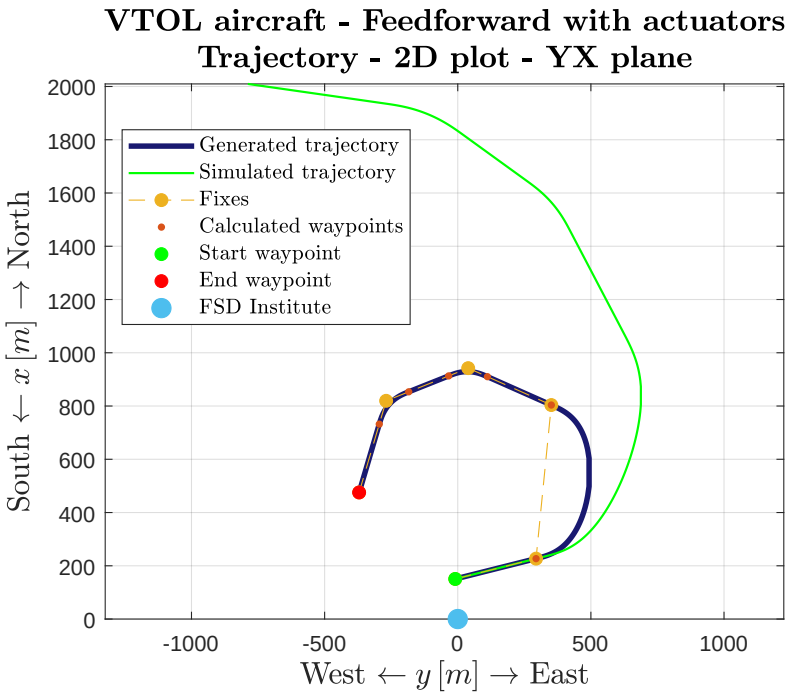

Figure 6.13: 2D trajectory in YX plane - Feedforward with actuators

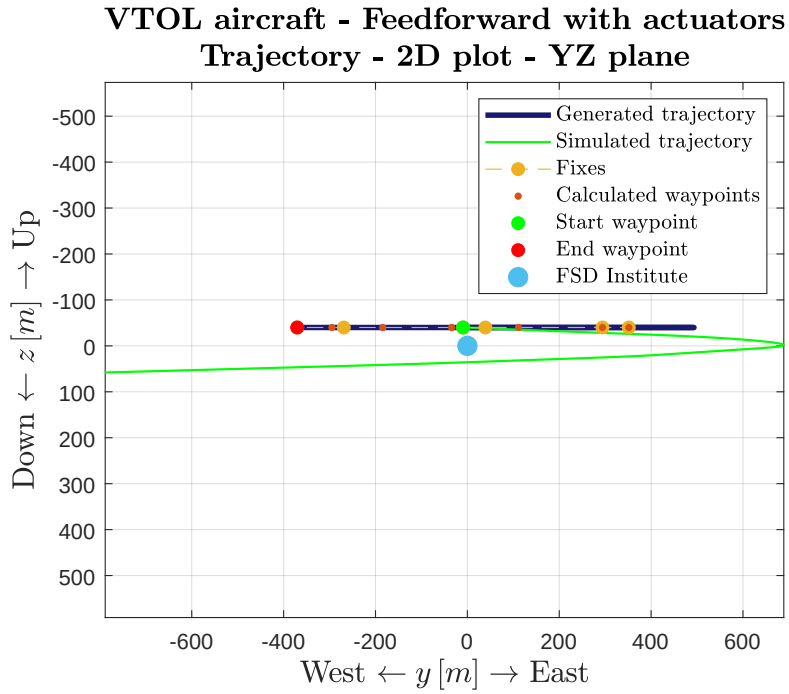

Figure 6.14: 2D trajectory in YZ plane - Feedforward with actuators

The same conclusions can be reached also analyzing the time histories of the position and speed components and the absolute value of the speed. In particular the fact that v_z remains always positive, which means that it points downwards,

causes a descent for which the aircraft passes the 0 m altitude and so the ground; obviously it is not physical.

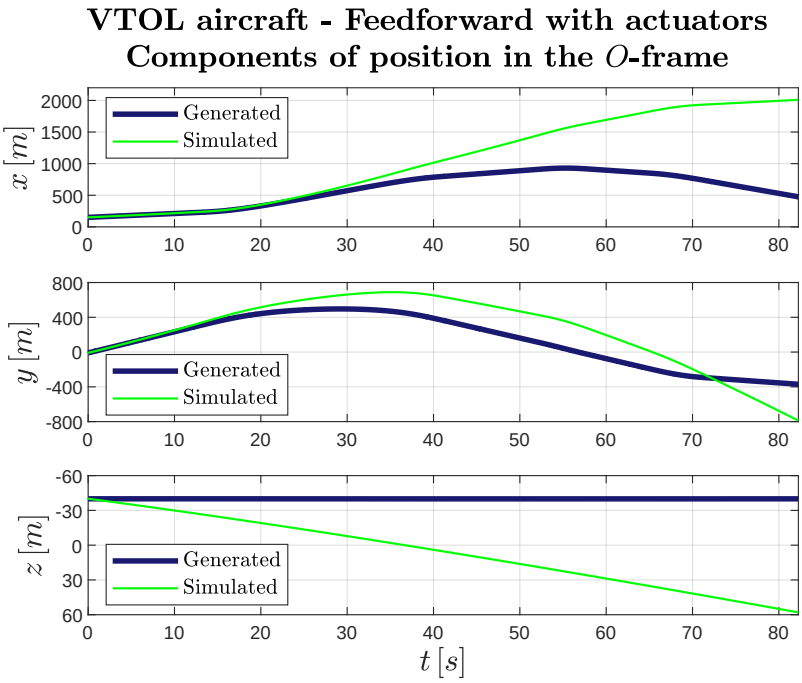

Figure 6.15: Time history of position components - Feedforward with actuators

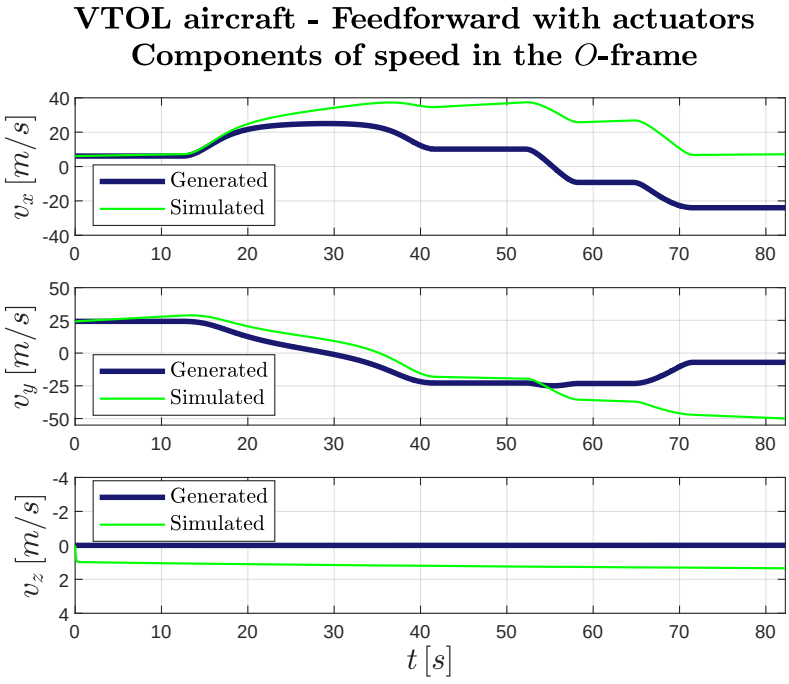

Figure 6.16: Time history of speed components - Feedforward with actuators

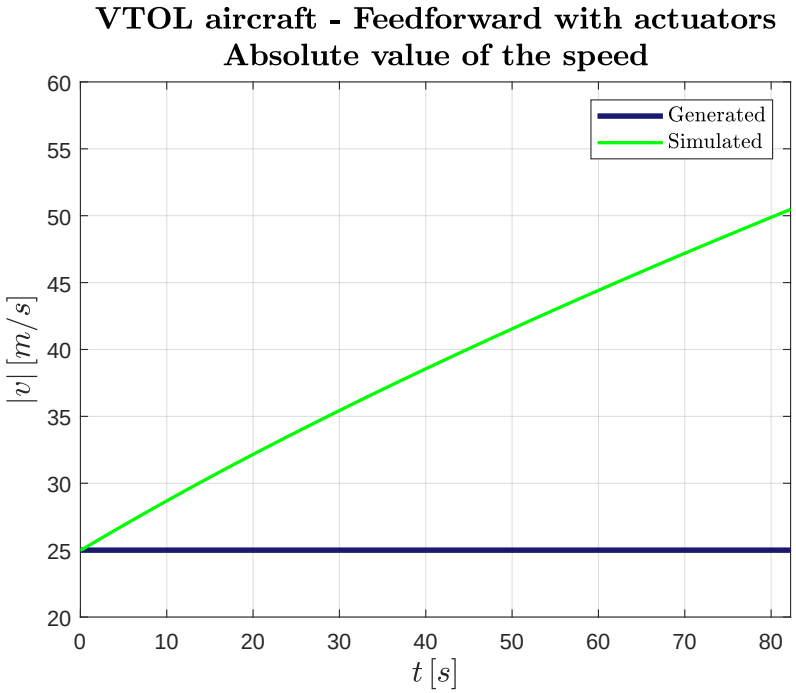

Figure 6.17: Time history of speed absolute value - Feedforward with actuators

The components of the command force are the same of Fig. 6.12.

6.3.3 Feedforward + feedback

In order to solve the problems exposed in the previous section, the addition of the feedback action is required. It is based on the control law of Eq. (6.2) where the gains are tuned in an experimental way; starting from low values and taking into account that the velocity loop is faster than the position one, the gains are increased until satisfactory results are obtained. They are reported in Table 6.3.

		x	y	z
K_{pos}	$[1/s^2]$	0.1	0.1	0.1
K_{vel}	$[1/s]$	1	1	1

Table 6.3: Feedback gains

Firstly the usefulness of the feedback control part is evident comparing the geometry of the simulated and the desired trajectories reported in Fig. 6.18 and Fig. 6.19.

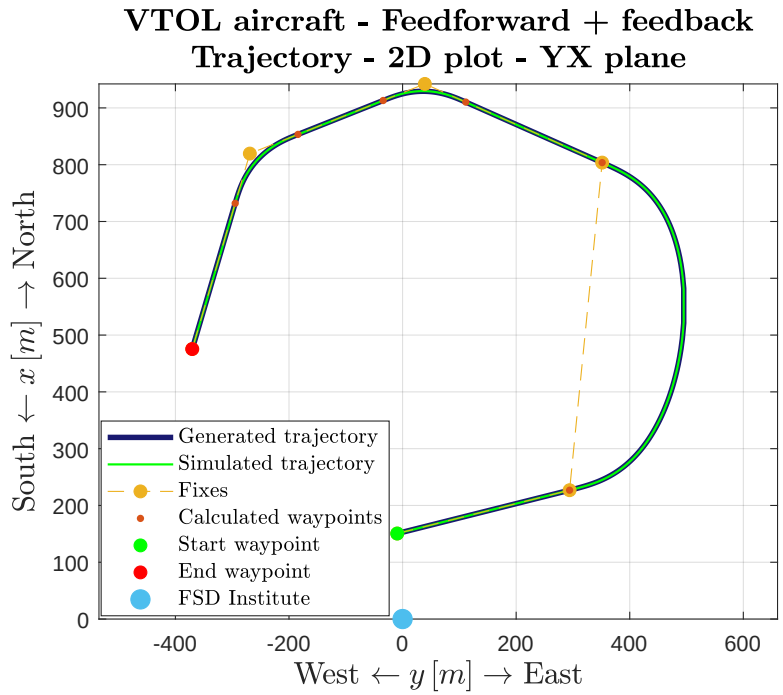

Figure 6.18: 2D trajectory in YX plane - Feedforward + feedback

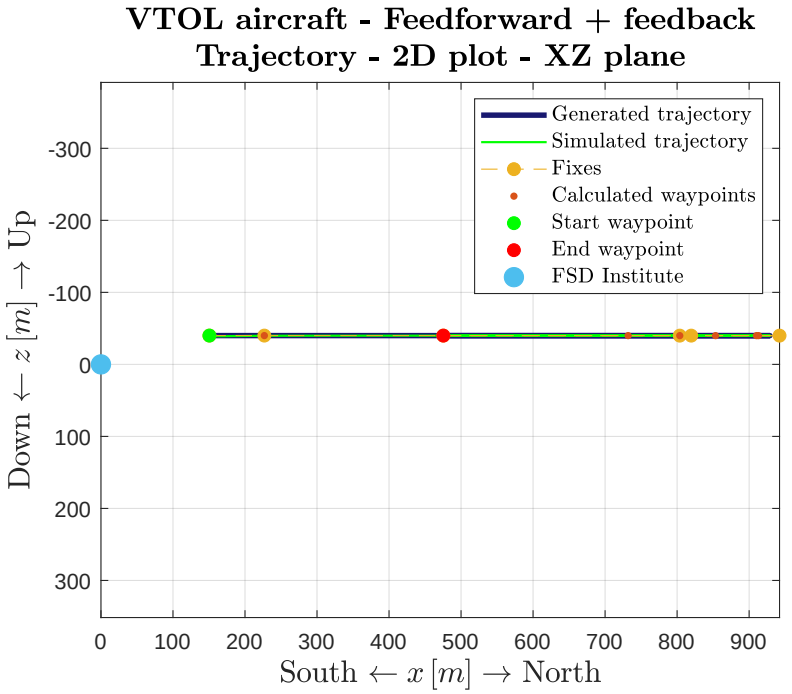

Figure 6.19: 2D trajectory in XZ plane - Feedforward + feedback

The evolution in time of the position and speed components and the absolute value of speed in Fig. 6.20, Fig. 6.21 and Fig. 6.22 shows that the feedback control action allows to reduce significantly the error and follow the trajectory in a proper

way. In particular, the initial error on the z components of position and speed is immediately brought to a low value.

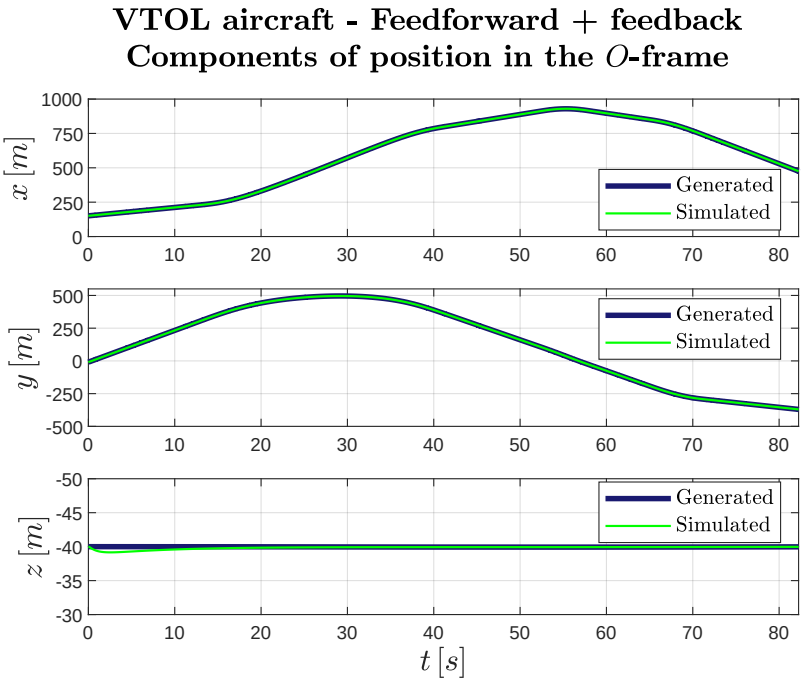

Figure 6.20: Time history of position components - Feedforward + feedback

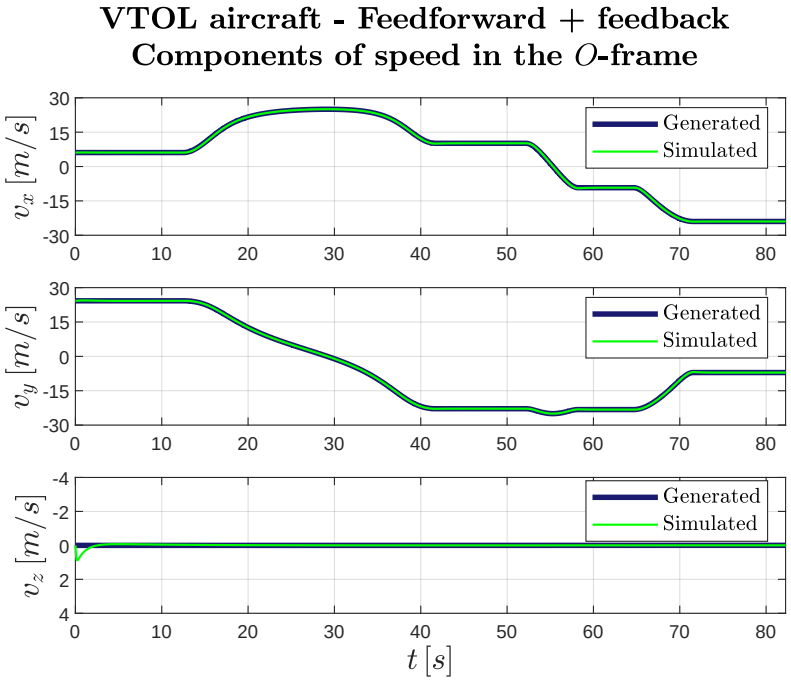

Figure 6.21: Time history of speed components - Feedforward + feedback

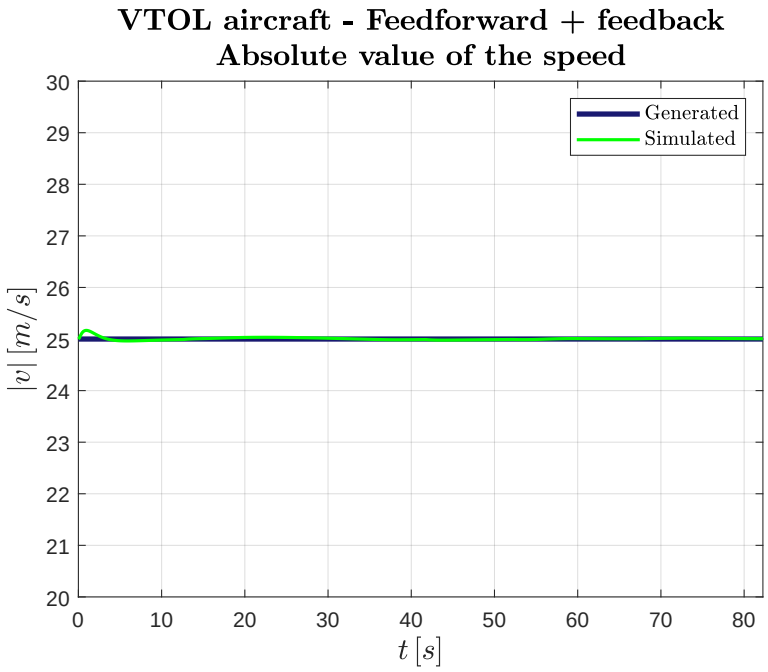

Figure 6.22: Time history of speed absolute value - Feedforward + feedback

Finally the components of the command force in Fig. 6.23 are very similar to the ones in Fig. 6.12 in the x and y directions while the most remarkable difference is in Z_K whose initial values, not equal to $-49.05\ N$, permit to overcome the initial error in the z direction.

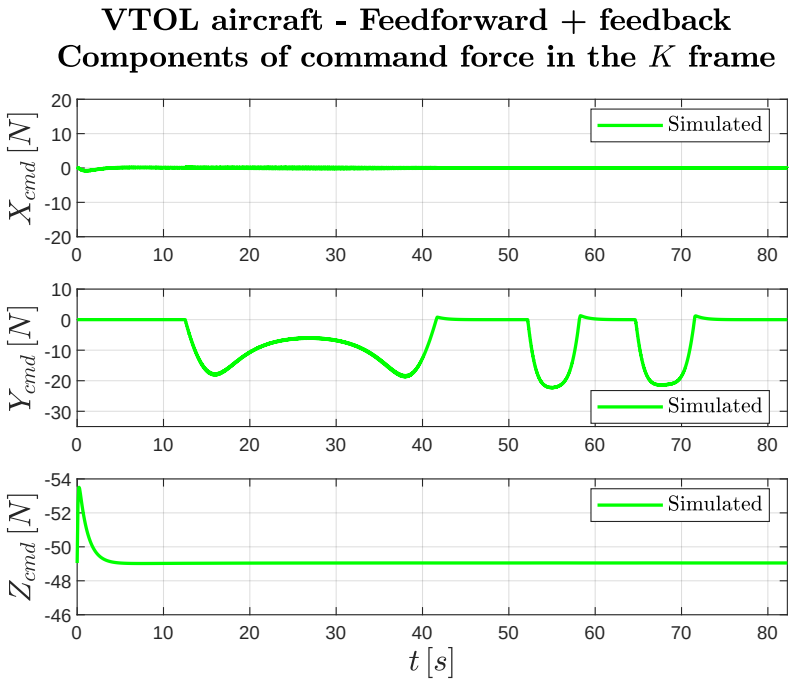

Figure 6.23: Time history of command force components in K frame - Feedforward + feedback

Chapter 7

Conclusion and further developments

This is the last chapter of the thesis and provides a final recap of the work done. Firstly, Section 7.1 briefly explains all the theoretical choices which have been taken to accomplish the requirements and shows all the reached results. Finally, Section 7.2 presents possible future works which can improve the obtained results.

7.1 Conclusion

The first conclusion is that the position loop of the VTOL transition aircraft dynamics can be described by two different models: the first one for the wingborne phase is based on the equations for a conventional airplane while the second one for the hover phase is characterized by the equations of quadrotor dynamics.

Secondly, the complete system containing both phases is flat and the flat outputs are the coordinates of position in NED frame, x, y and z. The implemented algorithm generates the trajectory by using four kinds of curves, namely first degree polynomials, fifth degree polynomials, double S trajectories and circular arcs, in order to describe all the flight phases and their related maneuvers.

The four flight plans, whose results and analysis are treated in Chapter 5, demonstrate that the flatness-based trajectory generator can provide trajectories which involve both wingborne and hover phases and can take into account all the fixed requests. A complete summary of the requirements and the respective solution is reported in Table 7.1 in order to be as clear as possible.

Requirement	**Solution**
3D trajectory	Set of scalar problems synchronized in time

Different types of trajectories between each couple of waypoints to take into account all the flight phases and maneuvers	Sequence of point-to-point segments
Continuity in position, speed and acceleration	First degree polynomials for straight lines and fifth degree polynomials for curves
Constant speed equal to 25 m/s during wingborne phase	Scaling in time for the fifth degree polynomials
Maximum value of $\dot{\chi}$ equal to 10 deg/s	Proper trajectory planning or variation of $\dot{\chi}_{des}$ or saturation with circular arc
Acceleration between hover and wingborne phase with tangential acceleration maximum value equal to 2 m/s^2 and deceleration for the opposite case	Double S trajectory

Table 7.1: Summary of requirements-solutions

Finally, the simulations of the complete scheme for wingborne phase introduce the advantages of flatness property by the point of view of trajectory control. The results demonstrate that a feedforward control which steers the system along the desired trajectory can be built using the flatness relationships, which are only algebraic. In addition, it is proved that only a feedforward control is not sufficient to follow the trajectory when the actuators transfer functions are inserted in the model but it is also shown that a simple feedback law can be added to the feedforward and allow to track the trajectory in a good way.

7.2 Further developments

Possible future works connected to the topic of this thesis are listed below.

- The first improvement is related to the trajectory control part. Since the gains are found through an experimental way, the study of the error dynamics would be a more rigorous method to tune the parameters. Subsequently, the criteria to select the references could consider also the error on the position because the implemented one can be inaccurate if the model is more complicated.

- The hover phase could be added to the complete control scheme in order to simulate a flight plan which involves both phases, such as the flight plan n.4 defined in Section 5.4. It consists not only in the insertion of the mathematical model in the simple VTOL model but also in the addition of the hover flatness relationships in the feedforward control part. An important aspect

would be to find an accurate method to switch the two models in order to deal with the transition between the two phases.

- Control strategies based on flatness theory, as the exact feedforward linearization, could be applied and tested to follow the desired trajectory provided by the flatness-based trajectory generator presented in this thesis.

- The complete set of equations of VTOL transition aircraft dynamics, which considers also the attitude loop, could be treated under a flatness approach. So, a new flat outputs vector would be defined and a larger set of flatness relationships would be found. The latter ones could be exploited by building a complete flatness-based trajectory generator and controller for the attitude and position loops.

- The simple VTOL model could be substituted by an higher fidelity one which considers not only the actuators but also the sensors, the ground contacts, the propellers and the control aerodynamic surfaces. In addition, also the wind and aeroelastic effects can be added to the model since two of the initial hypotheses are the zero wind condition and no aeroelasticity.

Appendix A

Matlab scripts

A.1 Geometry calculation

Fly-by

```
function wp_flyby = calculus_flyby(wp_0,wp_1,wp_2,v_k,chi_dot_des)

% Extrapolation of waypoints features
id_0 = wp_0(1);
x_0 = wp_0(2);
y_0 = wp_0(3);
z_0 = wp_0(4);
chi_0 = wp_0(5);
gamma_0 = wp_0(6);

x_1 = wp_1(2);
y_1 = wp_1(3);
z_1 = wp_1(4);
chi_1 = wp_1(5);
gamma_1 = wp_1(6);

x_2 = wp_2(2);
y_2 = wp_2(3);
z_2 = wp_2(4);
chi_2 = wp_2(5);
gamma_2 = wp_2(6);

% Creation of vectors
r_0 = [x_0,y_0,z_0]';
r_1 = [x_1,y_1,z_1]';
r_2 = [x_2,y_2,z_2]';

% Speed
v_k = v_k(3);
```

```
% Alpha
delta_chi = abs(chi_2-chi_1);

alpha_t = abs(pi-delta_chi);

% Turn rate
chi_dot = chi_dot_des*pi/180;

% Turn radius
r_c = v_k/chi_dot;

s_turn = abs(r_c/tan(alpha_t/2));

% S_2 and S_1
S_1 = s_turn;
S_2 = s_turn;

% New positions
r_3 = r_1+S_1*(r_0-r_1)/(norm(r_0-r_1));
r_4 = r_1+S_2*(r_2-r_1)/(norm(r_2-r_1));

% New waypoints
if (id_0 == 4) || (id_0 == 6) || (id_0 == 7)
wp_3 = [4;r_3;chi_1;gamma_1];
else
wp_3 = [0;r_3;chi_1;gamma_1];
end

wp_4 = [1;r_4;chi_2;gamma_2];

wp_flyby = [wp_3;wp_4];
```

Fly-over

```
function wp_flyover = calculus_flyover(wp_1, wp_2)

% Extrapolation of waypoints features
x_1 = wp_1(2);
y_1 = wp_1(3);
z_1 = wp_1(4);
chi_1 = wp_1(5);
gamma_1 = wp_1(6);

x_2 = wp_2(2);
y_2 = wp_2(3);
z_2 = wp_2(4);
chi_2 = wp_2(5);
gamma_2 = wp_2(6);

% Creation of vectors
r_1 = [x_1,y_1,z_1]';
```

```
r_2 = [x_2,y_2,z_2]';

% Distance and S
distance = sqrt((x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2);
S = 2/3*distance;

% New positions
r_3 = r_1+S*(r_2-r_1)/(norm(r_2-r_1));

% New waypoints
wp_1 = [0;r_1;chi_1;gamma_1];
wp_3 = [1;r_3;chi_2;gamma_2];

wp_flyover = [wp_1;wp_3];
```

Radius-to-fix

```
function wp_radiustofix = calculus_radiustofix(wp_1, wp_2)

% Extrapolation of waypoints features
x_1 = wp_1(2);
y_1 = wp_1(3);
z_1 = wp_1(4);
chi_1 = wp_1(5);
gamma_1 = wp_1(6);

x_2 = wp_2(2);
y_2 = wp_2(3);
z_2 = wp_2(4);
chi_2 = wp_2(5);
gamma_2 = wp_2(6);

% Creation of vectors
r_1 = [x_1,y_1,z_1]';
r_2 = [x_2,y_2,z_2]';

% New waypoints
wp_1 = [1;r_1;chi_2;gamma_2];

wp_radiustofix = [wp_1;zeros(6,1)];
```

Vertical fly-by

```
function wp_verticalflyby = calculus_verticalflyby(wp_0,wp_1,...
    wp_2,vert_flyby_dist)

% Extrapolation of waypoints features
x_0 = wp_0(2);
```

```
y_0 = wp_0(3);
z_0 = wp_0(4);
chi_0 = wp_0(5);
gamma_0 = wp_0(6);

x_1 = wp_1(2);
y_1 = wp_1(3);
z_1 = wp_1(4);
chi_1 = wp_1(5);
gamma_1 = wp_1(6);

x_2 = wp_2(2);
y_2 = wp_2(3);
z_2 = wp_2(4);
chi_2 = wp_2(5);
gamma_2 = wp_2(6);

% Creation of vectors
r_0 = [x_0,y_0,z_0]';
r_1 = [x_1,y_1,z_1]';
r_2 = [x_2,y_2,z_2]';

% S_1 and S_2
S_1 = vert_flyby_dist;
S_2 = vert_flyby_dist;

% New positions
r_3 = r_1+S_1*(r_0-r_1)/(norm(r_0-r_1));
r_4 = r_1+S_2*(r_2-r_1)/(norm(r_2-r_1));

% New waypoints
if z_2 - z_0 < 0
wp_3 = [2;r_3;chi_1;gamma_1];
else
wp_3 = [5;r_3;chi_1;gamma_1];
end
wp_4 = [3;r_4;chi_2;gamma_2];

wp_verticalflyby = [wp_3;wp_4];
```

A.2 Algorithms

Horizontal straight line

```
function [x,y,z,x_dot,y_dot,z_dot,x_dotdot,y_dotdot,z_dotdot,t,...
    t_prime] = horizontalstraightline(WP_0,WP_1,v_k)

% Extrapolation of waypoints data
% Waypoint 0
X_0 = WP_0(2);
```

```
Y_0 = WP_0(3);
Z_0 = WP_0(4);

% Waypoint 1
X_1 = WP_1(2);
Y_1 = WP_1(3);
Z_1 = WP_1(4);

% Speed
v_k = v_k(3);

% Time
distance = sqrt((X_1-X_0)^2+(Y_1-Y_0)^2+(Z_1-Z_0)^2);
delta_time = distance/v_k;
t = linspace(0,delta_time,1000);
t_prime = t;

% Construction of straight lines

% x direction
a_x_0 = X_0;
a_x_1 = (X_1-X_0)/delta_time;
x = a_x_0+a_x_1*t;

% y direction
a_y_0 = Y_0;
a_y_1 = (Y_1-Y_0)/delta_time;
y = a_y_0+a_y_1*t;

% z direction
a_z_0 = Z_0;
a_z_1 = (Z_1-Z_0)/delta_time;
z = a_z_0+a_z_1*t;

% First derivative
x_dot = a_x_1*ones(1,1000);
y_dot = a_y_1*ones(1,1000);
z_dot = a_z_1*ones(1,1000);

% Second derivative
x_dotdot = zeros(1,1000);
y_dotdot = zeros(1,1000);
z_dotdot = zeros(1,1000);
```

Horizontal curve

```
function [x,y,z,x_dot,y_dot,z_dot,x_dotdot,y_dotdot,z_dotdot,t,...
    t_prime] = horizontalcurve(WP_0,WP_1,v_k)

% Extrapolation of waypoints data
% Waypoint 0
```

```
X_0 = WP_0(2);
Y_0 = WP_0(3);
Z_0 = WP_0(4);
chi_0 = WP_0(5);
gamma_0 = WP_0(6);

% Waypoint 1
X_1 = WP_1(2);
Y_1 = WP_1(3);
Z_1 = WP_1(4);
chi_1 = WP_1(5);
gamma_1 = WP_1(6);

% Speed
v_k = v_k(3);

% COMPUTATION OF DELTA_TIME WITH STRAIGHT LINE
% Time
distance = sqrt((X_1-X_0)^2+(Y_1-Y_0)^2+(Z_1-Z_0)^2);
delta_time = distance/v_k;
t = linspace(0,delta_time,1000);

% Speed
v = [v_k,0,0]';

% v_O_0
M_OK_0 = [cos(chi_0)*cos(gamma_0) -sin(chi_0) ...
    cos(chi_0)*sin(gamma_0);sin(chi_0)*cos(gamma_0) cos(chi_0) ...
    sin(chi_0)*sin(gamma_0);-sin(gamma_0) 0 cos(gamma_0)];
v_O_0 = M_OK_0*v;

% v_O_1
M_OK_1 = [cos(chi_1)*cos(gamma_1) -sin(chi_1) ...
    cos(chi_1)*sin(gamma_1);sin(chi_1)*cos(gamma_1) cos(chi_1) ...
    sin(chi_1)*sin(gamma_1);-sin(gamma_1) 0 cos(gamma_1)];
v_O_1 = M_OK_1*v;

% x direction
v_x_0 = v_O_0(1);
v_x_1 = v_O_1(1);

% y direction
v_y_0 = v_O_0(2);
v_y_1 = v_O_1(2);

% z direction
v_z_0 = v_O_0(3);
v_z_1 = v_O_1(3);

% Acceleration
% x direction
acc_x_0 = 0;
acc_x_1 = 0;
```

```
% y direction
acc_y_0 = 0;
acc_y_1 = 0;

% z direction
acc_z_0 = 0;
acc_z_1 = 0;

% Construction of fifth degree polynomial

% x direction
% Definition of coefficients a_0 to a_2
a_x_0 = X_0;
a_x_1 = v_x_0;
a_x_2 = acc_x_0/2;

% Constant values
A_x = X_1-a_x_0-a_x_1*delta_time-a_x_2*delta_time^2;
B_x = v_x_1-a_x_1-2*a_x_2*delta_time;
C_x = acc_x_1-2*a_x_2;

% Definition of coefficients a_3 to a_5
a_x_5 = (3/delta_time^3)*(C_x/6+2*A_x/(delta_time^2)-B_x/delta_time);
a_x_4 = (B_x-3*A_x/delta_time-2*a_x_5*delta_time^4)/(delta_time^3);
a_x_3 = (A_x-a_x_4*delta_time^4-a_x_5*delta_time^5)/(delta_time^3);

x = a_x_0+a_x_1*t+a_x_2*t.^2+a_x_3*t.^3+a_x_4*t.^4+a_x_5*t.^5;

% y direction
% Definition of coefficients a_0 to a_2
a_y_0 = Y_0;
a_y_1 = v_y_0;
a_y_2 = acc_y_0/2;

% Constant values
A_y = Y_1-a_y_0-a_y_1*delta_time-a_y_2*delta_time^2;
B_y = v_y_1-a_y_1-2*a_y_2*delta_time;
C_y = acc_y_1-2*a_y_2;

% Definition of coefficients a_3 to a_5
a_y_5 = (3/delta_time^3)*(C_y/6+2*A_y/(delta_time^2)-B_y/delta_time);
a_y_4 = (B_y-3*A_y/delta_time-2*a_y_5*delta_time^4)/(delta_time^3);
a_y_3 = (A_y-a_y_4*delta_time^4-a_y_5*delta_time^5)/(delta_time^3);

y = a_y_0+a_y_1*t+a_y_2*t.^2+a_y_3*t.^3+a_y_4*t.^4+a_y_5*t.^5;

% z direction
% Definition of coefficients a_0 to a_2
a_z_0 = Z_0;
a_z_1 = v_z_0;
a_z_2 = acc_z_0/2;

% Constant values
A_z = Z_1-a_z_0-a_z_1*delta_time-a_z_2*delta_time^2;
```

```
B_z = v_z_1-a_z_1-2*a_z_2*delta_time;
C_z = acc_z_1-2*a_z_2;

% Definition of coefficients a_3 to a_5
a_z_5 = (3/delta_time^3)*(C_z/6+2*A_z/(delta_time^2)-B_z/delta_time);
a_z_4 = (B_z-3*A_z/delta_time-2*a_z_5*delta_time^4)/(delta_time^3);
a_z_3 = (A_z-a_z_4*delta_time^4-a_z_5*delta_time^5)/(delta_time^3);

z = a_z_0+a_z_1*t+a_z_2*t.^2+a_z_3*t.^3+a_z_4*t.^4+a_z_5*t.^5;

% First derivative
x_dot_1 = a_x_1+2*a_x_2*t+3*a_x_3*t.^2+4*a_x_4*t.^3+5*a_x_5*t.^4;
y_dot_1 = a_y_1+2*a_y_2*t+3*a_y_3*t.^2+4*a_y_4*t.^3+5*a_y_5*t.^4;
z_dot_1 = a_z_1+2*a_z_2*t+3*a_z_3*t.^2+4*a_z_4*t.^3+5*a_z_5*t.^4;

% Second derivative
x_dotdot_1 = 2*a_x_2+6*a_x_3*t+12*a_x_4*t.^2+20*a_x_5*t.^3;
y_dotdot_1 = 2*a_y_2+6*a_y_3*t+12*a_y_4*t.^2+20*a_y_5*t.^3;
z_dotdot_1 = 2*a_z_2+6*a_z_3*t+12*a_z_4*t.^2+20*a_z_5*t.^3;

% Rescaling the time to have constant speed along the path
mod_v = sqrt(x_dot_1.^2+y_dot_1.^2+z_dot_1.^2);

lambda_v = (v_k./mod_v);

v_vect = [x_dot_1;y_dot_1;z_dot_1]';
acc_vect = [x_dotdot_1;y_dotdot_1;z_dotdot_1];

lambda_acc = zeros(1,1000);
for i = 1:1000
    lambda_acc_num = -(v_k^2)*(v_vect(i,:)*acc_vect(:,i));
    lambda_acc(i) = lambda_acc_num/(mod_v(i)^4);
end

x_dot = lambda_v.*x_dot_1;
y_dot = lambda_v.*y_dot_1;
z_dot = lambda_v.*z_dot_1;

x_dotdot = x_dot_1.*lambda_acc+x_dotdot_1.*lambda_v.^2;
y_dotdot = y_dot_1.*lambda_acc+y_dotdot_1.*lambda_v.^2;
z_dotdot = z_dot_1.*lambda_acc+z_dotdot_1.*lambda_v.^2;

% New time
t_prime = zeros(1,1000);
for i = 2:1000
    t_prime(i) = t_prime(i-1)+(t(i)-t(i-1))/lambda_v(i-1);
end

% % COMPUTATION OF DELTA_TIME WITH CIRCULAR ARC
%
% % m1 and m2
% if (chi_0>=0 && chi_0<=pi)
%     m0 = tan(pi/2-chi_0);
% else
```

```
%     m0 = tan(3/2*pi-chi_0);
% end
%
% if (chi_1>=0 && chi_1<=pi)
%     m1 = tan(pi/2-chi_1);
% else
%     m1 = tan(3/2*pi-chi_1);
% end
%
% m0_per = -1/m0;
% m1_per = -1/m1;
%
% % q_0 and q_1
% q_0 = X_0-m0_per*Y_0;
% q_1 = X_1-m1_per*Y_1;
%
% % Center of the circle
% y_center = (q_0-q_1)/(m1_per-m0_per);
% x_center = m0_per*y_center+q_0;
% r_c = (sqrt((X_0-x_center)^2+(Y_0-y_center)^2)+...
%   sqrt((X_1-x_center)^2+(Y_1-y_center)^2))/2;
%
% delta_chi = abs(chi_1-chi_0);
%
% alpha_t = abs(pi-delta_chi);
%
% delta_phi = pi-alpha_t;
%
% arc_length = delta_phi*r_c;
% delta_time = arc_length/v_k;
% t = linspace(0,delta_time,1000);
```

Saturation with circular arc

```
function [x1,y1,z1,x_dot1,y_dot1,z_dot1,x_dotdot1,y_dotdot1,...
    z_dotdot1,t1,t_prime1] = saturationchidot(x,y,z,x_dot,y_dot,...
    z_dot,x_dotdot,y_dotdot,z_dotdot,t,t_prime,v_k,chi_dot_max)

x1 = zeros(1,1000);
y1 = zeros(1,1000);
z1 = zeros(1,1000);
x_dot1 = zeros(1,1000);
y_dot1 = zeros(1,1000);
z_dot1 = zeros(1,1000);
x_dotdot1 = zeros(1,1000);
y_dotdot1 = zeros(1,1000);
z_dotdot1 = zeros(1,1000);
t1 = zeros(1,1000);
t_prime1 = zeros(1,1000);

% Speed
```

```
v_k = v_k(3);

% Saturation of chi_dot

% Computation of chi
chi_dot = 1./(1+(y_dot./x_dot).^2).*...
    ((y_dotdot.*x_dot-y_dot.*x_dotdot)./(x_dot.^2));
chi_vecchio = atan2(y_dot,x_dot);
chi = zeros(1,1000);

for i = 1:1000
    if chi_vecchio(i) < 0
        chi(i) = chi_vecchio(i)+2*pi;
    else
        chi(i) = chi_vecchio(i);
    end
end

% Computation of gamma
gamma = atan(-(z_dot./y_dot).*sin(chi));

if max(abs(chi_dot))>chi_dot_max
    ind_chimajor10 = find(abs(chi_dot)>chi_dot_max);
    length_ind = ind_chimajor10(end)-ind_chimajor10(1)+1;

    r_c = sqrt((v_k^2-z_dot(ind_chimajor10(1):...
        ind_chimajor10(end)).^2))/(chi_dot_max);

    chi_3 = chi(ind_chimajor10(1)-1);
    chi_4 = chi(ind_chimajor10(end)+1);

    chi_dot_3 = chi_dot(ind_chimajor10(1)-1);

    gamma_3 = gamma(ind_chimajor10(1)-1);
    gamma_4 = gamma(ind_chimajor10(end)+1);

    if (chi_3 <= pi/2 && chi_4 >= pi)||(chi_4 <= pi/2 && chi_3 >= pi)
        ang_circ = abs((2*pi)-abs(chi_4-chi_3));
    else
        ang_circ = pi-(pi-abs(chi_4-chi_3));
    end

    % Rotation matrix
    M_OK = [cos(chi_3) -sin(chi_3);sin(chi_3) cos(chi_3)];

    % Initial position
    x_3 = x(ind_chimajor10(1)-1);
    y_3 = y(ind_chimajor10(1)-1);
    z_3 = z(ind_chimajor10(1)-1);

    % Circular arc
    if chi_dot_3>0
        phi_c = linspace(0,ang_circ,length_ind);
```

```
        t_circarc = linspace(t_prime(ind_chimajor10(1)),...
        t_prime(ind_chimajor10(1))+ang_circ/chi_dot_max,length_ind);

        phi_c_dot = (phi_c(4)-phi_c(3))/(t_circarc(4)-t_circarc(3));

        x_circarc = r_c.*sin(phi_c);
        y_circarc = r_c.*(1-cos(phi_c));

        x_dot_circarc = r_c.*cos(phi_c)*phi_c_dot;
        y_dot_circarc = r_c.*sin(phi_c)*phi_c_dot;

        x_dotdot_circarc = -r_c.*sin(phi_c)*phi_c_dot^2;
        y_dotdot_circarc = r_c.*cos(phi_c)*phi_c_dot^2;
    else
        phi_c = linspace(0,ang_circ,length_ind);

        t_circarc = linspace(t_prime(ind_chimajor10(1)),...
            t_prime(ind_chimajor10(1))+ang_circ/chi_dot_max,length_ind);

        phi_c_dot = (phi_c(4)-phi_c(3))/(t_circarc(4)-t_circarc(3));

        x_circarc = r_c.*sin(phi_c);
        y_circarc = - r_c.*(1-cos(phi_c));

        x_dot_circarc = r_c.*cos(phi_c)*phi_c_dot;
        y_dot_circarc = - r_c.*sin(phi_c)*phi_c_dot;

        x_dotdot_circarc = - r_c.*sin(phi_c)*phi_c_dot^2;
        y_dotdot_circarc = - r_c.*cos(phi_c)*phi_c_dot^2;
    end

    % Rotation from K-frame to NED-Local-frame
    pos_circarc = M_OK*[x_circarc;y_circarc];

    x_circarc = pos_circarc(1,:);
    y_circarc = pos_circarc(2,:);
    z_circarc = z(ind_chimajor10(1):(ind_chimajor10(end)));

    vel_circarc = M_OK*[x_dot_circarc;y_dot_circarc];

    x_dot_circarc = vel_circarc(1,:);
    y_dot_circarc = vel_circarc(2,:);
    z_dot_circarc = z_dot(ind_chimajor10(1):(ind_chimajor10(end)));

    acc_circarc = M_OK*[x_dotdot_circarc;y_dotdot_circarc];

    x_dotdot_circarc = acc_circarc(1,:);
    y_dotdot_circarc = acc_circarc(2,:);
    z_dotdot_circarc = z_dotdot(ind_chimajor10(1):...
        (ind_chimajor10(end)));

    % Translation from NED-Local-frame to O-frame
    x_circarc = x_3 + x_circarc;
    y_circarc = y_3 + y_circarc;
```

```
x1(1:(ind_chimajor10(1)-1)) = x(1:(ind_chimajor10(1)-1));
x1(ind_chimajor10(1):(ind_chimajor10(end))) = x_circarc;
x1((ind_chimajor10(end)+1):end) = (x_circarc(end)-...
    x(ind_chimajor10(end)+1))+x((ind_chimajor10(end)+1):end);
y1(1:(ind_chimajor10(1)-1)) = y(1:(ind_chimajor10(1)-1));
y1(ind_chimajor10(1):(ind_chimajor10(end))) = y_circarc;
y1((ind_chimajor10(end)+1):end) = (y_circarc(end)-...
    y(ind_chimajor10(end)+1))+y((ind_chimajor10(end)+1):end);
z1(1:(ind_chimajor10(1)-1)) = z(1:(ind_chimajor10(1)-1));
z1(ind_chimajor10(1):(ind_chimajor10(end))) = z_circarc;
z1((ind_chimajor10(end)+1):end) = (z_circarc(end)-...
    z(ind_chimajor10(end)+1))+z((ind_chimajor10(end)+1):end);

x_dot1(1:(ind_chimajor10(1)-1)) = x_dot(1:(ind_chimajor10(1)-1));
x_dot1(ind_chimajor10(1):(ind_chimajor10(end))) = x_dot_circarc;
x_dot1((ind_chimajor10(end)+1):end) = ...
    x_dot((ind_chimajor10(end)+1):end);
y_dot1(1:(ind_chimajor10(1)-1)) = y_dot(1:(ind_chimajor10(1)-1));
y_dot1(ind_chimajor10(1):(ind_chimajor10(end))) = y_dot_circarc;
y_dot1((ind_chimajor10(end)+1):end) = ...
    y_dot((ind_chimajor10(end)+1):end);
z_dot1(1:(ind_chimajor10(1)-1)) = z_dot(1:(ind_chimajor10(1)-1));
z_dot1(ind_chimajor10(1):(ind_chimajor10(end))) = z_dot_circarc;
z_dot1((ind_chimajor10(end)+1):end) = ...
    z_dot((ind_chimajor10(end)+1):end);

x_dotdot1(1:(ind_chimajor10(1)-1)) = ...
    x_dotdot(1:(ind_chimajor10(1)-1));
x_dotdot1(ind_chimajor10(1):(ind_chimajor10(end))) = ...
    x_dotdot_circarc;
x_dotdot1((ind_chimajor10(end)+1):end) = ...
    x_dotdot((ind_chimajor10(end)+1):end);
y_dotdot1(1:(ind_chimajor10(1)-1)) = ...
    y_dotdot(1:(ind_chimajor10(1)-1));
y_dotdot1(ind_chimajor10(1):(ind_chimajor10(end))) = ...
    y_dotdot_circarc;
y_dotdot1((ind_chimajor10(end)+1):end) = ...
    y_dotdot((ind_chimajor10(end)+1):end);
z_dotdot1(1:(ind_chimajor10(1)-1)) = ...
    z_dotdot(1:(ind_chimajor10(1)-1));
z_dotdot1(ind_chimajor10(1):(ind_chimajor10(end))) = ...
    z_dotdot_circarc;
z_dotdot1((ind_chimajor10(end)+1):end) = ...
    z_dotdot((ind_chimajor10(end)+1):end);

t1(1:(ind_chimajor10(1)-1)) = t(1:(ind_chimajor10(1)-1));
t1(ind_chimajor10(1):(ind_chimajor10(end))) = t_circarc;
t1((ind_chimajor10(end)+1):end) = t_circarc(end)...
    -t(ind_chimajor10(end)+1)+t((ind_chimajor10(end)+1):end);

t_prime1(1:(ind_chimajor10(1)-1)) = ...
    t_prime(1:(ind_chimajor10(1)-1));
t_prime1(ind_chimajor10(1):(ind_chimajor10(end))) = t_circarc;
```

```
    t_prime1((ind_chimajor10(end)+1):end) = t_circarc(end)-...
        t_prime(ind_chimajor10(end)+1)+...
        t_prime((ind_chimajor10(end)+1):end);
else
    x1 = x;
    y1 = y;
    z1 = z;
    x_dot1 = x_dot;
    y_dot1 = y_dot;
    z_dot1 = z_dot;
    x_dotdot1 = x_dotdot;
    y_dotdot1 = y_dotdot;
    z_dotdot1 = z_dotdot;
    t1 = t;
    t_prime1 = t_prime;
end
```

Vertical acceleration and deceleration

```
function [x,y,z,x_dot,y_dot,z_dot,x_dotdot,y_dotdot,z_dotdot,t,...
    t_prime] = verticalaccdec(WP_0,WP_1,WP_2,v_k)

% Extrapolation of waypoints data
% Waypoint 0
id_0 = WP_0(1);
X_0 = WP_0(2);
Y_0 = WP_0(3);
Z_0 = WP_0(4);
chi_0 = WP_0(5);
gamma_0 = WP_0(6);

% Waypoint 1
X_1 = WP_1(2);
Y_1 = WP_1(3);
Z_1 = WP_1(4);
chi_1 = WP_1(5);
gamma_1 = WP_1(6);

% Waypoint 2
id_2 = WP_2(1);

% Speed
if (id_2 == 3) || (id_0 == 3)
    if Z_1 - Z_0 < 0
        v_k_0 = v_k(1);
        v_k_1 = v_k(2);
    else
        v_k_0 = -v_k(2);
        v_k_1 = -v_k(1);
    end
    v_mean = abs(v_k_0+v_k_1)/2;
```

```
else
    v_k_0 = v_k(1);
    v_k_1 = v_k(1);
    v_mean = 1;
end

% Max speed and acceleration
v_max = 2;
acc_max = 2;

% COMPUTATION OF DELTA_TIME WITH A STRAIGHT LINE
% Time
distance = sqrt((X_1-X_0)^2+(Y_1-Y_0)^2+(Z_1-Z_0)^2);
delta_time = distance/v_mean;
t = linspace(0,delta_time,1000);

% Speed
% v_O_0
v_O_0 = [0,0,-v_k_0]';

% v_O_1
v_O_1 = [0,0,-v_k_1]';

% x direction
v_x_0 = v_O_0(1);
v_x_1 = v_O_1(1);

% y direction
v_y_0 = v_O_0(2);
v_y_1 = v_O_1(2);

% z direction
v_z_0 = v_O_0(3);
v_z_1 = v_O_1(3);

% Acceleration
% x direction
acc_x_0 = 0;
acc_x_1 = 0;

% y direction
acc_y_0 = 0;
acc_y_1 = 0;

% z direction
acc_z_0 = 0;
acc_z_1 = 0;

% Construction of fifth degree polynomial

% x direction
% Definition of coefficients a_0 to a_2
a_x_0 = X_0;
a_x_1 = v_x_0;
```

```
a_x_2 = acc_x_0/2;

% Constant values
A_x = X_1-a_x_0-a_x_1*delta_time-a_x_2*delta_time^2;
B_x = v_x_1-a_x_1-2*a_x_2*delta_time;
C_x = acc_x_1-2*a_x_2;

% Definition of coefficients a_3 to a_5
a_x_5 = (3/delta_time^3)*(C_x/6+2*A_x/(delta_time^2)-B_x/delta_time);
a_x_4 = (B_x-3*A_x/delta_time-2*a_x_5*delta_time^4)/(delta_time^3);
a_x_3 = (A_x-a_x_4*delta_time^4-a_x_5*delta_time^5)/(delta_time^3);

x = a_x_0+a_x_1*t+a_x_2*t.^2+a_x_3*t.^3+a_x_4*t.^4+a_x_5*t.^5;

% y direction
% Definition of coefficients a_0 to a_2
a_y_0 = Y_0;
a_y_1 = v_y_0;
a_y_2 = acc_y_0/2;

% Constant values
A_y = Y_1-a_y_0-a_y_1*delta_time-a_y_2*delta_time^2;
B_y = v_y_1-a_y_1-2*a_y_2*delta_time;
C_y = acc_y_1-2*a_y_2;

% Definition of coefficients a_3 to a_5
a_y_5 = (3/delta_time^3)*(C_y/6+2*A_y/(delta_time^2)-B_y/delta_time);
a_y_4 = (B_y-3*A_y/delta_time-2*a_y_5*delta_time^4)/(delta_time^3);
a_y_3 = (A_y-a_y_4*delta_time^4-a_y_5*delta_time^5)/(delta_time^3);

y = a_y_0+a_y_1*t+a_y_2*t.^2+a_y_3*t.^3+a_y_4*t.^4+a_y_5*t.^5;

% z direction
% Definition of coefficients a_0 to a_2
a_z_0 = Z_0;
a_z_1 = v_z_0;
a_z_2 = acc_z_0/2;

% Constant values
A_z = Z_1-a_z_0-a_z_1*delta_time-a_z_2*delta_time^2;
B_z = v_z_1-a_z_1-2*a_z_2*delta_time;
C_z = acc_z_1-2*a_z_2;

% Definition of coefficients a_3 to a_5
a_z_5 = (3/delta_time^3)*(C_z/6+2*A_z/(delta_time^2)-B_z/delta_time);
a_z_4 = (B_z-3*A_z/delta_time-2*a_z_5*delta_time^4)/(delta_time^3);
a_z_3 = (A_z-a_z_4*delta_time^4-a_z_5*delta_time^5)/(delta_time^3);

z = a_z_0+a_z_1*t+a_z_2*t.^2+a_z_3*t.^3+a_z_4*t.^4+a_z_5*t.^5;

% First derivative
x_dot = a_x_1+2*a_x_2*t+3*a_x_3*t.^2+4*a_x_4*t.^3+5*a_x_5*t.^4;
y_dot = a_y_1+2*a_y_2*t+3*a_y_3*t.^2+4*a_y_4*t.^3+5*a_y_5*t.^4;
z_dot = a_z_1+2*a_z_2*t+3*a_z_3*t.^2+4*a_z_4*t.^3+5*a_z_5*t.^4;
```

```
% Second derivative
x_dotdot = 2*a_x_2+6*a_x_3*t+12*a_x_4*t.^2+20*a_x_5*t.^3;
y_dotdot = 2*a_y_2+6*a_y_3*t+12*a_y_4*t.^2+20*a_y_5*t.^3;
z_dotdot = 2*a_z_2+6*a_z_3*t+12*a_z_4*t.^2+20*a_z_5*t.^3;

t_prime = t;
```

Vertical curve

```
function [x,y,z,x_dot,y_dot,z_dot,x_dotdot,y_dotdot,z_dotdot,t,...
    t_prime] = verticalcurve(WP_0,WP_1,v_k)

% Extrapolation of waypoints data
% Waypoint 0
X_0 = WP_0(2);
Y_0 = WP_0(3);
Z_0 = WP_0(4);
chi_0 = WP_0(5);
gamma_0 = WP_0(6);

% Waypoint 1
X_1 = WP_1(2);
Y_1 = WP_1(3);
Z_1 = WP_1(4);
chi_1 = WP_1(5);
gamma_1 = WP_1(6);

% Speed
v_k = v_k(2);

% COMPUTATION OF DELTA_TIME WITH A STRAIGHT LINE
% Time
distance = sqrt((X_1-X_0)^2+(Y_1-Y_0)^2+(Z_1-Z_0)^2);
delta_time = distance/v_k;
t = linspace(0,delta_time,1000);

% Speed
if Z_1 - Z_0 < 0
    % v_O_0
    v_O_0 = [0,0,-v_k]';

    % v_O_1
    v = [v_k,0,0]';
    M_OK_1 = [cos(chi_1)*cos(gamma_1) -sin(chi_1) ...
        cos(chi_1)*sin(gamma_1);sin(chi_1)*cos(gamma_1) cos(chi_1) ...
        sin(chi_1)*sin(gamma_1);-sin(gamma_1) 0 cos(gamma_1)];
    v_O_1 = M_OK_1*v;
else
    % v_O_0
    v = [v_k,0,0]';
```

```
    M_OK_0 = [cos(chi_0)*cos(gamma_0) -sin(chi_0) ...
        cos(chi_0)*sin(gamma_0);sin(chi_0)*cos(gamma_0) cos(chi_0) ...
        sin(chi_0)*sin(gamma_0);-sin(gamma_0) 0 cos(gamma_0)];
    v_O_0 = M_OK_0*v;

    % v_O_1
    v_O_1 = [0,0,v_k]';
end

% x direction
v_x_0 = v_O_0(1);
v_x_1 = v_O_1(1);

% y direction
v_y_0 = v_O_0(2);
v_y_1 = v_O_1(2);

% z direction
v_z_0 = v_O_0(3);
v_z_1 = v_O_1(3);

% Acceleration
% x direction
acc_x_0 = 0;
acc_x_1 = 0;

% y direction
acc_y_0 = 0;
acc_y_1 = 0;

% z direction
acc_z_0 = 0;
acc_z_1 = 0;

% Construction of fifth degree polynomial

% x direction
% Definition of coefficients a_0 to a_2
a_x_0 = X_0;
a_x_1 = v_x_0;
a_x_2 = acc_x_0/2;

% Constant values
A_x = X_1-a_x_0-a_x_1*delta_time-a_x_2*delta_time^2;
B_x = v_x_1-a_x_1-2*a_x_2*delta_time;
C_x = acc_x_1-2*a_x_2;

% Definition of coefficients a_3 to a_5
a_x_5 = (3/delta_time^3)*(C_x/6+2*A_x/(delta_time^2)-B_x/delta_time);
a_x_4 = (B_x-3*A_x/delta_time-2*a_x_5*delta_time^4)/(delta_time^3);
a_x_3 = (A_x-a_x_4*delta_time^4-a_x_5*delta_time^5)/(delta_time^3);

x = a_x_0+a_x_1*t+a_x_2*t.^2+a_x_3*t.^3+a_x_4*t.^4+a_x_5*t.^5;
```

```
% y direction
% Definition of coefficients a_0 to a_2
a_y_0 = Y_0;
a_y_1 = v_y_0;
a_y_2 = acc_y_0/2;

% Constant values
A_y = Y_1-a_y_0-a_y_1*delta_time-a_y_2*delta_time^2;
B_y = v_y_1-a_y_1-2*a_y_2*delta_time;
C_y = acc_y_1-2*a_y_2;

% Definition of coefficients a_3 to a_5
a_y_5 = (3/delta_time^3)*(C_y/6+2*A_y/(delta_time^2)-B_y/delta_time);
a_y_4 = (B_y-3*A_y/delta_time-2*a_y_5*delta_time^4)/(delta_time^3);
a_y_3 = (A_y-a_y_4*delta_time^4-a_y_5*delta_time^5)/(delta_time^3);

y = a_y_0+a_y_1*t+a_y_2*t.^2+a_y_3*t.^3+a_y_4*t.^4+a_y_5*t.^5;

% z direction
% Definition of coefficients a_0 to a_2
a_z_0 = Z_0;
a_z_1 = v_z_0;
a_z_2 = acc_z_0/2;

% Constant values
A_z = Z_1-a_z_0-a_z_1*delta_time-a_z_2*delta_time^2;
B_z = v_z_1-a_z_1-2*a_z_2*delta_time;
C_z = acc_z_1-2*a_z_2;

% Definition of coefficients a_3 to a_5
a_z_5 = (3/delta_time^3)*(C_z/6+2*A_z/(delta_time^2)-B_z/delta_time);
a_z_4 = (B_z-3*A_z/delta_time-2*a_z_5*delta_time^4)/(delta_time^3);
a_z_3 = (A_z-a_z_4*delta_time^4-a_z_5*delta_time^5)/(delta_time^3);

z = a_z_0+a_z_1*t+a_z_2*t.^2+a_z_3*t.^3+a_z_4*t.^4+a_z_5*t.^5;

% First derivative
x_dot_1 = a_x_1+2*a_x_2*t+3*a_x_3*t.^2+4*a_x_4*t.^3+5*a_x_5*t.^4;
y_dot_1 = a_y_1+2*a_y_2*t+3*a_y_3*t.^2+4*a_y_4*t.^3+5*a_y_5*t.^4;
z_dot_1 = a_z_1+2*a_z_2*t+3*a_z_3*t.^2+4*a_z_4*t.^3+5*a_z_5*t.^4;

% Second derivative
x_dotdot_1 = 2*a_x_2+6*a_x_3*t+12*a_x_4*t.^2+20*a_x_5*t.^3;
y_dotdot_1 = 2*a_y_2+6*a_y_3*t+12*a_y_4*t.^2+20*a_y_5*t.^3;
z_dotdot_1 = 2*a_z_2+6*a_z_3*t+12*a_z_4*t.^2+20*a_z_5*t.^3;

% Rescaling the time to have constant velocity along the path
mod_v = sqrt(x_dot_1.^2+y_dot_1.^2+z_dot_1.^2);

lambda_v = (v_k./mod_v);

v_vect = [x_dot_1;y_dot_1;z_dot_1]';
acc_vect = [x_dotdot_1;y_dotdot_1;z_dotdot_1];
```

```
lambda_acc = zeros(1,1000);
for i = 1:1000
    lambda_acc_num = -(v_k^2)*(v_vect(i,:)*acc_vect(:,i));
    lambda_acc(i) = lambda_acc_num/(mod_v(i)^4);
end

x_dot = lambda_v.*x_dot_1;
y_dot = lambda_v.*y_dot_1;
z_dot = lambda_v.*z_dot_1;

x_dotdot = x_dot_1.*lambda_acc+x_dotdot_1.*lambda_v.^2;
y_dotdot = y_dot_1.*lambda_acc+y_dotdot_1.*lambda_v.^2;
z_dotdot = z_dot_1.*lambda_acc+z_dotdot_1.*lambda_v.^2;

% New time
t_prime = zeros(1,1000);
for i = 2:1000
    t_prime(i) = t_prime(i-1)+(t(i)-t(i-1))/lambda_v(i-1);
end
```

Horizontal acceleration

```
function [x,y,z,x_dot,y_dot,z_dot,x_dotdot,y_dotdot,z_dotdot,t,...
    t_prime] = horizontalacceleration(WP_0,WP_1,v_k,acc_limits)

% Extrapolation of waypoints data
% Waypoint 0
id_0 = WP_0(1);
X_0 = WP_0(2);
Y_0 = WP_0(3);
Z_0 = WP_0(4);
chi_0 = WP_0(5);
gamma_0 = WP_0(6);

% Waypoint 1
id_1 = WP_1(1);
X_1 = WP_1(2);
Y_1 = WP_1(3);
Z_1 = WP_1(4);
chi_1 = WP_1(5);
gamma_1 = WP_1(6);

% Speed
if id_0 == 3
    v_k_0 = v_k(2);
else
    v_k_0 = v_k(1);
end
v_k_1 = v_k(3);
v_mean = (v_k_0+v_k_1)/2;
```

```
% Max acceleration and jerk
acc_max = acc_limits(1);
j_max = acc_limits(2);

% Distance
distance = sqrt((X_1-X_0)^2+(Y_1-Y_0)^2+(Z_1-Z_0)^2);

% Position initialization
q_i = 0;
q_f = distance;

% Time
Tj_1 = acc_max/j_max;
Ta = Tj_1 + (v_k_1-v_k_0)/acc_max;
Tv = (q_f-q_i)/v_k_1 - Ta/2*(1 + v_k_0/v_k_1);

t = linspace(0,Ta + Tv,1000);

ind_1 = find(t < Tj_1);
ind_2 = find(t < Ta-Tj_1);
ind_2 = ind_2(ind_1(end)+1:end);
ind_3 = find(t < Ta);
ind_3 = ind_3(ind_2(end)+1:end);
t_1 = t(ind_1);
t_2 = t(ind_2);
t_3 = t(ind_3);
t_4 = t(ind_3(end)+1:end);

q_1 = q_i + v_k_0*t_1 + j_max/6*t_1.^3;
q_1_dot = v_k_0 + j_max/2*t_1.^2;
q_1_dotdot = j_max*t_1;

q_2 = q_i + v_k_0*t_2 + (acc_max/6)*(3*t_2.^2 - 3*Tj_1*t_2 + Tj_1^2);
q_2_dot = v_k_0 + acc_max*(t_2 - Tj_1/2);
q_2_dotdot = acc_max*ones(1,length(t_2));

q_3 = q_i + (v_k_1+v_k_0)*Ta/2 - v_k_1*(Ta-t_3) + j_max/6*(Ta-t_3).^3;
q_3_dot = v_k_1 - j_max/2*(Ta-t_3).^2;
q_3_dotdot = j_max*(Ta - t_3);

q_4 = q_i + (v_k_1+v_k_0)*Ta/2 + v_k_1*(t_4-Ta);
q_4_dot = v_k_1*ones(1,length(t_4));
q_4_dotdot = zeros(1,length(t_4));

q = [q_1,q_2,q_3,q_4];
q_dot = [q_1_dot,q_2_dot,q_3_dot,q_4_dot];
q_dotdot = [q_1_dotdot,q_2_dotdot,q_3_dotdot,q_4_dotdot];

% Rotation matrix
M_OK = [cos(chi_1)*cos(gamma_1) -sin(chi_1) ...
    cos(chi_1)*sin(gamma_1);sin(chi_1)*cos(gamma_1) cos(chi_1) ...
    sin(chi_1)*sin(gamma_1);-sin(gamma_1) 0 cos(gamma_1)];

% Position
```

```
x = zeros(1,1000);
y = zeros(1,1000);
z = zeros(1,1000);

% First derivative
x_dot = zeros(1,1000);
y_dot = zeros(1,1000);
z_dot = zeros(1,1000);

% Second derivative
x_dotdot = zeros(1,1000);
y_dotdot = zeros(1,1000);
z_dotdot = zeros(1,1000);

for i=1:1000
    pos = M_OK*[q(i),0,0]';

    x(i) = pos(1);
    y(i) = pos(2);
    z(i) = pos(3);

    vel = M_OK*[q_dot(i),0,0]';

    x_dot(i) = vel(1);
    y_dot(i) = vel(2);
    z_dot(i) = vel(3);

    acc = M_OK*[q_dotdot(i),0,0]';

    x_dotdot(i) = acc(1);
    y_dotdot(i) = acc(2);
    z_dotdot(i) = acc(3);
end

x = X_0 + x;
y = Y_0 + y;
z = Z_0 + z;

t_prime = t;
```

Horizontal deceleration

```
function [x,y,z,x_dot,y_dot,z_dot,x_dotdot,y_dotdot,z_dotdot,t,...
    t_prime] = horizontaldeceleration(WP_0,WP_1,WP_2,v_k,acc_limits)

% Extrapolation of waypoints data
% Waypoint 0
id_0 = WP_0(1);
X_0 = WP_0(2);
Y_0 = WP_0(3);
Z_0 = WP_0(4);
```

```
chi_0 = WP_0(5);
gamma_0 = WP_0(6);

% Waypoint 1
id_1 = WP_1(1);
X_1 = WP_1(2);
Y_1 = WP_1(3);
Z_1 = WP_1(4);
chi_1 = WP_1(5);
gamma_1 = WP_1(6);

% Waypoint 2
id_2 = WP_2(1);

% Speed
v_k_0 = v_k(3);
if id_2 == 3
    v_k_1 = v_k(2);
else
    v_k_1 = v_k(1);
end
v_mean = (v_k_0+v_k_1)/2;

% Max acceleration and jerk
acc_max = acc_limits(1);
acc_min = -acc_max;
j_max = acc_limits(2);

% Distance
distance = sqrt((X_1-X_0)^2+(Y_1-Y_0)^2+(Z_1-Z_0)^2);

% Position initialization
q_i = 0;
q_f = distance;

% Time
Tj_2 = acc_max/j_max;
Td = Tj_2 + (v_k_0-v_k_1)/acc_max;
Tv = (q_f-q_i)/v_k_0 - Td/2*(1 + v_k_1/v_k_0);

t = linspace(0,Tv + Td,1000);

ind_1 = find(t < Tv);
ind_2 = find(t < Tv+Tj_2);
ind_2 = ind_2(ind_1(end)+1:end);
ind_3 = find(t < Tv+Td-Tj_2);
ind_3 = ind_3(ind_2(end)+1:end);
t_1 = t(ind_1);
t_2 = t(ind_2);
t_3 = t(ind_3);
t_4 = t(ind_3(end)+1:end);

q_1 = q_i + v_k_0*t_1;
q_1_dot = v_k_0*ones(1,length(t_1));
```

```
q_1_dotdot = zeros(1,length(t_1));

q_2 = q_f - (v_k_0+v_k_1)Td/2 + v_k_0(t_2-Tv) - j_max/6*(t_2-Tv).^3;
q_2_dot = v_k_0 - j_max/2*(t_2-Tv).^2;
q_2_dotdot = -j_max*(t_2-Tv);

q_3 = q_f - (v_k_0+v_k_1)Td/2 + v_k_0(t_3-Tv) + ...
    acc_min/6*(3*(t_3-Tv).^2 - 3*Tj_2*(t_3-Tv) + Tj_2^2);
q_3_dot = v_k_0 + acc_min*(t_3-Tv-Tj_2/2);
q_3_dotdot = acc_min*ones(1,length(t_3));

q_4 = q_f - v_k_1*(Tv+Td-t_4) - j_max/6*(Tv+Td-t_4).^3;
q_4_dot = v_k_1 + j_max/2*(Tv+Td-t_4).^2;
q_4_dotdot = -j_max*(Tv+Td-t_4);

q = [q_1,q_2,q_3,q_4];
q_dot = [q_1_dot,q_2_dot,q_3_dot,q_4_dot];
q_dotdot = [q_1_dotdot,q_2_dotdot,q_3_dotdot,q_4_dotdot];

% Rotation matrix
M_OK = [cos(chi_1)*cos(gamma_1) -sin(chi_1) ...
    cos(chi_1)*sin(gamma_1);sin(chi_1)*cos(gamma_1) cos(chi_1) ...
    sin(chi_1)*sin(gamma_1);-sin(gamma_1) 0 cos(gamma_1)];

% Position
x = zeros(1,1000);
y = zeros(1,1000);
z = zeros(1,1000);

% First derivative
x_dot = zeros(1,1000);
y_dot = zeros(1,1000);
z_dot = zeros(1,1000);

% Second derivative
x_dotdot = zeros(1,1000);
y_dotdot = zeros(1,1000);
z_dotdot = zeros(1,1000);

for i=1:1000
    pos = M_OK*[q(i),0,0]';

    x(i) = pos(1);
    y(i) = pos(2);
    z(i) = pos(3);

    vel = M_OK*[q_dot(i),0,0]';

    x_dot(i) = vel(1);
    y_dot(i) = vel(2);
    z_dot(i) = vel(3);

    acc = M_OK*[q_dotdot(i),0,0]';
```

```
    x_dotdot(i) = acc(1);
    y_dotdot(i) = acc(2);
    z_dotdot(i) = acc(3);
end

x = X_0 + x;
y = Y_0 + y;
z = Z_0 + z;

t_prime = t;
```

Hovering

```
function [x,y,z,x_dot,y_dot,z_dot,x_dotdot,y_dotdot,z_dotdot,t,...
    t_prime] = hovering(WP_1,t_hover)

% Extrapolation of waypoints data
% Waypoint 1
id_1 = WP_1(1);
X_1 = WP_1(2);
Y_1 = WP_1(3);
Z_1 = WP_1(4);
chi_1 = WP_1(5);
gamma_1 = WP_1(6);

% Position
x = X_1*ones(1,1000);
y = Y_1*ones(1,1000);
z = Z_1*ones(1,1000);

% First derivative
x_dot = zeros(1,1000);
y_dot = zeros(1,1000);
z_dot = zeros(1,1000);

% Second derivative
x_dotdot = zeros(1,1000);
y_dotdot = zeros(1,1000);
z_dotdot = zeros(1,1000);

t = linspace(0,t_hover,1000);
t_prime = t;
```

A.3 Other scripts

Shift in case of saturation

```
function [x_shifted,y_shifted,z_shifted] = shift(x,y,z,s)

x_shifted = x;
y_shifted = y;
z_shifted = z;

for i = 1:s-2
    x_shifted((i*1000+1):((i+1)*1000)) = x((i*1000+1):...
        ((i+1)*1000))+x_shifted(i*1000)-x_shifted(i*1000+1);
    y_shifted((i*1000+1):((i+1)*1000)) = y((i*1000+1):...
        ((i+1)*1000))+y_shifted(i*1000)-y_shifted(i*1000+1);
    z_shifted((i*1000+1):((i+1)*1000)) = z((i*1000+1):...
        ((i+1)*1000))+z_shifted(i*1000)-z_shifted(i*1000+1);
end
```

Main code

```
%% Mass and gravity acceleration
g = 9.81;
mass = 5;

%% Parameters used in Simulink
% Coordinates of the FSD Institute
wp_FSD = [48.266185;11.668320;478];

% Vertical fly-by distance
vert_flyby_dist = 5;

% Duration of hovering
t_hover = 10;

% Characteristic speeds
v_hover = 0;
v_verticalflyby = 2;
v_wingborne = 25;
v_k = [v_hover,v_verticalflyby,v_wingborne];

% Prescribed limits
acc_max = 2;
j_max = 2;
acc_limits = [acc_max,j_max];

% Desired chi_dot
chi_dot_des = 7.63;

% Maximum chi_dot
chi_dot_max = 10/180*pi;

%% Waypoint List
% Altitude
```

```
h = 518;

% Definition of waypoints
id_0 = 9;
mu_0 = 48.267539;
lambda_0 = 11.668193;
h_0 = h-40;
wp_0 = [id_0,mu_0,lambda_0,h_0]';

id_1 = 4;
mu_1 = 48.267539;
lambda_1 = 11.668193;
h_1 = h;
wp_1 = [id_1,mu_1,lambda_1,h_1]';

id_2 = 5;
mu_2 = 48.268225;
lambda_2 = 11.672281;
h_2 = h;
wp_2 = [id_2,mu_2,lambda_2,h_2]';

id_3 = 3;
mu_3 = 48.273412;
lambda_3 = 11.673054;
h_3 = h;
wp_3 = [id_3,mu_3,lambda_3,h_3]';

id_4 = 1;
mu_4 = 48.274658;
lambda_4 = 11.668848;
h_4 = h;
wp_4 = [id_4,mu_4,lambda_4,h_4]';

id_5 = 1;
mu_5 = 48.273555;
lambda_5 = 11.664697;
h_5 = h;
wp_5 = [id_5,mu_5,lambda_5,h_5]';

id_6 = 6;
mu_6 = 48.270460;
lambda_6 = 11.663331;
h_6 = h;
wp_6 = [id_6,mu_6,lambda_6,h_6]';

id_7 = 7;
mu_7 = 48.270460;
lambda_7 = 11.663331;
h_7 = h;
wp_7 = [id_7,mu_7,lambda_7,h_7]';

id_8 = 8;
mu_8 = 48.270460;
lambda_8 = 11.663331;
```

```
h_8 = h+20;
wp_8 = [id_8,mu_8,lambda_8,h_8]';

wp_list = [wp_0,wp_1,wp_2,wp_3,wp_4,wp_5,wp_6,wp_7,wp_8];

% Dimension of waypoint list
dim = size(wp_list);
m = dim(1,1);
n = dim(1,2);

% Computation of the number of added waypoints
d = 0;
for i=1:n
    if wp_list(1,i)==1 || wp_list(1,i)==2 || wp_list(1,i)==4
        d = d+1;
    end
end

% Computation of the new total number of way-points
s = n+d;

% Definition of index vector used in ExtrapolationNewWP
j = 0;
ind_vector = [];
for i = 1:n
    if wp_list(1,i) == 1 || wp_list(1,i) == 2 || wp_list(1,i) == 4
        j = j+1;
        ind_vector = [ind_vector,j];
        j = j+1;
        ind_vector = [ind_vector,j];
    else
        j = j+1;
        ind_vector = [ind_vector,j];
        j = j+1;
    end
end

dim_vector = (s-1)*1000;

% Definition of final_selector used in the EliminationOfDuplicates
final_selector = [1:1:1000];
for i = 1:s-2
    adding_vector=[final_selector(end)+2:1:final_selector(end)+1000];
    final_selector=[final_selector,adding_vector];
end

%% Launch SIMULATOR
sim trajectory_generator

%% Plot
plot_traj
```

Acronyms

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
ARINC	Aeronautical Radio Incorporated
CAD/CAM	Computer-Aided Design/Computer-Aided Manufacturing
CG	Center of Gravity
CoM	Center of Mass
DME	Distance Measurement Equipment
DoF	Degree of Freedom
ECEF	Earth-Centered Earth-Fixed
eVTOL	electric Vertical Take-Off and Landing
FAA	Federal Aviation Administration
FFF	Fast-Forward Flight
FSD	Flight System Dynamics
HEV	Hybrid Electric Vehicle
HF	Hover Flight
ICAO	International Civil Aviation Organization
IF	Initial Fix
INDI	Incremental Nonlinear Dynamic Inversion
MAV	Micro Air Vehicle
NAVAIDs	Navigational Aids

NDB	Non Directional Beacon
NED	North East Down
NURBS	Non-Uniform Rational B-Spline
PBN	Performance-based Navigation
PhD	Doctor of Philosophy
PID	Proportional Integral Derivative
PVTOL	Planar Vertical Take-Off and Landing
RF	Radius to a Fix
RNAV	Area Navigation
RTCA	Radio Technical Commission for Aeronautics
SISO	Single Input Single Output
SFF	Slow-Forward Flight
SoC	State of Charge
TF	Track to a Fix
TUM	Technical University of Munich
UAV	Unmanned Aerial Vehicle
VOR	Very High Frequency Omnidirectional Radio Range
V/STOL	Vertical/Short Take-Off and Landing
VTOL	Vertical Take-Off and Landing
WGS84	World Geodetic System 1984

Bibliography

[1] Jean Levine, Michel Fliess, Philippe Martin, Pierre Rouchon. *Flatness and Defect of Non-linear Systems: Introductory Theory and Examples*. International Journal of Control, vol. 61, no. 6, 1327-1361, 1995.

[2] Jean Levine. *Analysis and Control of Nonlinear Systems: A Flatness-based Approach*. Springer-Verlag Berlin, 2009.

[3] Jean Levine. *Are there New Industrial Perspectives in the Control of Mechanical Systems?* Advances in Control (Highlights of ECC'99), Springer-Verlag London, 197-226, 1999.

[4] Tudor-Bogdan Airimitoaie, Gemma Prieto Aguilar, Loic Lavigne, Christophe Farges, Franck Cazaurang. *Convertible Aircraft Dynamic Modelling and Flatness Analysis*. 9th Vienna International Conference on Mathematical Modelling, vol. 51, issue 2, 25-30, 2018.

[5] Pedro Castillo, Rogelio Lozano, Alejandro E. Dzul. *Modelling and Control of Mini-flying Machines*. Springer-Verlag London, 2005.

[6] Giuseppe Notarstefano, John Hauser. *Modeling and Dynamic Exploration of a Tilt-Rotor VTOL Aircraft*. 8th IFAC Symposium on Nonlinear Control Systems, vol. 43, issue 14, 119-124, 2010.

[7] Gerardo Ramon Flores, Juan Escareño, Rogelio Lozano, Sergio Salazar. *Quad-Tilting Rotor Convertible MAV: Modeling and Real-Time Hover Flight Control*. Journal of Intelligent and Robotic Systems, vol. 65, 457–471, 2012.

[8] Juan Escareño, Sergio Salazar, Rogelio Lozano. *Modelling and Control of a Convertible VTOL Aircraft*. 45th IEEE Conference on Decision and Control, San Diego, USA, 2006.

[9] Sabir Abdelhay, Alia Zakriti. *Modeling of a Quadcopter Trajectory Tracking System Using PID Controller*. 12th International Conference Interdisciplinarity in Engineering, Procedia Manufacturing, vol. 32, 564-571, 2019.

[10] Simone Formentin, Marco Lovera. *Flatness-based Control of a Quadrotor Helicopter via Feedforward Linearization*. 50th IEEE Conference on Decision and Control and European Control Conference, Orlando, USA, 2011.

[11] Jean Levine, Dinh Viet Nguyen. *Flat Output Characterization for Linear Systems Using Polynomial Matrices*. Systems and Control Letters, vol. 48, issue 1, 69-75, 2003.

[12] Michiel J. Van Nieuwstadt, Richard M. Murray. *Real Time Trajectory Generation for Differentially Flat Systems*. International Journal of Robust and Nonlinear Control, vol. 8, issue 11, 995–1020, 1998.

[13] Daniel Mellinger, Vijay Kumar. *Minimum Snap Trajectory Generation and Control for Quadrotors*. 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 2011.

[14] Barış Başpinar, Emre Koyuncu. *Differential Flatness-based Optimal Air Combat Maneuver Strategy Generation*. AIAA Scitech Forum, San Diego, USA, 2019.

[15] Amit Ailon. *Closed-form Feedback Controllers for Set-point and Trajectory Tracking for the Nonlinear Model of Quadrotor Helicopters*. 7th IFAC Symposium on Robust Control Design, Aalborg, Denmark, 2012.

[16] Marius Beul, Sven Behnke. *Analytical Time-optimal Trajectory Generation and Control for Multirotors*. International Conference on Unmanned Aircraft Systems (ICUAS), 2016.

[17] Marius Beul, Sven Behnke. *Fast Full State Trajectory Generation for Multirotors*. International Conference on Unmanned Aircraft Systems (ICUAS), Miami, USA, 2017.

[18] Michael Shomin, Ralph Hollis. *Differentially Flat Trajectory Generation for a Dynamically Stable Mobile Robot*. 2013 IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, 2013.

[19] Martina Joševski, Dirk Abel. *Flatness-based Trajectory Planning for the Battery State of Charge in Hybrid Electric Vehicles*. 8th IFAC Symposium on Advances in Automotive Control, vol. 49, issue 11, 134-140, 2016.

[20] Michael Treuer, Tobias Weissbach, Günter Scheffknecht, Veit Hagenmeyer. *Trajectory Planning for Flatness-based Two-degree-of-freedom Control of a Pumped Storage Power Station*. 17th IFAC World Congress, vol. 41, issue 2, 11080-11085, Seoul, Korea, 2008.

[21] Thomas Ruppel, Karl Lukas Knierim, Oliver Sawodny. *Analytical Multi-Point Trajectory Generation for Differentially Flat Systems with Output Constraints*. 18th IFAC World Congress, vol. 44, issue 1, 950-955, Milan, Italy, 2011.

[22] Luigi Biagiotti, Claudio Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. Springer-Verlag Berlin Heidelberg, Bologna, 2008.

[23] Knut Graichen, Veit Hagenmeyer, Michael Zeitz. *A New Approach to Inversion-based Feedforward Control Design for Nonlinear Systems*. Automatica, vol. 41, issue 12, 2033-2041, 2005.

[24] Philippe Martin, Santosh Devasia, Brad Paden. *A Different Look at Output Tracking: Control of a VTOL Aircraft*. Automatica, vol. 32, issue 1, 101-107, 1996.

[25] Arom Boekfah. *Nonlinear Output-Transition Control for Nonminimum-Phase Systems: the VTOL Example*. 2017 2nd International Conference on Control and Robotics Engineering, Bangkok, Thailand, 2017.

[26] Veit Hagenmeyer, Emmanuel Delaleau. *Exact Feedforward Linearization Based on Differential Flatness*. International Journal of Control, vol. 76, issue 6, 537–556, 2003.

[27] Veit Hagenmeyer, Emmanuel Delaleau. *Exact Feedforward Linearization Based on Differential Flatness: the SISO Case*. Nonlinear and Adaptive Control, 161–170, Springer Berlin Heidelberg, 2003.

[28] Veit Hagenmeyer, Emmanuel Delaleau. *Robustness Analysis of Exact Feedforward Linearization Based on Differential Flatness*. Automatica, vol. 39, issue 11, 1941-1946, 2003.

[29] Andreas Rauh, Ekaterina Auer. *Modeling, Design, and Simulation of Systems with Uncertainties*. Springer-Verlag Berlin Heidelberg, 2011.

[30] T. John Koo, Shankar Sastry. *Differential Flatness Based Full Authority Helicopter Control Design*. 38th IEEE Conference on Decision and Control, Phoenix, USA, 1999.

[31] Volker Schneider. *Trajectory Generation for Integrated Flight Guidance*. PhD thesis, Institute of Flight System Dynamics, Technical University of Munich, academic year 2017-2018, supervisor Univ. Prof. Dr.-Ing. Florian Holzapfel.

[32] Florian Holzapfel. *Flight System Dynamics 2. Chapter 4: Nonlinear Equations of Motions*. Lectures of Flight System Dynamics 2, Institute of Flight System Dynamics, Technical University of Munich.

[33] International Civil Aviation Organization. *Performance-based Navigation (PBN) Manual*. ICAO Doc 9613 AN-937, Montréal, Canada, 3^{rd} Edition 2008.

[34] U.S. Department of Transportation, Federal Aviation Administration, Flight Technologies Division. *Instrument Procedures Handbook*. FAA-H-8083-16B, 2017.

`http://www.faa.gov/regulations_policies/handbooks_manuals/aviation/instrument_procedures_handbook/`

[35] Rong-Shine Lin. *Real-time Surface Interpolator for 3-D Parametric Surface Machining on 3-axis Machine Tools*. International Journal of Machine Tools and Manufacture, vol. 40, issue 10, 1513–1526, 2000.

[36] https://upload.wikimedia.org/wikipedia/commons/2/24/Hawker_Siddeley_Harrier_GR1%2C_UK_-_Air_Force_AN2256552.jpg

[37] https://newatlas.com/agustawestland-aw609-tiltrotor/21466/

[38] https://it.wikipedia.org/wiki/Bell_Boeing_V-22_Osprey#/media/File:V22-Osprey.jpg