Politecnico di Torino

POLITECNICO DI TORINO

Master's Degree
in Aerospace Engineering

Academic Year 2021-2022

Master's Degree Thesis

Formation flight of multiple UAVs using Artificial Potential Field Algorithm

Academic supervisors
Prof. Piero Gili
Angelo Lerro

Company supervisors
(Leonardo - Aircraft Division)
Alberto Chiesa
Umberto Papa

Candidate
Alessandro Favia
Matr. 268918

POLITECNICO DI TORINO

Master's Degree
in Aerospace Engineering

Academic Year 2021-2022

Master's Degree Thesis

Formation flight of multiple UAVs using Artificial Potential Field Algorithm

Academic supervisors
Prof. Piero Gili
Angelo Lerro

Candidate
Alessandro Favia
Matr. 268918

Company supervisors
(Leonardo - Aircraft Division)
Alberto Chiesa
Umberto Papa

A Lucia, Michele e Raffaella.

Declaration

This master thesis has been realized thanks to a mutual collaboration between *Politecnico di Torino* and *Leonardo Aircraft* so every result, *Matlab* code and *Simulink* model is of their own. The Author hereby declare that this thesis represents his own work which has been done after registration for the degree of the Master's Degree in Aerospace Engineering at *Politecnico di Torino*, and has not been previously included in a thesis or dissertation submitted to this or any other institution for a degree, diploma or other qualifications.

Abstract

In the near future *Unmanned Aerial Vehicles (UAVs)* will become increasingly important, especially for their great advantage of reducing risks regarding human presence in the field. The main issue with these platforms lies in the fact that their capabilities to operate outside of a military or segregated environment are quite limited. The achieved degree of autonomy is also limited and in fact most of them are remotely controlled or semi-autonomous, without the possibility to identify or handle out of the ordinary situations. The concept of *Manned - Unmanned Teaming (MUM-T)*, i.e. the ability of unmanned aircraft to support manned ones during a mission, is particularly important. This thesis, which is the result of the collaboration between Politecnico di Torino and Leonardo Company (Aircraft Division) and which has been developed during an internship in the aforementioned company, fits into this context. Using *Matlab/Simulink* a fixed-wing UAV model, that includes main flight control surfaces and autopilots, has been designed in order to simulate aircraft dynamics. As the leader-follower architecture has been adopted, a five-aircraft formation control algorithm based on PID (*Proportional - Integral - Derivative*) controllers has been implemented. PID is one of the simplest forms of automatic control that only allows to create statically controlled formations. Thus, the autonomy of the unmanned platforms is nullified and the safe feasibility of the controlled formation is not guaranteed. For this reason it was decided to change the approach, generating a system based on *Artificial Potential Fields (APF)*. Artificial potentials allow the overlapping of different aspects which should be considered (to perform a specific manoeuvre, to maintain a safe distance from the other components of the formation, to avoid an obstacle, the command authority to be used), all weighted through a cost function. In practice, the "desired" position of the UAV is commanded, but it is left with the autonomy to implement the command basing on needs. This is a clear breakthrough both in terms of the use of *self-awareness* of the platform and its *decision-making* capabilities. After studying the operating principle of the algorithm in its static (not time-variant) version, a two-dimensional Simulink model (which works only in the horizontal plane) has been created. This model has been subsequently extended to the vertical plane in order to obtain a formation control algorithm in three-dimensional space. In both cases the code is able to perform manoeuvrers and formation geometry variations in real time and it can be controlled by the user with a joystick. Along with the capability to avoid both other formation components and obstacles, any follower aircraft can also avoid flying over no-fly zones. The system can be easily adapted to the selected type of aircraft and

to mission needs by tuning specific parameters. Trajectories, attitude and potentials visualization is carried out using both real-time and post-production animations.

Sommario

Nel prossimo futuro i velivoli senza pilota (*Unmanned Aerial Vehicles - UAVs*) acquisiranno sempre più importanza, soprattutto per il grande vantaggio di ridurre i rischi inerenti alla presenza umana sul campo. Il problema più grande di queste piattaforme risiede nel fatto che le loro capacità di operare al di fuori di un ambiente militare o segregato sono piuttosto limitate. Il grado di autonomia ad oggi raggiunto da tali velivoli è anch'esso limitato ed infatti la maggior parte di essi sono controllati da remoto o sono semi-autonomi, senza capacità di riconoscere o gestire situazioni al di fuori del compito ordinario. Particolare rilevanza assume il concetto di *Manned - Unmanned Teaming (MUM-T)*, ovvero la capacità dei velivoli *unmanned* di affiancare in missione quelli con equipaggio. Questa tesi, frutto della collaborazione tra Politecnico di Torino e Leonardo Company (Divisione Velivoli) e sviluppata durante un periodo di tirocinio curricolare nella suddetta azienda, si inserisce all'interno di questo contesto. Grazie all'utilizzo del software *Matlab/Simulink* si è innanzitutto costruito un modello di UAV ad ala fissa comprendente superfici mobili primarie e autopiloti, utilizzato per simulare la dinamica del velivolo. Avendo adottato un'architettura *leader-follower*, è stato implementato un algoritmo di controllo di formazione a cinque velivoli basato su controllori PID (*Proportional - Integral - Derivative*). Il PID è una delle forme più semplici di controllo automatico che permette unicamente di realizzare formazioni comandate in maniera statica. Questo azzera l'autonomia delle piattaforme *unmanned* e non garantisce la realizzabilità in sicurezza della formazione comandata. Per questa ragione si è deciso di cambiare approccio, generando un sistema basato su potenziali artificiali (*Artificial Potential Fields - APF*). I potenziali artificiali consentono la sovrapposizione di diversi aspetti da considerare (il dover eseguire una specifica manovra, il mantenere una distanza sicura dagli altri componenti della formazione, evitare un ostacolo, l'autorità di comando da usare), tutti ponderati tramite una funzione di costo. In pratica, la posizione "desiderata" dell'UAV viene comandata, ma ad esso è lasciata l'autonomia di implementare il comando tenendo conto della situazione. Si tratta di un netto passo avanti sia in termini di uso della *self-awareness* della piattaforma che delle sue capacità di *decision making*. Dopo aver studiato il funzionamento dell'algoritmo nella sua versione statica (non tempo-variante), si è costruito un modello Simulink bidimensionale (i.e. agente unicamente sul piano orizzontale). Successivamente tale modello è stato esteso anche al piano verticale in maniera da ottenere un algoritmo di controllo di formazione nello spazio. In entrambi i casi il sistema è capace di effettuare manovre e variazioni della geometria della

formazione in tempo reale ed eventualmente di essere controllato dall'utente attraverso joystick. Ai vari velivoli follower, che si evitano tra loro e che evitano ostacoli, si può anche ordinare di non sorvolare no-fly zones. Il sistema è adattabile al tipo di velivolo utilizzato e alle esigenze di missione mediante taratura di parametri specifici. La visualizzazione di traiettorie, dell'assetto e dei potenziali è effettuata sia mediante animazioni real-time che in post-produzione.

Contents

List of Tables

List of Figures

"One thing I have learned in a long life: that all our science, measured against reality, is primitive and childlike – and yet it is the most precious thing we have."

[A. Einstein, Letter to Hans Muehsam (July 1951), Einstein Archives 38-408]

Chapter 1

Introduction

This dissertation is the result of a collaboration between *Politecnico di Torino* and *Leonardo Aircraft*, being the Author's MSc Thesis in Aerospace Engineering. Starting from April 2022 the Author had the valuable opportunity to attend a six-months in person internship at the *Leonardo Aircraft* headquarters sited in Turin. During this period large part of the project, which will be here described, has been realized. It would be highlighted that every result derives from a long *trade-off* procedure and that not all the tested versions will be illustrated in this dissertation for the sake of shortness.
Formation control of unmanned vehicles has received a lot of attention in the past decades, with application to the coordination of various type of plants (aircraft, spacecraft, robots, underwater vehicles). The main advantage of unmanned vehicles is the minimization of human mistakes and pilot injury risks, especially in the military sector. On the other hand benefits of *formation control* are numerous, such as improved mission capability, efficiency and energy saving. There could be various degrees of autonomy: every actor can be *remote piloted* or completely *autonomous*. It could be easily imagined how great interest lies in studying how *autonomous unmanned vehicles* (potentially flanked by *manned* ones) can flight together in formation in order to successfully accomplish a specific mission. In this chapter an introduction about general aspects of formation flight is presented. First of all, a brief footnote of history will be included, explaining why multiple aircraft formation flight has been an important topic in the past. At a later time, general theoretical aspects will be illustrated as section formation pattens and the *Manned-Unmanned Teaming (MUM-T)* concept.

1.1 Historical background

By *formation flying* we mean the flight of two or more aircraft travelling and manoeuvring together in a disciplined, synchronized, predetermined manner [1]. The history of this discipline starts in World War I (1914-1918), when single pilot *fighter aircraft* escorted *reconnaissance* ones. This last type of vehicles, whose primary aim was to visually identify enemy troops and settlements, could fly at high altitudes (which for that time were about $24000\,ft$) to avoid interception thanks to their powerful engines. Advantages of flying in

formation were soon discovered observing how fighting together increased aerial victories and the German air force provided some of the basic rule of formation flight, some of whom are still used in the present day. After World War I formation flying remained the centre of attention also thanks to the development of air shows: aircraft began not to be perceived as dangerous and first *airliners* were built. The development of *jet engines* and the increase of aircraft velocity and manoeuvrability made formation flying more and more an elite discipline, forcing pilots to focus on training. In Fig. 1.1 *Frecce Tricolori*, the aerobatic demonstration team of the Italian Air Force, are depicted.

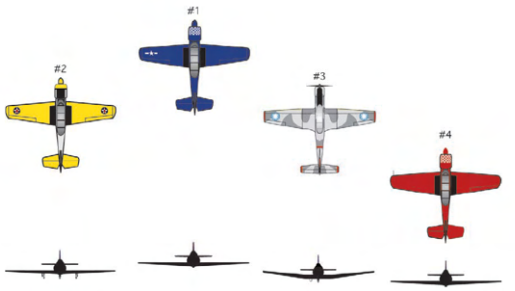
Figure 1.1: Frecce Tricolori (from [2])

1.2 Formation flight organization

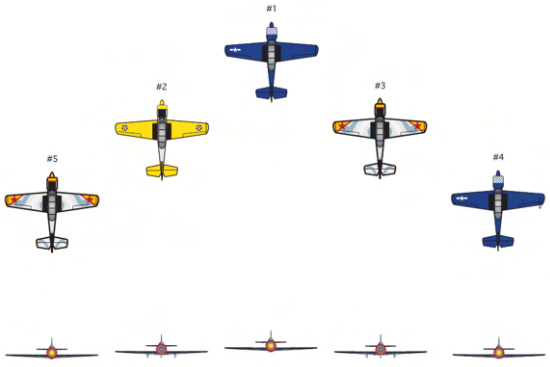
Before delving into autonomous flight concepts some basic features of piloted (manned) formation flight will be examined, starting from terminology. First of all, a group of more than one aircraft is called a *flight*. The smallest unit of formation is the *section*, composed of a *leader* and a *wingman*. The *leader*, that is usually the most experienced pilot, have the responsibility to safely conduct the flight and plan the mission. Other pilots, regardless of their number, are called *wingman*: they should maintain formation integrity, providing mutual support and follow leader's commands. Aircraft arrangement inside the formation can be various, generating a number of formation *"shapes"*. During a mission multiple formation shapes can be selected, depending on the scenario. The basic formation configurations are *fingertip*, *echelon* and *line astern*, while most other configurations are variations of these [3].
If a ***line astern*** (or *trail*, Fig. 1.2(a)) formation is chosen, every wingman follows the leader in trail. The distance between aircraft can vary, depending on mission requirements. During World War I it was one of the most common pattern but it was soon found to be

inefficient, especially because aircraft could not visually communicate among themselves. The ***fingertip*** formation (Fig 1.2(b)) was created in order to overcome this problem. It takes its name from the shape of a hand if viewed from above: if the pattern is similar to the right hand it is called *"fingertip strong right"* while if the selected one is the left, *"fingertip strong left"*. There are three wingman (two *sections*), each of which should maintain relative inclination and lateral/vertical spacing from the leader. If the number of aircraft is odd there will be a *section* made up of one aircraft (instead of two). In this case the formation is called ***vic*** (Fig. 1.2(c)) and the concept of *phantom wingman* is adopted: the leader, in fact, gives command as if an additional plane existed. The ***echelon*** formation (Fig. 1.2(d)) is a configuration where every aircraft is on the right or on the left of the leader. Usually employed during traffic patterns or surveillance missions, its main advantage is the high range of vision of every participant. If the inclination between aircraft is null, the formation is called ***line abreast*** (or *wall*). The ***box*** (or *diamond*, Fig. 1.2(e)) formation is particularly difficult as every aircraft must fly very close to each other. Being compact and manoeuvrable, it is considered as a building-block to generate bigger formations. In our thesis an additional shape has been created by the Author, called *arrow* formation. Conceived as a hybrid between *box* and *vic*, in this case the leader is surrounded by wingmen and the geometry resembles an arrowhead.

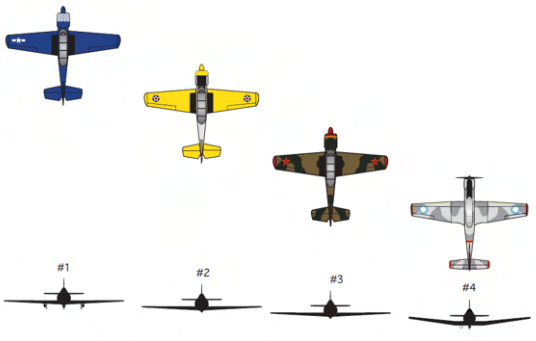
(a) *Line astern* formation



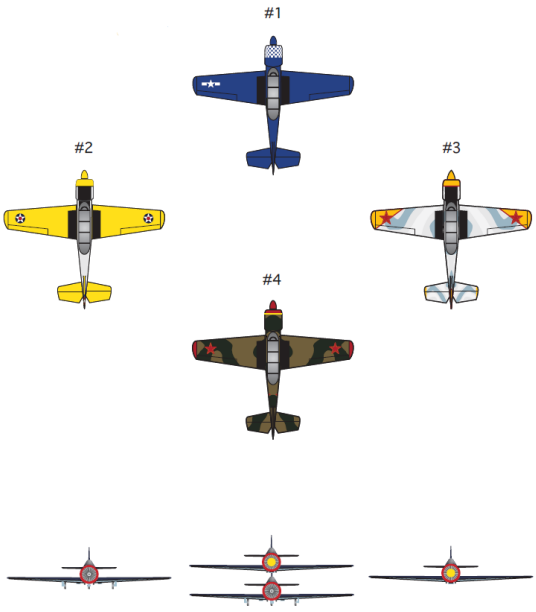
(b) *Fingertip* formation



(c) *Vic* formation



(d) *Echelon* formation



(e) *Box* formation

Figure 1.2: Basic formations (from [3])

Piloted (manned) formation flight is generally performed using eyesight; key concepts, in this case, are *recognition* and *anticipation* [3]. Taking as an example a section (two aircraft), the basic idea of *recognition* is that the wingman has to recognize its motion in relation to the leader and, if the above-mentioned motion is unwanted, it should make corrections. Movements are usually perceived using fixed reference on the leader, observing if they move from the wingman point of view. On the other hand, *anticipation* means that pilots should be ready for every change of the formation, thinking about the manoeuvre in advance. The *autonomous (unmanned) formation flight* approach, as can be imagined, directly derives from the piloted one. The main difficulty in this case is that formation control algorithms should be extremely reliable, having a short response time. Advantages and characteristics of autonomous formation flight will be explained in depth in Sect. 2.1.

1.3 *Manned – Unmanned Teaming (MUM-T)*

In the last few years the concept of *Manned-Unmanned Teaming (MUM-T)* has become more and more established, as it is clear that it will become a pillar of *Future Combat Air Systems (FCAS)*. MUM-T has been defined, by the United States Army Aviation Centre (USAACE), *"the synchronized employment of soldier, manned and unmanned air and ground vehicles, robotics, and sensors to achieve enhanced situational understanding, greater lethality, and improved survivability"* [4]. The basic idea is to create a *system* in which *manned* vehicles can operate jointly with *unmanned* ones, whose purpose is to assist them during the mission. Particular attention is paid to *Unmanned Aerial Vehicles*

(UAV): according to [5], the value of the UAVs global market will reach ca. fourteen billion dollars in 2026, with a 300 % increase compared to 2017. UAVs are changing the way in which civil and military operation are conducted, improving tasks like data and image acquisition of areas of interest, localization and tracking of specific targets, map building, communication relays, pipeline surveying, border patrolling, military operations, policing duties, persistent wide area surveillance, search and rescue, and traffic surveillance [6]. Moreover, in a world where *Advanced Air Mobility (AAM)* will characterize future cities, it will be crucial to define the reciprocal relationship between manned and unmanned vehicles. According to the *NATO STANAG 4586*, communication complexity can be summarized using five *Levels of Interoperability (LOIs)*:

Levels of Interoperability	
"The ability of robots to operate in synergy to the execution of assigned missions and the capability of diverse systems and organizations to work together, sharing data, intelligence and resources"	
Level 1	Indirect receipt and/or transmission of sensor product and associated metadata
Level 2	Direct receipt of sensor product data and associated metadata from the UAV
Level 3	Control and monitoring of the UAV payload unless specified as monitor only
Level 4	Control and monitoring of the UAV, unless specified as monitor only, less launch and recovery
Level 5	Control and monitoring of UAV launch and recovery unless specified as monitor only

Table 1.1: NATO's STANAG 4586 - *Levels of Interoperability* (from [6])

While current technology is able to successfully perform up to Level 4 (e.g. *Boeing AH-64 Apache*), within the scope of MUM-T the most challenging goal is to reach the Level 5 of LOIs, that would allow manned aircraft to control and monitor UAVs with the possibility to launch and recover it. In this scenario, it can be imagined how one of the most important technologies on which MUM-T will be based is the development of *AI-based navigation algorithms*. This thesis can be included in this context as its main purpose is to demonstrate how, using *Artificial Potential Field (APF)* algorithms, *manned* and *unmanned* aircraft can fly together in formation.

Chapter 2

Formation control and motion planning of multiple UAVs: the state of the art

In this chapter, an introduction about the state of the art of formation control and motion planning is presented. At the beginning, the research topics and the key concepts will be illustrated, focusing on differences and similarities between the previous subjects. Different *formation control strategies* will be investigated, along with advantages and disadvantages. Subsequently, specific attention will be paid on *path planning*, in particular to its *"cooperative"* meaning. Practical applications and historical background have been discussed in the previous chapter, together with the purpose of the thesis. However, choices made will be illustrated here, opening the door to following explanations of the project in the next sections.

2.1 Research areas

Autonomous formation flight is, at the moment, a prolific research area in the aerospace community. The reasons for studying how a number of aircraft could fly closely and autonomously are numerous. First of all, it has been demonstrated that flying in the wing-tip vortex of an aircraft exhibit fuel saving [7] [8]. Particularly useful during long range missions, it can be explained observing the reduction of induced drag of the trailing wingmen, which need less energy to maintain its speed [9]. This behaviour has been observed in many species of birds, especially the migratory ones, which travel in "V" formation during long journeys. Another reason for designing robust formation flight algorithms lies in using them in the civil sector, with the urgent demand for aircraft coordination within high density airspace, especially near airports [10]. *Autonomous Aerial Refuelling (AEF)* is another research area of interest, both in civil and military sector [11] [12]. From the military point of view, the interaction between multiple *Unmanned Aerial Vehicle (UAV)* and (possibly) manned aircraft is a very

important issue. As can be seen in [13] the *Ghost Bat* program of the Royal Australian Air Force (RAAF), for example, aim to develop a loyal-wingman system together with Boeing, with its planned entry into service date not so far in time.

Another increasing trend is the development of more sophisticated *unmanned vehicles*, which can operate in multiple mission scenarios. The reduction of risks regarding the human presence on the field is only one of the reasons why these systems are becoming so common. The degree of autonomy reached today does not allow them to act completely independently, so they are generally *semi-autonomous* because the human intervention is needed to assure a reasonable reliability. In this context, adopting a *fleet* of UAVs instead of a single one could be an interesting solution since it can assure an increase in mission capability as well as augmented redundancy.
There are, in this regard, two main research areas particularly prolific in the aerospace sector: *formation control* and *cooperative motion planning*.

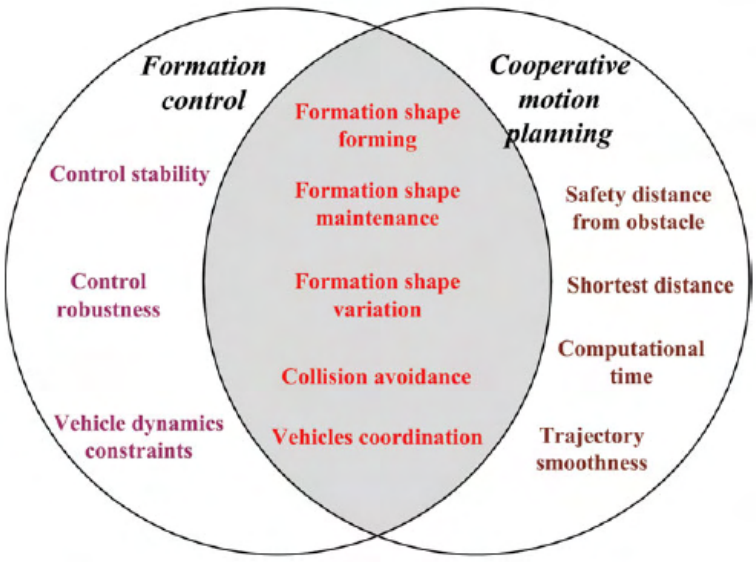


Figure 2.1: Autonomous formation flight research areas (from [14])

In Fig. 2.1 differences and similarities are illustrated. The main aim of *formation control* is to drive every single agent to reach a prescribed constraint on his state [15]. Consensus-based controller are one of the most studied solutions for the problem, together with flexible algorithms which implement *collision-avoidance* features. Anyway, formation control does not act on the trajectory of the formation seen as a whole: its aim is to ensure that every aircraft is in a defined position with respect to a certain reference system. Control stability, robustness and vehicle dynamics constraints (like aircraft *inertia*) are factor to be taken into account, but they are not the only ones. *Cooperative motion planning* intervenes to fill the gap: in fact, it deals with formation trajectory optimization, taking into account the peculiarity of having multiple agents acting together [16]. In this case the attention is drawn on trajectory smoothness (especially for aircraft, which should operate inside their *flight envelope*), computational time, obstacle avoidance and shortest

path for fuel saving. Having said that, it can be imagined that *formation control* and *cooperative motion planning* share some important features. Every vehicle, for example, should be controlled taking care of collision avoidance between its counterparts and if an obstacle appears the trajectory of every vehicle must be generated so that it can be efficiently avoided and the formation gradually recovered.

2.2 Formation control

The reason multiple aircraft could put in formation is mainly because they can assure better performances in a certain mission and because of tactical reasons, such as lowering radar signature. The aforementioned *"V shape"* (or *"Vic formation"*), for example, assure a good view of the neighbouring situation, but could not be so useful in small mission area as the *"line astern"* figure. Besides, when the mission scenario changes dynamically the formation must adapt changing his geometry, taking care of avoiding other vehicles and deadlock situations (the aircraft must not block the path of the other ones). That said, the importance of a robust formation control strategy comes to light. According to [14], there are three types of features that a control strategy could implement:

- *Formation generation and maintenance (Type 1)*: from a random condition (e.g. random positions and headings), the formation should be formed and maintained
- *Formation maintenance during trajectory tracking (Type 2)*: the formation should be maintained during a manoeuvre or a predefined trajectory
- *Formation shape variation and re-generation (Type 3)*: if an obstacle appears, it should be avoided either with a formation trajectory deviation from the nominal one or a temporary variation of the formation shape, which should be restored at later time.

The importance of this partition lies in the fact that it can be useful to classify the three types of control strategies that will be briefly described in the following lines.

2.2.1 The *leader-follower* formation control approach

The first control strategy is called *"leader-follower approach" (LFA)*. An aircraft is elected as *leader*, it has all the information about the trajectory to follow to reach a certain point in the space and its motion define the one of all the others, called *"followers"*. Their purpose is to maintain a certain distance from the *leader* geometrically calculated from the desired formation shape. This approach has been extensively studied in literature, with numerous variations on the theme. In [17], every element (in this case, robots) is considered as point-mass and the control law aim to nullify distance and angular errors then the desired values. A similar approach can be noticed, for example, in [18]. In other cases collision-avoidance features can be implemented, as in [19] and [20]. The *leader-follower approach* is frequently adopted because *Type 1*, *Type 2* and *Type 3* could be guaranteed. Moreover, a *virtual leader* (instead of the physical one) could be implemented.

The main problem of this approach is that there must be a good quality communication between all the aircraft (especially with the leader) in order to transmit position data. These uncertainties could be partially modelled using, among others, machine-learning techniques.

2.2.2 The *virtual structure* and the *behaviour-based* formation control approaches

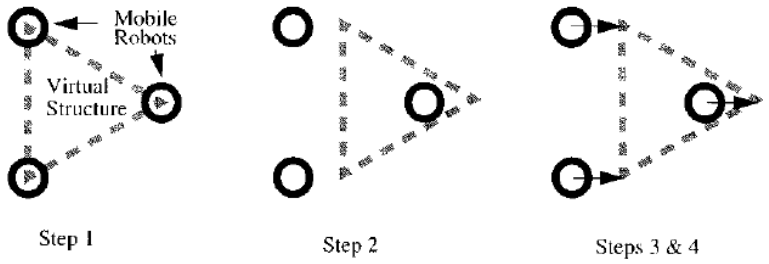


Figure 2.2: Steps in the virtual structure control algorithm for moving in formation (from [21])

The second control strategy is called *"virtual structure" approach* [21]. In this case there is no distinction between aircraft, which are equally important. The aim of every vehicle is to minimize the position error with respect to a vertex of the virtual structure (the *shape* of the formation), treated as a rigid body. The virtual structure is translated to follow a predetermined trajectory, forcing every aircraft to move to the subsequent vertex position (Fig. 2.2). As compared to the previous strategy, one of the advantages is the reliability in relation to the *Type 1* strategy, but the major drawback is that collision avoidance (*Type 2*) is particularly difficult to implement.

The third control strategy is the *"behaviour-based" approach* [22]. The formation control problem is solved using a *hybrid vector-weighted control function*. In particular, different control schemes (e.g. *move-to-goal*, *avoid-static-obstacle* and *maintain-formation*) are developed and the general control scheme for the specific mission is the result of the weighting of them (using gain values). The main problem of this approach is that the above-mentioned general control scheme does not depend on kinematic/dynamic characteristics of the robots, so the study of system stability become quite complex to pursue. However, all type of formation control strategies could be implemented, including collision avoidance. In Tab. 2.1 characteristics of the three formation control approaches are briefly summarised.

2.2.3 The formation control approach adopted for the thesis

During the initial phase of requirement definition, some important must-have features of the novel algorithm at the centre of the thesis came to light. First, the code has to consider the presence of a *manned aircraft*, followed by a number of *unmanned aircraft*. Every

UAV must follow the *manned* autonomously in order to generate a predefined formation pattern. Contacts between vehicles must be avoided, and *collision avoidance* should be implemented to avoid fixed or mobile obstacles. That said, it has been established that the most reasonable approach was the *leader-follower*, especially because the presence of a manned *leader* was the most realistic scenario. Relatively simple to design, this approach has another important advantage, which is the *centralized communication architecture*. In fact, all data needed to carry out the separation between aircraft result in a direct link between the *leader* and the single *follower*, with a reduction in the overall quantity of exchanged data as compared to a *decentralized communication architecture*. However, the author would underline that this type of approach does not exclude a configuration in which the leader is *unmanned*.

2.3 Cooperative formation path planning

As said in previous sections, *cooperative formation path planning* (or *cooperative motion planning*, as mentioned before) is a field of study which works in parallel with *formation control*. In general, path planning consists in finding an optimal or near-optimal path from the starting state to the target state that avoids obstacles based on one or some performance indicators [23]. For a 3-D path planning algorithm, for example, the problem can be schematized as [24]:

$$P_s(x_s, y_s, z_s, \theta_s, \phi_s) \overset{r(t)}{\rightarrow} P_f(x_f, y_f, z_f, \theta_f, \phi_f) \tag{2.1}$$

where P is a point, subscripts s and f means *start* and *final*, and $r(t)$ is a path (Fig. 2.3).

The path choice, for a single-robot case, depends on various factors such as lowest working cost, shortest walking route and shortest walking time. Extending the problem for a formation of N vehicles, a number of N paths have to be calculated taking into account additional performance costs like:

- *Internal collision avoidance*: vehicles should avoid others inside the formation
- *Formation behaviour*: if a formation pattern should be maintained, paths are influenced
- *Cooperation behaviour*: there are two different forms of cooperation behaviour, *time cooperative behaviour* and *time and position cooperative behaviour*. The main difference lies on the fact that the formation could be broken in encountering an obstacle (first case) or it should be always maintained (second case).
- *Total distance*: the path optimization should take into consideration not only the single N-th path length, but also the sum of distances.

There are several ways to categorise path-planning algorithms. For example, they could be divided according to the know degree of environmental information: global-map based (*global path planning*) or local-map based (*local path planning*) [23]. In the first case it is

Methods	Advantages	Disadvantages	Formation maintenance types	Platforms
Leader-follower	• Easy to be designed and implemented • Efficient communication within the system	• Highly dependent on the leader vehicle • Lack of the feedback from the follower to the leader	*Type 1*, *Type 2* and *Type 3*	• Widely adopted across various platforms
Virtual structure	• Good performance in shape keeping • Good representation of the relationship and the coordination between each vehicle in the formation	• Not flexible for shape deformation • Not easy for collision avoidance	*Type 1* and *Type 2*	• Most applications seen on mobile robots • Less application on unmanned vehicles
Behaviour-based	• Capable of dealing with multi-task mission	• Not easy to mathematically express the system behaviour • Difficult to prove and guarantee the system stability	*Type 1*, *Type 2* and *Type 3*	• Mobile robots and UGVs are two popular platforms

Table 2.1: Formation control strategies (from [14])

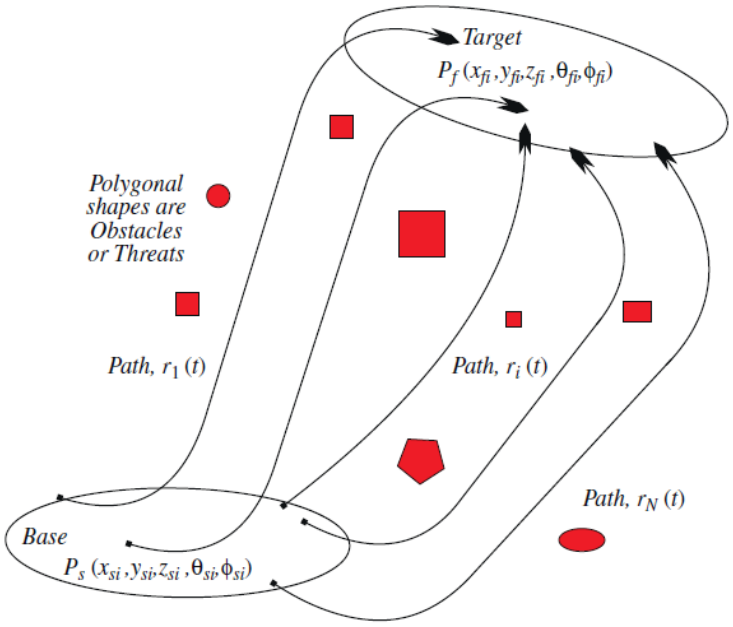


Figure 2.3: The path planning problem (from [24])

assumed that the environment is completely known, especially the obstacle position and shape: an environmental model is created (using, for example, a grid decomposition) and the path is calculated. In the second case the surrounding is sensed only using *sensors*, so the path is based on limited perception of the environment and it should be adjusted in real time.
Another classification refers to the *deterministic* or *heuristic* approach (Fig. 2.4). In the deterministic approach, the solution for the trajectory exists and it is exact (being made up of a finite number of step): if nothing changes, the output of the simulation remains the same no matter how many times the same is run. It can be considered *complete* and *consistent*. Conversely, a heuristic approach is used if an exact solution does not exist or if it is difficult to find. In this case the solution is *near-optimal* since it is the result of an approximation and it generally change if the simulation is repeated.

2.3.1 The Artificial Potential Field method (references)

The Artificial Potential Field is a powerful path-planning algorithm that was first proposed by Oussama Khatib in 1986. Being the selected method for the thesis, it will be extensively explained in Ch. 5 but there it will be briefly introduced in order to compare it with other solutions provided in literature. In general, the Artificial Potential Field method generates, in every point in which the physical space has been discretized, a potential field. There, an *attractive* potential is centred in the target point and influences all the surrounding space while *repulsive* potentials are centred in the obstacles and act only in certain areas around them. The *steepest descent* direction can be calculated doing the gradient of the

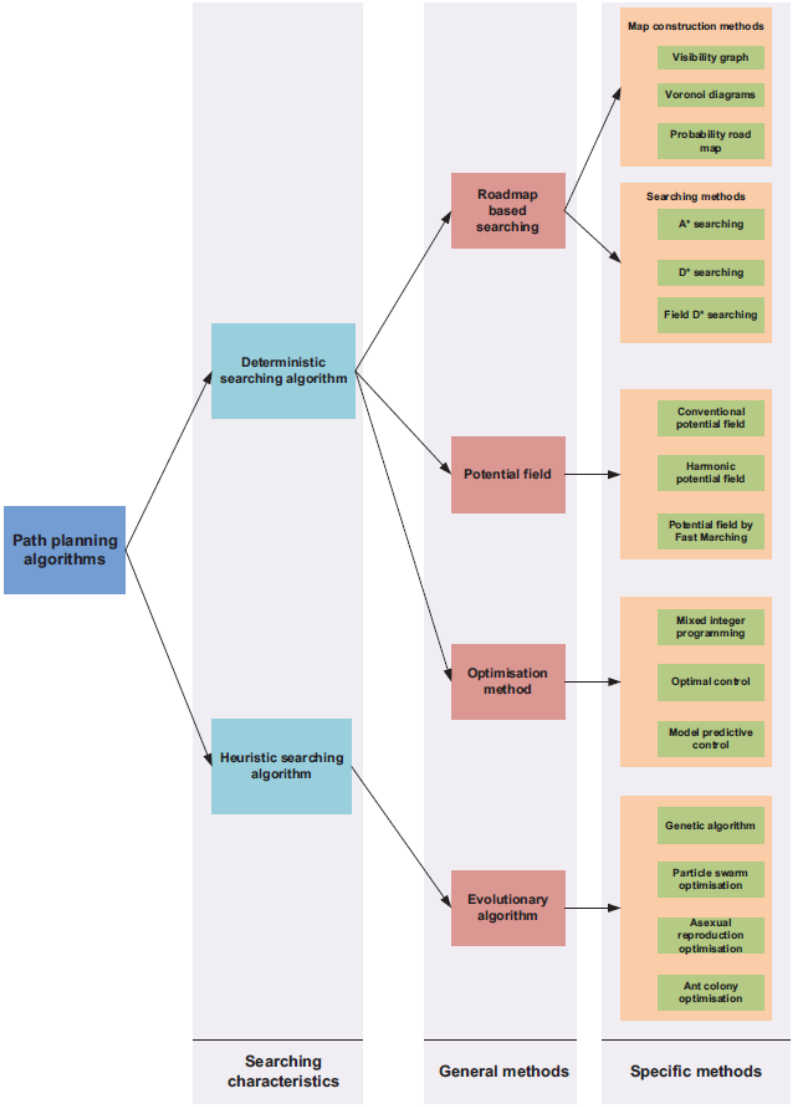


Figure 2.4: Deterministic and heuristic path planning categorization (from [14])

potential which, in this case, corresponds to a *force*:

$$F_{total} = F_{att} + F_{rep} = -\nabla(U_{att} + U_{rep}) \tag{2.2}$$

When used for formation path planning other type of potential should be implemented, in order to avoid collision between robots and maintain formation shape. Fig. 2.5 represents an example of a potential used in this thesis for formation control. A particular version of APF is called *"Fast Marching Method based Potential Field"* [25]. Being the result of the application of the Voronoi Fast Marching (VFM) method to the APF, it generates the potential field simulating the propagation of an electromagnetic

wave (a *viscosity map* is extracted from the discretized space grid). In this case the algorithm is faster, even though more complicated to implement.

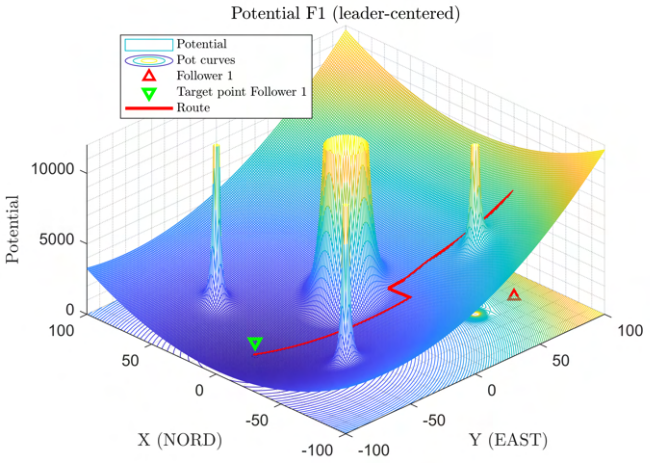


Figure 2.5: Example of potential used for formation path planning

2.3.2 The optimal control method

The *optimal control method* represents another approach to the formation path planning. In this case, the problem is faced dividing it into several sub-problems (one for each vehicle), solved using numerical optimization and considering a set of constraints. A basic formulation is available in [26]. Let denote the initial state of the vehicle with $\boldsymbol{x}_{init}$ and the desired (final) state with $\boldsymbol{x}_f$. The planning horizon of the algorithm T (how far in time the route is calculated) depends on a number of factors, like the distance over which the space has been discretized. The *cost function* for the i^{th} time step is $\ell_i(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{x}_f)$ where $\boldsymbol{u}_i$ is the input vector. The problem can be formulated as follow:

$$\min_{\boldsymbol{x}_i, \boldsymbol{u}_i} J_T = \sum_{i=0}^{T-1} \ell_i(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{x}_f) + f(\boldsymbol{x}_T, \boldsymbol{x}_f) \tag{2.3}$$

$$\begin{aligned} \text{subject to: } & \boldsymbol{x}_{i+1} = \boldsymbol{A}\boldsymbol{x}_i + \boldsymbol{B}\boldsymbol{u}_i \\ & \boldsymbol{x}_0 = \boldsymbol{x}_{init} \\ & \boldsymbol{x}_i \in \mathcal{X} \\ & \boldsymbol{u}_i \in \mathcal{U} \\ & (x_i, y_i) \in \mathcal{D} \\ & (x_i, y_i) \notin \mathcal{O}) \end{aligned}$$

where $(\boldsymbol{A}, \boldsymbol{B})$ represent matrices of the linear state space model (longitudinal), (x_i, y_i) denotes the position of the vehicle in the plane, the set $\mathcal{D}$ represents the discretized physical space and the set $\mathcal{O}$ characterizes the obstacles. Sets $\mathcal{X}$ and $\mathcal{U}$ represent the dynamic and kinematic constraints of the vehicle, such as maximum turn rate and minimum speed restrictions. Using *Mixed Integer Linear Programming (MILP)*, the problem could be solved. The main disadvantage of this method is the solver high computation complexity, which make it difficult to use it for on-line applications.

2.3.3 The evolutionary algorithm

The *evolutionary algorithm (EA)* is a *heuristic* algorithm inspired to biological evolution process. In case of path planning, it mimics the natural selection considering every path as an individual which mutate with time. Using a number of constraints and selection criteria the best path is chosen for every vehicle. When used for the cooperative path planning, it generally consists of two processes: a *N-th vehicle process* calculate the optimal trajectory for every single vehicle separately while the *master process* takes into consideration the cooperative behaviour. In particular, it re-evolves every path considering collision avoidance and distances inside the formation. The computational speed for this type of algorithm should be taken into consideration, because it could not be suitable for on-line planning.

2.3.4 Cooperative formation path planning approach adopted for the thesis

The approaches as above are summarized in Tab. 2.2. When a single vehicle is involved in the simulation, grid and map-based algorithms are preferred, like *D-star* [27] and *rapidly-exploring random trees* [28] respectively. For formation path planning, where the computational time is extremely important (algorithms become more and more complicated with the increase in the number of the vehicles), APF and Evolutionary Algorithms are the first choice. In addition to be faster than others, they produce smooth trajectories which can be easily followed by the UAVs.

For the purpose of this thesis, a "traditional" *Artificial Potential Field* has been chosen for its simplicity and effectiveness. It has been extensively used for unmanned aerial vehicles (in contrast to *Fast Marching*, which has been experimented only on mobile robots) and it can be easily updated or adapted. In fact, there are several potential functions in literature to be used in different situations, like the *Morse potential* [29] or the one proposed by Khatib [30].

Methods	Deployment platform	Algorithm completeness	Algorithm consistency	Cooperative behaviour type	Capability for multi-optimisation
Potential field method	Mobile robots, AUV, UAV	Incomplete	Consistent	• Time and position cooperative behaviour • Time cooperative behaviour	No
Fast marching method based potential field	Mobile robots	Complete	Consistent	• Time and position cooperative behaviour • Time cooperative behaviour	Yes
Evolutionary algorithm	Mobile robots, UAV	Probabilistic complete	Inconsistent	• Time cooperative behaviour	Yes
Optimal control method	Mobile robots, UAV, AUV	Complete	Consistent	• Time and position cooperative behaviour • Time cooperative behaviour	Yes

Table 2.2: Cooperative multi-vehicle path planning algorithms (from [14])

Chapter 3

The dynamic model of UAV using MATLAB Simulink

This chapter represents the introduction of the model of the aircraft used in the thesis. At first, a summary of theory of longitudinal and lateral-directional linearised dynamics will be illustrated, focusing on state-space representation. The general characteristics and performances of the UAV will be described, as trim conditions and saturations. At a later time there will be a detailed description of the Simulink model with control fundamentals, controllers tuning and code explanation. Having a solid understanding of the aircraft will be necessary to test the APF-based algorithm in the following chapters.

3.1 Linearised dynamics: general characteristics

3.1.1 Reference systems, kinematic, navigation and dynamics equations

The term *"flight dynamics"* generally refers to the study of how forces acting on an aircraft determine his velocity and attitude with respect to time. In order to describe briefly the equations involved, it is necessary to clarify which *reference systems* are included in this dissertation. Considering a point P in space, the first reference system is called *North-East-Down (NED)*: having its origin in P and being a right-handed coordinates, its x, y, z axes point toward north, east and down respectively. For flight dynamics problems, NED coordinates are considered *inertial* neglecting, for example, the effect of the Earth rotation. The *Earth-Centred-Inertial (ECI)* has its origin in the centre of the Earth with z pointing toward north and x toward the Aries constellation. It is fixed with respect to the celestial sphere but it is generally used only for *navigation problems*, in which long periods of time are involved. Finally, the *Aircraft-Body-Coordinates (ABC)* axes are centred in the centre of gravity of the aircraft with x, y, z direction which could be arbitrarily chosen. In many cases the *stability axes* are used, which correspond to the *wind axes* (i.e. aircraft-centred axes with x pointing toward the velocity direction) in equilibrium conditions.

The reference systems defined, it is possible to describe the *equations of motion* of

the aircraft. These twelve equations aim to describe completely how an aircraft moves in space and are:

- Kinematic equations (3)
- Navigation equations (3)
- Dynamics equations (6)

First, consider a NED and an ABC reference systems. If we want to pass from the first to the second, we should essentially move the origin of NED in the centre of gravity of the aircraft and rotate it. The theory suggest that the order of rotation is fundamental: in fact, rotations are *not commutable*. For aircraft, is frequently used $Z \rightarrow Y \rightarrow X$ (*Tait-Bryan*) and the angles are called *Euler angles* (Fig. 3.1).

The rotation matrix $R_{NED \rightarrow ABC}$ is, as usual, the product of the ones derived from every rotation around a single axis:

$$\begin{aligned} R_{NED \rightarrow ABC} &= R_\varphi R_\theta R_\psi \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi \\ 0 & -\sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{3.1}$$

The *kinematic equations*, whose proof is not included there for the sake of brevity, describe the relation between angular velocities in body coordinates $\omega_B = [p, q, r]^T$ and time variation of Euler angles $[\dot{\varphi}, \dot{\theta}, \dot{\psi}]^T$:

$$\begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\varphi\tan\theta & \cos\varphi\tan\theta \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \frac{\sin\varphi}{\cos\theta} & \frac{\cos\varphi}{\cos\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \leftarrow \textit{Kinematic Equations} \tag{3.2}$$

In case of longitudinal motion ($r \simeq 0, \varphi \simeq 0$) the $\dot{\theta} = q$ while for lateral-directional motion ($q \simeq 0, \theta \simeq 0$), $\dot{\varphi} = p$.

The *navigation equations* derive directly from the relation between NED and ABC coordinates. If we define P_{NED} as:

$$\boldsymbol{P}_{NED} = \begin{bmatrix} x_{NED} \\ y_{NED} \\ z_{NED} \end{bmatrix} = R^T_{NED \rightarrow ABC} \begin{bmatrix} x_{ABC} \\ y_{ABC} \\ z_{ABC} \end{bmatrix} \tag{3.3}$$

Deriving Eq. 3.3 we find:

$$\begin{aligned} \dot{\boldsymbol{P}}_{NED} = \begin{bmatrix} \dot{x}_{NED} \\ \dot{y}_{NED} \\ \dot{z}_{NED} \end{bmatrix} &= R^T_{NED \rightarrow ABC} \begin{bmatrix} u_{ABC} \\ v_{ABC} \\ w_{ABC} \end{bmatrix} \\ &= R^T_{NED \rightarrow ABC} \dot{\boldsymbol{V}}_B \leftarrow \textit{Navigation Equations} \end{aligned} \tag{3.4}$$

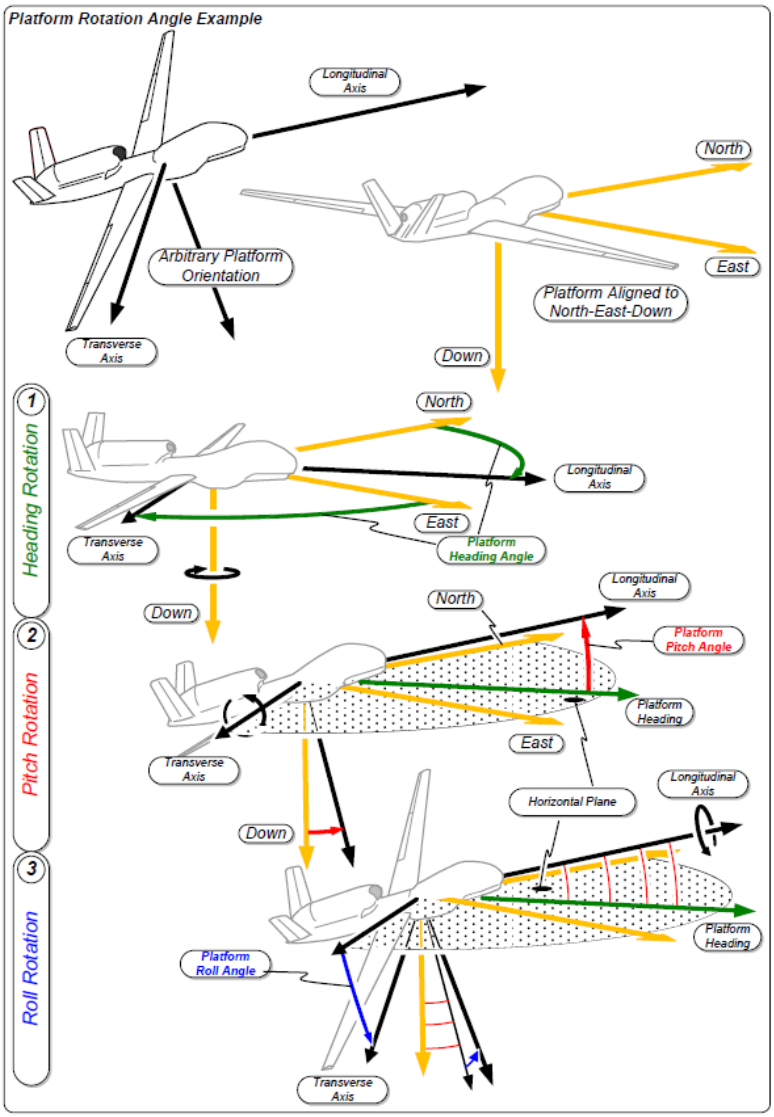


Figure 3.1: Euler angles example (from [31])

Velocities in body frame $\dot{\boldsymbol{V}}_B$ should be known using a Pitot tube on the aircraft. However, it should be noticed that *wind* has not been considered in this dissertation.

Let now introduce the *flight dynamics equation* in their general form. It is possible to define the equations of motion of the aircraft from Newton's second law:

$$\begin{cases} \sum \boldsymbol{F} = \frac{d}{dt}(m\boldsymbol{v}) \\ \sum \boldsymbol{M} = \frac{d}{dt}(\boldsymbol{H}) \end{cases} \tag{3.5}$$

The summation of all external forces is equal to the time rate of change of the momentum of the body, while the summation of all the external moments is equal to the

time rate of change of the angular momentum. The system 3.5 consist of two vector equations, which can be divided into six scalar ones. Expressing all terms involved:

$$\begin{cases} F_x = m(\dot{u} + qw - rv) \\ F_y = m(\dot{v} + ru - pw) \\ F_z = m(\dot{w} + pv - qu) \\ L = \dot{p}J_x - \dot{r}J_{xz} - pqJ_{xz} + qr(J_z - J_y) \underline{- J_{yz}(q^2 - r^2) - J_{xy}(\dot{q} - 2p)} \\ M = \dot{q}J_y - pr(J_x - J_z) - (r^2 - p^2)J_{xz} \underline{- J_{xy}(\dot{p} + qr) - J_{yz}(\dot{r} - pq)} \\ N = \dot{r}J_z - \dot{p}J_{xz} + pq(J_y - J_x) + qrJ_{xz} \underline{- J_{xy}(p^2 - q^2) - J_{yz}(\dot{q} + rp)} \end{cases} \tag{3.6}$$

where the underlined terms exist only if the motion is not inside the aircraft plane of symmetry ($J_{xy}, J_{yz} \neq 0$), F_x, F_y, F_z are the forces acting along $x_{body}, y_{body}, z_{body}$ and L, M, N the external moments around the same axes. A deeper explanation can be found in [32]. At this point, let imagine two conditions:

- *Initial condition*: uniform rectilinear motion (symmetric) $\rightarrow p_{eq}, q_{eq}, r_{eq}, v_{eq} = 0$
- *Generic condition*: immediately after a disturbance (e.g. a *doublet*)

For the generic condition:

$$\begin{cases} u = u_{eq} + \Delta u \\ v = \cancel{v_{eq}} + \Delta v \\ w = w_{eq} + \Delta w \\ p = \cancel{p_{eq}} + \Delta p \\ q = \cancel{q_{eq}} + \Delta q \\ r = \cancel{r_{eq}} + \Delta r \end{cases} \tag{3.7}$$

Substituting Eq. 3.7 in Eq. 3.6, if the disturbance is *small*, second-order terms can be neglected obtaining the *linearised small-disturbance equations of motion in* <u>*body axes*</u> (rigid body):

$$\begin{cases} F_x = m(\dot{\Delta u} + qw_{eq}) \\ F_y = m(\dot{\Delta v} + ru_{eq} - pw_{eq}) \\ F_z = m(\dot{\Delta w} - qu_{eq}) \\ L = \dot{p}J_x - \dot{r}J_{xz} \\ M = \dot{q}J_y \\ N = \dot{r}J_z - \dot{p}J_{xz} \end{cases} \tag{3.8}$$

Using the Eq. 3.8 it is possible to study the longitudinal and lateral-directional motion separately: in fact, involved variables are different while all values for the initial condition (equilibrium) are known.

3.1.2 Longitudinal and lateral-directional linearised dynamics

The longitudinal motion of the aircraft is usually studied considering *wind axes*, because it is easier to turn aerodynamics coefficients into equations. Lift L and drag D are parallel to z_{wind} and x_{wind} and trigonometric functions applies only to the thrust T. Variables are, in this case, $[V, \alpha, q]$ where V is the airspeed and α is the angle of attack.

$$\begin{cases} F_x = m\dot{V} \\ F_z = -mV_{eq}(q - \dot{\alpha}) \\ M = \dot{q}J_y \end{cases} \tag{3.9}$$

At this point, forces and moment should be written explicitly. Considering the *small perturbations theory*:

$$\begin{cases} F_x = \cancel{F_{x,eq}} + \Delta F_x \\ F_z = \cancel{F_{z,eq}} + \Delta F_z \\ M = \cancel{M_{eq}} + \Delta M \end{cases} \tag{3.10}$$

where $F_{x,eq}, F_{z,eq}$ and $M_{eq} = 0$ at equilibrium. $\Delta F_x, \Delta F_z$ and ΔM can be calculated considering the difference between an initial and generic condition. Adding the kinematic equation $q = \dot{\theta}$ we obtain the complete set of ***explicit linearised longitudinal equations of motion in wind axes*** ($\alpha_t \rightarrow$ thrust incidence):

$$\begin{cases} \boldsymbol{m\dot{V} = \Delta T \cos(\alpha_t) - T_{eq}\Delta\alpha \sin(\alpha_t) - \Delta D - W(\theta - \Delta\alpha)} \\ \boldsymbol{m(-V_{eq}q + V_{eq}\dot{\alpha}) = -\Delta T \sin(\alpha_t) - T_{eq}\Delta\alpha \cos(\alpha_t) - \Delta L} \\ \boldsymbol{J_y\dot{q} = \frac{\partial M}{\partial V}\Delta V + \frac{\partial M}{\partial \alpha}\Delta\alpha + \frac{\partial q}{\partial V}\Delta q + \frac{\partial M}{\partial \dot{\alpha}}\Delta\dot{\alpha}} \\ \boldsymbol{q = \dot{\theta}} \end{cases} \tag{3.11}$$

Rearranging this equations is possible to write a system in the form $\dot{\boldsymbol{x}} = A_{lon}\boldsymbol{x}$, where A_{lon} is called *plant (longitudinal) matrix* and $\boldsymbol{x} = [V, \alpha, q, \theta]^T$ is the *state (longitudinal) vector*.

The lateral-directional case can be obtained in a similar way, also if the system is written with respect to *body axes* and *virtual moments of inertia* J'_x, J'_z, J'_{xz} are included. The complete set of ***explicit linearised lateral-directional equations of motion in body axes*** is:

$$\begin{cases} \boldsymbol{\dot{v} = \frac{Y_v}{m}v + \frac{Y_p}{m}p + (\frac{Y_r}{m} - V_{eq})r + g\varphi} \\ \boldsymbol{\dot{p} = (\frac{L_v}{J'_x} + J'_{zx}N_v)v + (\frac{L_p}{J'_x} + J'_{zx}N_p)p + (\frac{L_r}{J'_x} + J'_{zx}N_r)r} \\ \boldsymbol{\dot{r} = (\frac{N_v}{J'_z} + J'_{zx}L_v)v + (\frac{N_p}{J'_z} + J'_{zx}L_p)p + (\frac{N_r}{J'_z} + J'_{zx}L_r)r} \\ \boldsymbol{\dot{\varphi} = p} \\ \boldsymbol{\dot{\psi} = r} \end{cases} \tag{3.12}$$

which can be written, as the longitudinal case, in the form $\dot{\boldsymbol{x}} = A_{latdir}\boldsymbol{x}$, where A_{latdir} is called *plant (lateral-directional) matrix* and $\boldsymbol{x} = [v, p, r, \varphi, \psi]^T$ is the *state (lateral-directional) vector*. However, this state vector is not unique: in our thesis, for example, it will be used the *side-slip angle* β instead of the velocity along y_{body}.

3.1.3 State space modelling

The *state-space approach* is one of the most used methods in control theory for Linear Time-Invariant systems (LTI), based on the definition of *state variables* and *state equations*. Generally speaking, the state variables of a system are a minimum set of variables $x_n(t)$ that, when known at time t_0 and along with the input, are sufficient to determine the state of the system at any other time $t > t_0$ [32]. Knowing the aircraft equation of motion (Eq. 3.11 and 3.12) they can be written, as partially mentioned before, in the form:

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{\eta} \\ \boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{\eta} \end{cases} \tag{3.13}$$

where:

- $\boldsymbol{x}$ is the state vector
- $\boldsymbol{\eta}$ is the control vector
- $\boldsymbol{y}$ is the output vector
- $\boldsymbol{A}$ is the plant matrix
- $\boldsymbol{B}$ is the input matrix
- $\boldsymbol{C}$ and $\boldsymbol{D}$ are, in our case, an identity and a null matrix respectively

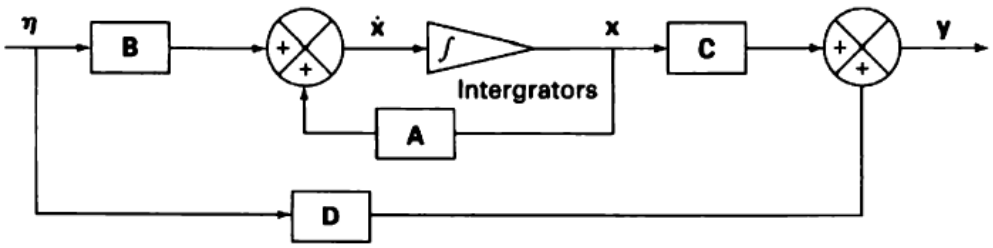


Figure 3.2: Linear state equations (from [31])

A block diagram of 3.13 is represented in Fig. 3.2. If $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ and $\boldsymbol{\eta}$ are known and $\boldsymbol{C} = \boldsymbol{I}$, we can describe how the aircraft behaves over time. It should be specified that all the matrices depend on altitude, centre of gravity and velocity, but for the purposes of this thesis they will remain constant. Moreover, $\boldsymbol{x}$, $\boldsymbol{\eta}$ and $\boldsymbol{y}$ are defined as variation

with respect to the trim conditions: it means that the state vector consist of *increments* $\Delta x_1, \Delta x_2 ..., \Delta x_n$ (where n is the *order of the system*).
For the longitudinal motion, vectors are:

$$\boldsymbol{x}_{lon} = \begin{bmatrix} V \\ \alpha \\ q \\ \theta \end{bmatrix} = \boldsymbol{y}_{lon} \quad \boldsymbol{\eta}_{lon} = \begin{bmatrix} i_H \\ \delta_T \end{bmatrix} \tag{3.14}$$

with i_H = *elevators deflection* and δ_T = *throttle* ($\boldsymbol{C} = \boldsymbol{I}$). In the same way, for the lateral-directional motion:

$$\boldsymbol{x}_{latdir} = \begin{bmatrix} \beta \\ p \\ r \\ \varphi \\ \psi \end{bmatrix} = \boldsymbol{y}_{latdir} \quad \boldsymbol{\eta}_{latdir} = \begin{bmatrix} \delta_A \\ \delta_R \end{bmatrix} \tag{3.15}$$

where δ_A = *ailerons deflection* and δ_R = *rudder deflection*. For the purpose of this thesis all matrices have been provided by *Leonardo Aircraft* and are considered known.

3.2 The aircraft: general characteristics and performances

When designing a cooperative formation control algorithm, the first step is to choose an aircraft model. In general, there could be different types of plant in the same formation (it is reasonable to suppose that the manned leader and the unmanned wingmen are different) but in this thesis, for the sake of simplicity, there will be only a single aircraft model. In particular, *Leonardo Aircraft* has provided matrices of a plant model that will be briefly described in the following lines. The aircraft in question is a generic example of *Unmanned Aerial Vehicle* of type *medium-altitude long-endurance*. The main characteristic of this UAV class is their capacity to fly at an altitude between 10000 ft and 30000 ft ($\simeq 3000 - 9000$ m) for about $24 - 48$ hours. Examples of this type of aircraft are the *MQ-1 Predator* produced by *General Atomics* (Fig. 3.3) and the *Falco Xplorer* produced by *Leonardo* (Fig. 3.4). The geometry has not been provided as the flight physics modelling was sufficient for the purpose of the thesis. Considering the state and control vectors in Eq. 3.14 and Eq. 3.15, matrices for the longitudinal and lateral-directional motions are:

$$
\begin{aligned}
A_{long} &= \begin{bmatrix} 0.0162 & 0.1250 & 0 & 0 \\ -0.0056 & -1.8534 & 0 & 0 \\ 0 & 0 & -1.5413 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\
B_{long} &= \begin{bmatrix} 1.9642 & 5.6405 \\ -0.3100 & 0.0003 \\ -23.2807 & -0.7499 \\ 0 & 0 \end{bmatrix} \\
A_{latdir} &= \begin{bmatrix} -0.1106 & -0.0061 & -0.9915 & 0.1230 \\ -0.8556 & -4.5955 & 1.6498 & 0 \\ 1.2308 & -0.2812 & -0.2675 & -0.0001 \\ 0 & 1.0000 & -0.0046 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\
B_{latdir} &= \begin{bmatrix} 0.0015 & -0.0298 \\ 24.7178 & 0.5901 \\ -0.3433 & -0.8961 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}
\end{aligned}
\tag{3.16}
$$

Figure 3.3: The *MQ-1 Predator* UAV

All the elements of the matrices are dimensional and units of measurement are the "International System of Units" ones, with m for length, $\frac{m}{s}$ for velocities, rad for angles and $\frac{rad}{s}$ for angular velocities. As mentioned before, the plant matrix A and the input matrix B change with altitude, centre of gravity and velocity and, consequently, the dynamic behaviour of the vehicle varies in the flight envelope. In this dissertation, however, this phenomenon has not been considered because it would have complicated the simulink code and it has not been considered important for the purpose of the study. What is extremely important to know are the *trim conditions* (Tab. 3.1) to which

Figure 3.4: The *Leonardo "Falco Xplorer"* UAV at Paris Air Show 2019

matrices refer. In fact, once the state vector (which contains *increments*) has been calculated, this conditions should be added in order to obtain absolute values.

V_{trim} $(\frac{m}{s})$	H_{trim} (ft)	α_{trim} (deg)	θ_{trim} (deg)	$i_{H,\,trim}$ (deg)	$\delta_{A,\,trim}$ (deg)	$\delta_{R,\,trim}$ (deg)	T_{trim} (kg)
82.3111	10000	$\simeq 3.5$	$\simeq 3.5$	$\simeq 0.5$	0	0	$\simeq 600$

Table 3.1: UAV trim conditions

To build the plant model, it was necessary to know some characteristic and performance of the UAV in order, for example, to limit the maximum deflection of mobile surfaces or to guarantee the respect of the flight envelope. This values will be used mainly in *saturation blocks* inside PIDs or LQRs (*Linear-Quadratic Regulator*).

3.3 The Simulink model of the aircraft

Once the aircraft has been chosen, the following step consists in trying to simulate its dynamic behaviour. The cooperative formation control system generates high - level commands which guarantees the separation between leader, followers and obstacles. As we will see, the nature of these commands and the type of variable which APF controls could be different on a case-to-case basis and their choice is made during the design phase. However, the output that the formation control system generates must be converted in variables which the state-space system can accept. There the *aircraft Simulink model* will be described in order to understand how concepts from the previous sections have been applied to the code.

First of all, it would be necessary to draw attention on the *solver*. MATLAB Simulink basically solves a system of equations implicitly built by the user through a block diagram interface. During the initial design phase no changes were applied to the default options (*"Variable-step"* solver of automatic selected type) but, when including more than two

$\boldsymbol{V}$ $(\frac{m}{s})$	$\boldsymbol{\Delta\delta_A}$ (deg)	$\boldsymbol{\Delta i_H}$ (deg)	$\boldsymbol{\Delta\delta_R}$ (deg)	$\boldsymbol{H_{max}}$ (ft)	$\boldsymbol{\dot{h}}$ $(\frac{m}{s})$	$\boldsymbol{T_{max}}$ (kg)	$\boldsymbol{\varphi_{max}}$ (deg)
$[60-140]$	± 30	± 20	± 20	25000	± 5.08	$\simeq 2000$	± 20

Table 3.2: UAV performances and mobile surfaces deflection limits

followers controlled by the *Artificial Potential Field* (Ch.6 and Ch.7) at the hearth of this thesis, computational performance became very poor. For this reason it has been decided to use a discrete solver with option in Tab. 3.3.

Simulation time	**Solver Selection**		**Solver Details**
	Type	Solver	
Depends on scenario	*Fixed-step*	*Discrete*	$dt = \frac{1}{100}$ *sec*

Table 3.3: Simulink solver options

In Fig. 3.7 the overview of the plant model is depicted. Inputs of the block are the components of control vectors $\boldsymbol{\eta}_{lon}$ (Eq. 3.14) and $\boldsymbol{\eta}_{latdir}$ (Eq. 3.15) deriving from mobile surfaces block. They enter in *plant_long* and *plant_latdir* subsystems, which include the state-space equations reproduced as Fig. 3.5.

The initial conditions inside the *Discrete Integrator* (for both longitudinal and lateral-directional blocks) are all equal to zero: we want the aircraft trimmed at $t = t_0$. $\boldsymbol{C} = \boldsymbol{I}$ and $\boldsymbol{D} = \boldsymbol{0}$ have not been considered because they do not affect the simulation. In fact, having chosen a $\boldsymbol{C}$ matrix equal to $\boldsymbol{I}$, the output corresponds to the entire state vector, divided in our case in longitudinal $\boldsymbol{y}_{lon} = \boldsymbol{x}_{lon}$ and lateral-directional $\boldsymbol{y}_{latdir} = \boldsymbol{x}_{latdir}$ components. We want to remember that this motion separation is possible thanks to the *small-perturbations* hypothesis applied to linearised dynamics equations. The trim conditions should be added to the output of *plant_long* and *plant_latdir* in order to obtain absolute values. The *flight path angle* γ has been calculated under the assumption that the aerodynamics characteristic of the vehicle never changes during the flight (the angle between the $C_L = 0$ direction and x_{body} do not vary):

$$\Delta\gamma = \Delta\alpha - \Delta\theta \tag{3.17}$$

Integrating the angular velocity r the *yaw angle* ψ could be archived. The initial condition for this discrete integrator depends on the desired initial *heading angle*. This comes from the fact that when building the *Heading Autopilot* in Sect. 3.5, the yaw angle and the heading angle have been considered coincident.

Thanks to the *navigation equations* (Fig. 3.6), it is possible to calculate positions and velocities in NED frame. Being z_{NED} pointed toward the centre of the Earth, changing its sign the altitude h is obtained. First of all, we need velocities in body frame $\boldsymbol{V}_{body} = [u, v, w]^T$ considering that the longitudinal state-space equations originate from force and moments equilibrium in wind frame. The relations between this coordinate systems (Eq.

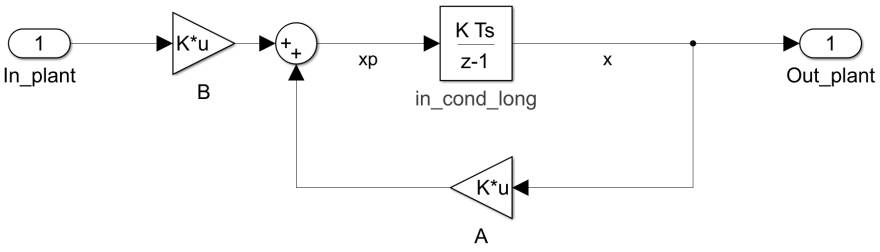


Figure 3.5: The longitudinal state equations block

3.18) pass through the angle of attack α, the side-slip angle β and the *airspeed* V, that is the velocity vector to which x_{wind} is parallel.

$$\begin{cases} u = V\cos(\alpha)\cos(\beta) \\ v = V\sin(\beta) \\ w = V\sin(\alpha)\cos(\beta) \end{cases} \tag{3.18}$$

At the same time the rotation matrix between NED and body (ABC) frame is generated (Eq. 3.1), transposed and multiplied with $\boldsymbol{V}_{body}$, yielding to the *velocity vector in NED frame* $\boldsymbol{V}_{NED}$:

$$\boldsymbol{V}_{NED} = \begin{bmatrix} V_{NORD} \\ V_{EAST} \\ V_{DOWN} \end{bmatrix} = \boldsymbol{R}_{ABC \to NED} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{3.19}$$

Integrating the NED velocities it is possible to obtain the position vector in the inertial frame $\boldsymbol{P}_{NED}$ that will be used for formation control as well as for visualization purpose. The initial conditions for all the *Discrete Integrator* involved are extremely important because they represent the *initial position* $\boldsymbol{P}_{NED,0}$ *of the aircraft in NED frame*. When considering multiple vehicles, different values must be selected in order to avoid overlapping.

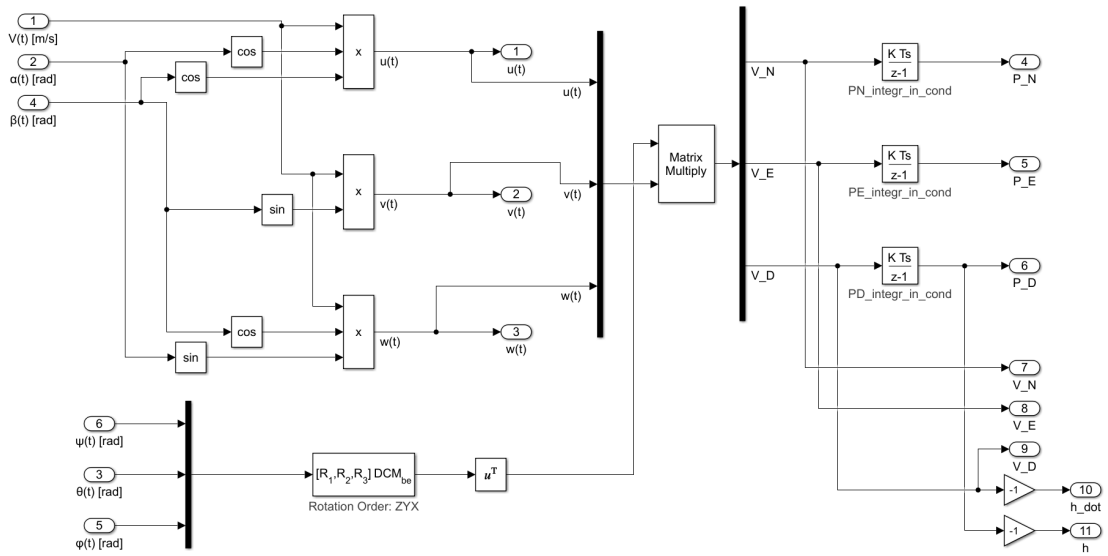


Figure 3.6: The navigation equations

3.4 Control surfaces

The inputs of the plant are essentially the aircraft controls, that are throttle and deflections of the mobile surfaces of the aircraft expressed in radians. Such deflections result from a control system that calculates them based on a "desired" that comes, in our case, from the *Autopilots* block. It is therefore necessary to design these control laws, building the block *Flight surfaces control and autothrottle*. As can be seen from Fig. 3.8, inputs of the subsystem are essentially two: on the one hand there are the autopilot control and on the other the plant data, useful for the calculation of incoming errors from the various control algorithms. As we will see, only for the leader aircraft the autopilot command can be substituted with the *joystick* ones, so that the user can directly control its trajectory.

3.4.1 *Linear Quadratic Regulator* (LQR) fundamentals

Among the various control algorithms present in literature it was decided to use an *LQR (Linear Quadratic Regulator)* for mobile surfaces management as it is simple to implement as well as very effective. The theory of optimal control is concerned with operating a generic dynamic system minimizing a cost function. The history of optimal problems starts with Galileo Galilei and his geometrical problems. Studying the motion of a body under gravity, he was able to calculate the time of descent of an object falling along an inclined plane. Assuming that the shortest path from a point A to a point B is a straight line, he asked himself if it was also the one that would have taken the shortest time. In 1696 Johann Bernoulli resumed the dilemma and challenged *"the most acute mathematicians of the entire world"* to solve *the Brachistochrone Problem* [33]:

"Given two points on a plane at different heights, what is the shape of the wire

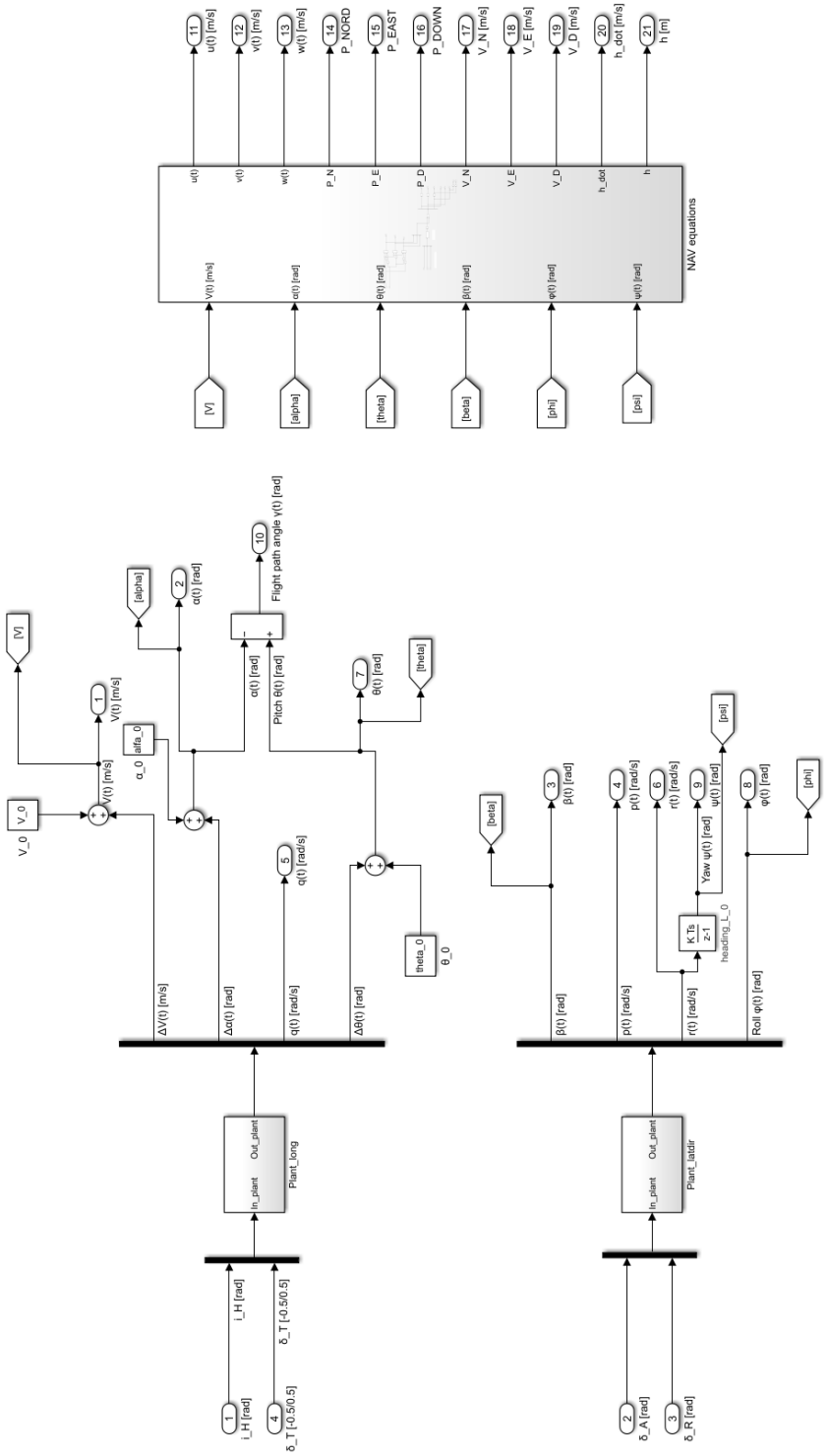


Figure 3.7: The plant model overview

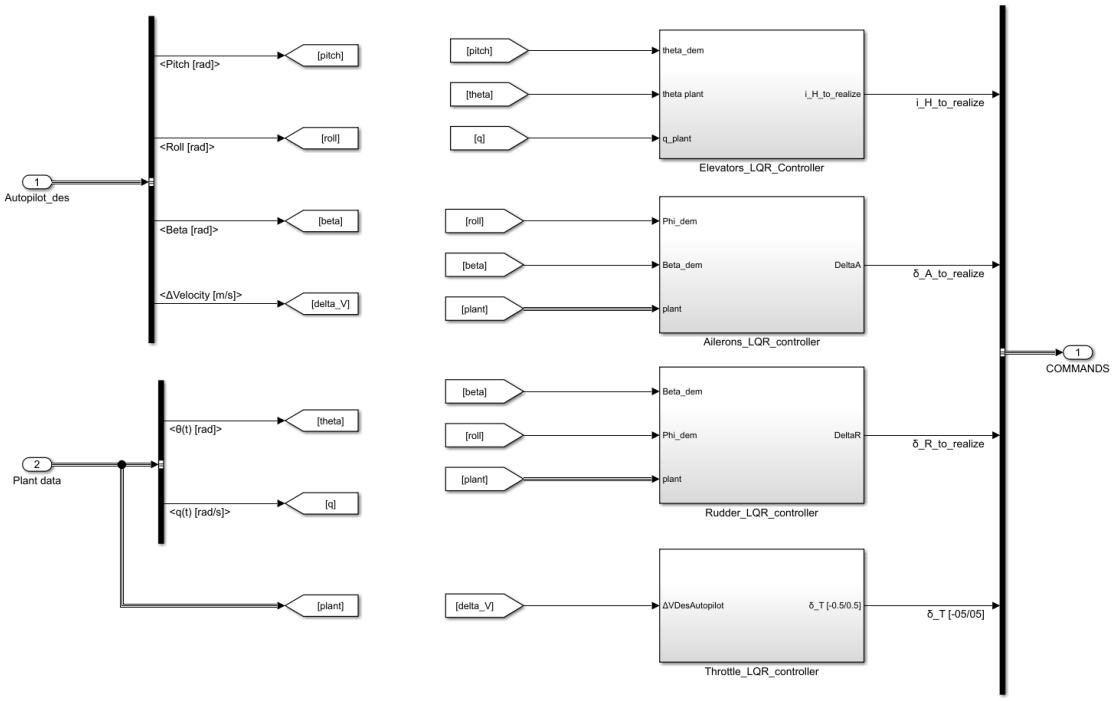


Figure 3.8: Flight surfaces control and autothrottle block

down which a bead will slide (without friction) under the influence of gravity so as to pass from the upper point to the lower point in the shortest amount of time?"

The solution is the cycloid: this type of challenge aroused interest in this type of problems. One of the first aerospace applications was released by A.E. Bryson [34] and was related to the development of an optimization algorithm which calculate the fastest ascent trajectory from a start to an end point.

Mathematically speaking, the *Linear Quadratic Regulator* is based on the calculation of a performance index J_∞ which governs the performance of the closed-loop system:

$$J_\infty = \int_0^\infty (\boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{\eta}^T \boldsymbol{R} \boldsymbol{\eta}) dt \tag{3.20}$$

where $\boldsymbol{x}$ is the state vector and $\boldsymbol{\eta}$ = the control/input vector. As we can see, the quadratic terms involve two weighting matrices:

- $\boldsymbol{Q}$ is positive semidefinite and penalizes the states
- $\boldsymbol{R}$ is positive definite and penalizes the input

The user has the possibility to tune the control algorithm changing $\boldsymbol{Q}$ and $\boldsymbol{R}$, giving more importance to a vector over the other. To find the solution to the *optimal control problem*,

the input $\boldsymbol{\eta}$ that minimize J_∞ subject to the constraint $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{\eta}$ should be found. It should be remembered that this control algorithm applies on *full-state feedback loops*, for which $\boldsymbol{x}$ is completely measurable and this relation is valid:

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{\eta} \\ \boldsymbol{\eta} = \boldsymbol{C}\boldsymbol{x} \end{cases} \tag{3.21}$$

The optimal control law consists in finding K_{LQR} given by:

$$K_{LQR} = -R^{-1}B^T P \tag{3.22}$$

where the P matrix is the solution of the *Algebraic Riccati Equation*:

$$A^T P + PA - PBR^{-1}B^T P = -Q \tag{3.23}$$

For the purpose of our thesis the MATLAB *lqr* command has been used, giving in input the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. Images of the Simulink model for all the control blocks related to mobile surfaces can be found in Appendix A.

3.4.2 Elevators and throttle control algorithms

For the longitudinal part of the code two different LQR have been designed, one for the elevators and one for the throttle. The separation of the control loops acting on the same dynamic model is advisable because of the very weak correlation between them, so there is no real gain in optimizing them by building an overall control law. In order to tune the various gains inside $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices a Simulink model has been created in which the plant is subjected to a *Step* signal. In so doing, the response of the system over time can be observed and modified according to the necessity. *Saturation* blocks are present (see Sect. 3.2) together with *Discrete Integrator* and for this reason it was necessary to control the *"windup"*. Characteristic of systems in which output is limited to a specific interval, it is due to the presence of the integrator. The control system does not perceive the *Saturation* block because it is inserted directly inside the output line and for this reason it generates a not-limited command, depending on the input error. If this command exceeds the saturation limits the output becomes a constant signal and the system, which is of closed-loop type, behaves like an *open-loop* one. The *Discrete Integrator*, in this case, continues to integrate the error and generates an ever-increasing control request making the system unstable. This phenomenon is mitigated using of an *"antiWindUp"* block which excludes the integration line once saturation limits have been reached.

Let's start to describe the procedure for designing the longitudinal LQRs. First of all, matrices $\boldsymbol{A}_{long}$ and $\boldsymbol{B}_{long}$ should be known (Sect. 3.2) so that an elementary plant block as Fig. 3.5 can be created. The MATLAB *lqr* command accepts the following syntax:

$$[K_{LQR}, S, e] = lqr(\widetilde{A}, \widetilde{B}, Q, R)$$

where $\boldsymbol{K}_{LQR}$ is the matrix we are interested in. $\widetilde{\boldsymbol{A}}$ and $\widetilde{\boldsymbol{B}}$ depend on the system and $\boldsymbol{Q}$ and $\boldsymbol{R}$ should be chosen by the user. Actually, $\widetilde{\boldsymbol{A}}$ and $\widetilde{\boldsymbol{B}}$ could not be exactly equal to $\boldsymbol{A}_{long}$ and $\boldsymbol{B}_{long}$ (or $\boldsymbol{A}_{latdir}$ and $\boldsymbol{B}_{latdir}$) but they could be modified, for example, adding

integrals of the values of the state vector $\boldsymbol{x}_{long}$ (or $\boldsymbol{x}_{latdir}$). Experience in control theory and the degree of knowledge of the aircraft characteristics lead the user to choose the correct $\widetilde{\boldsymbol{A}}$ and $\widetilde{\boldsymbol{B}}$ for the LQR. In our case, it has been decided to control *elevators* using errors of q, θ and $\int \theta$:

$$\widetilde{\boldsymbol{A}}_{i_H} = \begin{bmatrix} -1.5413 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \widetilde{\boldsymbol{B}}_{i_H} = \begin{bmatrix} -23.2807 \\ 0 \\ 0 \end{bmatrix} \quad \text{with:} \quad \widetilde{\boldsymbol{x}}_{i_H} = \begin{bmatrix} \Delta q \\ \Delta\theta \\ \Delta(\int \theta) \end{bmatrix} \quad \widetilde{\boldsymbol{\eta}}_{i_H} = [i_H] \tag{3.24}$$

Through a *trial-and-error* procedure, knowing how $\boldsymbol{Q}_{long,i_H}$ and $\boldsymbol{R}_{long,i_H}$ penalize states or inputs, the following matrices have been chosen. In Fig. 3.9 the response of the elevator LQR is depicted.

$$\boldsymbol{Q}_{long,i_H} = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \quad \boldsymbol{R}_{long,i_H} = [0.06] \quad \boldsymbol{K}_{long,i_H} = \begin{bmatrix} -0.6604 \\ -4.1545 \\ -0.4082 \end{bmatrix} \tag{3.25}$$

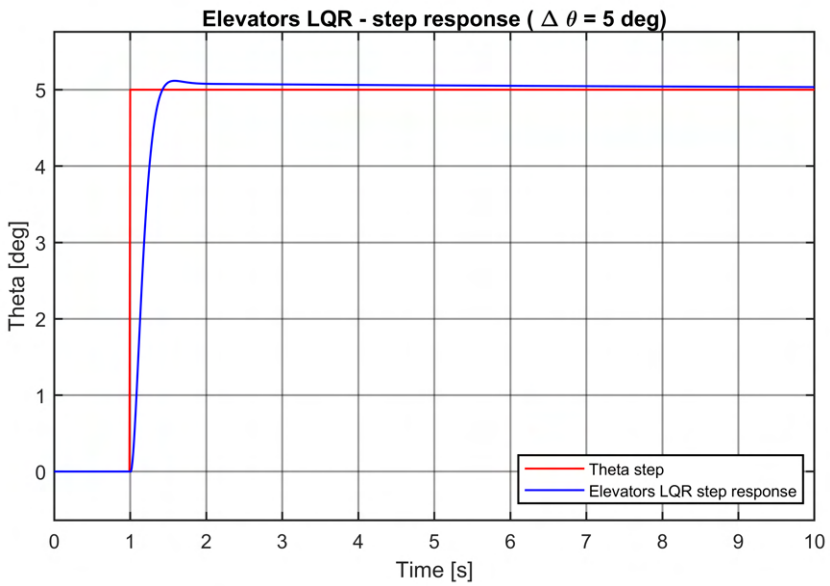


Figure 3.9: Elevators LQR step response

The *throttle* is controlled by means of a ΔV required by the *Autopilot* block, which is transformed into a value between $[-0.5; 0.5]$ (remember that the trim condition is $\delta_{Th,0} = 0.5$, see Tab. 3.1). After a few trials it was found that the better results were obtained using the velocity error ΔV and its integral $\int \Delta V$ inside $\widetilde{\boldsymbol{A}}_{Th}$. In Fig. 3.10 the response of the throttle LQR is depicted.

$$\widetilde{\boldsymbol{A}}_{Th} = \begin{bmatrix} 0.0162 & 0 \\ 1 & 0 \end{bmatrix} \quad \widetilde{\boldsymbol{B}}_{Th} = \begin{bmatrix} 5.6405 \\ 0 \end{bmatrix} \quad \text{with:} \quad \widetilde{\boldsymbol{x}}_{Th} = \begin{bmatrix} \Delta V \\ \Delta(\int V) \end{bmatrix} \quad \widetilde{\boldsymbol{\eta}}_{Th} = [\delta_{Th}] \tag{3.26}$$

The $\boldsymbol{Q}_{long,Th}$ and $\boldsymbol{R}_{long,Th}$ matrices, together with $\boldsymbol{K}_{long,Th}$, are:

$$\boldsymbol{Q}_{long,Th} = \begin{bmatrix} 1 & 0 \\ 0 & 0.01 \end{bmatrix} \quad \boldsymbol{R}_{long,Th} = [0.06] \quad \boldsymbol{K}_{long,Th} = \begin{bmatrix} 4.1031 \\ 0.4082 \end{bmatrix} \tag{3.27}$$

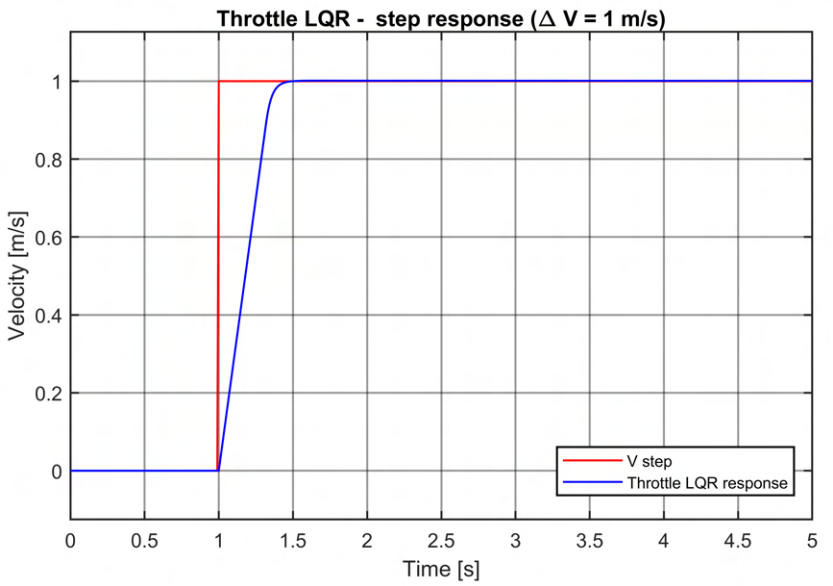


Figure 3.10: Throttle LQR step response

3.4.3 Ailerons and rudder control algorithms

As far as lateral-directional control is concerned, the following procedure is slightly different from the previous one. The *rudder* deflection causes an unwanted roll motion, which must be corrected. Designing the ailerons and rudder LQRs separately this effect would have not been considered, making the aircraft unstable. The adopted solution is to use a control system that synergistically involves both mobile surfaces. The LQR state vector $\widetilde{\boldsymbol{x}}_{latdir}$ is composed by all the lateral-directional state variables to which $\int \beta$ and $\int \varphi$ have been added. The $\boldsymbol{K}_{LQR,A}$ and $\boldsymbol{K}_{LQR,R}$ matrices have been calculated using a single *lqr* function. It should be noted that, for reasons due to the *Autopilot* block design (which will be clarified later), the directional motion is controlled through the sideslip angle β (instead of the yaw angle ψ).

$$\widetilde{\boldsymbol{A}}_{latdir} = \begin{bmatrix} -0.1106 & -0.0061 & -0.9915 & 0.1230 & 0 & 0 \\ -0.8556 & -4.5955 & 1.6498 & 0 & 0 & 0 \\ 1.2308 & -0.2812 & -0.2675 & -0.0001 & 0 & 0 \\ 0 & 1 & -0.0046 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\widetilde{\boldsymbol{B}}_{latdir} = \begin{bmatrix} 0.0015 & -0.0298 \\ 24.7178 & 0.5901 \\ -0.3433 & -0.8961 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{3.28}$$

$$\widetilde{\boldsymbol{x}}_{latdir} = \begin{bmatrix} \beta \\ p \\ r \\ \varphi \\ \int \beta \\ \int \varphi \end{bmatrix} \quad \widetilde{\boldsymbol{\eta}}_{latdir} = \begin{bmatrix} \delta_A \\ \delta_R \end{bmatrix}$$

The $\boldsymbol{Q}_{latdir}$ and $\boldsymbol{R}_{latdir}$ matrices, together with $\boldsymbol{K}_{latdir}$, are:

$$\boldsymbol{Q}_{latdir} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix} \boldsymbol{R}_{latdir} = \begin{bmatrix} 1 & 1 \end{bmatrix} \boldsymbol{K}_{latdir} = \begin{bmatrix} 0.4553 & 0.9682 \\ 0.8809 & 0.0099 \\ -0.2980 & -1.5343 \\ 1.3132 & -0.1620 \\ 0.2538 & 0.9672 \\ 0.3059 & -0.0803 \end{bmatrix} \tag{3.29}$$

3.5 Autopilots

The input of the *"Flight surfaces control and autothrottle"* block is a vector composed by four elements:

$$AP_{out} = \begin{bmatrix} \Delta\theta_{AP\_out} & \Delta\varphi_{AP\_out} & \Delta\beta_{AP\_out} & \Delta V_{AP\_out} \end{bmatrix} \tag{3.30}$$

This vector includes the variations of all the quantities that the aircraft needs to perform a certain manoeuvrer. These quantities are produced by the *"Autopilots"* subsystem, which we will briefly describe in this section. There are various autopilots in literature whose implementation depends, among other things, by the technological level of the vehicle. In our case we have chosen to implement:

- *"Altitude/Vertical velocity autopilot"*
- *"Heading autopilot"*

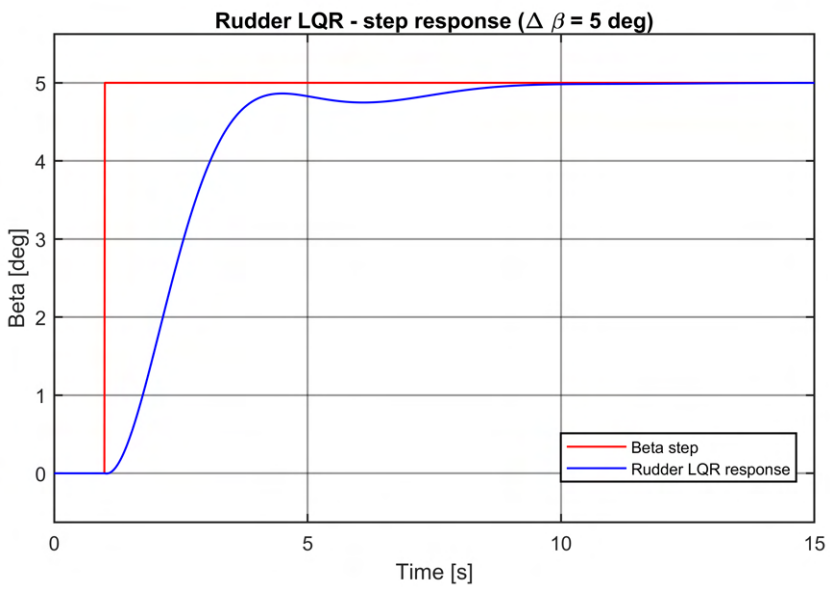


Figure 3.11: Rudder LQR step response

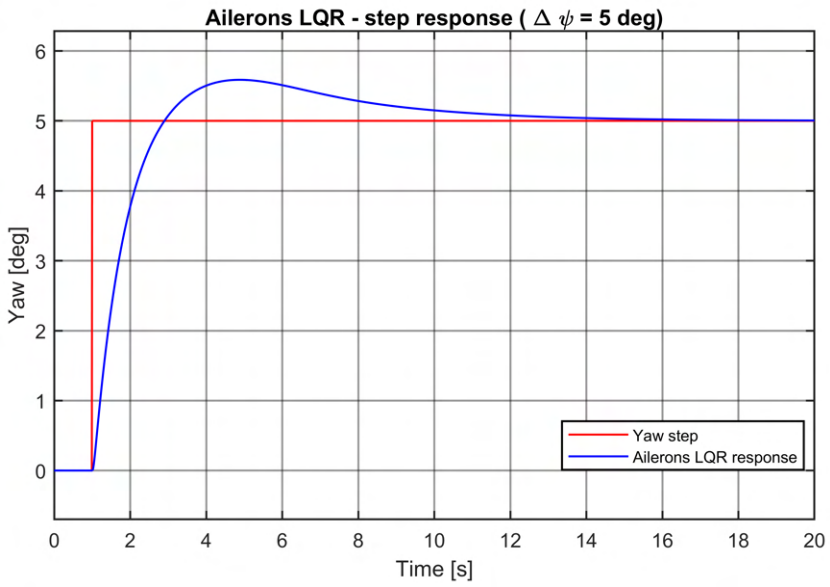
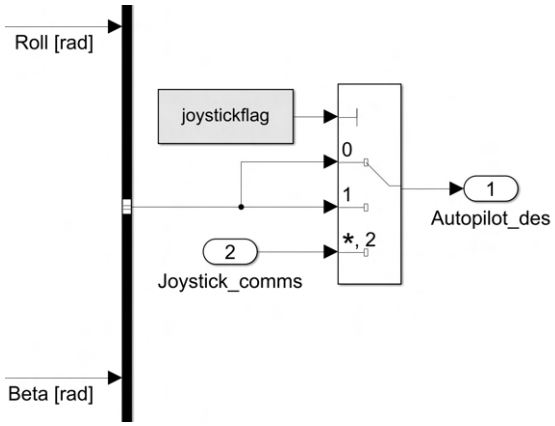


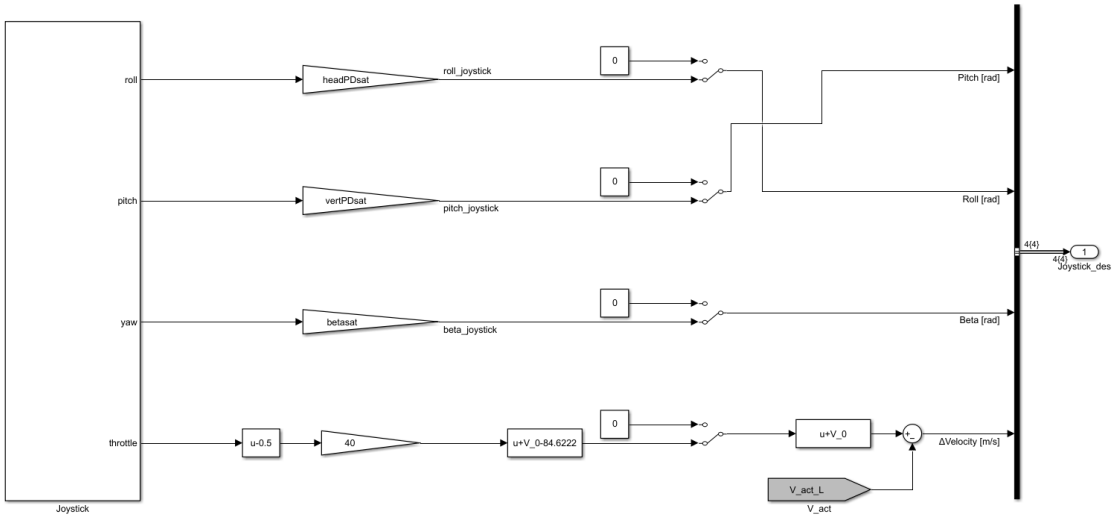
Figure 3.12: Ailerons LQR step response

The outputs of this control blocks are, respectively, $\Delta\theta_{AP\_out}$ and $\Delta\varphi_{AP\_out}$. The β_{AP_out} has been set to zero because it has been considered that the aircraft can perform only *right turns*. It has not been necessary to implement an additional algorithm for the velocity because the control logic for the manoeuvres (which will be described later) produces a ΔV that can be directly used in the *throttle LQR controller*. In Fig. A.5

and Fig. A.6 in Appendix A a overall view of the Simulink subsystem is represented for formation control using PID and APF.
Rather than using the autopilots, the leader aircraft can be directly controlled by the user, who can activate a *"User-controlled leader mode"* changing the value of the variable *joystickflag* inside the initialization code (Fig. 3.13a). In this case the output values in Eq. 3.30 do not originate from PIDs but they are generated using a joystick. The *Pilot joystick* MATLAB block produce normalized outputs, which should be handled using saturation values for pitch, roll, beta and velocity (Fig. 3.13b).



(a) Joystick switch in *Autopilots* subsystem



(b) Joystick subsystem

Figure 3.13: Joystick implementation for leader control

3.5.1 *Proportional Integral Derivative (PID)* fundamentals

For the *Autopilot* block, Proportional-Integral-Derivative (PID) controllers have been employed. The science of *continuous control* has one of its origins in the 17th century with the "centrifugal governor" invented by Christiaan Huygens. In the following centuries, a number of distinguished researchers tried to understand how to control and stabilize a mechanical system: some examples are J.C. Maxwell, J. Watt and E. Routh. However, only in 1922 a formal definition of a PID algorithm was published by engineer N. Minorsky [35] and PID controller was used for automatic ship steering. In 1930 this theory was used in electromechanical actuators while in 1970 PID was at the foundation of F-16 autopilots.

The PID mathematical theory is based on three "versions" of the error: the *error itself* $e(t)$, its *derivative* $\dot{e(t)}$ and its *integral* $\int e(t)$. In so doing, we are capable to obtain information about its evolution in the present, in the past and predict how it could reasonably vary in the future.

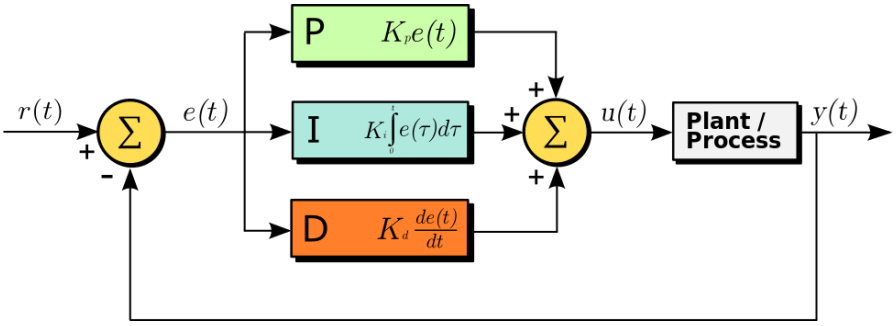


Figure 3.14: PID controller (from [36])

$$u(t) = K_P\, e(t) + K_I \int_0^t e(t)\, dt + K_D\, \dot{e(t)} \tag{3.31}$$

The action of every gain could be explained in this way [37]:

- The proportional term acts on the rise time but the steady state error is invariant
- The integrative term erases the steady state error but worsen the transitory response
- The derivative term decreases the overshoot improving the transitory term, reducing anyway stability robustness

Steady state error is a general way to describe an unwanted effect of one perturbation acting on the system. A good example is crosswind forcing the aircraft to constantly slide with respect to its longitudinal trajectory. A balance of these three effects is one of the main fundamentals of control system design. As for the LQR controller, gain values can be obtained through a *trial-and-error* procedure, observing the system response and changing them until it is considered satisfying. In this thesis, however, we had to face a difficult problem. The plant of *"Flight surfaces control and autothrottle"* LQRs was

the aircraft state space model, so they could be tuned quite easily. The *Autopilot* PIDs belong to an outer loop so their "plant" include, besides the aircraft, the mobile surfaces controllers. For this reason, it has been decided to use the MATLAB *PID Tuner App* to tune every autopilot and insert the calculated gains inside the PID subsystems. For each control loop, it has been selected the most adequate control structure, sometimes avoiding the use of Integral path if steady state error effects are negligible.

3.5.2 *Altitude/Vertical velocity* and *Heading* autopilots

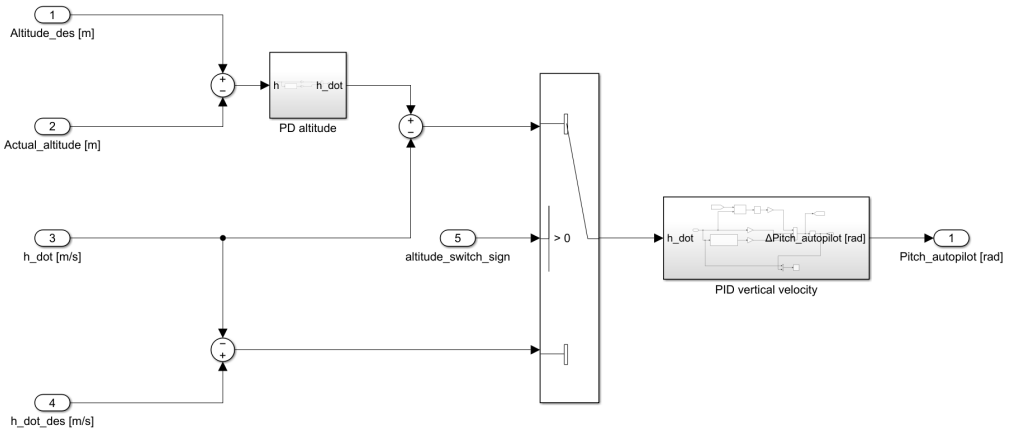


Figure 3.15: Altitude/Vertical velocity autopilot

The *Altitude/Vertical velocity autopilot* is represented in Fig. 3.15 and is composed of a "PD altitude" controller and a "PID vertical velocity" controller. The user can select, using a *Switch* and a *altitude_switch_sign* flag, to control the aircraft with a desired altitude or a desired vertical velocity. Outputs of both controllers are, respectively, $\Delta\dot{h}_{des}$ and $\Delta\theta_{des}$. This values are limited using *Saturation* blocks (Tab. 3.4).

$\Delta\dot{h}\,(\frac{m}{s})$		$\Delta\theta\,(deg)$	
Leader	Follower	Leader	Follower
5.08	5.08	4	5

Table 3.4: *Altitude/Vertical velocity autopilot* saturations

Gains (calculated using MATLAB *PID Tuner App*) are included in Tab. 3.5, step responses in Fig. 3.16 and Fig. 3.17 while an overview of the blocks can be found in Appendix A.

The *Heading autopilot* is represented in Fig. 3.18. As previously described for the *PD altitude* block, only proportional and derivative gains are present. In this case the input is the difference between the desired heading angle and the actual yaw angle (heading and yaw are considered identical) and the output is limited between $\pm 15\,deg$. Using the

PD altitude	*PID vertical velocity*
$P\_alt = 0.687476326769988$	$P\_clvel = 0.0162847317690211$
$(-)$	$I\_clvel = 0.030029471845295$
$D\_alt = 0.385200077951434$	$D\_clvel = 0.00153832455761427$

Table 3.5: *Altitude/Vertical velocity autopilot* gains

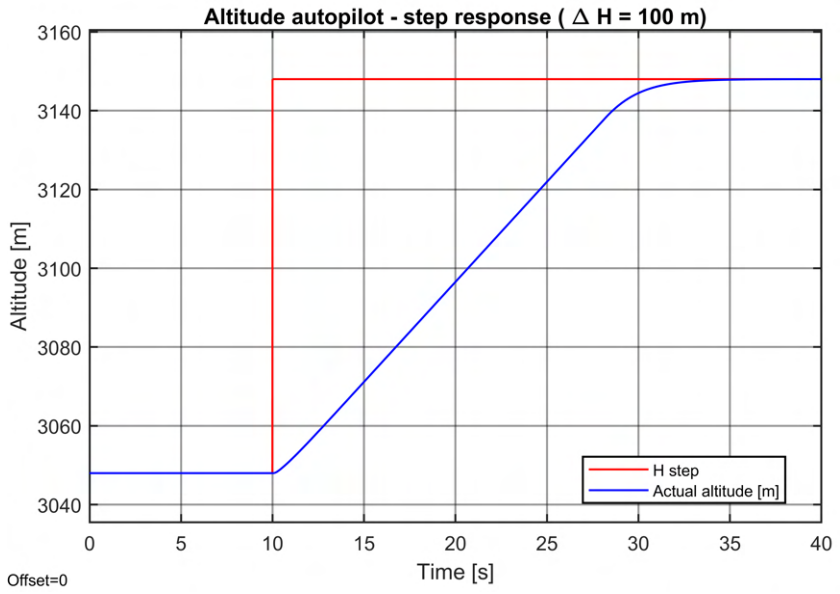


Figure 3.16: *PD altitude* step response

MATLAB *PID Tuner App* gains in Tab. 3.6 have been obtained, the step response is depicted in Fig. 3.18 while an overview of *PD roll* block is included in Appendix A.

Once the autopilots were designed, it was necessary to build a control logic that would allow them to be used to the fullest. A MATLAB function has been written and inserted in *Autopilot choice* block. The necessary data requested by autopilots are:

- h_{des}
- $\dot{h}_{des}$
- $altitude\_switch\_sign$
- $heading_{des}$
- β_{des}
- V_{des}

Using a *Switch* function, nine different *"fundamental flight modes"* have been considered:

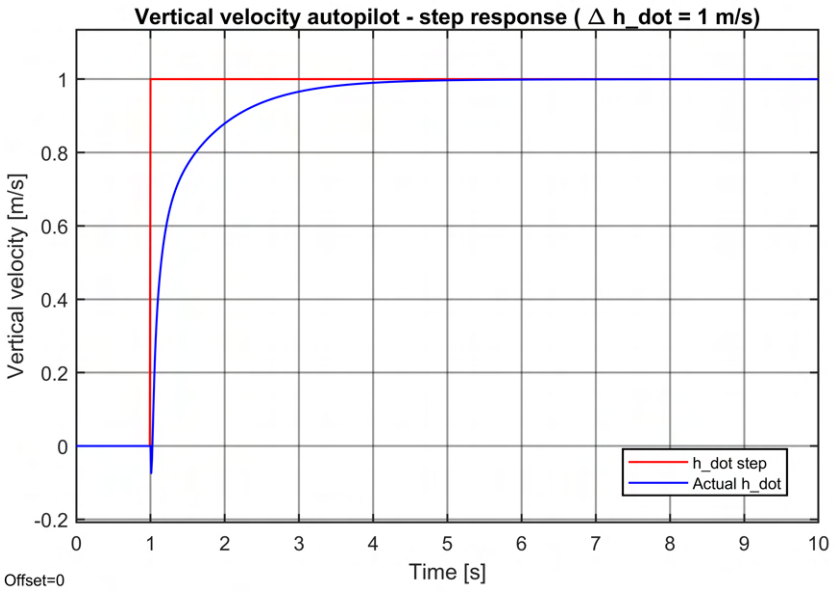


Figure 3.17: *PID vertical velocity* step response

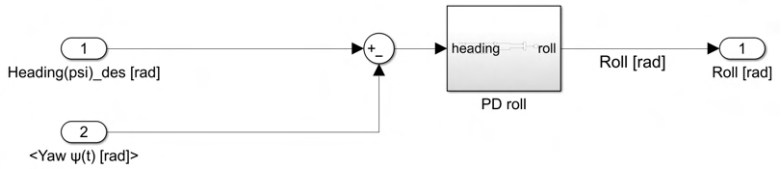


Figure 3.18: *Heading autopilot* block

- Reach the desired altitude ($ACflag = 1$)
- Maintain the desired altitude ($ACflag = 1.5$)
- Reach the desired vertical velocity ($ACflag = 2$)
- Reach the desired heading angle with constant altitude ($ACflag = 3$)
- Reach the desired heading angle with variable altitude ($ACflag = 4$)
- β changing with constant altitude ($ACflag = 5$)
- β changing with variable altitude ($ACflag = 6$)
- Reach the desired velocity with constant altitude ($ACflag = 7$)
- Reach the desired velocity with variable altitude ($ACflag = 8$)

Combining this flight modes, the user can create complex *manoeuvres*. The complete *Autopilot choice* code for a *follower* aircraft is available in Appendix B. A thing should

PD roll
$P\_head = 1.6592310034488$
$D\_head = 1.02787603790321$

Table 3.6: *Heading autopilot* gains

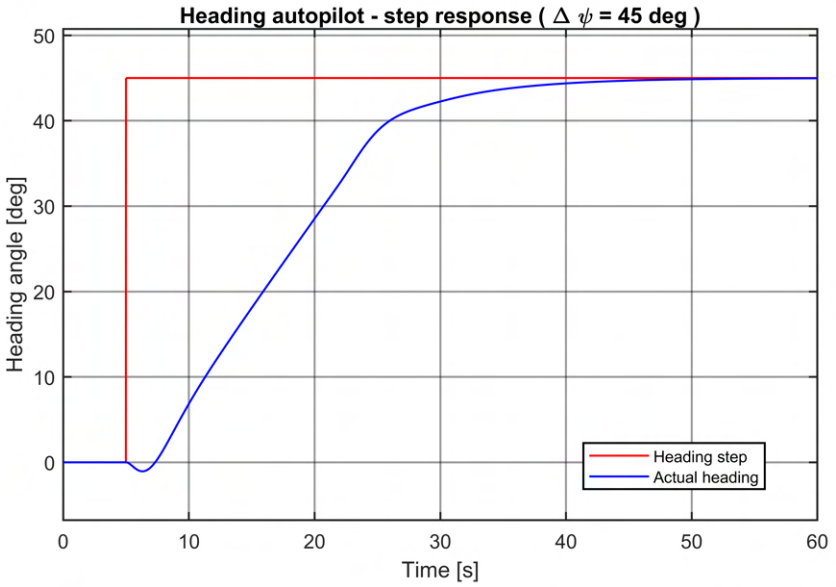


Figure 3.19: *Heading autopilot* step response

be clarified, that is the meaning of the *FormACTIVE* flag. As we will see, every follower in the formation can operate in two ways:

- *"Autonomous flight" mode*: the follower aircraft execute the manoeuvre independently. The formation control algorithm (PID or APF-based) is turned off and collision-avoidance is not guaranteed. ($FormACTIVE = 0$)
- *"Formation flight" mode*: the follower aircraft flights in formation. It ignores the ordered manoeuvre and gets in a defined position around the leader. This position can be user-selected and different *formation shapes* can be built. ($FormACTIVE = 1$)

3.6 Manoeuvres logic and visualization

The *Autopilot* block needs a specific vector of data to execute one of the fundamental flight modes. Moreover, if more than one fundamental flight mode should be used during a single simulation (to produce a complete manoeuvre), different vectors at specific instants should be sent to the subsystem. In this thesis it has been decided to adopt a programmatic

approach using the MATLAB *From Workspace* block. In the initialization code (thanks to a *Switch*) a predefined manoeuvre is selected, producing a *manoeuvre* vector with the following syntax:

$$manoeuvre = \begin{bmatrix} t_{start1} & APflag_1 & alt_{c,1} & \dot{h}_{c,1} & yaw_{c,1} & beta_{c,1} & V_{c,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ t_{startN} & APflag_N & alt_{c,N} & \dot{h}_{c,N} & yaw_{c,N} & beta_{c,N} & V_{c,N} \end{bmatrix} \tag{3.32}$$

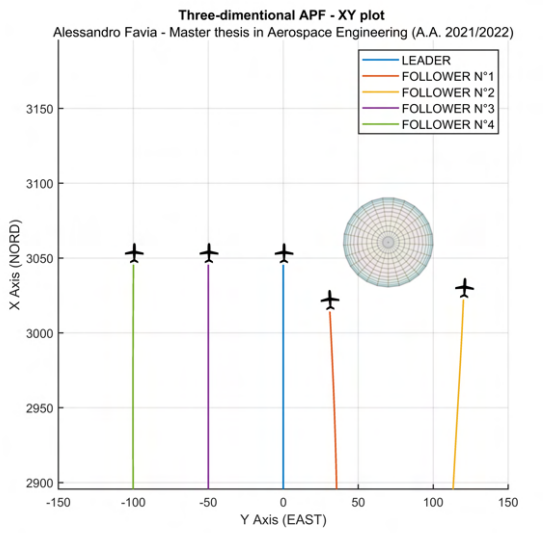
The following code is an example of how a manoeuvre could be built. The first part corresponds to a straight flight with constant altitude. In this case $APflag = 1$ and the *Altitude autopilot* is selected. Being *alt_c* ("altitudine_comandata", desidered altitude) equal to $H0$ (that is H at $t = 0\,sec$) nothing changes. The other values (*h_dot_c*, *yaw_c*...) are useless because they are not read by the model (see *Autopilot switch* in Appendix A). The first string $c1$ is produced. At $t_2 = 25\,sec$ (line 21) the second manoeuvre is activated. For the *From Workspace* block, in fact, the first value of the string is the time at which the same string is passed to the Simulink model. Differently then the first one, the second manoeuvre use the "fundamental flight mode" number four ($APflag = 4$) and the useful variables are $h\_dot\_c = 2\,\frac{m}{s}$ and $yaw\_c = 130\,deg$. The second string is $c2$ and, combined with $c1$, produces the vector *manoeuvre*.

```
1   case 'manovra5'
2       % Straight flight
3       APflag=1;
4       alt_c=H0; % [m]
5       h_dot_c=0; % useless
6       yaw_c=0; % useless
7       beta_c=0; % useless
8       V_c=0; % useless
9       c1=[APflag alt_c h_dot_c yaw_c beta_c V_c];
10      t1=0; % [s]
11      m1=[t1 c1];
12
13      % Turn with h_dot= 2 m/s
14      APflag=4;
15      alt_c=0; % useless
16      h_dot_c=2;
17      yaw_c=deg2rad(130);
18      beta_c=0; % useless
19      V_c=0; % useless
20      c2=[APflag alt_c h_dot_c yaw_c beta_c V_c];
21      t2=25; % [s]
22      m2=[t2 c2];
23
24      manoeuvre=[m1;m2];
```

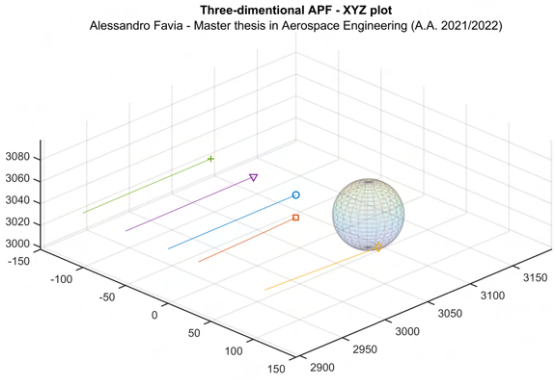
Once data has been produced, they should be visualized. This is a very important point because using videos and images it can be verified the accuracy of the simulation. We visualize data in two ways:

- Generating two-dimensional charts and tables in post-processing
- Generating three-dimensional charts both in real-time and post-processing

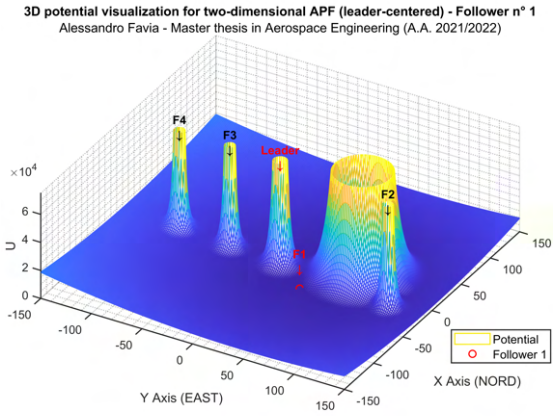
The first point is quite simple and it will not be explained. What is is not so simple is the second point, especially the real-time visualization. In order to generate axes during the Simulink simulation, the MATLAB *S-Function* tool should be used. In our case, our starting point has been the MATLAB *3DScope* block by G. Campa [38]. However, the basic code has been modified to accommodate the needs of the thesis. For example, the view is centred on the leader aircraft and formations can be clearly observed from more than one angle simultaneously. Fixed and mobile obstacles are included and they can be cylindrical (to simulate no-fly zones) or spherical (to simulate missiles or other aircraft). An example of this can be found in Fig. 3.20. Observing this figure, it can be noticed that the attitude of the aircraft can not been visualized. Using the MATLAB *Aerospace Blockset*, a post-processing animation has been created using the *Aero.Animation* tool (Fig. 3.21).
As we will see when dealing with APF, potentials should be visualized. In our thesis it has been done in a post-processing phase, saving potential data in the workspace. The two-dimensional APF case is quite easy to represent because the potential can be modelled as a 3D surface and two axes are physical length. The three-dimensional APF, in contrast, has been a difficult task to tackle because the three axes all represent length and the data we have to deal with are four-dimensional. The issue has been solved using a *scatter3* function, modifying the transparency of points in order to observe only zones with high-potential values (i.e. near aircraft and obstacles).



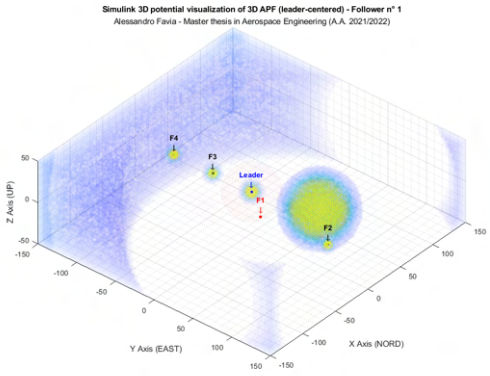
(a) 2D visualization



(b) 3D visualization



(c) Potential visualization (APF 2D)



(d) Potential visualization (APF 3D)

Figure 3.20: Visualization examples

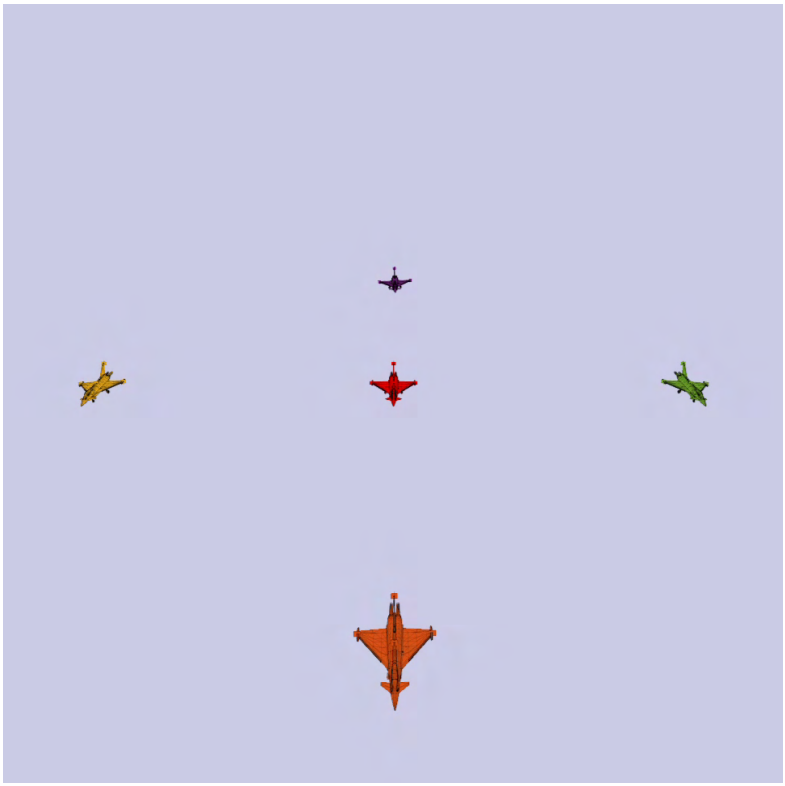


Figure 3.21: Attitude visualization using MATLAB *Aero.Animation*

Chapter 4

Formation control using Proportional Integral Derivative controllers

In Chapter 3 we have described the way in which the model of the aircraft and the logic for the manoeuvrers have been designed. Using the *manoeuvre* matrix (Sect. 3.6) every aircraft is completely independent from each other, that is it follows only user commands without sensing other followers. The next step is to create a simple cooperative formation control which allows the followers to maintain a relative position with respect to the leader. In this initial development phase, its simplicity lies in the fact that the *collision avoidance* is not considered. After examining the state of the art of formation control and motion planning (Ch. 2) and before facing the problem of the *Artificial Potential Field*, it has been decided to study how a PID-based formation algorithm can be used in a real application. An algorithm of this type is particularly useful to study because, being the PID control easy to implement, it allows us to understand the hidden mechanisms of formation flight adopting a systemic approach. Of particular importance is the YF-22 example, proposed by M.R. Napolitano et al. [18] [39]. In this case, three physical small-scale aircraft models were built, with a geometry based on the F-22 Raptor. In addition to propulsion, landing gear and other fundamental subsystems, every vehicle was provided with data acquisition and R/C systems in order to acquire data and be manually driven from the ground. During test flights, a piloted take-off was performed for all three models. Once the cruise altitude ($\simeq 150\,m$) was reached a *follower operational mode* was activated on two aircraft, testing their formation controller capabilities. For the purpose of our thesis this work has been fundamental. From the design of the formation control laws point of view, it allowed us to understand what the correct steps to follow were and laid the foundations for the APF-based algorithm.

4.1 Relative positions calculation in follower-centred reference system

When dealing with formation control, the description of *formation patten* goes hand in hand with the calculation of distances between aircraft. In a leader-follower architecture the formation is built around the leader, which is obviously the most important agent of all. For this reason it could be useful to represent the formation patten in a leader-centred coordinate systems. Nevertheless, formation commands should deal with *Autopilot* blocks of every follower whose input are in follower-centred reference system. In conclusion, it is necessary to understand how relative distances should be handled and which reference system should be used.

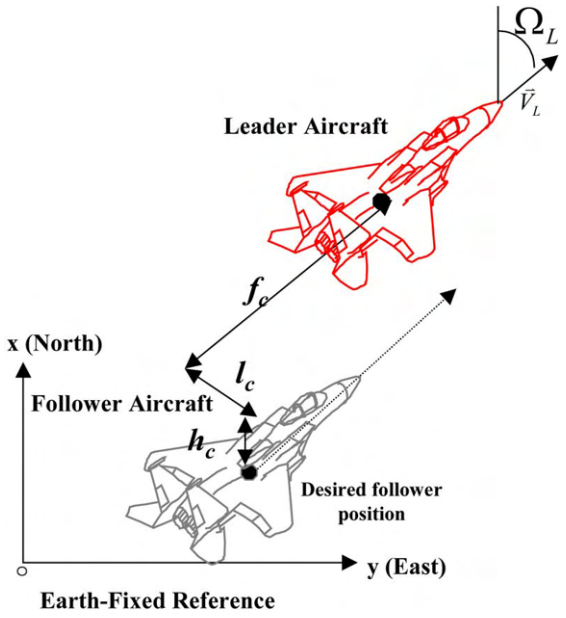


Figure 4.1: formation patten (from [18])

As suggested in [39] the formation control can be decomposed in two different problems, namely calculation of position errors on horizontal and vertical planes. First of all, an inertial NED reference system should be identified (the "Earth-Fixed Reference" in Fig. 4.1 - see Sect. 3.1.1). In our case and for how the Simulink model was structured, this system is defined for $t = 0\,s$ with the X axis pointing north, the Y axis pointing east and the Z axis pointing down. As we saw earlier, the initial position of every aircraft can be defined changing the *Initial Conditions* of the *Discrete Integrators* in the navigation equations. PN_0, PE_0 e PD_0 of leader and followers are all defined with respect to the NED system so they are absolute distances. Followers must occupy a certain position relative to the leader, which can vary with the mission. This position, seen as a vector in a *leader-centred reference frame*, can be defined using its three components:

- *Forward clearance* f_c

- *Lateral clearance* l_c
- *Height clearance* h_c

The main purpose of the PID-based formation control is to drive the followers toward their desired positions. These PIDs needs *position errors* in a follower-centred system, errors that we have to calculate. A clarification is a must: in this case we deal with position errors and for this reason we have to include initial conditions. When we will debate about APF we will calculate *velocity* errors and initial conditions will disappear.

Figure 4.2 shows the block for the calculation of the formation errors. PN, PE and PD derive from the plant block and are in NED frame. The relative altitude Δh_{LF} can be easily computed as a difference between h_L and h_F while for the *altitude error* the height clearance h_c should be added:

$$h_{formation\,error} = \Delta h_{LF} + h_c + [-1 \cdot (PD_{0,F} - PD_{0,L})] \tag{4.1}$$

The term $(PD_{0,F} - PD_{0,L})$ is introduced because if the initial condition coincides with the target point the formation error should be zero. Forward and lateral formation errors are treated differently and heading angles ψ_L and ψ_F should be introduced. Being l_c and f_c known in a *leader reference frame*, these distances should be transformed with respect to the follower:

$$\begin{bmatrix} f_{c,F} \\ l_{c,F} \end{bmatrix} = \begin{bmatrix} cos(\psi_F - \psi_L) & sin(\psi_F - \psi_L) \\ -sin(\psi_F - \psi_L) & cos(\psi_F - \psi_L) \end{bmatrix} \begin{bmatrix} f_c \\ l_c \end{bmatrix} - \begin{bmatrix} PN_{0,F} \\ PE_{0,F} \end{bmatrix} \tag{4.2}$$

The corresponding formation error are:

$$\begin{bmatrix} f_{formation\,error} \\ l_{formation\,error} \end{bmatrix} = \begin{bmatrix} cos(\psi_F) & sin(\psi_F) \\ -sin(\psi_F) & cos(\psi_F) \end{bmatrix} \begin{bmatrix} \Delta PN + (PN_{0,F} - PN_{0,L}) + f_{c,F} \\ \Delta PE + (PE_{0,F} - PE_{0,L}) + l_{c,F} \end{bmatrix} \tag{4.3}$$

The reason for using heading angles is that the X axis orientation of the follower (or leader)-centred reference frame could not be parallel to X_{NED} at $t = 0\,s$ and it varies during turns.

4.2 PID formation control block

Once $f_{formation\,error}$, $l_{formation\,error}$ and $h_{formation\,error}$ have been calculated, they are sent to the *PID formation control* block. During the block design, in addition to the creation of the actual control algorithms, peculiar problems have been addressed. As mentioned in Sect. 3.5.2 two operational modes are available:

- *"Autonomous flight"* $\rightarrow formACTIVE = 0 \rightarrow$ *formation control: OFF*
- *"Formation flight"* $\rightarrow formACTIVE = 1 \rightarrow$ *formation control: ON*

Using a *Step* block the user can change the *formACTIVE* value once during the simulation. Different approaches could have been followed in order to change the flag

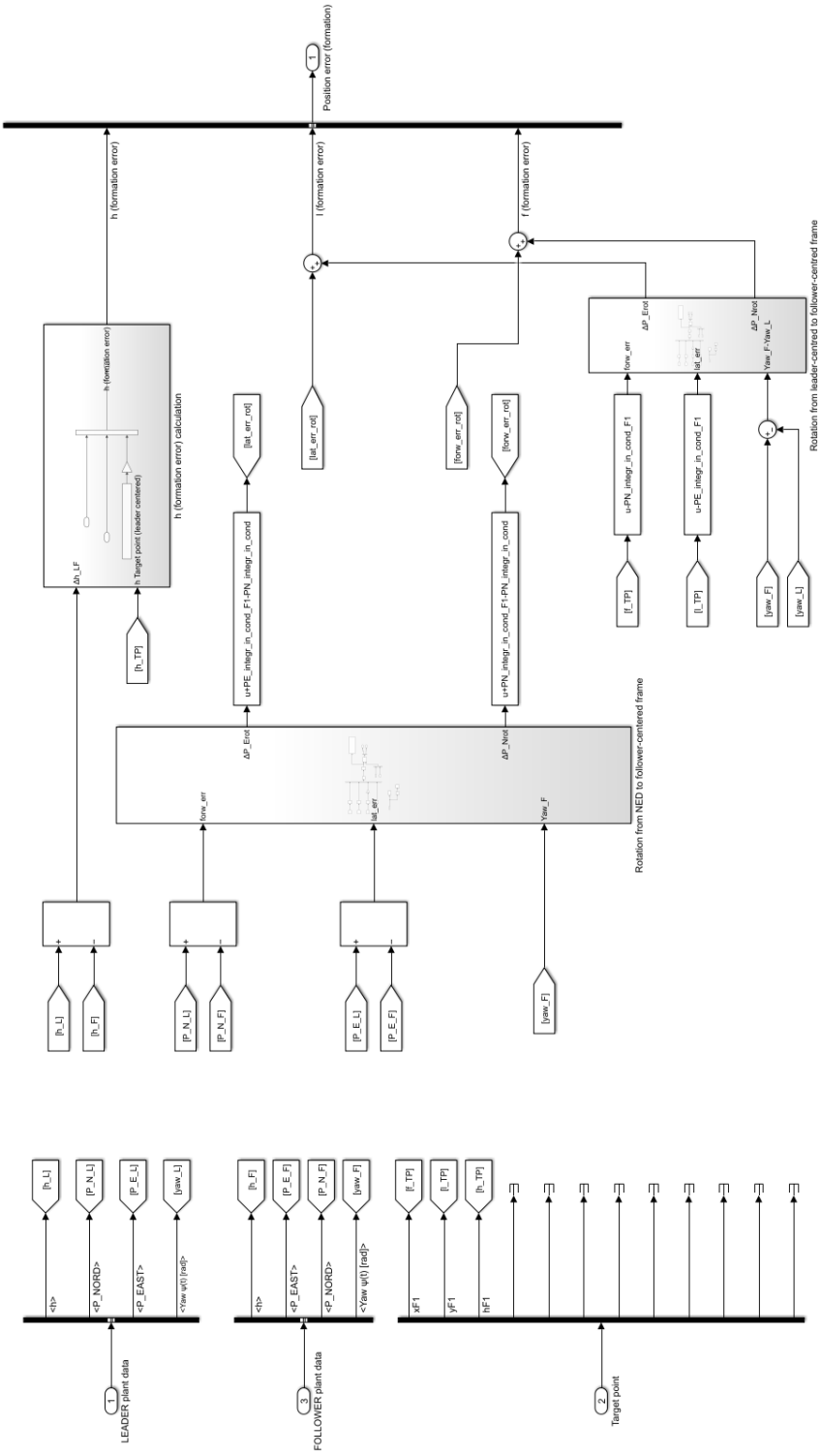


Figure 4.2: The position error subsystem overview (follower n°1)

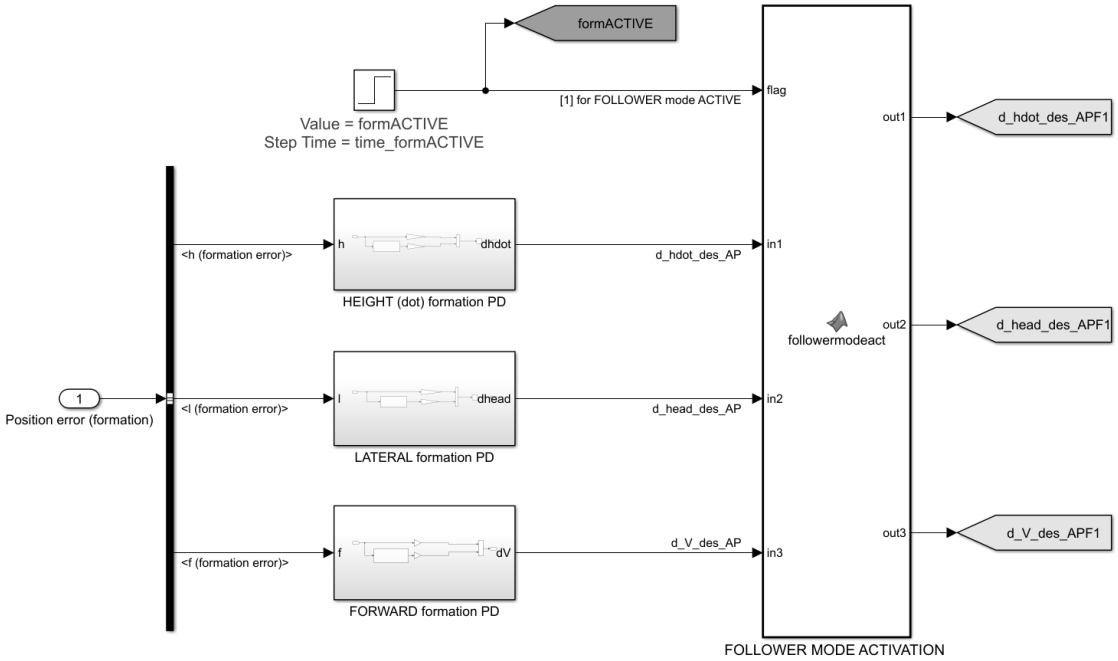


Figure 4.3: *PID formation control* block

more than once (using a *From Workspace* block similarly to *manoeuvres*, for example), but they have not been considered necessary for the purpose of this thesis.

In Fig. 4.3 an overview of the *PID formation control* block is depicted. In order to be coherent with autopilots a basic control switch called *"Follower mode activation"* has been created. If $formACTIVE = 1$ every input port is connected to the corresponding output while if $formACTIVE = 0$ every output is null. The choice of the operational mode affects the *Autopilots* block also (Fig. A.5). On the first hand it controls the *Autopilot choice* block, bypassing *manoeuvre* inputs if formation control is active. On the second hand it determines the type of vertical autopilot chosen for the simulation. The *Height (dot) formation PD* generate a vertical velocity value as output so, when $formACTIVE = 1$, the *Vertical velocity autopilot* must be selected and $altitude\_switch\_sign = -1$.

4.2.1 PID tuning

At this point it is possible to introduce the formation PID controllers and their gains, starting from the *"Lateral formation PD (position)"* block (Fig. A.10). It can be guessed that it is a proportional-derivative control (it was not necessary the use of the integrative line) which, receiving in input the $l_{formation\,error}$ generates a $\Delta\psi$. As others control blocks in this thesis, gains have been obtained through a *trial-and-error* procedure, imposing a step-like lateral clearance l_c and observing how the error tends to zero. The output has

been limited to $[-35\,deg;\, 35\,deg]$ in order to avoid excessive trajectory deviations. Gains are shown in Tab. 4.1 while the step response is depicted in Fig. 4.4.

Lateral formation PD (position)
$KP\_l\_fPD\_F = 0.06$
$KD\_l\_fPD\_F = 0.3$

Table 4.1: Lateral formation PD (position) block gains

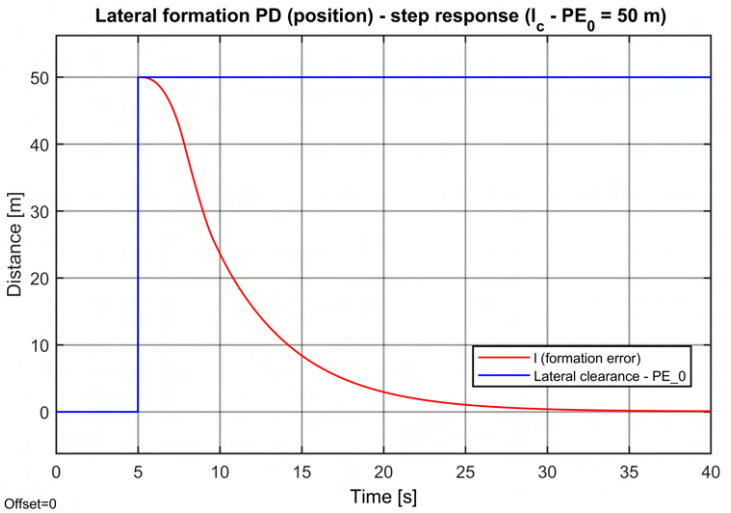


Figure 4.4: Lateral formation PD (position) block step response

The *Forward formation PD (position)* block (Fig. A.11) is similar to the previous one, not having the Integrator line. In this case the velocity saturation (between $[60;\, 140]\,\frac{m}{s}$ has been inserted in the *Throttle LQR* (Sect. 3.4.2) in order to limit both *manoeuvre* and PD signals. Being the output of the PD a ΔV_{AP}, the initial velocity V_0 should be added in order to obtain an absolute value. Gains are shown in Tab. 4.2 while the step response is depicted in Fig. 4.5.

Forward formation PD (position)
$KP\_f\_fPD\_F = 2$
$KD\_f\_fPD\_F = 2.5$

Table 4.2: Forward formation PD (position) block gains

The *Height (dot) formation PD (position)* block (Fig. A.12) is basically identical to the other PDs. In this case the saturation is identical to the *PD altitude* autopilot one, avoiding $\dot{h}$ values not included in $\pm 1000\,\frac{ft}{min}$ ($\pm 5.08\,\frac{m}{s}$). Gains can be found in Tab. 4.3 while the step response in Fig. 4.6.

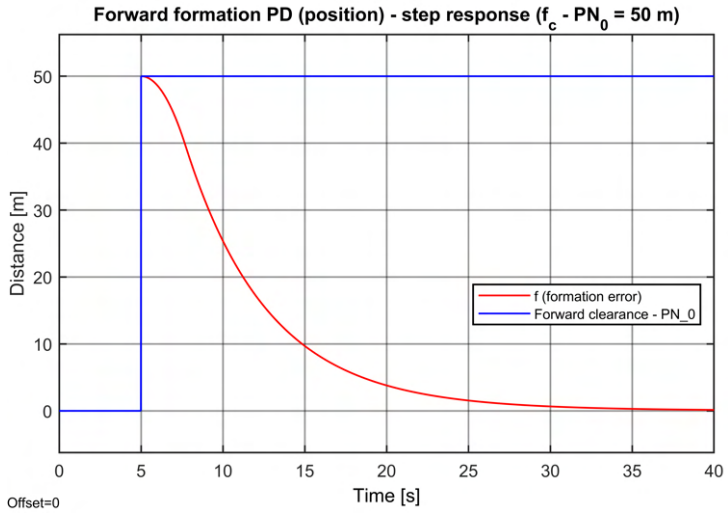


Figure 4.5: Forward formation PD (position) block step response

Height (dot) formation PD (position)
$KP\_h\_fPD\_F = 0.035$
$KD\_h\_fPD\_F = 0.2$

Table 4.3: Height (dot) formation PD (position) block gains

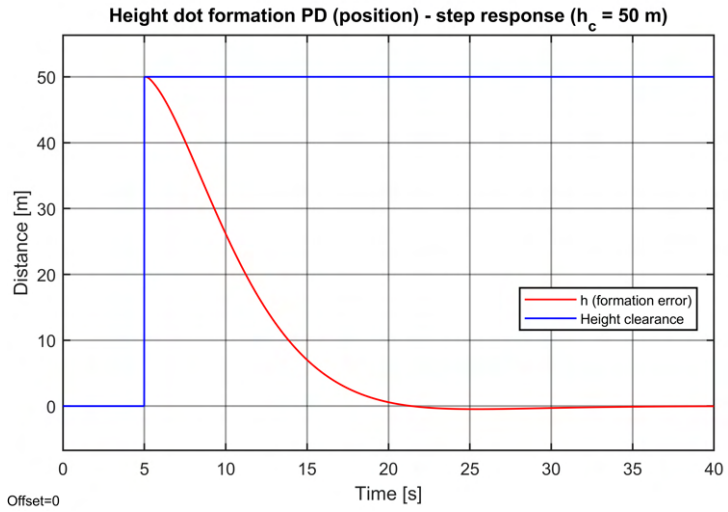


Figure 4.6: Height (dot) formation PD (position) block step response

4.3 Section formations

As previously explained, the formation control algorithm receives in input the *target points*, that are the relative positions of every follower with respect to the leader (in a *leader-centred frame*). In this thesis, seven different basic formation options have been implemented [40]:

- `'initialpositions'`
- `'vic'`
- `'echelon'`
- `'line_abreast'`
- `'line_astern'`
- `'box'`
- `'arrow'`

Every option corresponds to a `case` of the formation `switch` implemented in the initialization code of the model. During a mission it could be useful to change the pattern of the formation, both varying its shape and dimension. For this reason a `'manoeuvre'`-like approach has been adopted, using the block *From Workspace*.

```
        case 'line_astern + echelon'

        % LINE ASTERN
        xF1=-35; %[m] to modify
        xF2=xF1*2; xF3=xF1*3; xF4=xF1*4;
        yF1=0; yF2=0; yF3=0; yF4=0;
        hF1=0; hF2=0; hF3=0;hF4=0;
        targetpointF1=[xF1,yF1,hF1];
        targetpointF2=[xF2,yF2,hF2];
        targetpointF3=[xF3,yF3,hF3];
        targetpointF4=[xF4,yF4,hF4];
        c1=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
        t1=0; % [s]
        f1=[t1 c1];
        minfiguredist1=[t1 norm([xF1,yF1])]; % used to limit repulsive
            radius

         % ECHELON
        xF1=-35; %[m] to modify
        ang=45; %[deg] to modify
        xF2=2*xF1; xF3=3*xF1; xF4=4*xF1;
        yF1=xF1*tand(ang); yF2=2*yF1; yF3=3*yF1; yF4=4*yF1;
        hF1=0; hF2=0; hF3=0;hF4=0;
        targetpointF1=[xF1,yF1,hF1];
        targetpointF2=[xF2,yF2,hF2];
        targetpointF3=[xF3,yF3,hF3];
```

```
26          targetpointF4=[xF4,yF4,hF4];
27          c2=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
28          t2=50; % [s]
29          f2=[t2  c2];
30          minfiguredist2=[t1  norm([xF1,yF1])]; % used to limit repulsive
                radius
31
32          sectform=[f1;f2];
33          minfiguredist=[minfiguredist1;minfiguredist2];
```

The listing as above is an example of the string passed to the section formation logic. Two different section formations have been selected, *line-astern* and *echelon*. Every formation is composed by four points in space, that are the follower positions. Selecting only a few data (like `xF1` and `ang` for *echelon*) all the distances are calculated through geometric relations. Coordinates for every formation shape are included in a vector `f` where the first element is the time of activation of the specific string while the `sectform` consist of the combination of every `f` vector. `minfiguredist` is used for the APF model and it is composed by the shortest geometrical distance between aircraft in the same formation.

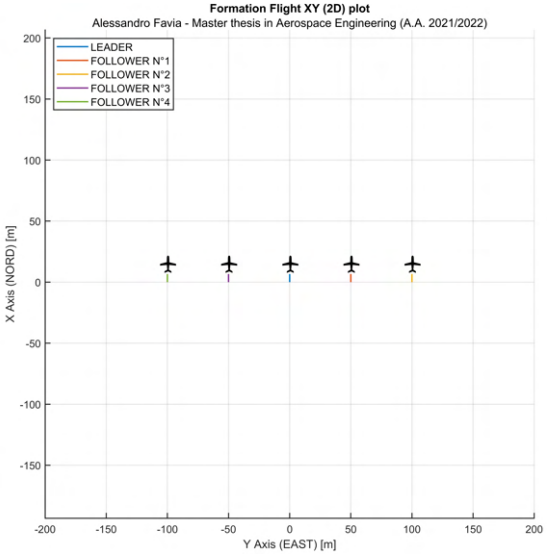
4.4 Simulation

Manipulating the `sectform` and `manoeuvre` vectors, various scenarios can be simulated. In this section we will illustrate a basic one, whose purpose is to test the code and investigate on advantages and disadvantages of a PID approach for formation control. Flag values for this scenario are:

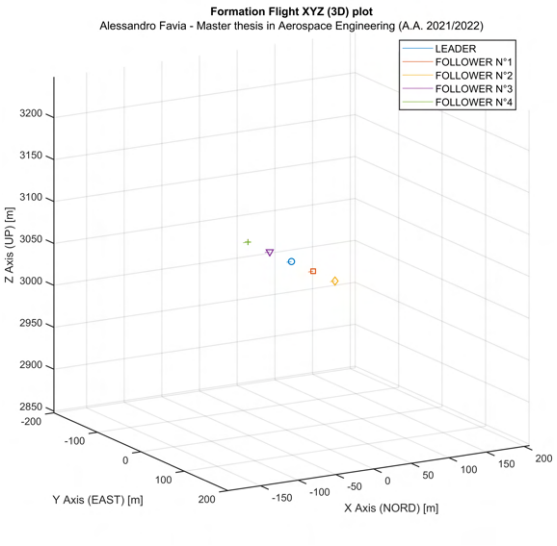
- `manoeuvre` → `'manovra5'`
- `sectform` → `'in_cond + line_astern + echelon'`

In Fig. 4.7 two and three-dimensional representations are depicted. For the first twenty seconds every aircraft remains in its initial position (*line abreast* figure) with respect to the leader, with a straight trajectory without any change in altitude (Fig. 4.7(a)(b)). Formation errors are equal to zero and PIDs do not intervene. At $t = 20\,s$ the *manoeuvre* string generates a right turn ($\psi_{des} = 180\,deg$) with altitude increase ($\dot{h}_{des} = 2\,\frac{m}{s}$), as can be seen in Fig. 4.7(c)(d). During this turn followers are ordered to change the geometry of the formation in a *line astern* type. It can be observed that aircraft initially closer to the centre of curvature (followers number one and two) have more difficulty reaching their target point (Fig. 4.7(e)(f)). This is probably due to the fact that they have to turn in the opposite direction of the manoeuvre and successively compensate for the roll overshoot. At $t = 100\,s$ another formation patten is ordered, that is the *echelon*: in this case every aircraft manages to reach its position quite easily (Fig. 4.7(i)(j)). The code used for this simulation is included in Appendix B.2.
What can be seen is that every aircraft follows the leader accurately and places itself the formation. From the computational point of view the model is particularly fast because no space grids have to be handled (unlike the APF case) while the main problem is the

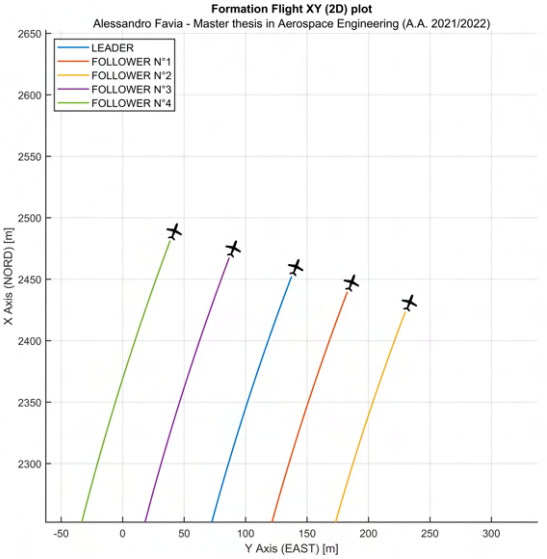
complete lack of *collision avoidance*, inasmuch aircraft have no consciousness of their surrounding (clearances are kept minimizing the distance error only).



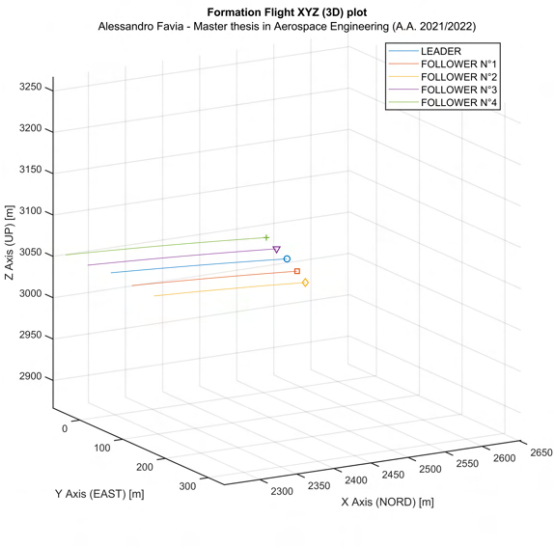
(a) 2D view (XY) - $t = 0\,s$



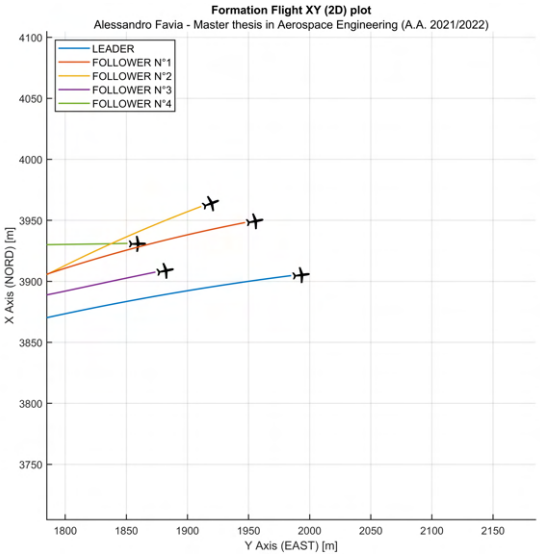
(b) 3D view - $t = 0\,s$



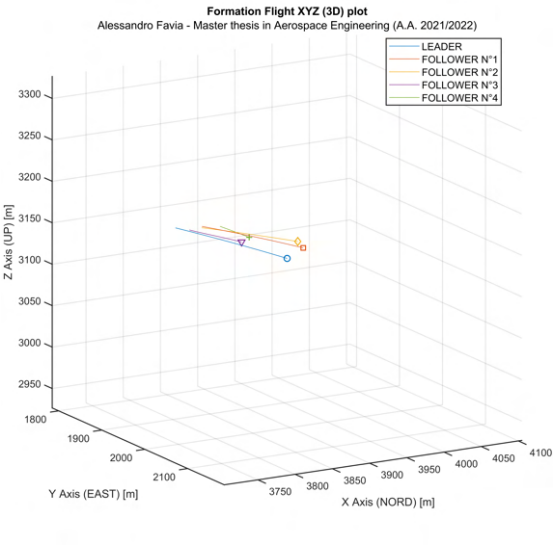
(c) 2D view (XY) - $t = 30\,s$



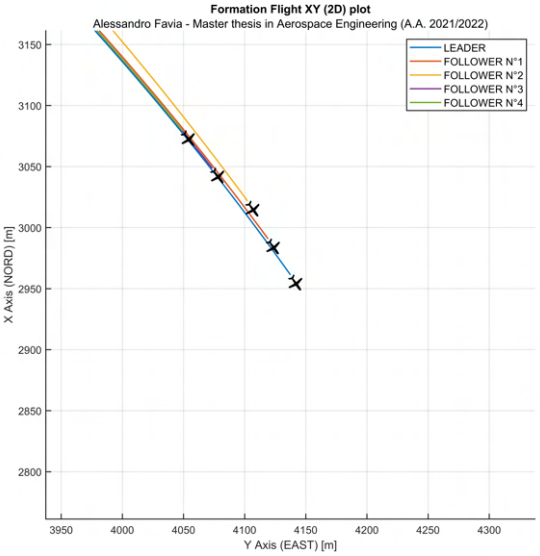
(d) 3D view - $t = 30\,s$



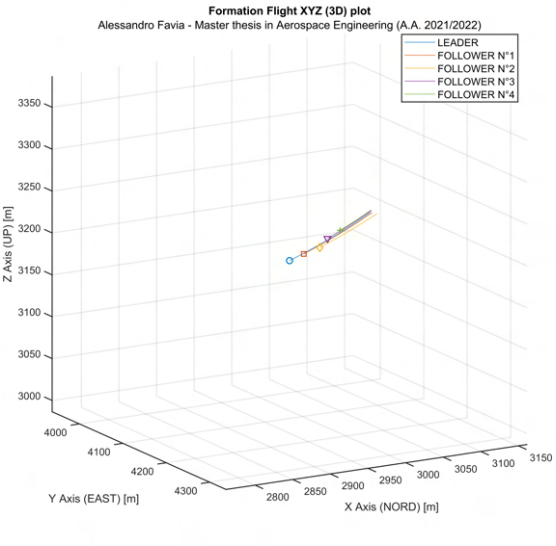
(e) 2D view (XY) - $t = 60\,s$



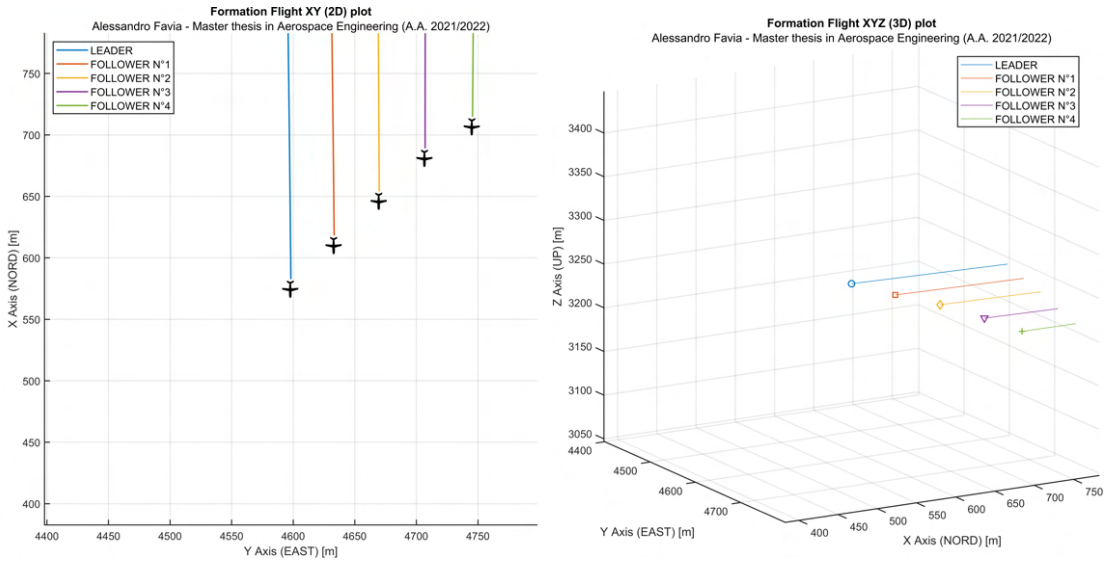
(f) 3D view - $t = 60\,s$



(g) 2D view (XY) - $t = 90\,s$



(h) 3D view - $t = 90\,s$



(i) 2D view (XY) - $t = 120\,s$ (j) 3D view - $t = 120\,s$

Figure 4.7: Formation PID model simulation example

Chapter 5

The Artificial Potential Field Algorithm - theoretical explanation

The main purpose of this thesis is to develop a cooperative formation control algorithm based on the *Artificial Potential Field (APF)* method. Also if it has been briefly illustrated in Sect. 2.3.1, it will be deeply explained in this chapter. At first, theoretical basis of *generalized coordinates*, *configuration space* and *operational space*, together with potentials definitions, will be explained. Subsequently it will pass to APF used for cooperative formation control, defining characteristics of the method and illustrating examples from the literature. Finally, *gradient descent algorithm* and *planning techniques* will be introduced, laying the foundations for future chapters.

5.1 Generalized coordinates, configuration and operational space

Motion planning is an active field of study in literature, in a world where robot *autonomy* and *reliability* are becoming increasingly important. In this thesis we could easily substitute the word "robot" with UAV, but we should remember that the field of application of motion planning is extremely extended including, for example, robotized arms or autonomous car driving. According to [41], being $\mathcal{B}$ a robot not subjected to kinematic constraints and $\mathcal{W}$ an Euclidean space (2D or 3D), motion planning can be described through a *canonical problem*:

> *"Given an initial and a final posture of $\mathcal{B}$ in $\mathcal{W}$, find if exists a path, i.e., a continuous sequence of postures, that drives the robot between $\mathcal{B}$ and the obstacles $\mathcal{O}_1, ..., \mathcal{O}_p$; report a failure if such a path does not exist."*

More than one robot can exist in the same Euclidean space, complicating the problem. Moreover, every robot could know the geometry of the environment in advance (*off-line*

planning) or not, and in the latter case it could employ sensors to obtain informations about its surrounding (*on-line planning*). That said, it can be imagined how complicated could be defining a *motion planning* algorithm.
Important definitions that will be useful in the next pages are the *configuration space* and *operational space* ones. From analytical mechanics we know that a system can be described using a limited set of parameters, which are the *generalized coordinates*. If we have a robotic arm, for example, knowing angles of all the joints we could draw exactly in which position the robotic arm is. Generalized coordinates tell us everything we need to know to characterize the movement of the robot in space: if we added a parameter, it would be completely useless. These coordinates generates a vector space called *configuration space*, which represents all the allowed configurations of the robot (including the effect of obstacles). The main problem is that planning a trajectory in this space is not so simple, because we do not have a direct representation of the Euclidean space we want to move in. At this point, the *operational space* comes to our aid. Being equal to the Euclidean space, in this case the position and orientation of the robot are intuitive. Creating a control system that let us specify the trajectory in the *operational space* and that subsequently transforms coordinates in *configuration space* would be extremely convenient.

5.2 The traditional APF approach

The *Artificial Potential Field* algorithm fits into this context, being an elegant and efficient solution to the *on-line* motion planning problem. Basically, the point $\boldsymbol{q}$ that represents the robot in the *operational space* moves under the influence of a *potential field* U, sum of *attractive* and *repulsive* terms. In each point, an *artificial force* can be calculated as $-\nabla U(\boldsymbol{q})$, whose direction is the *steepest descent direction* toward $\boldsymbol{q}_{goal}$. The first appearance of this type of algorithm can be observed in 1986 in the paper *"Real-Time Obstacle Avoidance for Manipulators and Mobile Robots"* by Oussama Khatib [30] while different variations on the theme have been published until today.
The ***total potential*** $U_{tot}(\boldsymbol{q})$ has been defined as the sum of attractive and repulsive components [41]:

$$U_{tot}(\boldsymbol{q}) = U_a(\boldsymbol{q}) + U_r(\boldsymbol{q}) \tag{5.1}$$

while the *total force* $F_{tot}(\boldsymbol{q})$, considering a unique $\boldsymbol{q}_{goal}$ and p obstacles, is:

$$F_{tot}(\boldsymbol{q}) = -\nabla U(\boldsymbol{q}) = F_a(\boldsymbol{q}) + \sum_{i=1}^{p} F_{r,i}(\boldsymbol{q}) \tag{5.2}$$

The ***attractive potential*** (Fig. 5.1(a)) can be both *parabolodic* and *conical*. In our application the first solution has been selected, that is the one illustrated in [30]:

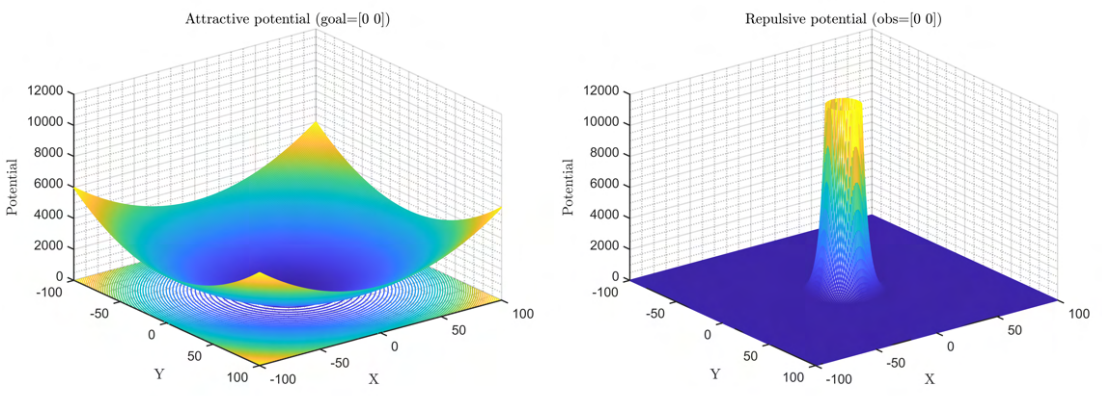
$$U_a(\boldsymbol{q}) = \frac{1}{2} K_a \|e(\boldsymbol{q})\|^2 \tag{5.3}$$

where $e(\boldsymbol{q}) = \boldsymbol{q}_g - \boldsymbol{q} = \boldsymbol{x}_g - \boldsymbol{x}$ ($\boldsymbol{x} = [x, y, z]^T$). The term $K_a > 0$ should be tuned in order to force the robot to reach the target point.

The ***repulsive potential*** (Fig. 5.1(b)) is added in order to consider obstacles with the idea of building a potential barrier around them, considering both their physical dimension (R_{obs}) and a specific tolerance (*toll*). In this case the function is of *piecewise-type* because we want the repulsive potential to act inside an established *range of influence* $\eta_{0,i}$. In general, $U_r(\boldsymbol{q})$ should be non-negative, continuous and differentiable, tending to infinity as we approach the obstacle. Khatib [30] proposed (*FIRAS function*):

$$U_{r,i}(\boldsymbol{q}) = \begin{cases} \frac{1}{2}K_{r,i}(\frac{1}{\eta_i(\boldsymbol{q})} - \frac{1}{\eta_{0,i}})^2 & \text{if } \eta_i(\boldsymbol{q}) \leq \eta_{0,i} \\ 0 & \text{if } \eta_i(\boldsymbol{q}) > \eta_{0,i} \end{cases} \tag{5.4}$$

where $\eta_{0,i} = R_{obs} + toll$, $\eta_i(\boldsymbol{q}) = |\boldsymbol{x} - \boldsymbol{x}_{obs}|$ and $K_{r,i} > 0$ is a repulsive constant.



(a) Attractive potential

(b) Repulsive potential

Figure 5.1: Potential example

An important issue of potential function is the existence of *local minima*, i.e. places where $F_{tot} = 0$. In this case the algorithm could produce an error, because the robot is trapped there (Fig. 5.2). As specified in [41] if every obstacle is modelled as a *sphere* (as in our case) and no zero-curvature obstacles exist, local minima do not appear.

5.3 APF for cooperative formation control

The approach described in the section above has been called "traditional" because it considers a static environment with a single robot. In our thesis, things are more complicated. First of all, multiple agents are considered in order to generate a formation. Every robot must not collide with the others, and it must arrange itself in a predefined position following the leader. Finally, if obstacles exist, they must be avoided. As can be imagined, cooperative formation avoidance requires a different approach instead of the traditional one. The most popular one consists in adding other potential component to the algorithm. In [20], for example, the potential of every follower i can be divided in three parts:

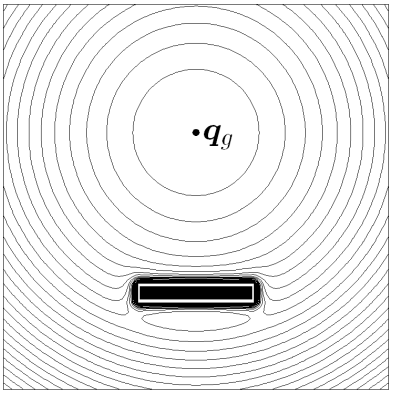


Figure 5.2: Local minimum example (from [41])

- *Inter-vehicle potential* U_i^j: consists in a sum of attractive and repulsive terms
- *obstacles potential* U_i^o: used for obstacle avoidance
- *leader potential* U_i^l: used to force followers to follow the leader

For every follower a $U_{i,tot}$ with a global a minimum exists. In this case the user does not have a direct control on the target point because the minimum is automatically generated. For the purpose of the thesis, it has been decided to adopt a different approach. In our case the potential of every follower is similar to [20], but the attractive part of U_i^j does not exists. As it will explained later, potentials will be built in a *leader-centred* frame and their components will be:

- *Inter-vehicle potential* U_i^j: repulsive potentials *only*, to avoid collisions between aircraft
- *obstacles potential* U_i^o: used for obstacle avoidance
- *target point potential* U_i^l: used to force the follower to arrange in the specific position inside the formation.

The APF used in this dissertation is defined *"pairwise"* because the potential for each agent is defined on the base of its state compared to that of one or some of the others [42].

5.4 *Gradient descent* algorithm and planning techniques

Once the potential has been created, it should be solved. Actually, the way this problem should be "solved" critically depends on its nature. In a static well-known environment (*off-line motion planning* problem), the trajectory could be computed once and then

followed by the robot. It could be the case of a robotic arm which operate in a isolated chamber, without and other moving object. If the environment changes with time, computing the entire trajectory could be useless because after a while an obstacle could interfere with the same trajectory. In both situations, however, the most used approach is the *gradient descent algorithm*. The basic idea is quite simple: starting at the initial configuration, the gradient is computed and a step in its the opposite direction is taken. The process is repeated until we reach the target point, which is where $U = 0$. In practice, this condition is often replaced with $\|\nabla U(q(i))\| < \epsilon$ where ϵ is a sufficiently small tolerance. According to [43], the algorithm can be summarized as follows:

Algorithm 1 Gradient Descent

Input: A means to compute the gradient $\nabla U(q(i))$ at a point q (e.g. APF)
Output: A sequence of points $\{q(0), ..., q(i)\} \rightarrow$ *route*

1: $q(0) = q_{start}$
2: $i = 0$
3: **while** $\nabla U(q(i)) \neq 0$ **do**
4: $\quad q(i+1) = q(i) + \alpha(i)\nabla U(q(i))$
5: $\quad i = i + 1$
6: **end while**

The notation $q(i)$ is used to denote the value of q at i-th iteration while the path consists in the sum of segments generated by $\{q(0), ..., q(i)\}$. The term $\alpha(i)$ is the *step size* and determines the length of every segment. It can be both constant and variable value, depending upon mission requirements. In this thesis, as we will analyse in the next chapter, $\alpha(i)$ will vary with the force $F_{tot}(q)$. The function we use to calculate the gradient in Algorithm 1 could considers all points of the grid. The *computational effort*, in this case, is a critical factor. When dealing with a lot of points, in fact, the *variable-time* simulation (Sect. 6.2) could be very slow. For this reason the question has been raised whether a way to optimize the algorithm exists. The *gradient descent* method, as can be seen from the literature, can basically be of three types:

- Batch gradient descent (BGD)
- Stochastic gradient descent (SGD)
- Mini Batch gradient descent (MBGD)

The *batch gradient descent* computes gradient considering all point of the grid. Conversely, the *stochastic gradient descent* essentially computes gradient using a single point of the grid. This method is computationally less expensive then BGD but introduces randomness, that is not the best. The *mini batch gradient descent* uses a *sub-grid* to compute the gradient, removing useless points. The main problem in cooperative formation control, however, is the choice of the physical limits of the mini-batch. In fact, if they are too small obstacle could not be recognised in time to be

avoided. In our thesis all methods have been investigated but for the sake of simplicity the BGD method has been implemented.

When the solution to the problem has been found, we need to transform it in something that can be interpreted by the plant. In fact, what we get from *gradient descent* can not be directly employed to control the aircraft. According to [41], using the force $F_{tot}(q)$, three *planning techniques* exist:

1. The force $F_{tot}(q)$ can be considered as a vector of *generalized forces* τ, inducing a motion which depends on the plant dynamics:

$$\tau = F_{tot}(q) \tag{5.5}$$

2. The force $F_{tot}(q)$ can be considered as a vector of *generalized accelerations* $\ddot{q}$:

$$\ddot{q} = F_{tot}(q) \tag{5.6}$$

   The effect of the robot, seen as a point mass, is independent on its dynamics.

3. The force $F_{tot}(q)$ can be considered as a vector of *generalized velocities* $\dot{q}$:

$$\dot{q} = F_{tot}(q) \tag{5.7}$$

Technically, all these methods can be used both for *on-line* and *off-line motion planning*. Using Eq. 5.5 we directly obtain control inputs for the robot: generated paths are smoother thanks to the "filtering" effect of the robot dynamics. If we would use Eq. 5.6 we should solve an inverse dynamic problem, substituting $\ddot{q}$ in the robot dynamic order in order to get forces. Eq. 5.7 could provide reference inputs to low-level controllers. In absence of local minima only Eq. 5.7 guarantees *asymptotic stability* of q_{goal} being, among other things, the fastest method. That said, in this thesis the third option has been chosen. If position formation PIDs in Sect. 4.2 are modified in order to accept *velocities as inputs* and the APF algorithm generates velocities as outputs, the follower aircraft can be controlled.

Chapter 6

Formation control using a two-dimensional Artificial Potential Field Algorithm

This chapter (together with Ch. 7) is probably one of the most important of all this dissertation: there the *two-dimensional APF-based formation control algorithm* will be explained in depth. First of all, a basic 2D cooperative formation control algorithm will be *off-line* prototyped using Matlab, understanding its operating principle. This code generates a *static* simulation, because the potential does not vary with time. Subsequently the same functions will be modified in order to be implemented in Simulink, working in a *time-varying* environment: both leader and followers APF blocks will be illustrated. Finally, simulations will be shown to highlight the code functioning.

6.1 Building the algorithm - a static 2D case

6.1.1 Potential functions analysis

Multiple potential functions are available in literature whose choice depends on the nature of the problem. We basically know that the attractive potential must be a concave function with a minimum in $\boldsymbol{x}_{goal}$ while the repulsive potential must be a nonnegative, continuous and differentiable function whose value should tend to infinity as we approach the obstacle. In this dissertation:

- The attractive potential is the one proposed by Khatib [30]:

$$U_a(\boldsymbol{q}) = \frac{1}{2} K_a \|e_g(\boldsymbol{x})\|^2 \tag{6.1}$$

- The repulsive potential is the one proposed by Dang [20]:

$$U_r(\boldsymbol{q}) = \frac{1}{2} K_{r1} \left(\frac{r_0 + K_{r2}}{\|d_{obs}\| + K_{r2}} - 1 \right)^2 \tag{6.2}$$

The attractive potential depends on a constant K_a and on the distance between the point in the grid $\boldsymbol{x}$ where we want to calculate its value and the position of the target point $\boldsymbol{x}_{goal}$:

$$e_g(\boldsymbol{x}) = \boldsymbol{x}_{goal} - \boldsymbol{x} \tag{6.3}$$

As can be observed in Fig. 6.1, K_a determines the steepness of the curve. In our case, a value of $K_a = 0.6$ has been chosen (Tab. 6.1).

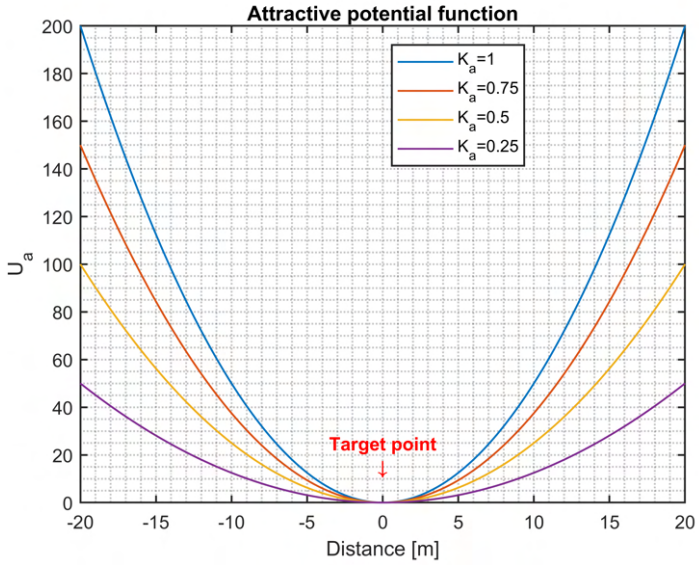


Figure 6.1: Attractive potential function

The repulsive potential is more complicated. In this case there are two different constants, K_{r1} and K_{r2}. $r_0 = (R_{obs} + toll)$ is the *repulsive radius* and d_{obs} is the difference between the point of the grid $\boldsymbol{x}$ and $\boldsymbol{x}_{obs}$:

$$\boldsymbol{d_{obs}} = \boldsymbol{x} - \boldsymbol{x}_{obs} \tag{6.4}$$

The effect of repulsive constants is different, as can be noticed in Fig. 6.2. If they are both equal to one, the function tends to a finite value for $d_{obs} = 0\,m$. If we reduce $K_{r2} = 0.01$, the function tends to infinity but the curve is quite close to the obstacle so it can not be employed in a APF algorithm. Raising $K_{r1} = 10000$ the function starts to be usable because it tends to infinity away from the obstacle. In this thesis it has been decided to combine these effects, with a high value of K_{r1} and a low value of K_{r2} (Tab. 6.1). For the leader, the K_{r1} constant is greater than the follower one in order to highlight the importance of avoiding the *manned aircraft* in the formation.

6.1.2 Code architecture

Once the potential functions have been analysed, the cooperative formation algorithm should be created. Before implementing it in Simulink it has been decided to study the

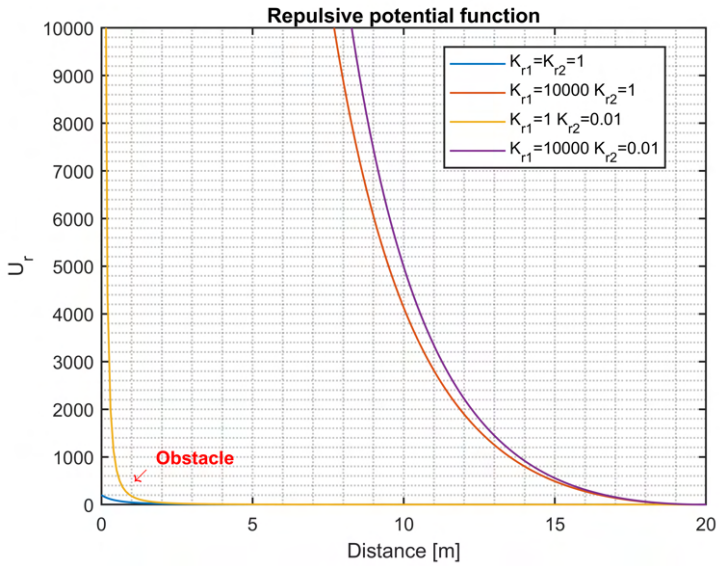


Figure 6.2: Repulsive potential function

	Target point	Leader	Follower
K_a	0.6	-	-
K_{r1}	-	10000	8000
K_{r2}	-	0.01	0.01

Table 6.1: Constants of the potential functions

operating principle of APF using Matlab. Our algorithm will be essentially made up of four functions:

- *Grid Generator* → Initialization code
- *Objects (relative) position calculator* → `ObjectPosCalc_AF`
- *Potential Generator* → `PotentialGenerator2D_AF` (or 3D)
- *Potential Solver* → `PotentialSolver_AF`

The potential, by nature, can be calculated in a specific point in space. Dealing with continuous quantities using algorithms is not practical while discrete quantities can be easily handled using lines of code. The space should be *discretized* transforming it in a two- or three-dimensional tensor, so the *Grid Generator* inside the initialization code fulfils this task. The listing below represents the code used for the three-dimensional case (the 2D code can be easily obtained removing all the terms which refers to the Z

axis). After choosing the half width of every side (`half_grid_ext_X`, `half_grid_ext_Y`, `half_grid_ext_Z`) and the distance between every point (`gridpointdist`), using the `ngrid` function the tensor is calculated.

Listing 6.1: *Grid Generator (3D)*

```
half_grid_ext_X=150; %[m]
half_grid_ext_Y=150; %[m]
half_grid_ext_Z=70; %[m]

gridpointdist=3; %[m]
n_grid_X=ceil((2*half_grid_ext_X)/gridpointdist);
n_grid_Y=ceil((2*half_grid_ext_Y)/gridpointdist);
n_grid_Z=ceil((2*half_grid_ext_Z)/gridpointdist);

X_ext_N=linspace(-half_grid_ext_X,half_grid_ext_X,n_grid_X);
Y_ext_E=linspace(-half_grid_ext_Y,half_grid_ext_Y,n_grid_Y);
Z_ext_H=linspace(-half_grid_ext_Z,half_grid_ext_Z,n_grid_Z);
[x_grid_N,y_grid_E,z_grid_H]=ndgrid(X_ext_N,Y_ext_E,Z_ext_H);
```

As can be seen in Sect. 6.1.1, we need *relative* distances between a point in the grid $\boldsymbol{x}$, the target point $\boldsymbol{x}_{goal}$ and obstacles $\boldsymbol{x}_{obs}$. What should be defined is the reference frame in which these values should be calculated. As will be described later, every potential in our code is computed in a *leader-centred reference frame*. The reason is that we consider the leader as the most advanced aircraft in the formation, having the best calculating capacity of all. The function "`ObjectPosCalc_AF`" is responsible to obtains these values, subtracting the $X_{L,NED}$ and $Y_{L,NED}$ positions of the leader to the follower ones and rotating the results using ψ_L.

Listing 6.2: *Objects position calculator* (extract)

```
diff_X_F1=x_F1-x_leader;
diff_Y_F1=y_F1-y_leader;

R=[cos(psi_leader) sin(psi_leader);-sin(psi_leader) cos(
    psi_leader)];
diff_rot_F1=(R*[diff_X_F1;diff_Y_F1]);

F1forw_d=diff_rot_F1(1);
F1lat_d=diff_rot_F1(2);
```

The next step is to calculate the total potential in every point of the grid. This is done using the "`PotentialGenerator2D_AF`" function. After defining all the constants K_a, K_{r1} and K_{r2} for leader and follower, the repulsive radius r_0 and after preallocating matrices, potential values are computed using two (or three in the 3D case) `for` loops. Moreover, when dealing with repulsive terms, using a `if` structure only points inside r_0 are selected. The attractive potential for the target point is calculated in a similar manner and the total potential for every follower is obtained adding all the terms. For the *static* simulation r_0 is considered constant whereas in Simulink it is a function of the aircraft velocity V. The *leader* position in the grid, which is $[x_{L,grid}, y_{L,grid}] = [0; 0]$ by default, can be changed modifying `Lforw_p` ($= x_{L,g}$) and `Llat_p` ($= y_{L,g}$). Two extra features, not used during simulations, have been included:

- The possibility to generate a potential without *target point* and with the coexistence of both attractive and repulsive parts centred in every follower. This operating principle is the first one explained in Sect. 5.3
- If the previous operating principle is selected, attractive and repulsive potential could use different radius r_0 in order to generate a sectionalized domain. This solution, according to [44], stabilizes the algorithm.

The following listing is an extract of the "`PotentialGenerator2D_AF`" function. Using a `switch` structure, the exact number of followers (between one and four) can be included.

Listing 6.3: *Potential Generator* (extract)

```
        Ua_F1 = zeros(n_grid_X,n_grid_Y);
        Ur_F1 = zeros(n_grid_X,n_grid_Y);
        Ut_F1 = zeros(n_grid_X,n_grid_Y);

        for ii = 1:n_grid_X
            for jj = 1:n_grid_Y
                if rho_obs_follo_2D(F1forw_d,F1lat_d,x_ext_N(ii),y_ext_E(
                    jj)) > Resterno_F1

                    Ua_F1(ii,jj) = 0.5*Ka_F1*rho_obs_follo_2D(F1forw_d,
                        F1lat_d,x_ext_N(ii),y_ext_E(jj)).^2-0.5*Ka_F1*(
                        Resterno_F1).^2;
                else
                    Ua_F1(ii,jj) = 0;
                end
                if rho_obs_follo_2D(F1forw_d,F1lat_d,x_ext_N(ii),y_ext_E(
                    jj)) <=Rinterno_F1

                    Ur_F1(ii,jj)=0.5*Kr1_F1*((r0_F1+Kr2_F1)./(
                        rho_obs_follo_2D(F1forw_d,F1lat_d,x_ext_N(ii),
                        y_ext_E(jj))+Kr2_F1)-1).^2;
                else
                    Ur_F1(ii,jj) = 0;
                end
            end
        end

        for ii = 1:n_grid_X % Target point
            for jj = 1:n_grid_Y
                Ut_F1(ii,jj) = 0.5*Ka_T*rho_obs_follo_2D(targetpointF1(1)
                    ,targetpointF1(2),x_ext_N(ii),y_ext_E(jj)).^2;
            end
        end

        potential_F1=Ui_L+Ut_F1;
```

At this point, the potential can finally be solved using a *gradient descent* method. There, an important difference exists between the *static* Matlab code and the *time-varying* Simulink one. The former is basically equal to the Algorithm 1 in Sect. 5.4, computing the entire route from the start to the target point. At the contrary, the latter does not

solve the entire potential because it would not be efficient, as we will see in the following sections. The listing below represents the code used for the *static* simulation. After initializing some parameters (`i`, `todo`, `max_iter` and `maxForce`), the index of the starting point is calculated using the `min` function. The gradient is then computed, generating the force vector `F` and the step size is obtained proportionally to the absolute value of `F`. The point `q` and its indexes can be employed to obtain the route through interpolation.

Listing 6.4: *Potential Solver (static Matlab 2D simulation)*

```
function [l_clear,f_clear,route_clear,index_clear,i] =
    PotentialSolver_AF(x_grid_N,y_grid_E,potential,Fforw_d,Flat_d,
    gamma_step)

%%% PARAMETERS INITIALIZATION
i=0; % # of iterations
todo=1; % for "while" loop
max_iter=500;
maxForce=10^-4;

% Find index of follo in leader-centered ref system
[~,inx_N]=min(abs(x_grid_N(1,:)-Fforw_d)); % Initial index of X follower
    pos
[~,iny_E]=min(abs(y_grid_E(:,1)-Flat_d)); % Initial index of Y follower
    pos
route_clear=[Fforw_d Flat_d];
index_clear=[inx_N iny_E];

%%% FIND THE MINIMUM! %%%
% Gradient calculation
[Fx,Fy]=gradient(-potential);

while todo==1

    %%% Force in current point (use "interp2")
    F=[interp2(Fx,inx_N,iny_E,'linear'),interp2(Fy,inx_N,iny_E,'linear')
        ];
    modF = max(10^-6,sqrt(F(1).^2+F(2).^2));

    %%% Compute step alpha
    alpha=gamma_step*sqrt(F(1)^2+F(2)^2);

    %%% Compute increment q
    q=alpha*(F./modF);

    %%% Compute new indexes
    inx_N = inx_N + q(1);
    iny_E = iny_E + q(2);

    if  modF<maxForce
        todo=0;
    else
        i=i+1;

```

```
40          if i==max_iter
41              todo=0;
42              fprintf('Too many iterations!!');
43          end
44      end
45
46      l_clear= interp2(y_grid_E,inx_N,iny_E,'linear');
47      f_clear= interp2(x_grid_N,inx_N,iny_E,'linear');
48      route_clear=[route_clear;[f_clear l_clear]];
49      index_clear=[index_clear;[inx_N iny_E]];
50  end
51  end
```

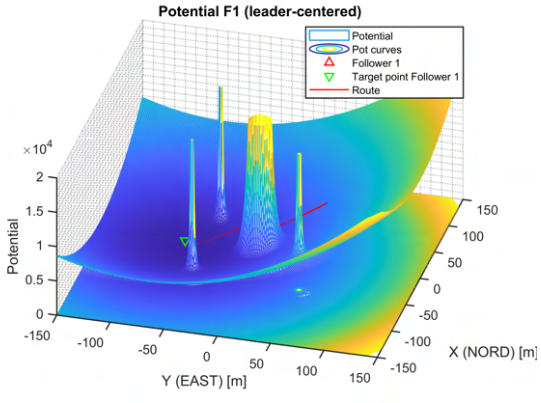
6.1.3 Static 2D simulation results

The main purpose of the static simulation is to understand how the *Artificial Potential Field* algorithm works and how it could be optimized for a *time-varying* environment. The initial conditions have been chosen to be different to the PID model ones (Sect. 4.4). Considering an inertial reference system (NED) corresponding to the initial position of the leader, starting points and target points of the aircraft are:

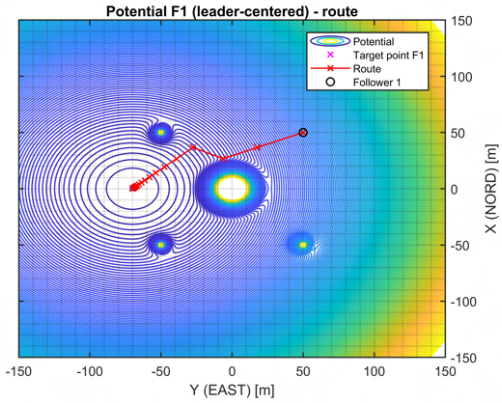
	Starting point $[x, y]_{st}$	Target point $[x, y]_{tg}$
Follower 1	[50,50]	[0,-70]
Follower 2	$[50, -50]$	[0,-90]
Follower 3	$[-50, -50]$	[80,-80]
Follower 4	$[-50,50]$	[30,-30]

Table 6.2: Static simulation - starting and target points

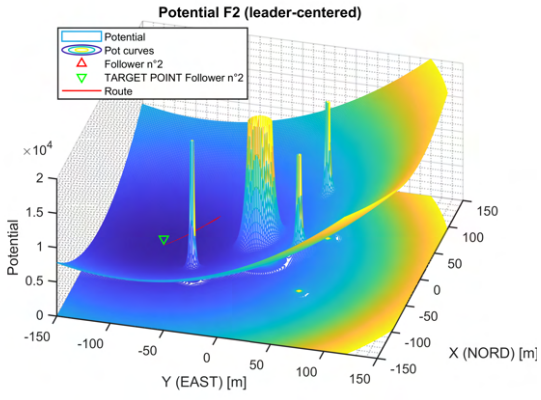
In Fig. 6.3 the results of the static simulation are depicted. As can be observed, the *gradient descent* algorithm follows the steepest descent direction, avoiding obstacles (as the leader aircraft in $[x, y]_L$ = [0,0]). The *step size* depends on the absolute value of the force (that is the slope of the potential), for this reason it decreases as we approach the target point. Observing the fourth follower, we can notice that it does not reach its target point but it remains trapped in the vicinity of the leader. This is a *saddle point*, and it is a typical disadvantage of the APF algorithm. Like the *local minima*, there the force F is exactly equal to zero so numerically no descent directions exist. In a static simulation this problem could be solved introducing *white noise* in the route in order to escape saddle points. A time-varying simulation is typically noisier than the static one, for this reason saddle points could be automatically escaped. This phenomenon, however, should be further studied.



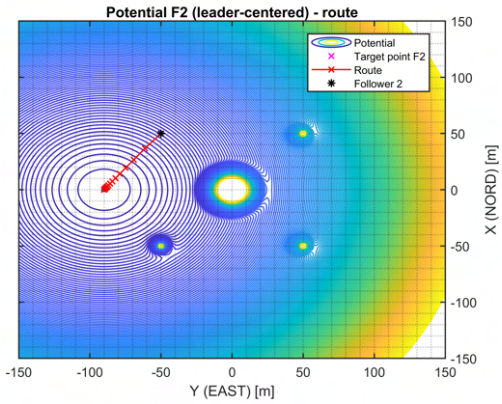
(a) 3D view (XY) - F1



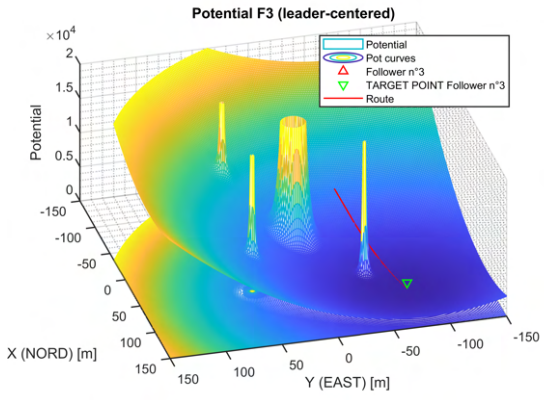
(b) 2D view (XY) - F1



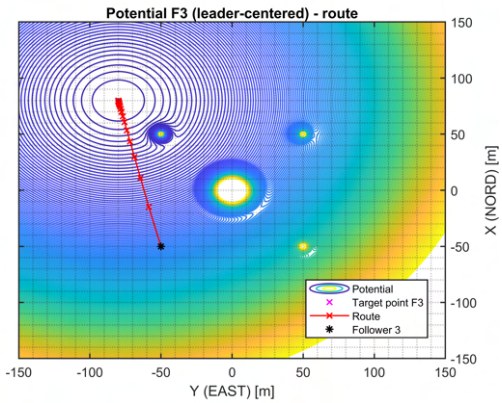
(c) 3D view (XY) - F2



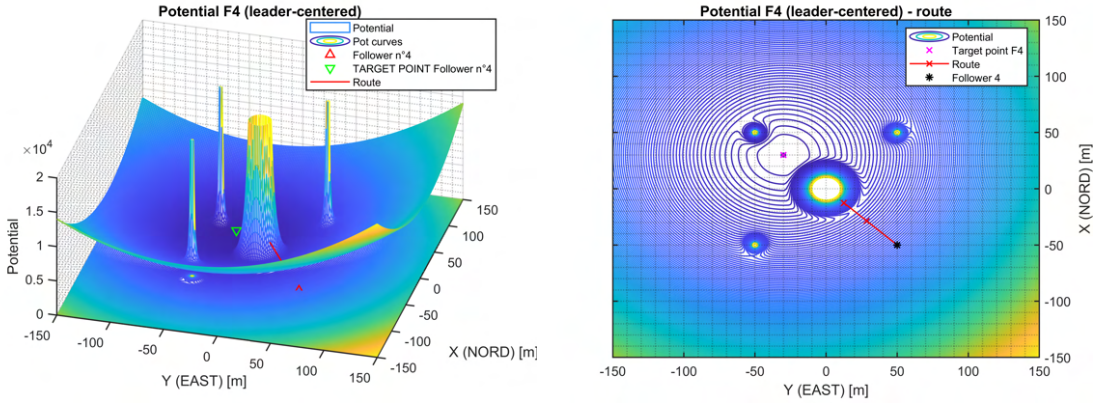
(d) 2D view (XY) - F2



(e) 3D view (XY) - F3



(f) 2D view (XY) - F3



(g) 3D view (XY) - F4

(h) 2D view (XY) - F4

Figure 6.3: Static two-dimensional APF example

6.2 The Simulink model - time varying 2D case

After studying the static algorithm, Matlab functions in Sect. 6.1.2 have been modified in order to work in a two-dimensional time-varying environment. The aircraft model is the same used for the *PID formation control model* (Sect. 4), with all its mobile surfaces and autopilot blocks. In this case, however:

- Relative distances and potentials in *leader-centred reference frame* are calculated by the leader. The *APF leader* block has been added.
- Potential tensors are transmitted to every follower and solved using the *gradient descent* algorithm. Only one step is executed and results are conveniently handled to be used by formation PIDs

Supposing a limited computational power, every APF block works at a frequency $f = 1\,Hz$. This hypothesis, as we will see, will affect our simulations (especially during turns) since it will introduce delayed commands.

6.2.1 Leader UAV - 2D APF block

In Fig. A.13 an overview of the APF leader block is depicted. First of all, NED positions and velocities are received in input together with the heading angle ψ_L. Using these data relative positions in leader-centred frame are calculated thanks to the "`ObjectPosCalc_AF`" function (Sect. 6.1.2). The same function can be employed to include *moving obstacles* in the simulation, provided that $x_{OBS,NED}$ and $y_{OBS,NED}$ are known. A *"Fixed-moving obstacle NED positions"* block (Fig. 6.4) has been created: assuming constant velocities and altitude, using a *Digital Clock* we obtain $x_{OBS,NED}$ and $y_{OBS,NED}$ that can subsequently transformed.

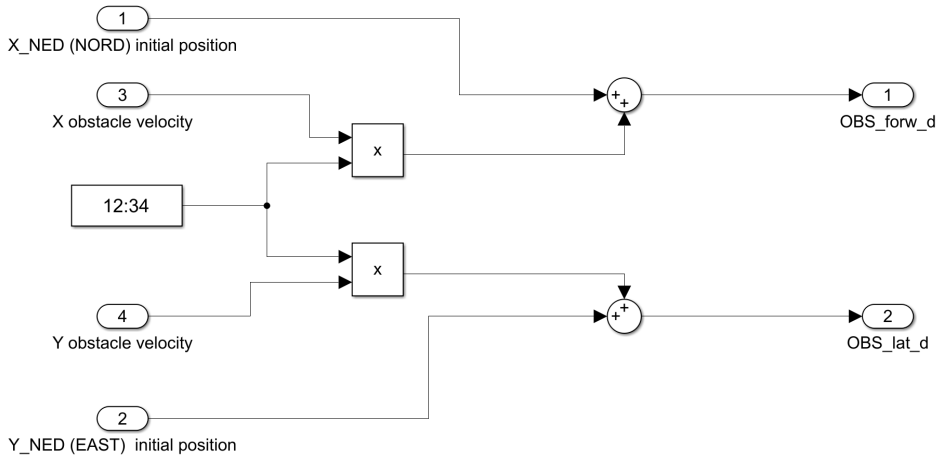


Figure 6.4: *Fixed-moving obstacle NED positions* block

Input values in Fig. 6.4, i.e. initial ($t = 0\,s$) positions and velocities of the obstacle should be initialized. Using `switch` multiple scenarios have been created, like the *no-fly zone* in the listing below. The `obs_on` flag is needed to activate the potential calculation in the `PotentialGenerator2D_AF_sim` function. `R_obs1` is the physical obstacle radius while `R_pot1` is the repulsive potential radius that is proportional to `R_obs1`, including a tolerance *toll* (in this case it is equal to `R_obs1`).

Listing 6.5: No-fly zone obstacle initialization example

```
1   case 'no-flyzone'
2       obs_on=1;
3       X_NED_NORD_obs1=4000;
4       Y_NED_EAST_obs1=-2000;
5       Z_NED_H_obs1=4000;
6       X_NED_vel_obs1=0;
7       Y_NED_vel_obs1=0;
8       Z_NED_vel_obs1=0;
9       R_obs1=1400;
10      R_pot1=R_obs1*2; %=R_obs1+toll
```

Once all positions in *leader-centred frame* have been calculated, the potential is computed using "`PotentialGenerator2D_AF_sim`". This function is similar to its static version, but some modifications have been introduced. Every aircraft, during the simulation, can change its velocity acting on the throttle. If an obstacle must be avoided at higher velocities then V_0, a faster platform response time is needed. In our model the inertia of the aircraft has been implicitly taken into consideration during all PID tuning and it mainly depends on plant matrices, which are constant. In order to facilitate obstacle avoidance, the potential generator function has been modified acting on the *repulsive radius*. Supposing a velocity interval of $[60 - 140]\,\frac{m}{s}$ (Tab. 3.2), the repulsive radius of every aircraft is calculated by a exponential function:

$$R_{rep} = R_{obs} + toll = R_{max}\left(\frac{R_{min}}{R_{max}}\right)^{\frac{V-140}{60-140}} \tag{6.5}$$

R_{max} and R_{min} should be chosen accurately in order not to interfere with the formation shape logic (Sect. 4.3). If we chose a wrong value of R_{max}, for example, followers could have difficulty reaching their target points. In our code it has been decided to compute R_{max} depending on the minimum aircraft distance in the formation patten, that is equal to the value of `minfiguredist`. As can be seen in the following listing, $R_{min,L} = 25\,m$ and $R_{min,F} = 20\,m$ are constant values while R_{max} varies:

$$\begin{aligned} R_{max,L} &= \max\left(25, \min\left(50, minfiguredist\right)\right) \\ R_{max,F} &= \max\left(20, \min\left(40, minfiguredist\right)\right) \end{aligned} \tag{6.6}$$

This means that if $V > V_0$ the repulsive radius increases and every follower "perceives" obstacles earlier, having more time to avoid them. Once potentials and distances have been calculated, they are sent to follower blocks where, thanks to *Bus selectors* they are extracted and manipulated.

Listing 6.6: *Potential Generator (Simulink version - extract)*

```
1
2  Rvarfun= @(Rmax,Rmin,V) Rmax*(Rmin/Rmax)^((V-140)/(60-140));
3
4  Rmin_L=35; %[m]
5  Rmax_L=max(Rmin_L,min(50,minfiguredist)); %[m]
6
7  Rmin_F=30; %[m]
8  Rmax_F=max(Rmin_F,min(40,minfiguredist)); %[m]
9
10 % Repulsive pot distance of the leader (VARIABLE!)
11 Rinterno_L=Rvarfun(Rmax_L,Rmin_L,VL); % [m]
12 Resterno_L=Rinterno_L+0; % [m]
13
14 % Repulsive pot distance of every follower (VARIABLE!)
15 Rinterno_F1 = Rvarfun(Rmax_F,Rmin_F,VF1); % [m]
16 Resterno_F1 = Rinterno_F1+0; % [m]
17 Rinterno_F2 = Rvarfun(Rmax_F,Rmin_F,VF2); % [m]
18 Resterno_F2 = Rinterno_F2+0; % [m]
19 Rinterno_F3 = Rvarfun(Rmax_F,Rmin_F,VF3); % [m]
20 Resterno_F3 = Rinterno_F3+0; % [m]
21 Rinterno_F4 = Rvarfun(Rmax_F,Rmin_F,VF4); % [m]
22 Resterno_F4 = Rinterno_F4+0; % [m]
```

6.2.2 Follower UAV - 2D APF block

In Fig. A.14 an overview of the APF follower block is depicted. Inputs of this block are potentials and distances (from the leader) together with ψ_L, ψ_F and XY grids produced using `ngrid`. The `PotentialSolver_AF_sim` function derives from its *static* version, but

there is an important difference that should be highlighted. In the *static* (not *time-variant*) version we calculated the entire route, stopping iterations when a specific condition was satisfied; in a *time-variant* environment this is completely useless. Suppose that, for every simulation step of the APF (i.e. being $f = 1\,Hz$, every second), we obtain the entire route. In order to calculate errors for formation PIDs, we should select a point from which we could extract two- or three-dimensional values (one for each direction). These values can be:

- *position* of one point P of the route
- *force* F of one point P of the route

Formation PIDs have to be created depending on the type of value we have chosen. In our case both solutions have been investigated but it has been decided to extract the *force* F so as to be coherent with Siciliano [41]. That said, we wonder *how many points* P *of the route we need to calculate every time step* (assumed that a force F can be associated to every P) *so that the solution can be efficiently employed in the Simulink model*. If we choose a random point of the route, obstacles will not be avoided because the aircraft would be forced to reach that point following a straight trajectory. For this reason we have decided to *stop before the first iteration, calculating the force in the point where the follower is*.
That said, the `PotentialSolver_AF_sim` function (in the listing below) will be now described. Parameters like `i`, `todo`, `maxiter` and `maxForce` are not initialized because they are not necessary for the model. Using `min` we can obtain the index of the actual position of the follower $INX_F = [inx\_N,\, inx\_E]^T$ in *leader-centred frame*, finding the point of the grid where the difference between x_{grid} (y_{grid}) and `Fforw_d` (`Flat_d`) is smaller. The gradient is calculated using `gradient` (the output is `[Fy,Fx]` to be coherent with `ngrid`) and the force `F` in INX_F is extracted. The absolute value of F is calculated (it is limited to 10^{-6} in order to avoid NaN errors) and the *descent direction* `dL` is obtained. As we described in Sect. 5.4, the output of the APF depends on `F` and consequently on `dL`. The main problem is that formation PIDs need vectors in *follower-centred frame*, so the descent direction should be rotated obtaining `d`:

$$\boldsymbol{d} = \begin{bmatrix} F_x \\ F_y \end{bmatrix}_{FOLLO} = \boldsymbol{R_{LF}} \cdot \boldsymbol{dL} = \begin{bmatrix} \cos(\psi_F - \psi_L) & \sin(\psi_F - \psi_L) \\ -\sin(\psi_F - \psi_L) & \cos(\psi_F - \psi_L) \end{bmatrix} \begin{bmatrix} F_x \\ F_y \end{bmatrix}_{LEAD} \tag{6.7}$$

At this point we should choose the type of command to send to the follower, which will characterize the input of formation PIDs. In the model explained in Sect. 4 commands are target points in follower-centred frame, so we dealt with *positions*; for the *Artificial Potential Field*, we extract a force F. In Sect. 5.4 we have described three *planning techniques* and we said that the best option for our case was Eq. 5.7:

$$\dot{q} = F(q)$$

We need to calculate velocities from the force F. These velocities (along X and Y) lead the follower to flight along a specific direction which is the one calculated by APF in order to avoid obstacles and reach safely the target point:

```
f_Vclear=min(5,0.15*abs(F(1)))*d(1)
l_Vclear=min(6,0.20*abs(F(2)))*d(2)
```
(6.8)

As can be observed from Eq. 6.8, `f_Vclear` and `l_Vclear` are obtained multiplying `d` by a term which depends on the corresponding F component. Using `min` we can insert a superior limit, avoiding excessive high speed.

Listing 6.7: *Potential Solver (Simulink version)*

```
function [l_Vclear,f_Vclear,dL] = PotentialSolver_AF_sim(x_grid_N,
    y_grid_E,potential,Fforw_d,Flat_d,psiL,psiF)
%%% PARAMETERS INITIALIZATION %%%
% Find index of follower in leader-centered ref system
[~,inx_N]=min(abs(x_grid_N(:,1)-Fforw_d)); % Index of X follower pos
[~,iny_E]=min(abs(y_grid_E(1,:)-Flat_d)); % Index of Y follower pos

%%% SOLVE! %%%
% Gradient calculation
[Fy,Fx]=gradient(-potential);

%%% Force in current point
F=[Fx(inx_N,iny_E),Fy(inx_N,iny_E)];
modF = max(10^-6,sqrt(F(1).^2+F(2).^2));

dL=(F./modF); % Descent direction in leader-centered system
diffangle=psiF-psiL;
R_LF=[cos(diffangle) sin(diffangle); -sin(diffangle) cos(diffangle)];
d=R_LF*dL';

% APF commands
l_Vclear=min(6,0.20*abs(F(2)))*d(2);
f_Vclear=min(5,0.15*abs(F(1)))*d(1);

end
```

6.2.3 Follower UAV - 2D errors calculation and PID tuning

`f_Vclear` and `l_Vclear`, that are velocities in *follower-centred reference frame*, can be used to calculate errors needed by formation PIDs. An overview of the error calculation block is depicted in Fig. A.15. Leader and followers NED velocities, obtained from their *Plant* blocks, are subtracted and rotated using the heading angle ψ_F:

$$\begin{bmatrix}\Delta V_{N,followerframe}\\ \Delta V_{E,followerframe}\end{bmatrix} = \boldsymbol{R}_{LF}\cdot\Delta\boldsymbol{V}_{NED} = \\ = \begin{bmatrix} cos(\psi_F) & sin(\psi_F)\\ -sin(\psi_F) & cos(\psi_F)\end{bmatrix}\begin{bmatrix}V_{N,F}-V_{N,L}\\ V_{E,F}-V_{E,L}\end{bmatrix} \tag{6.9}$$

Altitude error does not need to be rotated:

$$\Delta h = h_F - h_L \tag{6.10}$$

Formation errors are subsequently calculated:

$$\begin{cases} \dot{f}_{formationerror} = \dot{f}_{APF} - \Delta V_{N,followerframe} \\ \dot{l}_{formationerror} = \dot{l}_{APF} - \Delta V_{E,followerframe} \\ h_{formationerror} = h_{APF} - \Delta h \end{cases} \tag{6.11}$$

where $h_{APF} = 0\,m$ so that every follower can flight at the same altitude of the leader. This velocity errors obtained, they can be used in formation PIDs. The *Lateral velocity formation PID* (Fig. A.16) consist of a proportional, a derivative and an integral line. The output, $\Delta\varphi$, is limited between $[-30\,deg,\ 30\,deg]$ and it is added to the *Heading Autopilot* one (Fig. A.6). Thanks to a *trial-and-error* procedure, PID gains (Tab. 6.3) and step response (Fig. 6.5) can be obtained:

Lateral formation PID (velocity)
$KP\_ldot\_fPD = 0.15$
$KI\_ldot\_fPD = 0.03$
$KD\_ldot\_fPD = 0.05$
$Kpost\_ldot\_fPD = 2$

Table 6.3: Lateral formation PID (velocity) block gains

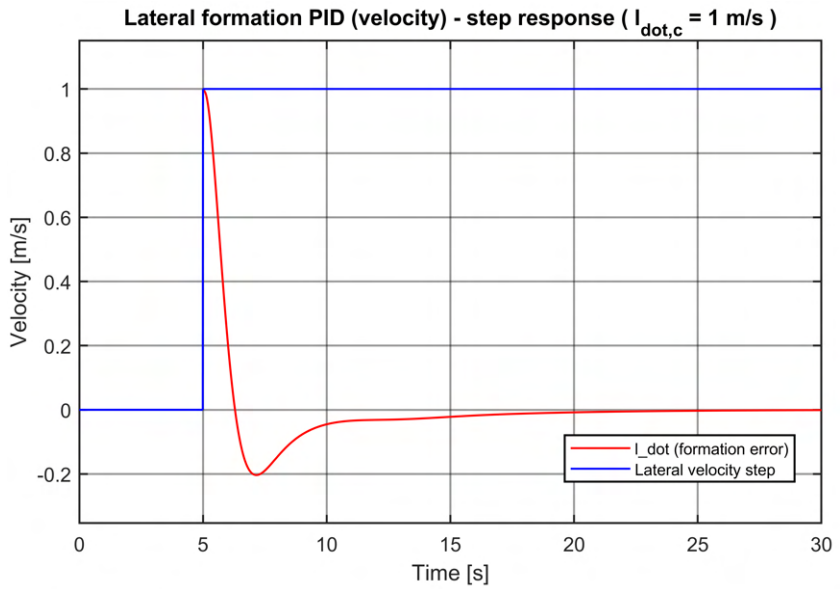


Figure 6.5: Lateral formation PID (velocity) block step response

The *Forward velocity formation PI* (Fig. A.17) does not include a derivative line inasmuch it has been observed that it causes signal disturbances. A *Saturation* has been inserted in the *Throttle LQR* (Fig. A.2) in order to limit the output between $[60\ \frac{m}{s},\ 140\ \frac{m}{s}]$ and the output is directly sent to the previous LQR. PID gains (Tab. 6.4) and step response (Fig. 6.6) are:

Forward formation PI (velocity)
$KP\_fdot\_fPD = 1$
$KI\_fdot\_fPD = 1$

Table 6.4: Forward formation PI (velocity) block gains

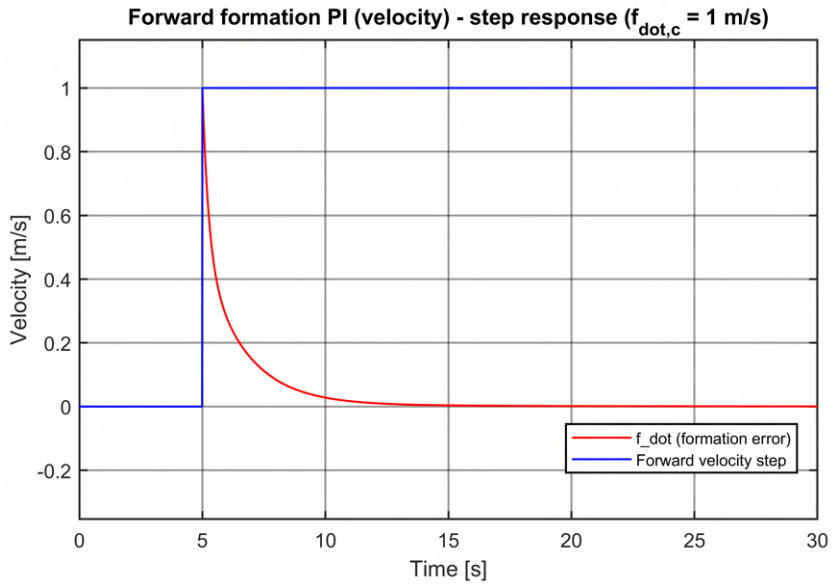


Figure 6.6: Forward formation PI (velocity) block step response

6.2.4 Simulation #1

The first simulation, similarly to Sect. 4.4, has been created using the `manoeuvre` string. In this case the *cooperative 2D APF-based formation control algorithm* is activated and drives each follower which follows the leader avoiding obstacles. The code used is included in Appendix B.3 while flag values for this scenario are:

- `manoeuvre` → `'manovra sim1 APF 2D'`
- `sectform` → `'sect sim1 APF 2D'`
- `obstacleflag` → `'no-flyzone sim1 APF 2D'`

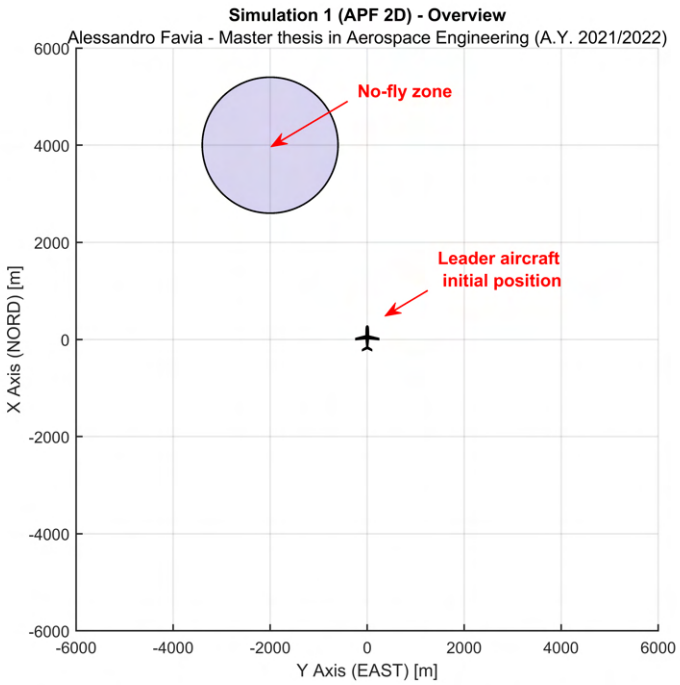
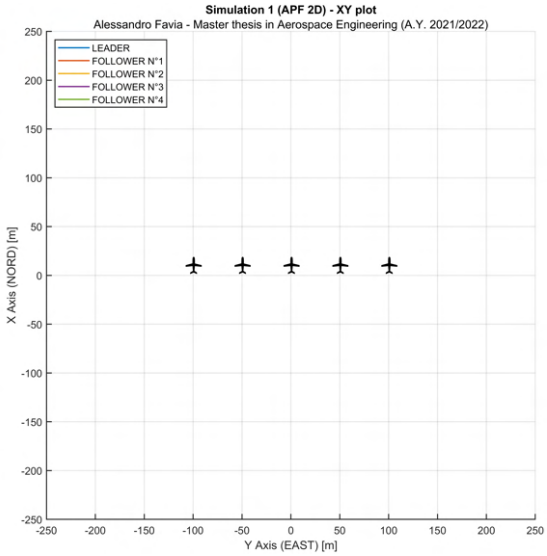
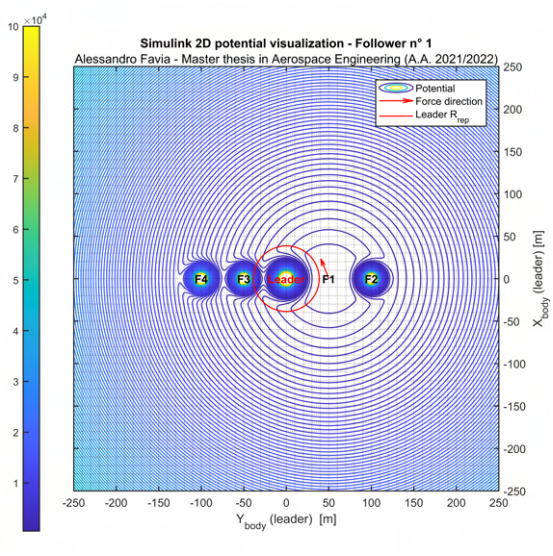


Figure 6.7: Simulation #1 overview (APF 2D)

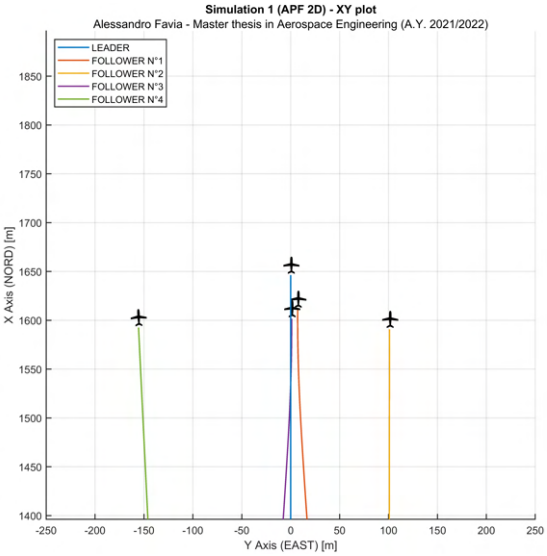
In Fig. 6.7 an overview of the scenario at $t = 0\,s$ is depicted. The leader aircraft position in NED reference frame is $[0{,}0, H_0]$ (where $H_0 = 3048\,m$) while followers are in *wall* formation. A *no-fly zone*, considered as a cylinder with infinite height and radius $R_{nfz} = 1400\,m$ has been centred in $C_{nfz} = [x_{nfz}, y_{nfz}] = [4000, -2000]\,m$. The *repulsive radius* of the *no-fly zone* has been calculated doubling R_{nfz}. The `manoeuvre` vector consists of three parts. First of all, every aircraft flights straight without any change in velocity. At $t = 40\,s$ a first turn is performed toward $\psi = -90\,deg$ avoiding the *no-fly zone* and at $t = 120\,s$ a second turn is performed toward $\psi = -180\,deg$. At the same time the formation patten changes using the the `sectform` vector. Starting from the *wall* formation, at $t = 5\,s$ a *vic* geometry is ordered while at $t = 120\,s$ the selected shape is *box*.



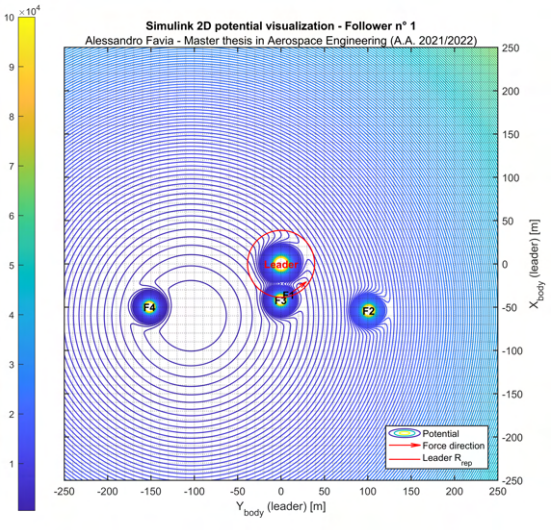
(a) S1 XY view - $t = 0\,s$



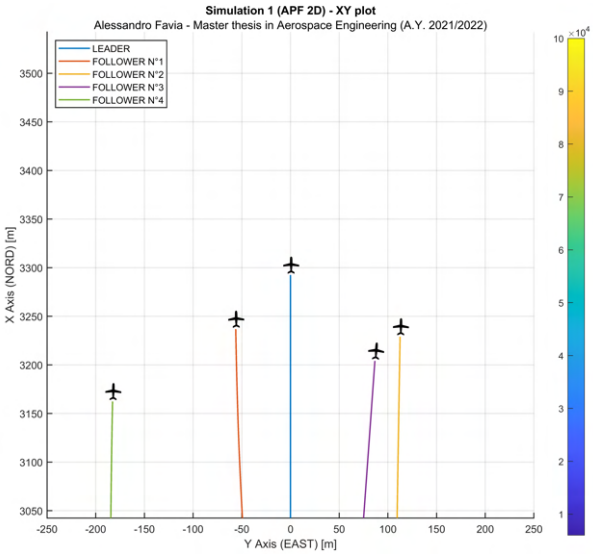
(b) S1 Potential visualization - $t = 0\,s$



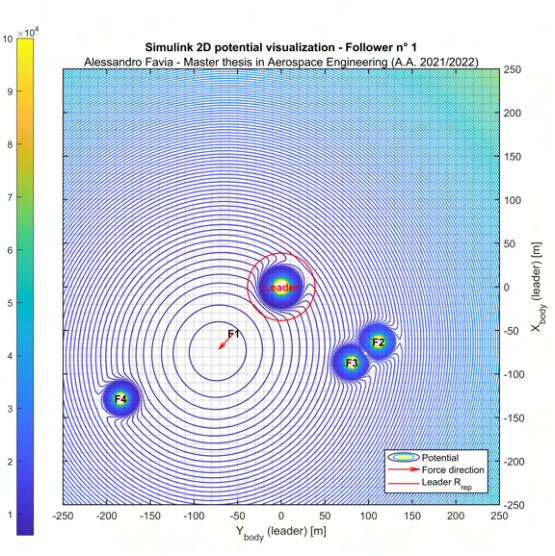
(c) S1 XY view - $t = 20\,s$



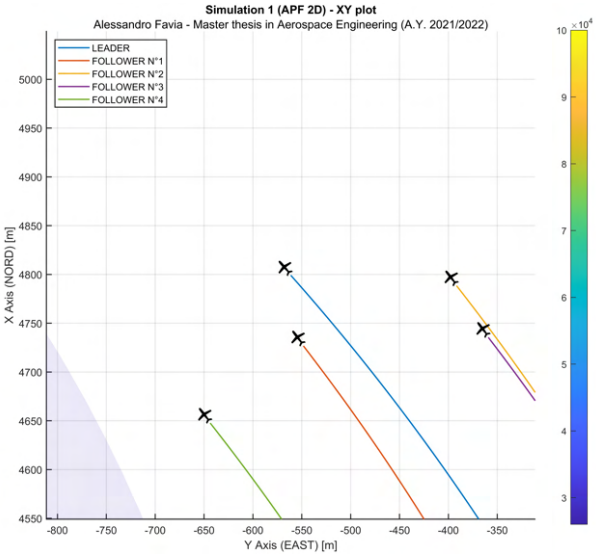
(d) S1 Potential visualization - $t = 20\,s$



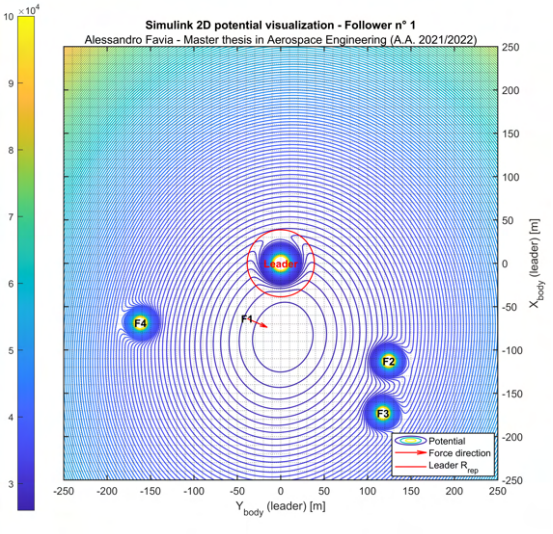
(e) S1 XY view - $t = 40\,s$



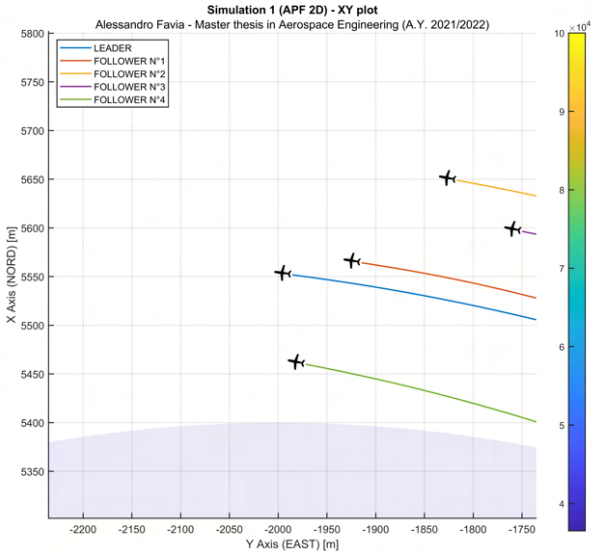
(f) S1 Potential visualization - $t = 40\,s$



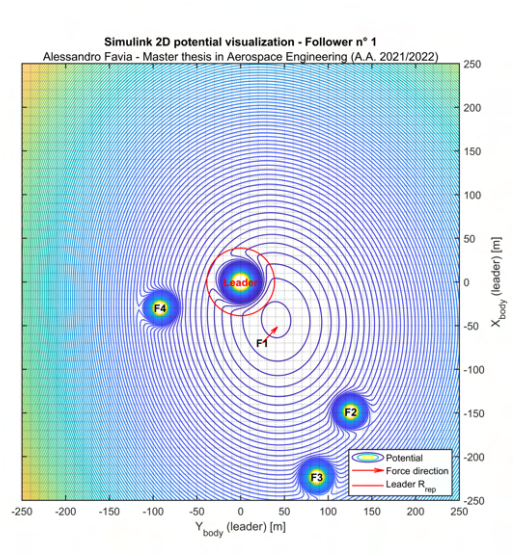
(g) S1 XY view - $t = 60\,s$



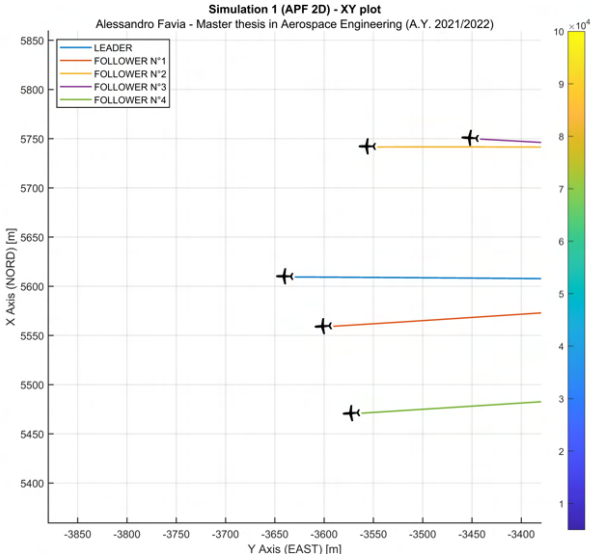
(h) S1 Potential visualization - $t = 60\,s$



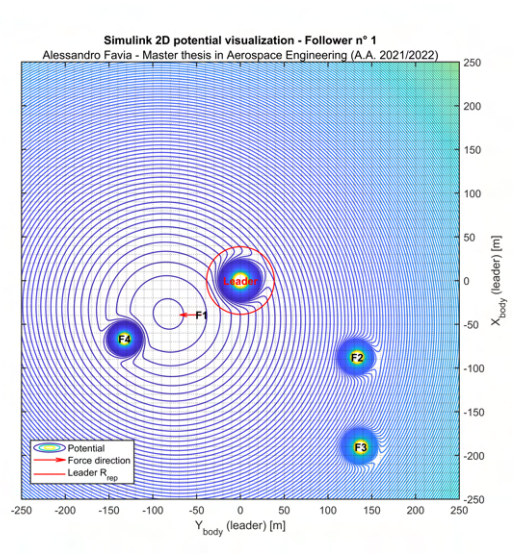
(i) S1 XY view - $t = 80\,s$



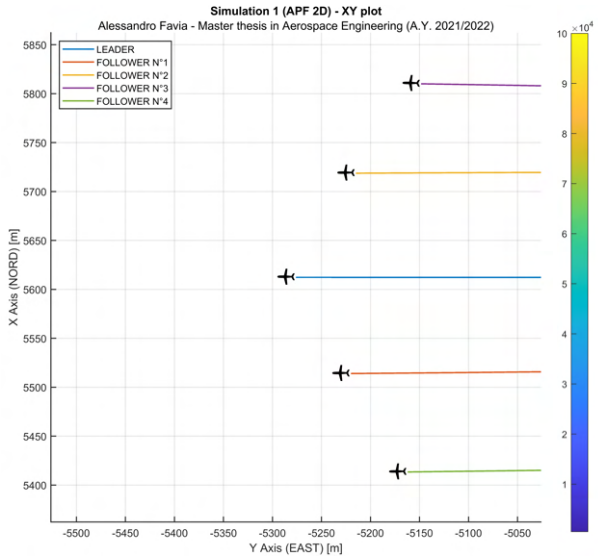
(j) S1 Potential visualization - $t = 80\,s$



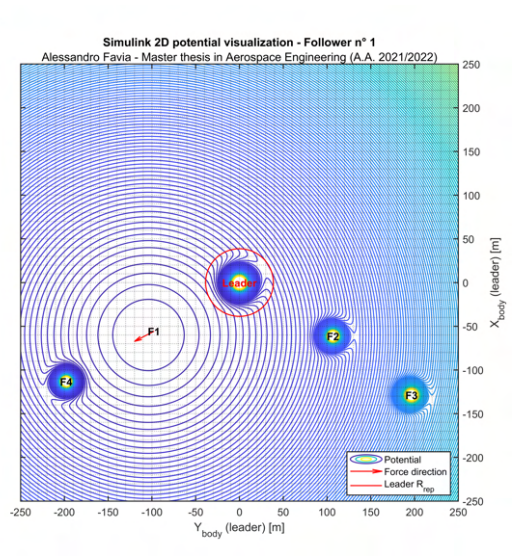
(k) S1 XY view - $t = 100\,s$



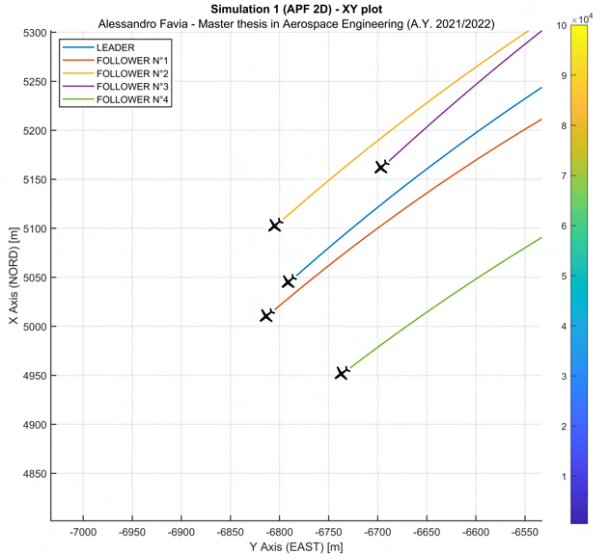
(l) S1 Potential visualization - $t = 100\,s$



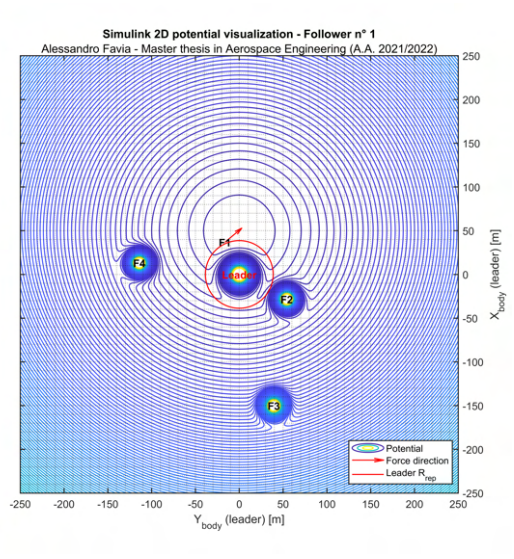
(m) S1 XY view - $t = 120\,s$



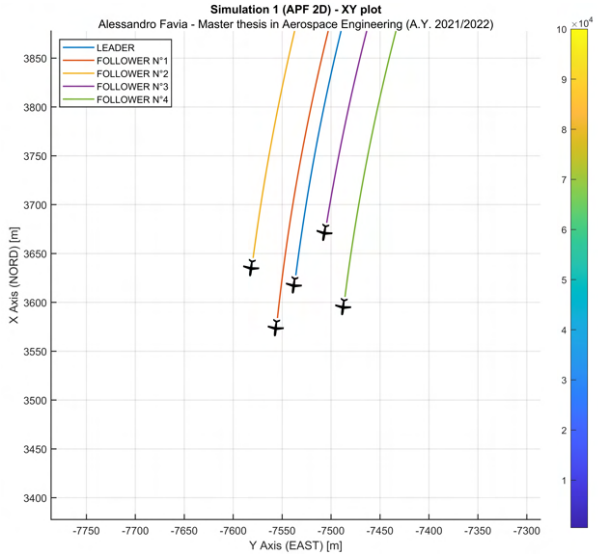
(n) S1 Potential visualization - $t = 120\,s$



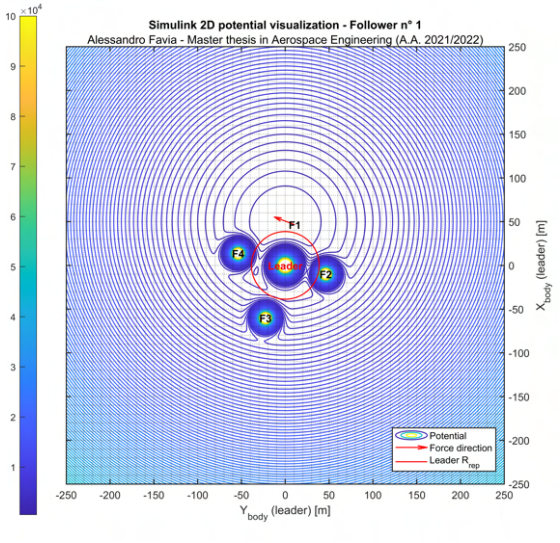
(o) S1 XY view - $t = 140\,s$



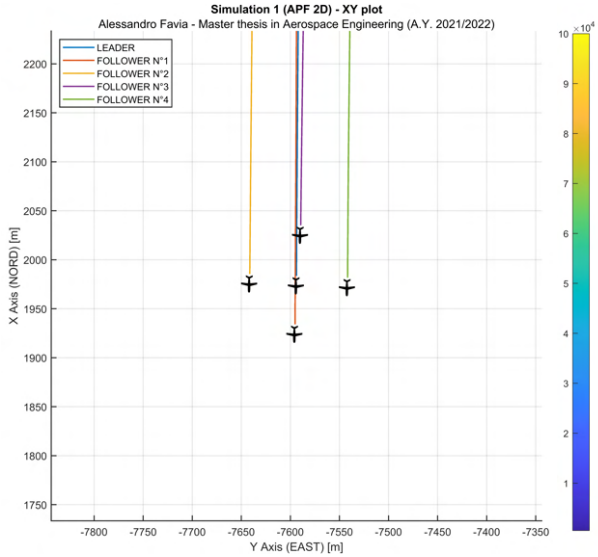
(p) S1 Potential visualization - $t = 140\,s$



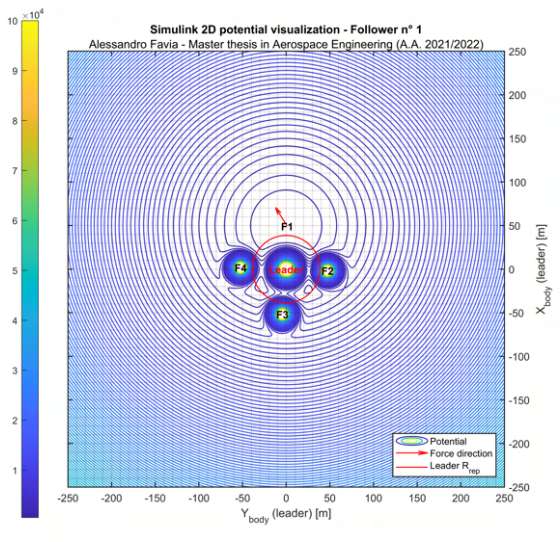
(q) S1 XY view - $t = 160\,s$



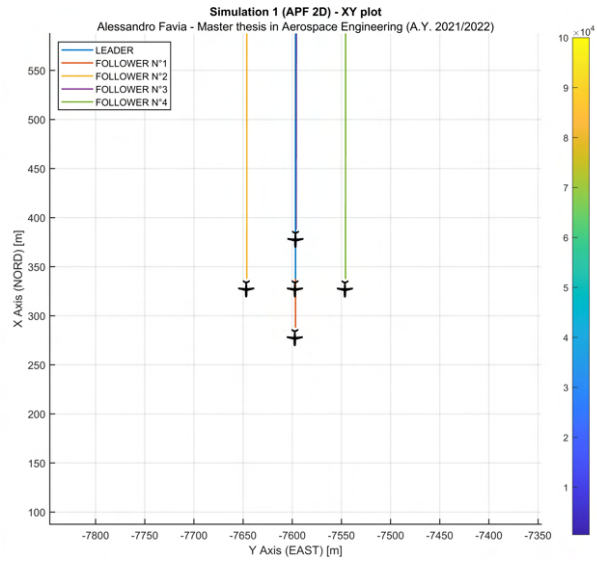
(r) S1 Potential visualization - $t = 160\,s$



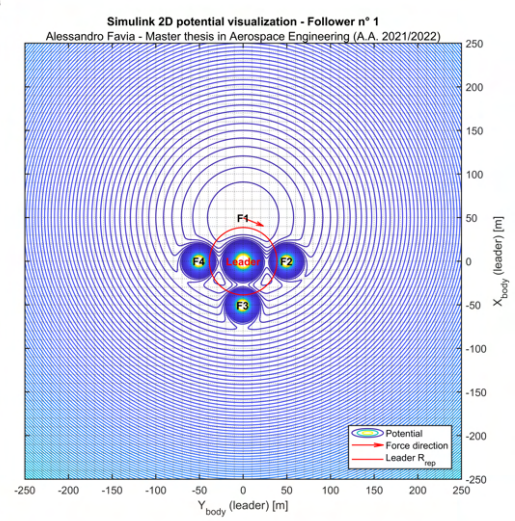
(s) S1 XY view - $t = 180\,s$



(t) S1 Potential visualization - $t = 180\,s$



(u) S1 XY view - $t = 200\,s$



(v) S1 Potential visualization - $t = 200\,s$

Figure 6.8: Simulation #1 results - 2D APF

In Fig. 6.8 results of the simulation can be observed. In particular, on the right side a *contour plot* represents the potential handled by the first follower in leader-centred reference frame and the leader repulsive circumference has been drawn. Around the other aircraft the repulsive part is recognisable by an increase in colour. For the sake of simplicity, we will refer to followers using abbreviations F1, F2, F3, F4 (e.g. F1 is the first follower). If the *wall* formation at $t = 0\,s$ can be easily identified in (a)(b) (F1 is already in its target point), the formation subsequently receives the order to generate the *vic* pattern. In (c)(d) we can see that F2 has successfully reached its position while F1, F3 and F4 repel themselves. The closest distance between F1 and F3 obtained during the manoeuvre strongly depends on attractive/repulsive constants, APF gains and aircraft inertia and could be reduced modifying these values. The *vic* pattern at $t = 40\,s$ (e)(f) is not complete yet but the leader begins its left turn. The *no-fly zone* influences the formation, as can be seen in (g)(h): the target point, which would be the same of (f) if no obstacles existed, moves right and push F1 away from the *no-fly zone*. In (i)(l) the first follower is on the right side of the leader while it should be on the left according to the *vic* formation. When the obstacle has been avoided and the heading direction is the desired one ($\psi = -90\,deg$ for the first turn), the correct *vic* shape is recovered (m)(n). At this point a second turn and a *box* shape are ordered and in the last images it can be seen how aircraft succeed in it, despite the short distance between them. This simulation demonstrates how this algorithm works in a complex scenario. Every follower tends to stay close to the leader and avoid obstacles with the possibility to change the formation shape in every moment. Varying few constants, the algorithm could be easily adapted to every type of aircraft because it does not depend directly to the plant geometry.

6.2.5 Simulation #2

In the second simulation, differently from the other ones, the `manoeuvre` vector has not been used. In this case the leader is controlled using a *Thrustmaster TCA Sidestick Airbus edition joystick*, whose Simulink block can be easily observed in Fig. 3.13. Being a 2D APF algorithm working on XY plane, only `roll_joystick` and `velocity_joystick` commands have been activated while `pitch_joystick` and `beta_joystick` ones have been set to zero. The `joystickflag` value has been set to 2 (as needed by the *Multiport Switch*). Using a combination of *To Workspace* and *From Workspace* blocks active outputs of the joystick have been saved in order to execute the same simulation multiple times. The code is included in Appendix B.4 while flag values are:

- `manoeuvre` → useless
- `sectform` → `'sect sim2 APF 2D'`
- `obstacleflag` → `'no-flyzone sim2 APF 2D'`

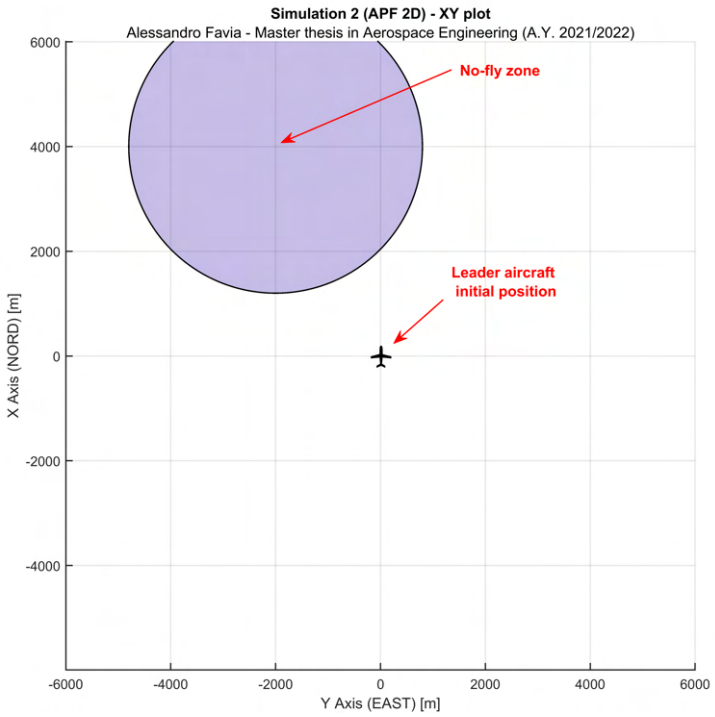


Figure 6.9: Simulation #2 overview (APF 2D)

In Fig. 6.9 an overview of the scenario at $t = 0\,s$ is depicted. As for the previous simulation, the leader aircraft position in NED reference frame is $[0{,}0, H_0]$ while followers are in *wall* formation. Using `sectform` the formation pattern autonomously changes two times: at $t = 50\,s$ the selected geometry is *box* while at $t = 100\,s$ it is *echelon*. The

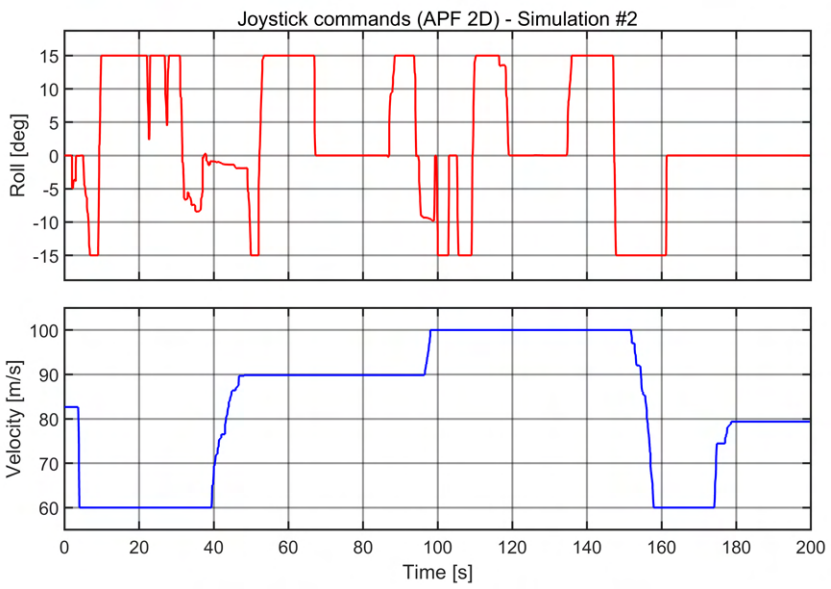
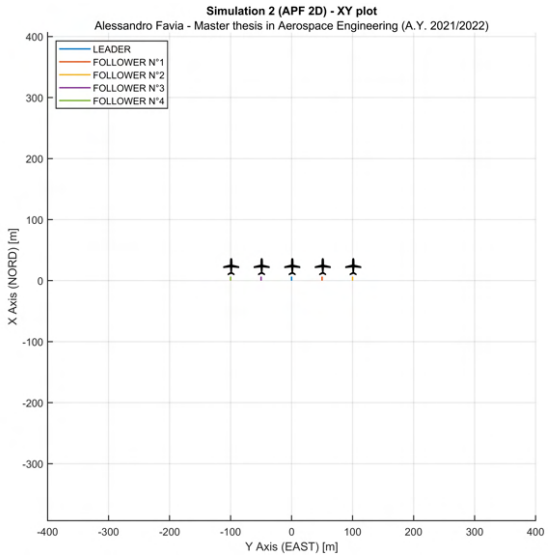
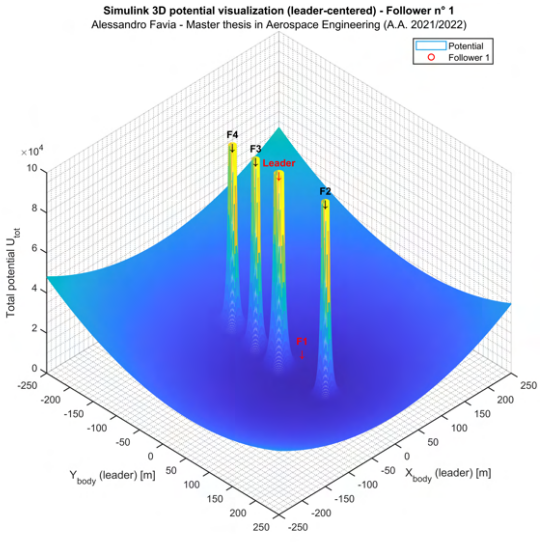


Figure 6.10: S2 Joystick commands

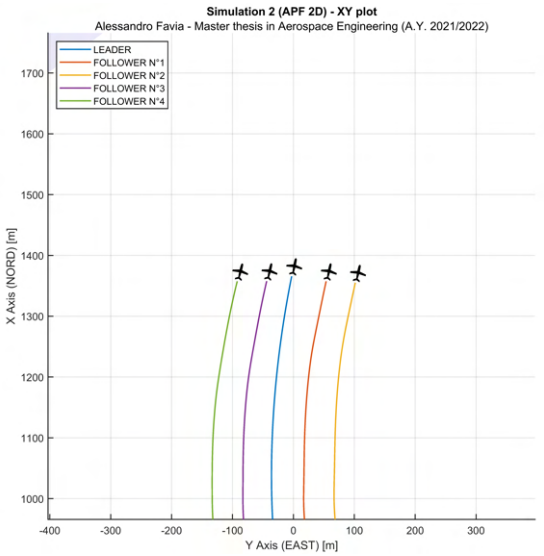
no-fly zone is always centred in $C_{nfz} = [x_{nfz}, y_{nfz}] = [4000, -2000]\, m$ also if its radius has been doubled. Joystick commands are shown in Fig. 6.10 where can be noticed that the roll angles has been limited to $\pm 15\, deg$ and velocities between $[60 - 100]\, \frac{m}{s}$. The initial velocity, being controlled with the joystick *slider*, has been set to $V_0 = 82.6566\, \frac{m}{s}$.



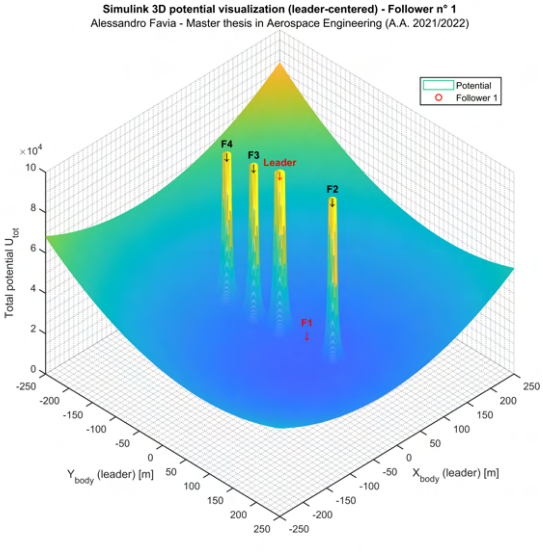
(a) S2 XY view - $t = 0\, s$



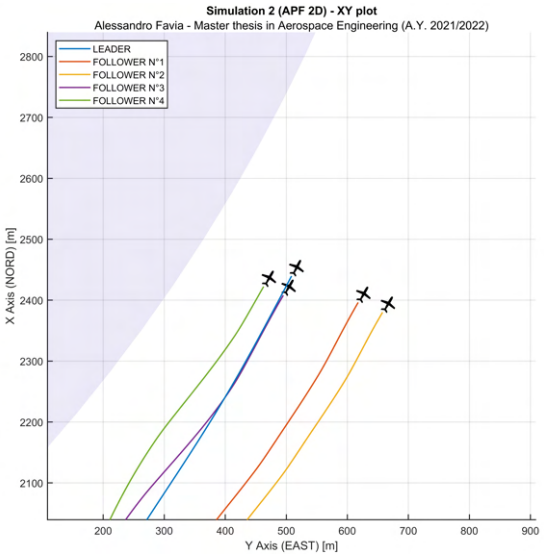
(b) S2 Potential visualization - $t = 0\, s$



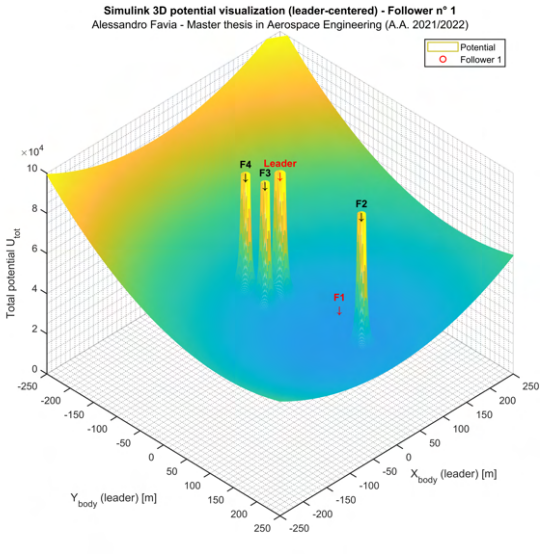
(c) S2 XY view - $t = 20\,s$



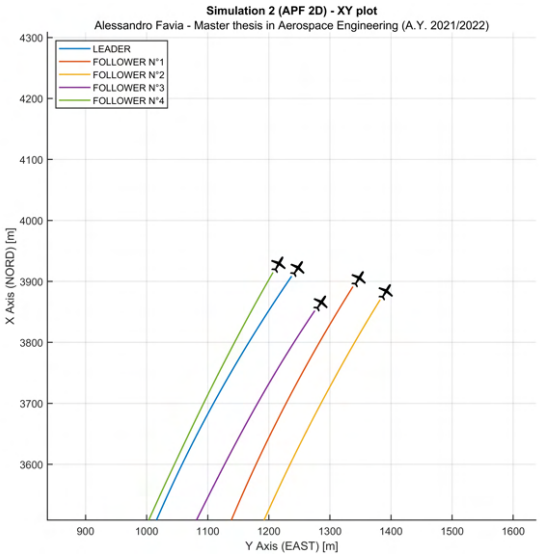
(d) S2 Potential visualization - $t = 20\,s$



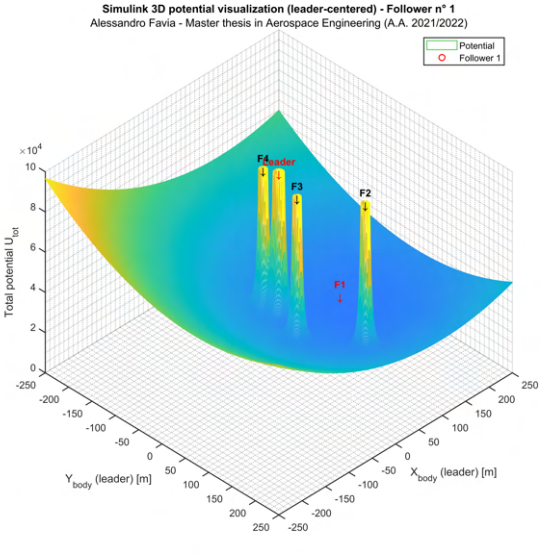
(e) S2 XY view - $t = 40\,s$



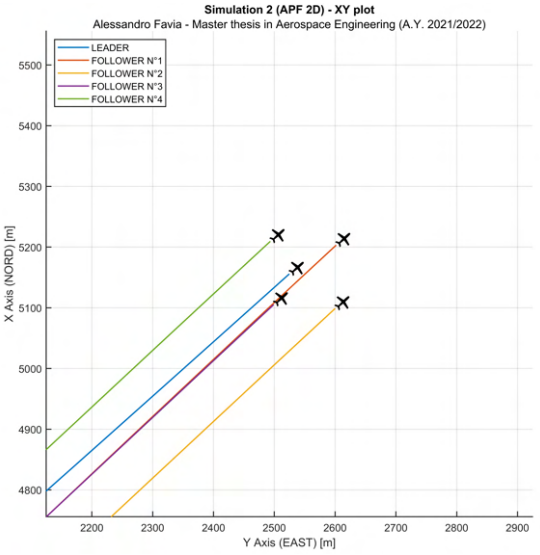
(f) S2 Potential visualization - $t = 40\,s$



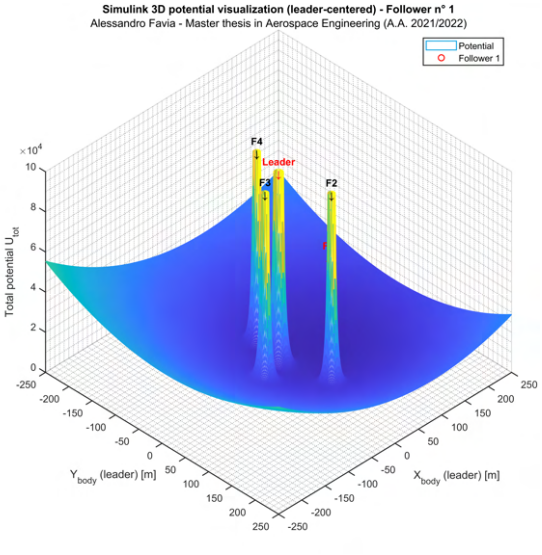
(g) S2 XY view - $t = 60\,s$



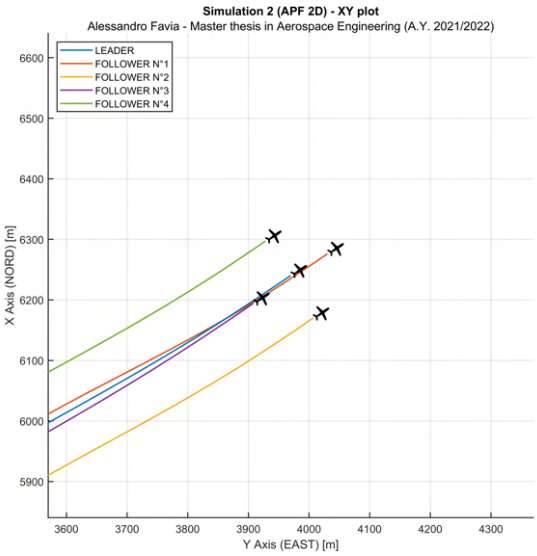
(h) S2 Potential visualization - $t = 60\,s$



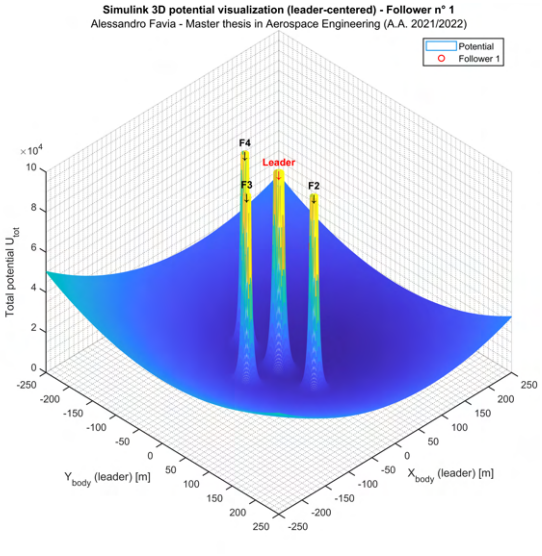
(i) S2 XY view - $t = 80\,s$



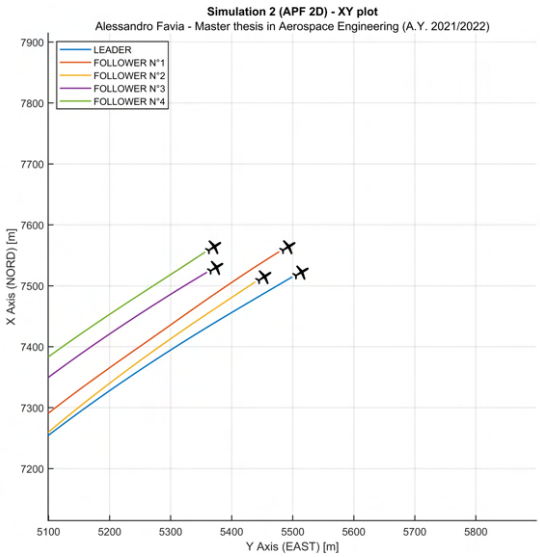
(j) S2 Potential visualization - $t = 80\,s$



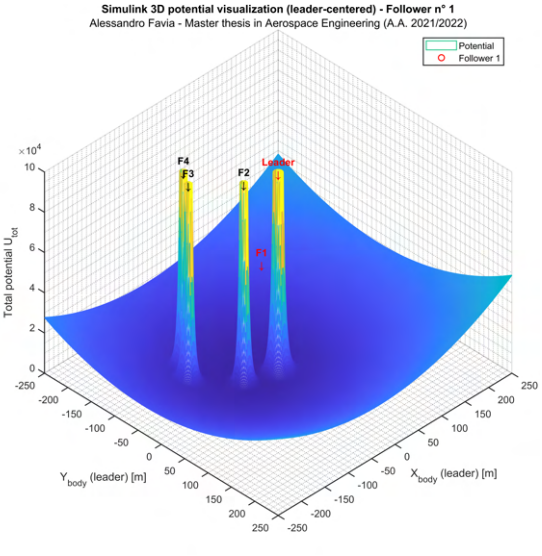
(k) S2 XY view - $t = 100\,s$



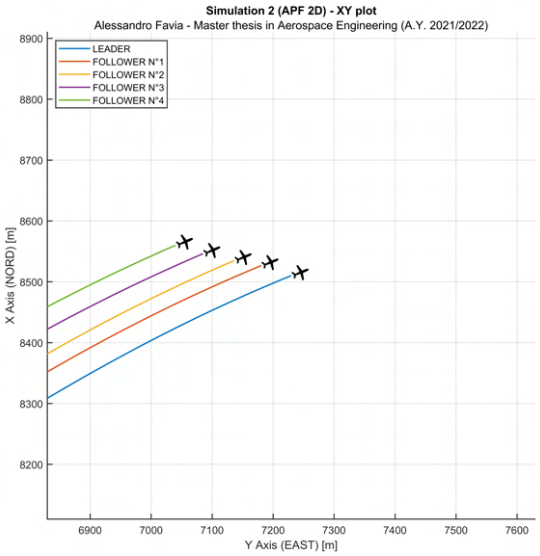
(l) S2 Potential visualization - $t = 100\,s$



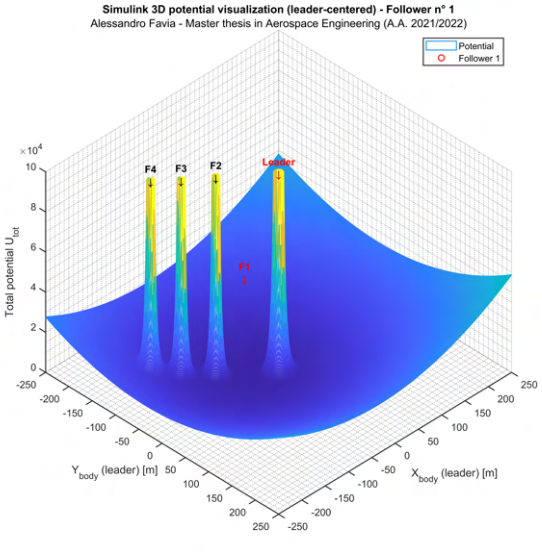
(m) S2 XY view - $t = 120\,s$



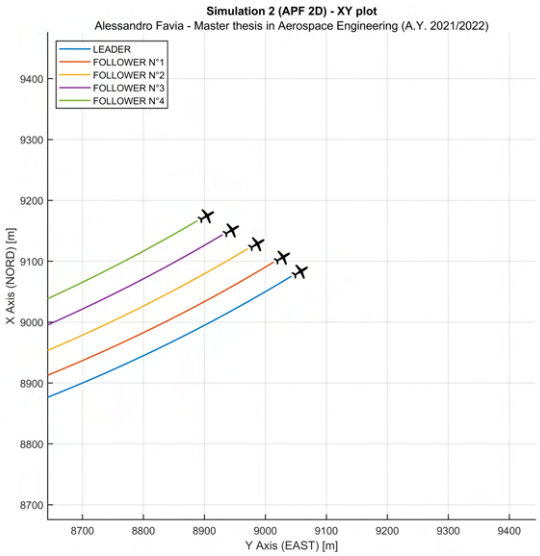
(n) S2 Potential visualization - $t = 120\,s$



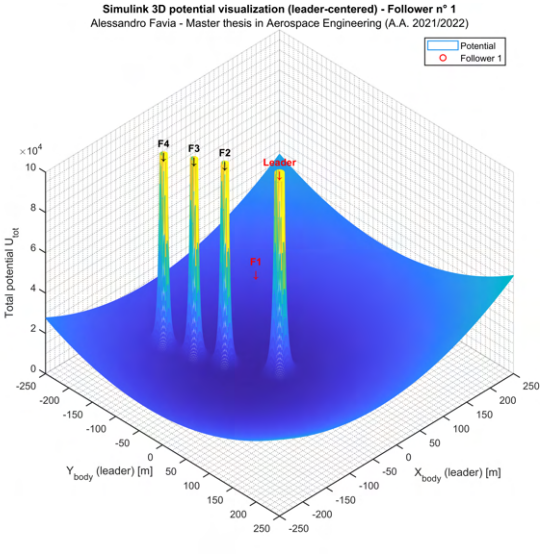
(o) S2 XY view - $t = 140\,s$



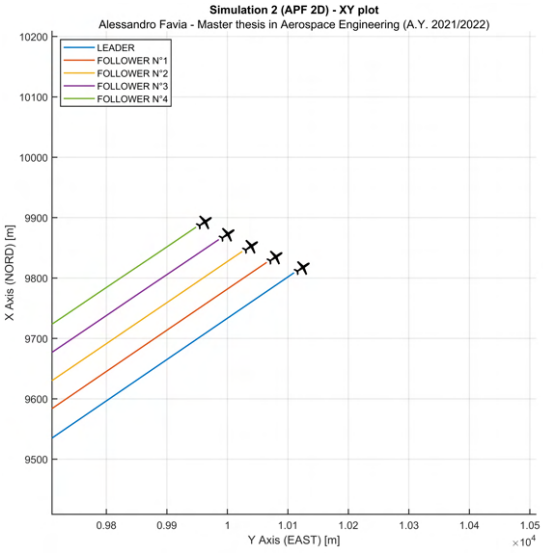
(p) S2 Potential visualization - $t = 140\,s$



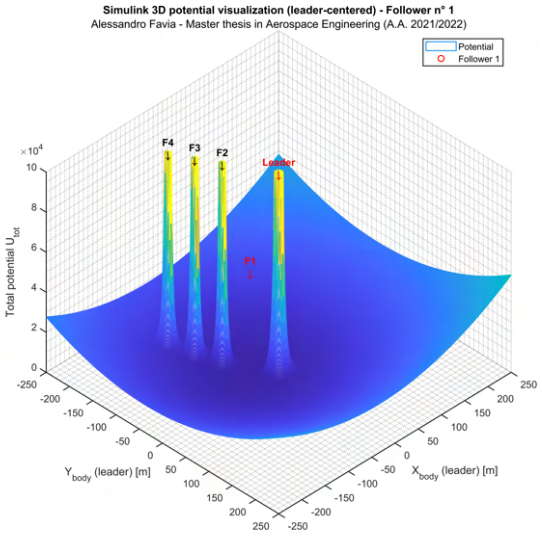
(q) S2 XY view - $t = 160\,s$



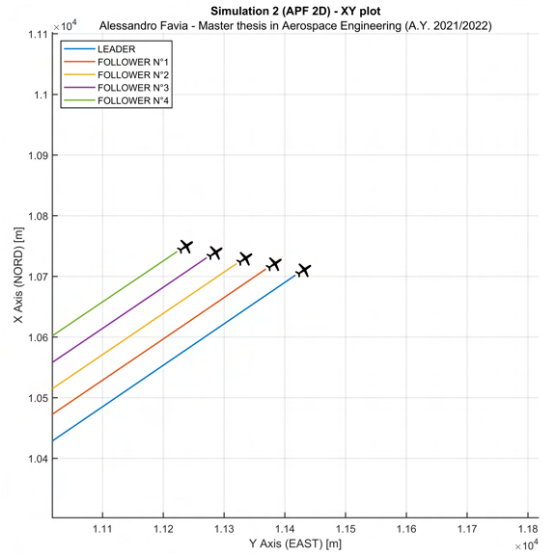
(r) S2 Potential visualization - $t = 160\,s$



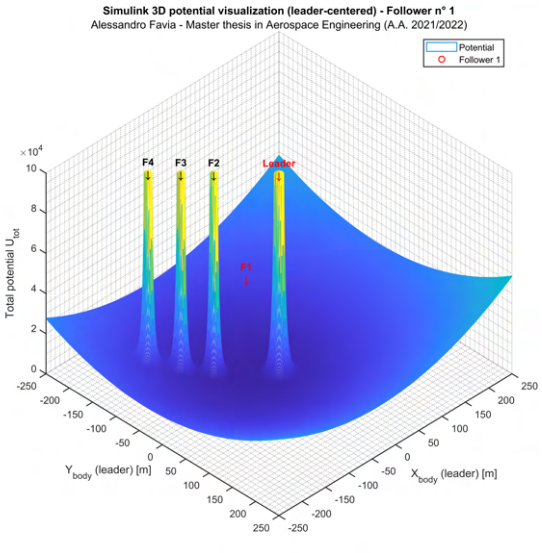
(s) S2 XY view - $t = 180\,s$



(t) S2 Potential visualization - $t = 180\,s$



(u) S2 XY view - $t = 200\,s$



(v) S2 Potential visualization - $t = 200\,s$

Figure 6.11: Simulation #2 results - 2D APF

The scenario will be now described. Differently from Simulation #1, the potential will be depicted using *surfaces* instead of *contour lines*. Joystick roll commands (Fig. 6.9(b)) are affected by joystick sensitivity and they may seem confusing, but thanks to aircraft inertia only held commands affect leader trajectory significantly. At $t = 0\,s$, as it has previously mentioned, the formation pattern is a *wall* type (Fig. 6.11(a)). Joystick commands correspond to initial conditions of the leader, so nothing change. At $t \simeq 5\,s$ a short left turn is executed and a velocity of $60\,\frac{m}{s}$ is ordered, followed by a right turn basically held for $\simeq 10\,s$. Observing Fig. 6.11(c)(d), the *wall* formation begins to warp due to the *no-fly zone* whose repulsive radius R_{pot} is equal to $5600\,m$ (it has been obtained doubling the actual physical radius $R_{obs} = 2800\,m$). The leader is piloted so that the *no-fly zone* can be avoided and its repulsive action become stronger as we approach C_{nfz}. At $t = 40\,s$ (Fig. 6.11(e)(f)) a left turn is executed in order to flank the obstacle and velocity is gradually increased to $V \simeq 90\,\frac{m}{s}$. Comparing Fig 6.11(b) and Fig. 6.11(f) it can be noticed how potential values are generally higher in the second case due to the *no-fly zone*. While F1 and F2 succeed in maintaining a quite undisturbed trajectory, F3 and F4 are conditioned by the presence of the leader and they are forced to fly very close to each other. As the distance from C_{nfz} increases the formation recovers its *wall* pattern until, at $t = 50\,s$, the *box* one is ordered: followers *target points* (i.e. relative positions in a *leader-centred reference frame*) change. Comparing Fig. 6.11(f) and Fig. 6.11(h) it can be observed how, for F1, the initial $TP_{F1,wall} = [0; 50]\,m$ becomes $TP_{F1,box} = [75; 0]\,m$. Once the obstacle has been overtaken, the *box* pattern can be easily recognized (Fig. 6.11(i)(j)) and it is preserved despite alternate left and right turns. At $t \simeq 100\,s$ (Fig. 6.11(k)(l)) a velocity of $V = 100\,\frac{m}{s}$ is selected and aircraft start generating the *echelon* geometry while the leader executes various turns (Fig. 6.11(m)(n)(o)(p)). At $t \simeq 150\,s$ the leader velocity is first reduced to $V = 60\,\frac{m}{s}$ then increased to $V \simeq 80\,\frac{m}{s}$ ($t \simeq 170\,s$) and it can be noticed how followers succeed in keeping the formation in every situation (Fig. 6.11(q)(r)(s)(t)(u)(v)). Thanks to this simulation the reliability of the algorithm has been demonstrated: every aircraft can maintain or change the formation shape during fast manoeuvres, avoiding obstacles. It should be specified that results could vary if different type of aircraft in the same formation are implemented. In this case, in fact, the algorithm should consider differences of *flight envelopes* and act accordingly.

6.2.6 Simulation #3

The main purpose of the third simulation is to demonstrate a particular feature of the algorithm, i.e. the possibility to split the formation so that a follower can fly over a specific point on the map. As for *Simulation #2*, the *Thrustmaster TCA Sidestick Airbus edition joystick* has been used to control roll angle and velocity of the leader (Fig. 3.13) and all the considerations made at the beginning of Sect. 6.2.5 are valid. Some differences with the previous model, however, should be highlighted. No obstacles are included in the scenario so every follower must avoid the other aircraft only. Flag values are:

- `manoeuvre` → useless
- `sectform` → included in the *Simulink model*
- `obstacleflag` → useless

The first important difference is that the `sectformfig` vector, which consist of the *target points* of every follower, is not created using the initialization code. *Target points* are generally defined, in the section formation `switch` structure, in a *leader-centred reference frame*. If we want a follower to reach a specific point on the map, it will reasonably provide in a *NED reference frame* so a *rototranslation* will be necessary. Thus, `sectformfig` depends on variables (x_L, y_L, ψ_L) which vary over time and it has to be recalculated every time step. This is done using the block in Fig. 6.12.

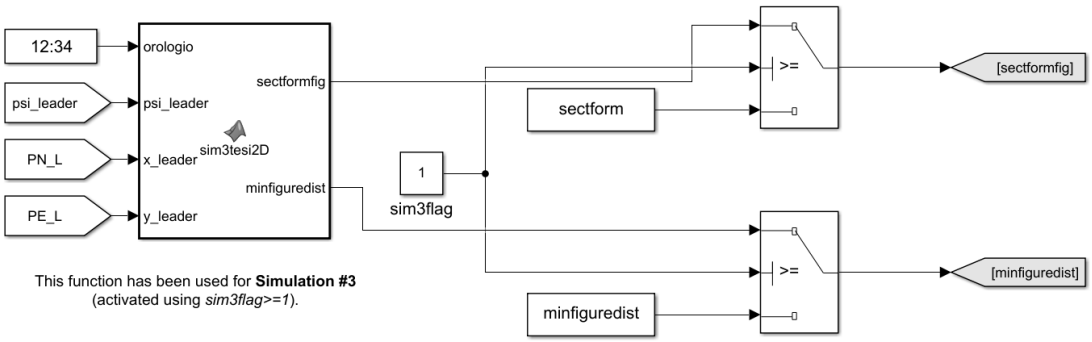


Figure 6.12: Simulation #3 *sectformfig* creation

The `sim3tesi2D` function code, included in Appendix B.6, is now described. The leader aircraft position in NED reference frame at $t = 0\,s$ is $[0{,}0, H_0]$ while followers are in *vic* formation, which is maintained for five seconds. At $t > t_2 = 5\,s$ the target point of the first follower changes, since its purpose is to reach the point (NED coordinates):

$$TP_{F1,NED} = [x_{TP,F1,NED}\,;\, y_{TP,F1,NED}] = [8000\,;\, 1250]\,m$$

This point is translated in the *leader-centred reference frame*. The `elseif` condition is characterized by:

```
(orologio>t2)&& ((8000-x_leader)>100 && (1250-y_leader)>100)
```

Thanks to this line of code, the first follower if forced to reach $TP_{F1,NED}$ until its forward and lateral distance from the same point is bigger than $100\,m$. If this condition is not met, the aircraft has successfully flew over $TP_{F1,NED}$. At this point a *echelon* figure is requested to all the followers.
The grid, in this case, should be larger than the other ones used in *Simulation #1* and *#2*. For this reason the `half_grid_ext_X` and `half_grid_ext_Y` values are equal to $1500\,m$ with `gridpointdist` equal to $6\,m$ (to avoid long computational times). The `PotentialSolver_AF_sim` (originally depicted in Listing 6.7) has been modified too, as can be seen in Listing 6.8. In order to force the follower to reach rapidly the point on the map $TP_{F1,NED}$, APF outputs depends on the absolute value of the distance between the target point and the current position of the follower. In so doing if the aircraft is far from its target point it tries to approach it in the shortest possible time while, if it is close enough, the APF formation control algorithm generates small commands in order to avoid *overshoots*. Joystick commands are shown in Fig. 6.13 where the initial velocity, controlled using a *slider*, has been set to $82.5003\,\frac{m}{s}$. As for *Simulation #2*, roll angle and velocity are limited between $[-15; 15]\,deg$ and $[60; 100]\,\frac{m}{s}$ respectively.

Listing 6.8: *Potential Solver* (*Simulink version* used in *Simulation #3*)

```
function [l_Vclear,f_Vclear,dL] = PotentialSolver_AF_sim(x_grid_N,
    y_grid_E,potential,Fforw_d,Flat_d,psiL,psiF)
%%% PARAMETERS INITIALIZATION %%%
% Find index of follower in leader-centered ref system
[~,inx_N]=min(abs(x_grid_N(:,1)-Fforw_d)); % Index of X follower pos
[~,iny_E]=min(abs(y_grid_E(1,:)-Flat_d)); % Index of Y follower pos

%%% SOLVE! %%%
% Gradient calculation
[Fy,Fx]=gradient(-potential);

%%% Force in current point
F=[Fx(inx_N,iny_E),Fy(inx_N,iny_E)];
modF = max(10^-6,sqrt(F(1).^2+F(2).^2));

dL=(F./modF); % Descent direction in leader-centered system
diffangle=psiF-psiL;
R_LF=[cos(diffangle) sin(diffangle); -sin(diffangle) cos(diffangle)];
d=R_LF*dL';

% APF commands (modified for Simulation 3)
if abs(targetpoint(2)-Flat_d)<50
    l_Vclear=min(6,0.15*abs(F(2)))*d(2);
else
    l_Vclear=min(24,15*abs(F(2)))*sign(d(2));
end
if abs(targetpoint(1)-Fforw_d)<50
    f_Vclear=min(5,0.2*abs(F(1)))*d(1);
else
    f_Vclear=min(21,10*abs(F(1)))*sign(d(1));
end
end
```

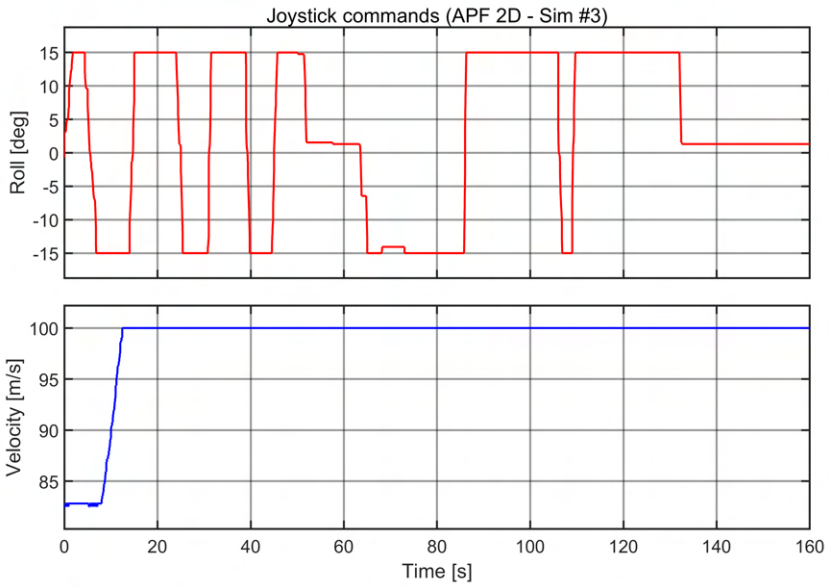
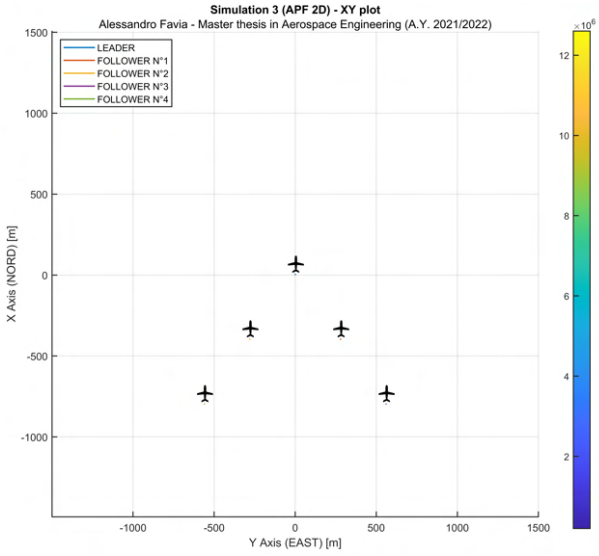
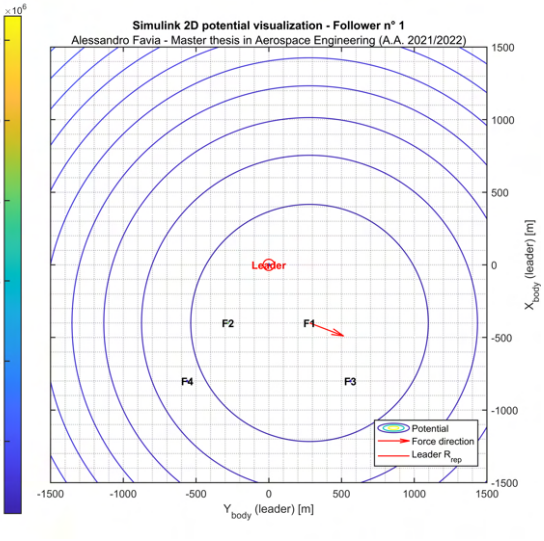


Figure 6.13: S3 Joystick commands

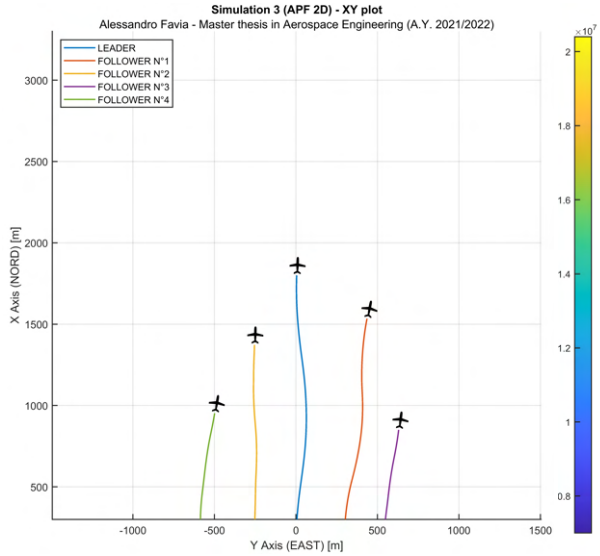
The simulation is now described observing Fig. 6.14, where both XY plot and potential of the first follower are depicted. For the sake of brevity, we will refer to followers using abbreviations (F1, F2, F3, F4). At $t = 0\,s$ (Fig. 6.14(a)(b)) the formation is a *vic* type, with larger distances then other scenarios in order to improve visualization. The leader aircraft performs right and left turns in sequence during all the simulation with a speed variation from $82.5003\,\frac{m}{s}$ to $100\,\frac{m}{s}$ at $t \simeq 10\,s$ (Fig. 6.13). At $t > t_2 = 5\,s$ the *target point* of F1 changes and F1 starts to move away from the formation: in Fig. 6.14(d)(f)(h) isolines are centred in $TP_{F1,NED}$ and the force direction points towards it. Its trajectory, as can be observed from Fig. 6.14(c)(e)(g), is not straight: this is probably caused by the high value of `gridpointdist` which cause a lower accuracy in gradient calculation. At $t \simeq 70\,s$, F1 flies nearby $TP_{F1,NED}$ (the purple area in Fig. 6.14(i) is centred in $TP_{F1,NED}$ and its radius is equal to $50\,m$). When the `elseif` condition of `sim3tesi2D` function (Appendix B.6) is met, a *echelon* formation is commanded to all the followers (Fig. 6.14(k)(l)). Using the joystick a right turn has been performed in order to hinder the formation control algorithm of F1 (Fig. 6.14(m)(n)) but, despite that, it has succeeded in reaching the correct position (Fig. 6.14(o)(p)(q)(r)). This simulation demonstrates how the algorithm that has been created in this thesis can be useful to control a follower aircraft which is not in close proximity of the leader. In future studies it will be necessary to understand how APF commands `f_Vclear` and `l_Vclear` can be further optimized, reducing the time needed to reach the target with a smooth trajectory.



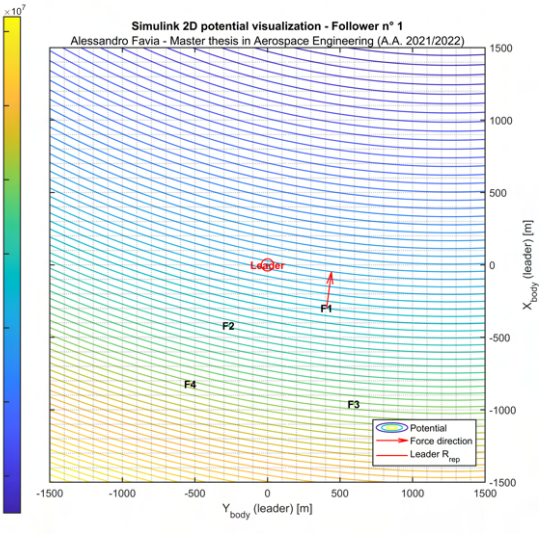
(a) S3 XY view - $t = 0\,s$



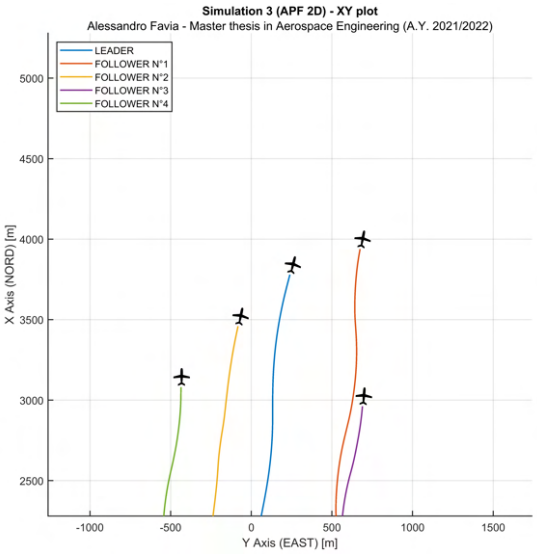
(b) S3 Potential visualization - $t = 0\,s$



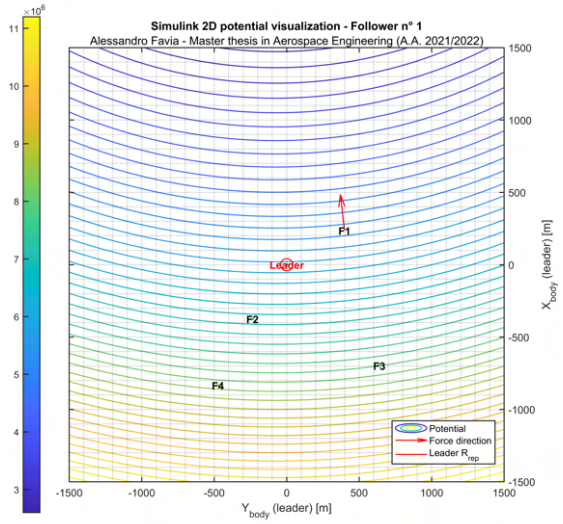
(c) S3 XY view - $t = 20\,s$



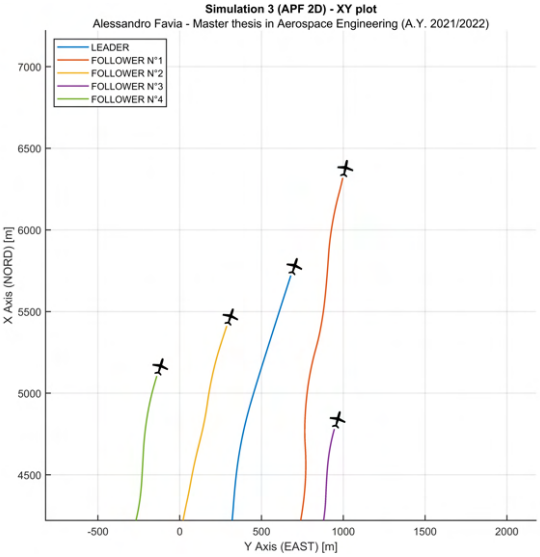
(d) S3 Potential visualization - $t = 20\,s$



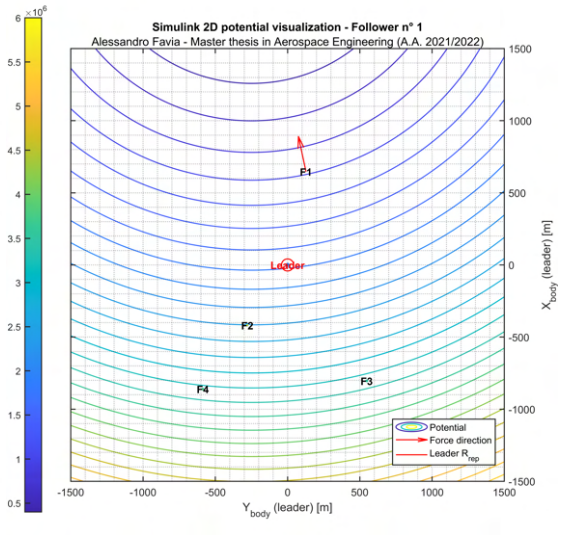
(e) S3 XY view - $t = 40\,s$



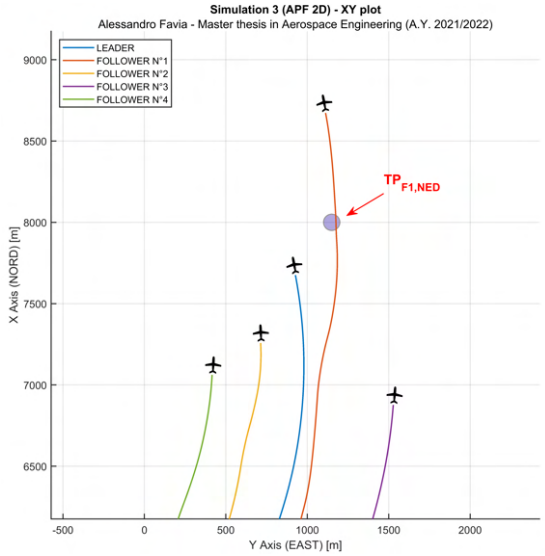
(f) S3 Potential visualization - $t = 40\,s$



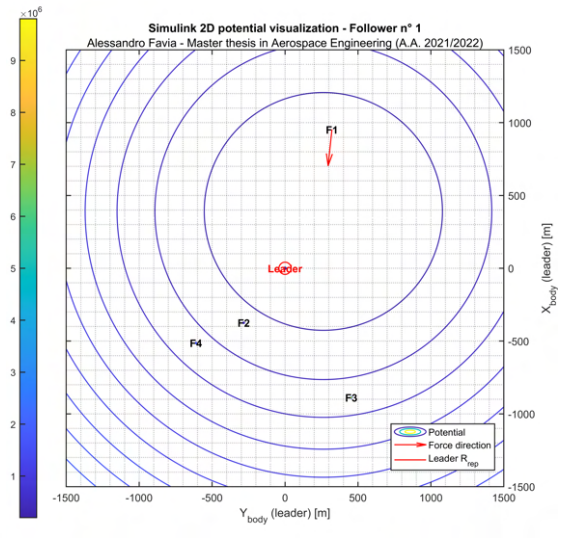
(g) S3 XY view - $t = 60\,s$



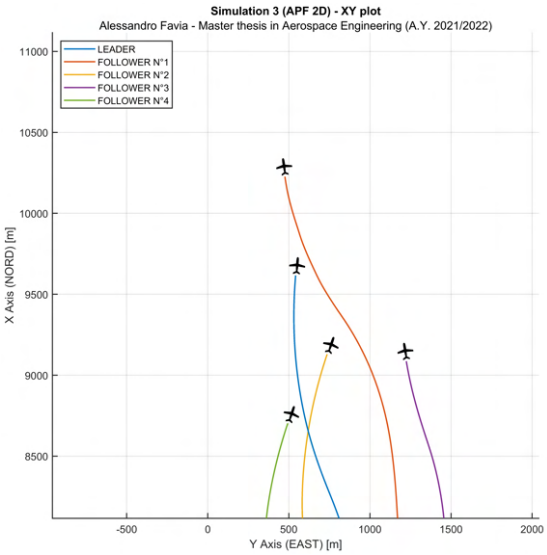
(h) S3 Potential visualization - $t = 60\,s$



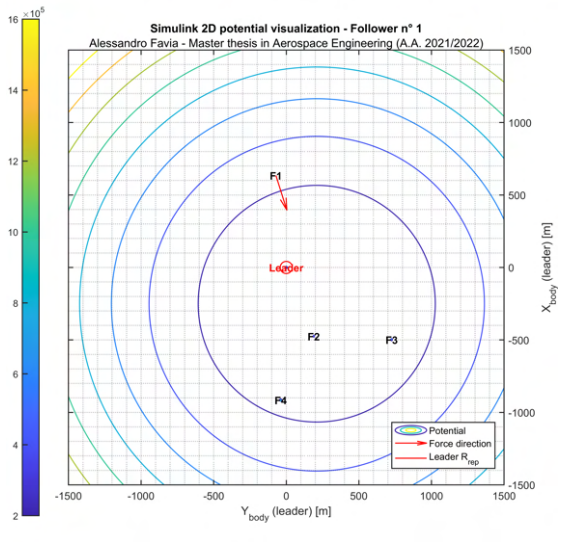
(i) S3 XY view - $t = 80\,s$



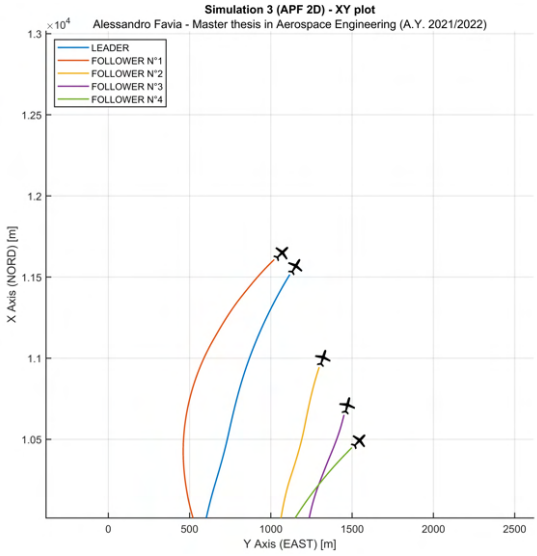
(j) S3 Potential visualization - $t = 80\,s$



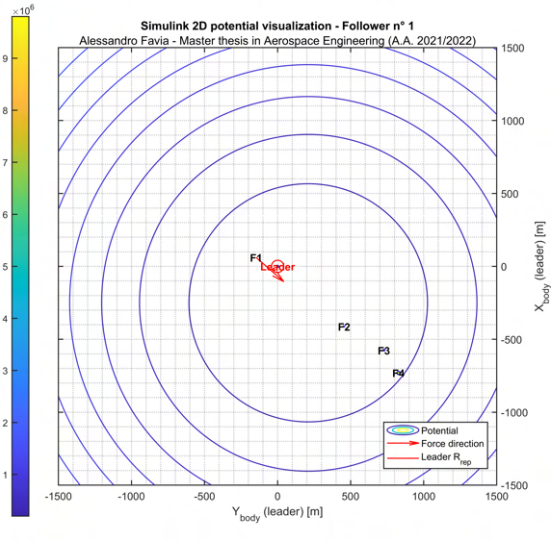
(k) S3 XY view - $t = 100\,s$



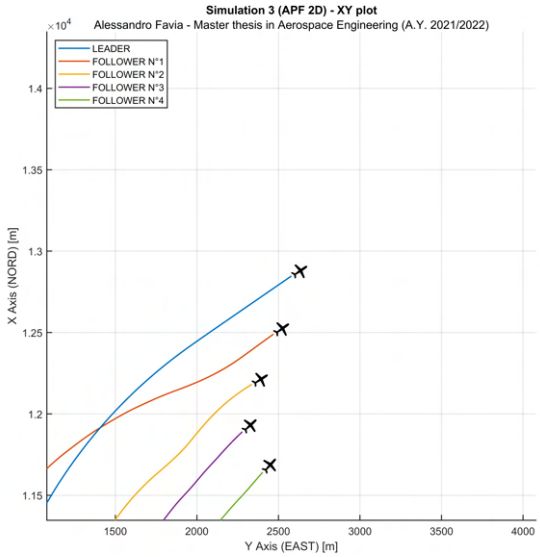
(l) S3 Potential visualization - $t = 100\,s$



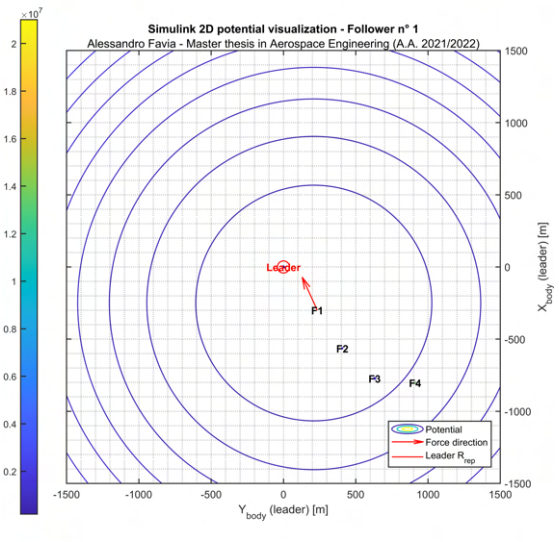
(m) S3 XY view - $t = 120\,s$



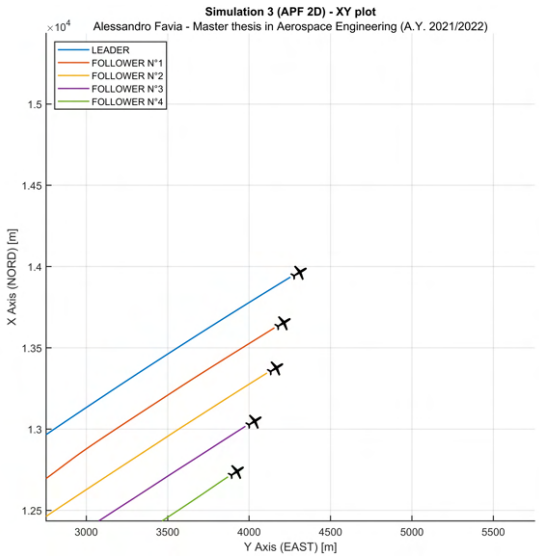
(n) S3 Potential visualization - $t = 120\,s$



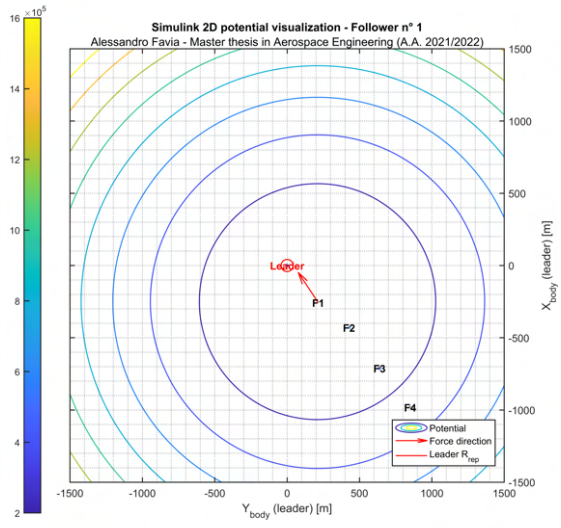
(o) S3 XY view - $t = 140\,s$



(p) S3 Potential visualization - $t = 140\,s$



(q) S3 XY view - $t = 160\,s$



(r) S3 Potential visualization - $t = 160\,s$

Figure 6.14: Simulation #3 results - 2D APF

Chapter 7

Formation control using a three-dimensional Artificial Potential Field Algorithm

In this chapter the *three-dimensional APF-based formation control algorithm* will be described. This model is similar to the one used in Sect. 6.2 with some adjustments in order to operate in three dimensions, as will be explained in the first section. One of the main problems in this case is the potential representation, a tensor of third order: the solution found will be described in the second section. Finally, thanks to a simulation, some code features be illustrated.

7.1 Code adaptations

Having verified the operating principle of the *two-dimensional APF-based cooperative formation control algorithm* it has been decided to extend its functioning to a three-dimensional case. The plant model of the aircraft (described in Ch. 3) does not change in contrast to all the APF blocks. Before describing how the Simulink model has been adapted, we will briefly focus on the *grid generation* (Sect. 6.1.2). If in the previous model the space was modelled as a *square*, in this case we deal with a *parallelepiped* filled with equidistant points and its spacing depends on the `gridpointdist` value. The number of points is higher than the 2D model (with equal XY dimensions) so the simulation could be quite slow, not allowing real-time scenarios. For this reason, 3D grids will be smaller and sparser than two-dimensional ones.

7.1.1 Leader UAV - 3D APF block

In Fig. A.18 an overview of the leader 3D APF block is depicted. First of all, relative positions in three-dimensional space should be calculated using the `ObjectPosCalc_AF_3D` function (listing 7.1). As we discussed in previous chapters, the grid is generated in order

to be leader-centred and it should be rotated so that it can be always aligned with leader body axes (to be more precise, we do not rotate the grid itself but rather relative distances of followers and obstacles). In the 2D case, dealing with a two-dimensional grid, the *heading angle* ψ_L was used. In the 3D case we need to consider other angles, as *pitch* θ_L and *roll* φ_L. For the sake of simplicity and to avoid long computational times we have added the pitch angle only, although the possibility to add φ_L rotation has been implemented.

Listing 7.1: *Objects position calculator - 3D case (extract)*

```
1        diff_X_F1=x_F1-x_leader;
2        diff_Y_F1=y_F1-y_leader;
3        diff_H_F1=h_F1-h_leader;
4
5        R_psi=[cos(psi_leader) sin(psi_leader) 0 ;-sin(psi_leader) cos(
             psi_leader) 0; 0 0 1];
6        R_theta=[cos(theta_leader) 0 -sin(theta_leader); 0 1 0; sin(
             theta_leader) 0 cos(theta_leader)];
7        R_phi=1; %[1 0 0; 0 cos(phi_leader) sin(phi_leader); 0 -sin(
             phi_leader) cos(phi_leader)];
8        R=R_psi*R_theta*R_phi;
9        diff_rot_F1=(R*[diff_X_F1;diff_Y_F1;diff_H_F1]);
10
11       F1forw_d=diff_rot_F1(1);
12       F1lat_d=diff_rot_F1(2);
13       F1height_d=diff_rot_F1(3);
```

In the two-dimensional APF Simulink model altitude h was considered constant and obstacles were defined using initial positions and velocities in *north-east* plane; quite the opposite, in this case obstacles are defined using $h_{0,obs}$ and $\dot{h}_{0,obs}$ also. The *"Fixed/Moving obstacle NED positions"* block has been modified, as can be observed in Fig. 7.1, considering a constant vertical velocity. The potential should be calculated taking altitude into consideration. A `for` loop (line 7 of the following listing) has been added to the `PotentialGenerator3D_AF` function in order to select points along Z_L. Attractive and repulsive functions are essentially the same of Sect. 6.1.1 with the only difference that vectors are made up of three components.

Listing 7.2: *Potential Generator* (extract) - 3D case

```
1        Ua_F1 = zeros(n_grid_X,n_grid_Y,n_grid_Z);
2        Ur_F1 = zeros(n_grid_X,n_grid_Y,n_grid_Z);
3        Ut_F1 = zeros(n_grid_X,n_grid_Y,n_grid_Z);
4
5        for ii = 1:n_grid_X
6            for jj = 1:n_grid_Y
7                for kk=1:n_grid_Z
8                    if rho_obs_follo_3D(F1forw_d,F1lat_d,F1height_d,
                         x_ext_N(ii),y_ext_E(jj),z_ext_H(kk)) > Resterno_F1
9
```

```
                    Ua_F1(ii,jj,kk) = 0.5*Ka_F1*rho_obs_follo_3D(
                        F1forw_d,F1lat_d,F1height_d,x_ext_N(ii),
                        y_ext_E(jj),z_ext_H(kk)).^2-0.5*Ka_F1*(
                        Resterno_F1).^2;
                else
                    Ua_F1(ii,jj,kk) = 0;
                end
                if rho_obs_follo_3D(F1forw_d,F1lat_d,F1height_d,
                    x_ext_N(ii),y_ext_E(jj),z_ext_H(kk)) <=Rinterno_F1

                    Ur_F1(ii,jj,kk)=0.5*Kr1_F1*((r0_F1+Kr2_F1)./(
                        rho_obs_follo_3D(F1forw_d,F1lat_d,F1height_d,
                        x_ext_N(ii),y_ext_E(jj),z_ext_H(kk))+Kr2_F1)
                        -1).^2;

                else
                    Ur_F1(ii,jj,kk) = 0;
                end
            end
        end
    end

    for ii = 1:n_grid_X
        for jj = 1:n_grid_Y
            for kk=1:n_grid_Z
                Ut_F1(ii,jj,kk) = 0.5*Ka_T*rho_obs_follo_3D(
                    targetpointF1(1),targetpointF1(2),targetpointF1(3)
                    ,x_ext_N(ii),y_ext_E(jj),z_ext_H(kk)).^2;
            end
        end
    end

    potential_F1=Ui_L+Ut_F1;
```

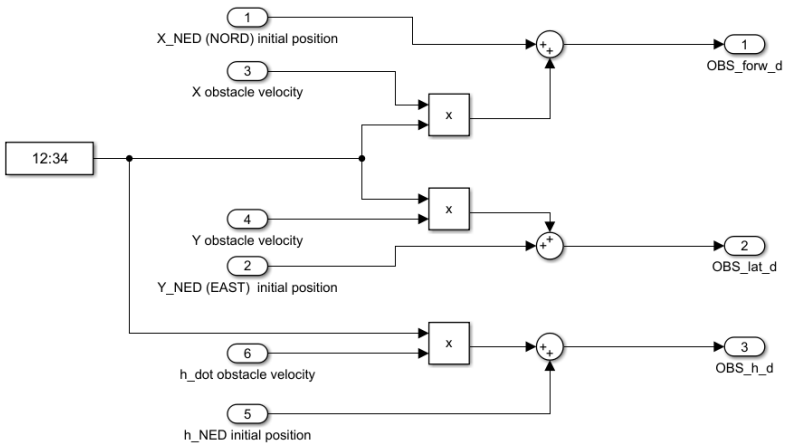


Figure 7.1: *Fixed/Moving obstacle NED positions* block - 3D version

7.1.2 Follower UAV - 3D APF block and errors calculation

In Fig. A.19 an overview of the follower 3D APF block is depicted. Comparing the model with its 2D version (Fig. A.14) it can be observed that some inputs have been added to the `PotentialSolver_AF_3D` function (Listing 7.3). Relative distance along Z_L is needed to find the current (approximate) position of the aircraft in the grid in the form of matrix index. Provided that for a 3D space `ngrid` generates three different tensors of third order, `min` has been used to select the point where the difference between space grid values and relative distances is minimum. Forces have been computed and the steepest descent direction `dL` has been rotated using $\Delta\psi$ and $\Delta\theta$ in order to obtain its components in a *follower-centred frame* `d`. Finally, commands `l_Vclear`, `f_Vclear` and `h_Vclear` are obtained.

Listing 7.3: *Potential Solver* - 3D case

```
function [l_Vclear,f_Vclear,h_Vclear,dL] = PotentialSolver_AF_3D(
    x_grid_N,y_grid_E,z_grid_H,potential,Fforw_d,Flat_d,Fheight_d,diff_psi
    ,diff_theta,diff_phi)

%%% PARAMETERS INITIALIZATION %%%
% Find index of follower in leader-centred ref system
[~,inx_N]=min(abs(x_grid_N(:,1,1)-Fforw_d)); % Index of X follower pos
[~,iny_E]=min(abs(y_grid_E(1,:,1)-Flat_d)); % Index of Y follower pos
[~,iny_H]=min(abs(z_grid_H(1,1,:)-Fheight_d)); % Index of Z follower pos

%%% SOLVE! %%%
[Fy,Fx,Fz]=gradient(-potential);

%%% Force in current point
F=[Fx(inx_N,iny_E,iny_H),Fy(inx_N,iny_E,iny_H),Fz(inx_N,iny_E,iny_H)];
modF = max(10^-6,sqrt(F(1).^2+F(2).^2+F(3).^2));
dL=(F./modF); % Descent direction in leader-centred system

R_psi=[cos(diff_psi) sin(diff_psi) 0 ;-sin(diff_psi) cos(diff_psi) 0; 0 0
    1];
R_theta=[cos(diff_theta) 0 -sin(diff_theta); 0 1 0; sin(diff_theta) 0 cos
    (diff_theta)];
R_phi=1;%[1 0 0; 0 cos(diff_phi) sin(diff_phi); 0 -sin(diff_phi) cos(
    diff_phi)]';
R_LF=R_psi*R_theta*R_phi;
d=R_LF*dL';

% APF commands
l_Vclear=min(6,0.15*abs(F(2)))*d(2);
f_Vclear=min(5,0.2*abs(F(1)))*d(1);
h_Vclear=min(5,0.15*abs(F(3)))*d(3);

end
```

At this point, commands should be used to control the follower aircraft taking into consideration that they are computed in a *follower-centred reference frame*. As we have done in the previous two-dimensional APF Simulink model, we have to calculate *velocity*

errors in this reference frame assumed that plant output velocities are in NED axes. A new function has been created, namely `NEDtofollower3D` (in the following listing). Similarly to Listing 7.1, rotation matrices (`R_psi`, `R_theta`, `R_phi`) are created and the difference between follower and leader NED velocities (`dV_N`, `dV_E`, `dh_dot`) are transformed in the corresponding values in *follower-centred reference frame* obtaining `dV_N_rot`, `dV_E_rot`, `dh_dot_rot`. This done differences between actual velocities and APF commands are computed and sent to the *"PD formation command (VELOCITY)"* block, previously described in Sect. 6.2.3. An overview of the error calculation block is depicted in Appendix (Fig. A.20).

Listing 7.4: *NEDtofollower3D* function

```
function [dV_N_rot,dV_E_rot,dh_dot_rot] = NEDtofollower3D(dV_N,dV_E,
    dh_dot,psi_f,theta_f,phi_f)

R_psi=[cos(psi_f) sin(psi_f) 0 ;-sin(psi_f) cos(psi_f) 0; 0 0 1];
R_theta=[cos(theta_f) 0 -sin(theta_f); 0 1 0; sin(theta_f) 0 cos(theta_f)
    ];
R_phi=1;%[1 0 0; 0 cos(phi_f) sin(phi_f); 0 -sin(phi_f) cos(phi_f)];
R=R_psi*R_theta*R_phi;

dV_rot=(R*[dV_N;dV_E;dh_dot]);

dV_N_rot=dV_rot(1);
dV_E_rot=dV_rot(2);
dh_dot_rot=dV_rot(3);

end
```

7.2 3D potential visualization

When dealing with three-dimensional potentials, a fundamental problem to be faced is the way we represent them. Observing simulations in Sect. 6.2.4 and Sect. 6.2.5 we can notice that, for a two-dimensional APF, *contour lines* or *mesh surfaces* can be employed. In our case, however, the third dimension represents altitude so it can not be used to visualize potential values. After some research into MATLAB documentation, the *"Visualizing Four-Dimensional Data"* [45] example has been found and in particular an approach similar to the one described in the *"Visualize Function of Three Variables"* section has been chosen. The `scatter3` function generates a 3D scatter plot, i.e. a plot composed of a set of spheres in space whose diameter can be selected by the user. In our 3D APF model physical space has been discretized, as a matter of fact, as a tensor of third order and potential is itself a tensor of this type. Using `scatter3` every point of the grid is plotted and coloured depending on its value of potential. Grids, previously generated with `ngrid`, are modified using the `pagetranspose` function. The potential *structure array* `out.potentialF1` is transformed using `squeeze`, which removes dimensions of length one. These changes are made in order to obtain variables that can be handled by `scatter3` . Point transparency (or *"alpha"*, as called in the MATLAB jargon) must be modified

somehow in order to visualize only high values of potentials (close to leader and follower). Using axes properties as `'AlphaDataMapping'`, `'AlphaDataMode'`, `'ALimMode'` and `'ALim'`, points transparency has been changed in order to be proportional to potential values. The main problem of a *scatter3-based* plot of this type is that the attractive potential, in particular the *target point*, can not be observed. In this point, in fact, potential value is the minimum so it has been made transparent for visualization purposes. Results strongly depend on `gridpointdist` which, as we have described in Sect. 6.1.2, define the number of points of the grid. Denser grids, in fact, generates better representations but critically lengthen computational time. A visualization example is depicted in Fig. 7.2, where `gridpointdist`$= 3\,m$ has been used.

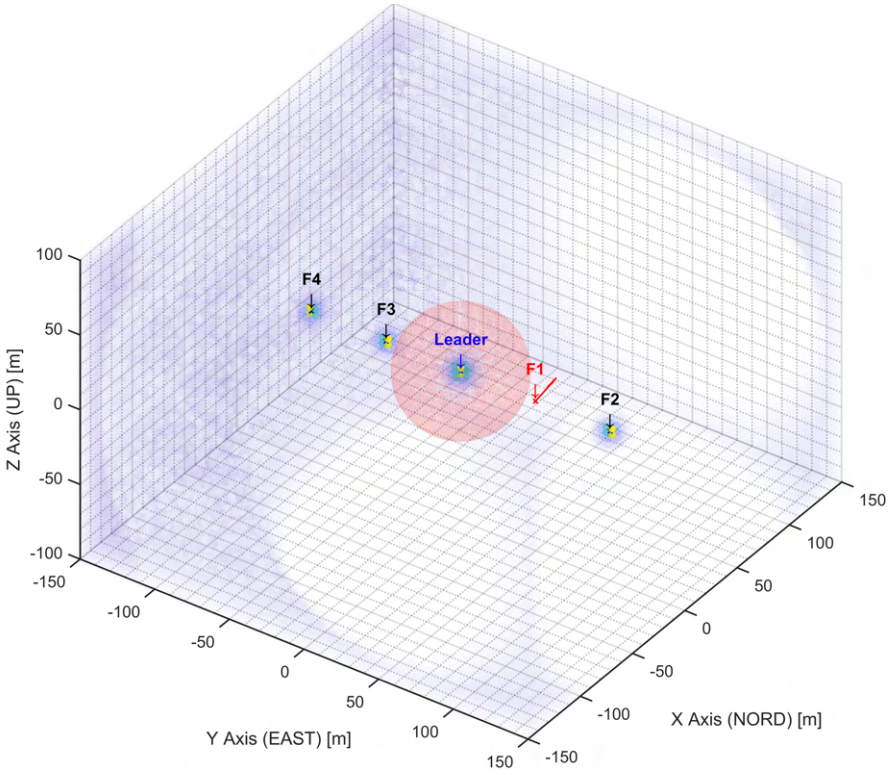


Figure 7.2: 3D APF potential visualization example

7.3 Simulation

In order to demonstrate how the model works, a three-dimensional simulation has been performed. The *grid*, as previously outlined, is generally sparser than the two-dimensional version: in this case a `gridpointdist` value of $4\,m$ has been selected. The flag values employed in this scenario, whose code is included in Appendix B.5, are:

- `manoeuvre` → `'manovra sim1 APF 3D'`

- `sectform` → `'sect sim1 APF 3D'`
- `obstacleflag` → `'sim1 APF 3D'`

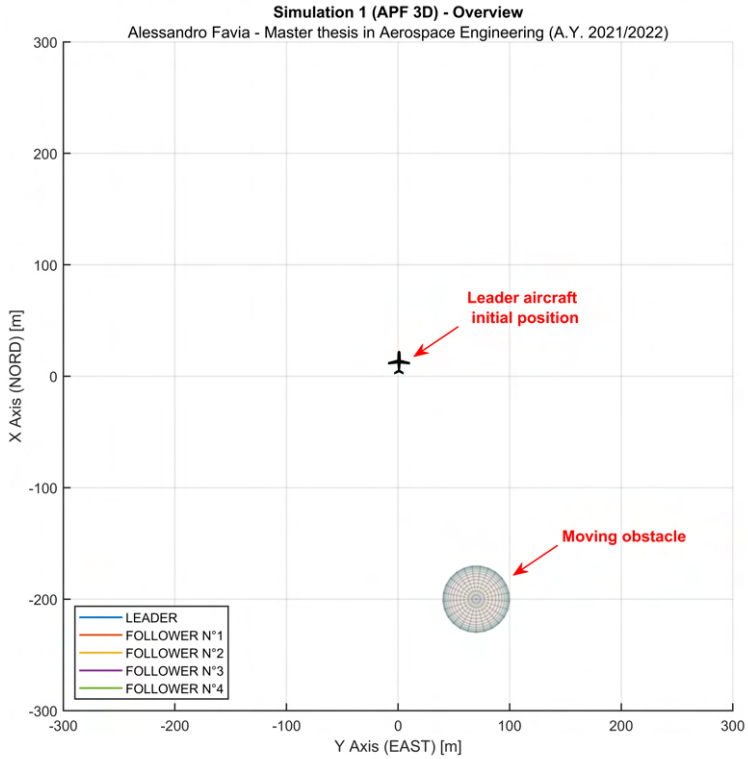
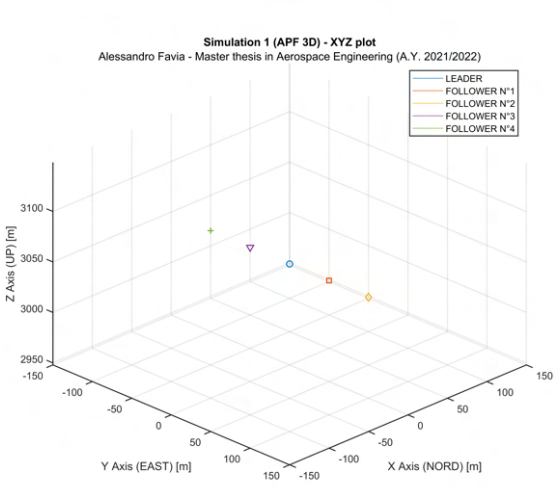
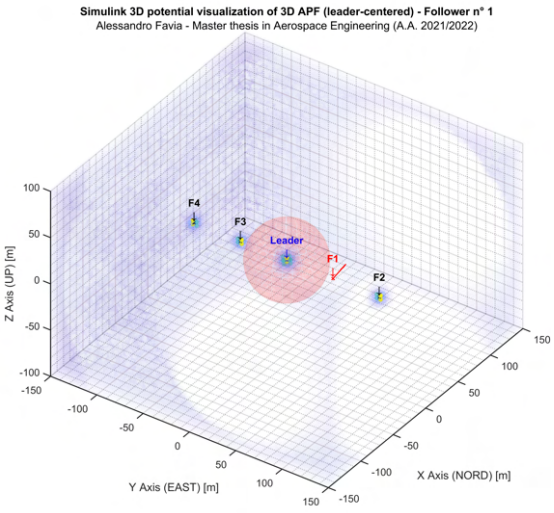


Figure 7.3: Simulation #1 overview (APF 3D)

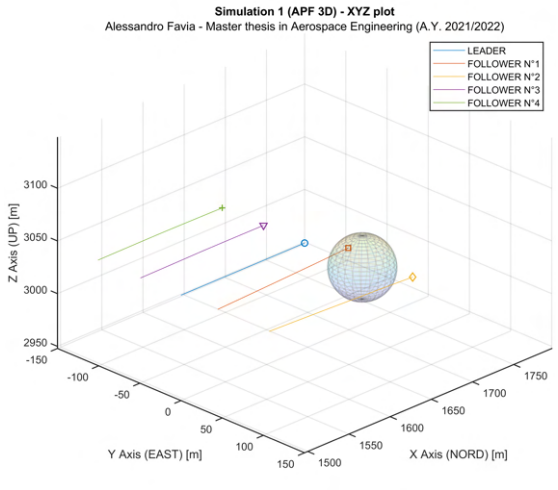
In Fig. 7.3 an overview of the scenario is depicted. A *moving obstacle* with physical radius $R_{obs} = 30\,m$ (and repulsive radius $R_{pot} = 60\,m$) has been inserted in $C_{mo} = [x_{mo}, y_{mo}] = [-200; 70]\,m$, having a velocity $V = V_{NORD} = 92.3111\,\frac{m}{s}$. The initial formation is of *wall* type and it is kept until $t = 25\,s$, when an *arrow* is ordered; finally, at $t = 70\,s$, a *line astern* formation is executed. As far as manoeuvres are concerned, every aircraft initially flights straight, without changing its altitude. At $t = 20\,s$ the leader starts a right turn with altitude change ($\dot{h} = 2\,\frac{m}{s}$), obtaining an *heading angle* $\psi_L = 60\,deg$. At a later time ($t = 90\,s$) a *flare* is performed. `hdotdes` is created using `linspace` obtaining an equally spaced vector of vertical velocities between $-10\,\frac{m}{s}$ and $10\,\frac{m}{s}$. Similarly, `trich` is an equally spaced vector of time values between $90\,s$ and $140\,s$. Using a `for` loop and the above-mentioned variables, a number of altitude variation commands have been piled in order to produce a continuous flare manoeuvre.



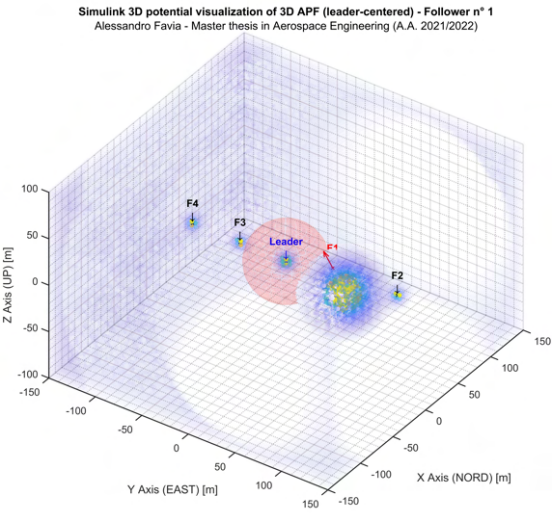
(a) S1 XYZ view - $t = 0\,s$



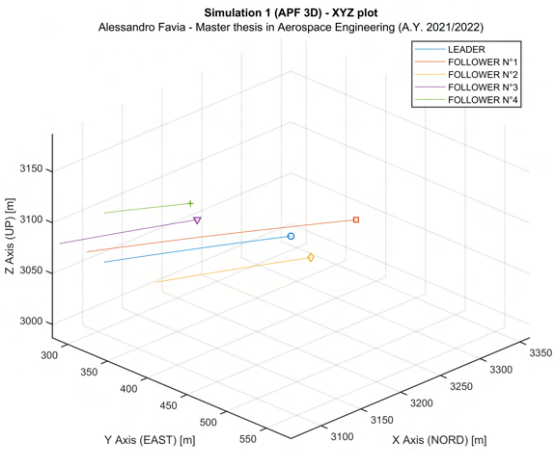
(b) S1 Potential visualization - $t = 0\,s$



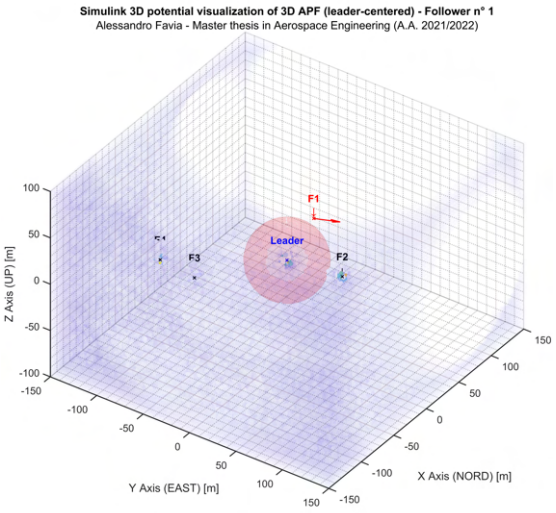
(c) S1 XYZ view - $t = 20\,s$



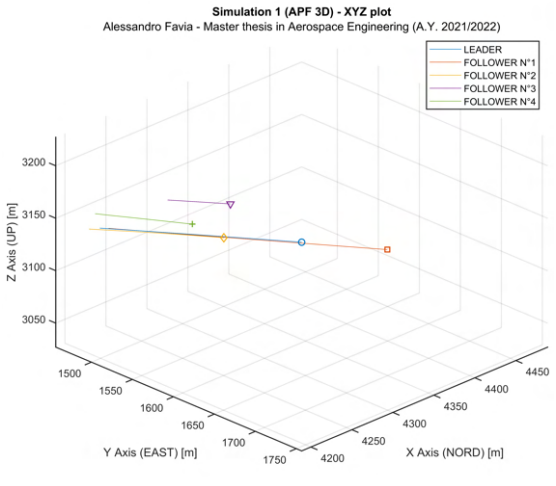
(d) S1 Potential visualization - $t = 20\,s$



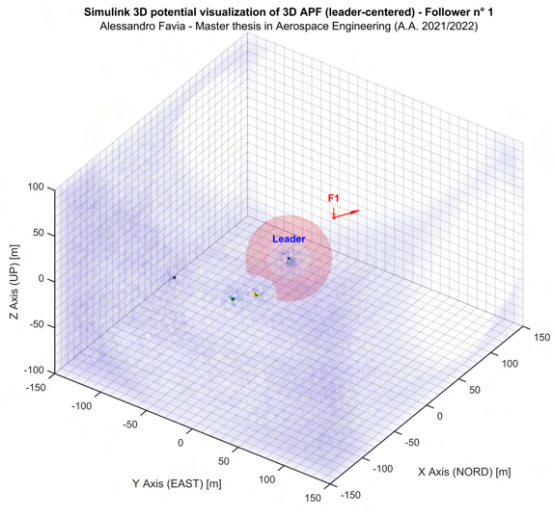
(e) S1 XYZ view - $t = 40\,s$



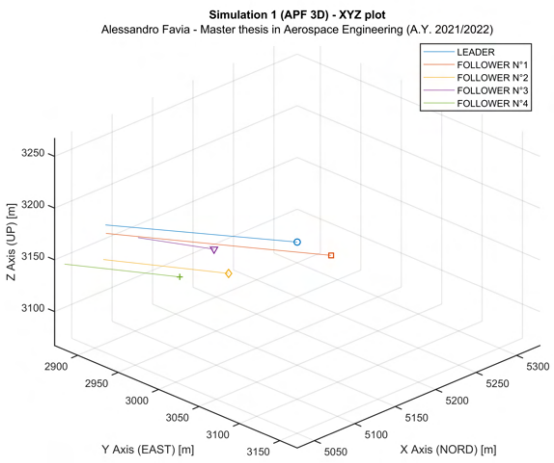
(f) S1 Potential visualization - $t = 40\,s$



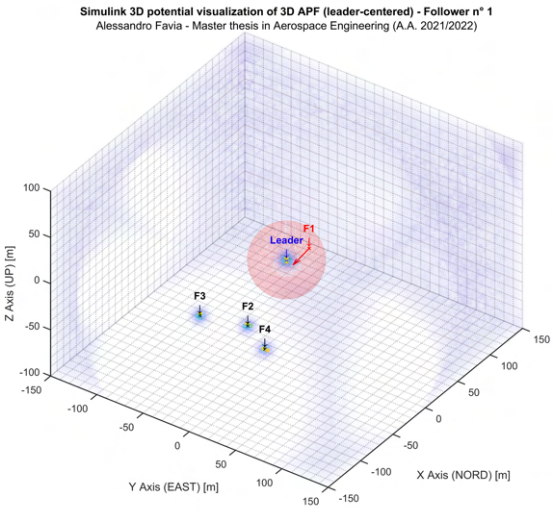
(g) S1 XYZ view - $t = 60\,s$



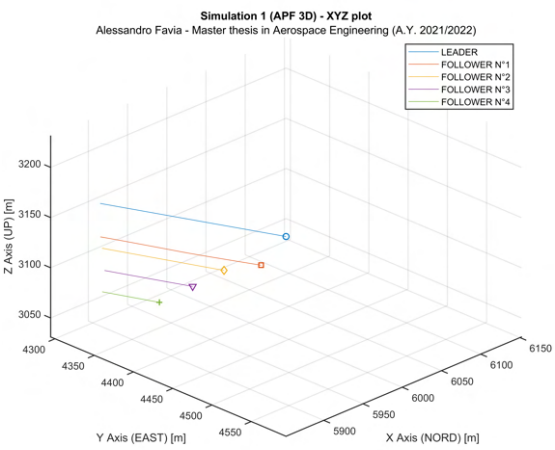
(h) S1 Potential visualization - $t = 60\,s$



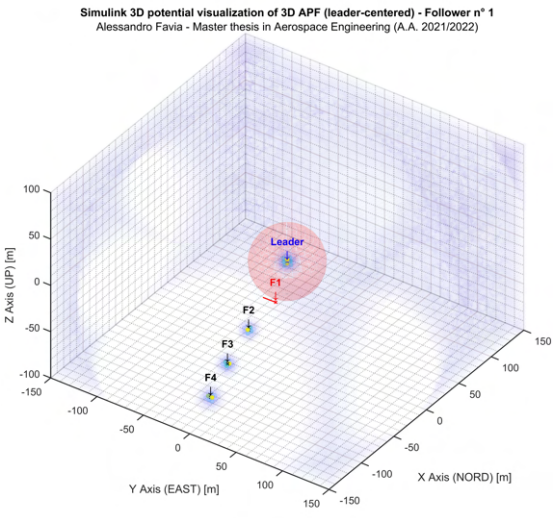
(i) S1 XYZ view - $t = 80\,s$



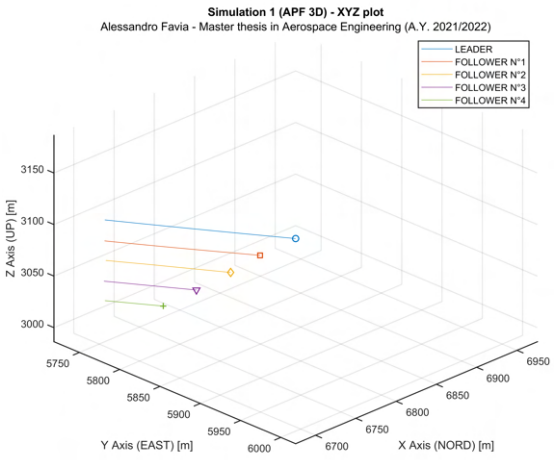
(j) S1 Potential visualization - $t = 80\,s$



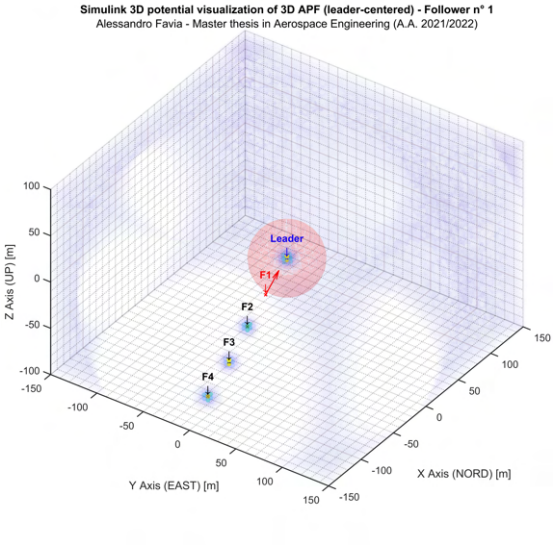
(k) S1 XYZ view - $t = 100\,s$



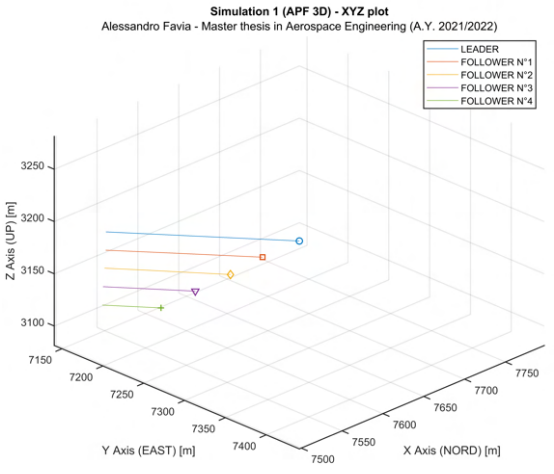
(l) S1 Potential visualization - $t = 100\,s$



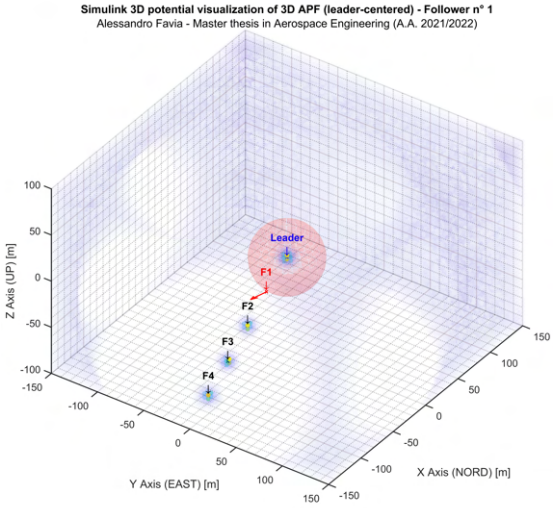
(m) S1 XYZ view - $t = 120\,s$



(n) S1 Potential visualization - $t = 120\,s$



(o) S1 XYZ view - $t = 140\,s$



(p) S1 Potential visualization - $t = 140\,s$

Figure 7.4: Simulation results - 3D APF

The scenario has been simulated and it can be observed in Fig. 7.4. While all aircraft are in *wall* formation (Fig. 7.4(a)(b)) the moving obstacle, whose velocity is greater than the aircraft one, reaches them from behind. F3 and F4 do not perceive the object while F1 and F2, as can be seen in Fig. 7.4(c), avoid it. It can be noticed from Fig. 7.4(d) how potential values of the object are particularly obvious. The potential sphere of influence of the leader is also depicted in red. After avoiding the obstacle, the right turn is ordered together with the *arrow* formation. In Fig. 7.4(e)(f) it can be seen how every aircraft is reaching its *target point* during the turn. The increase in altitude does not affect the formation shape, which is almost completed in Fig. 7.4(g)(h). At $t = 70\,s$ the *arrow* patter is broken and gradually changed to a *line astern* type. Fig. 7.4(i)(j) are particularly important because the effect of the APF along Z can be observed. The position of the first follower (in *leader-centred reference frame*) in the *arrow* is $TP_{F1,arrow} = [75; 0; 0]\,m$ while the subsequent *line astern* target point is $TP_{F1,line\,astern} = [-30; 0; -20]\,m$. The situation is quite similar to the *static* one (of F4, in that case) explained in Sect. 6.1.3. The Y coordinate is exactly the same, and if the APF was two-dimensional the aircraft could have difficulty in reaching its position. A *saddle point* would appear: the first follower, sure enough, would have reached a point nearby the leader reducing its speed only (without turning). Using a three-dimensional APF altitude can be changed: F1 avoids the leader reducing it (Fig. 7.4(i)(j)). At $t = 90\,s$ the *flare* manoeuvre is activated, and its effect is depicted in Fig. 7.4(k)(l). In this case the *line astern* formation is successfully completed (Fig. 7.4(m)(n)) and kept (Fig. 7.4(o)(p)) during the manoeuvre.

Chapter 8

Conclusions and future developments

This work deals with the design and modelling of a formation control algorithm of multiple UAVs based on the *Artificial Potential Fields (APF)* method. After a brief introduction on history, in Chapter 1 formation flight organization has been presented along with the *Manned-Unmanned Teaming (MUM-T)* concept. The state of the art of formation control and motion planning has been summarized in Chapter 2, focusing on various type of methods and choosing the best solution for the case. The basic aircraft, a medium-altitude long-endurance fixed-wing unmanned aerial vehicle (UAV), has been selected and simulated in Chapter 3: starting from the linearised dynamics, *main control surfaces* and *autopilots* have been modelled together with *manoeuvre logic* and *visualization* blocks. A first PID-based formation algorithm has been simulated in Chapter 4, observing how in this case *collision avoidance* can not be guaranteed. At this point the *Artificial Potential Field* method has been introduced (Chapter 5), starting from the traditional single-robot approach to the multiple-robot one through the description of the *gradient descent algorithm* and of different *planning techniques*. The aforementioned multiple-robot two-dimensional algorithm has been modified (Chapter 6) in order to be coherent with the purpose of the thesis and has been tested in a *static* environment. The same code has been later adapted to the *Simulink* time-varying environment and three simulations have been performed so that its operation has been verified. Finally, in Chapter 7 the three-dimensional version has been created in order to control the aircraft along all axes.
This work is particularly suitable for future developments. Specifically:

- the *plant dynamics* could be improved considering non-linear equations, observing how the algorithm reacts to an altitude/velocity-variable behaviour of the aircraft
- multiple type of aircraft could be selected, like a *fighter* for the leader and a UAV for every follower: in this case differences in performance must be taken into account. Moreover, there may be a need to change the behaviour of the control system automatically adjusting APF gain values by complex control laws. In this

thesis, for example, the *repulsive radius* of every follower changes with their velocity.

- other types of *attractive* and *repulsive potential functions* could be examined in depth in order to understand if they could improve general performance of the code. *Saddle points* and *local minima* should be further investigated so that it could be excluded that they may constitute a problem.

Appendix A

Simulink model images

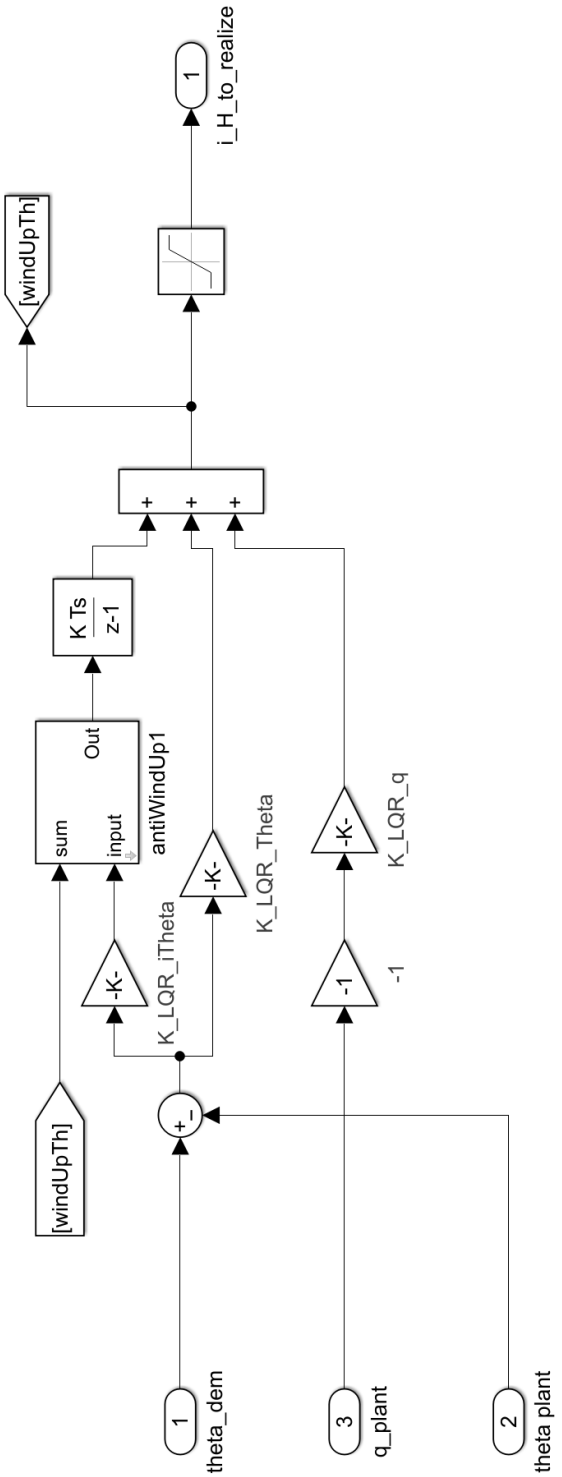


Figure A.1: Elevators LQR block - Simulink model

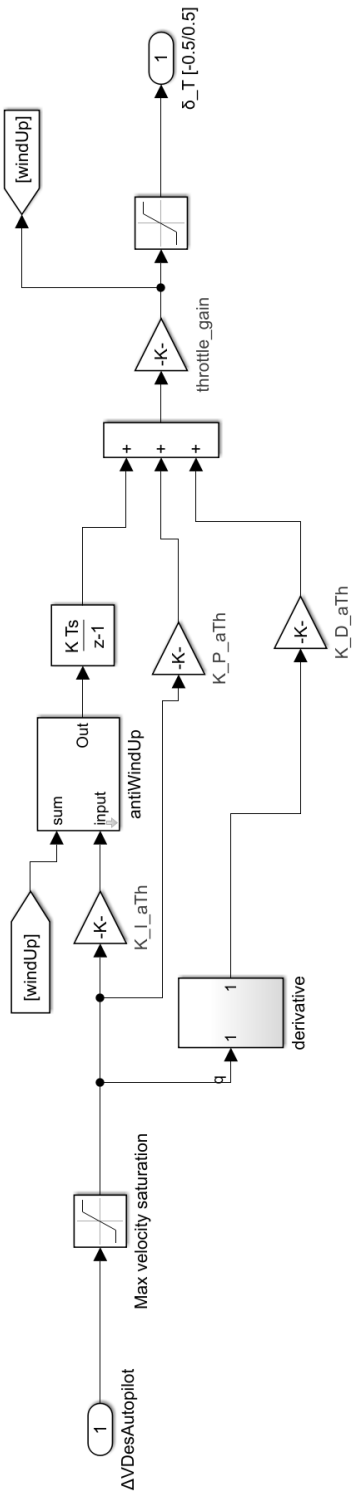


Figure A.2: Throttle LQR block - Simulink model

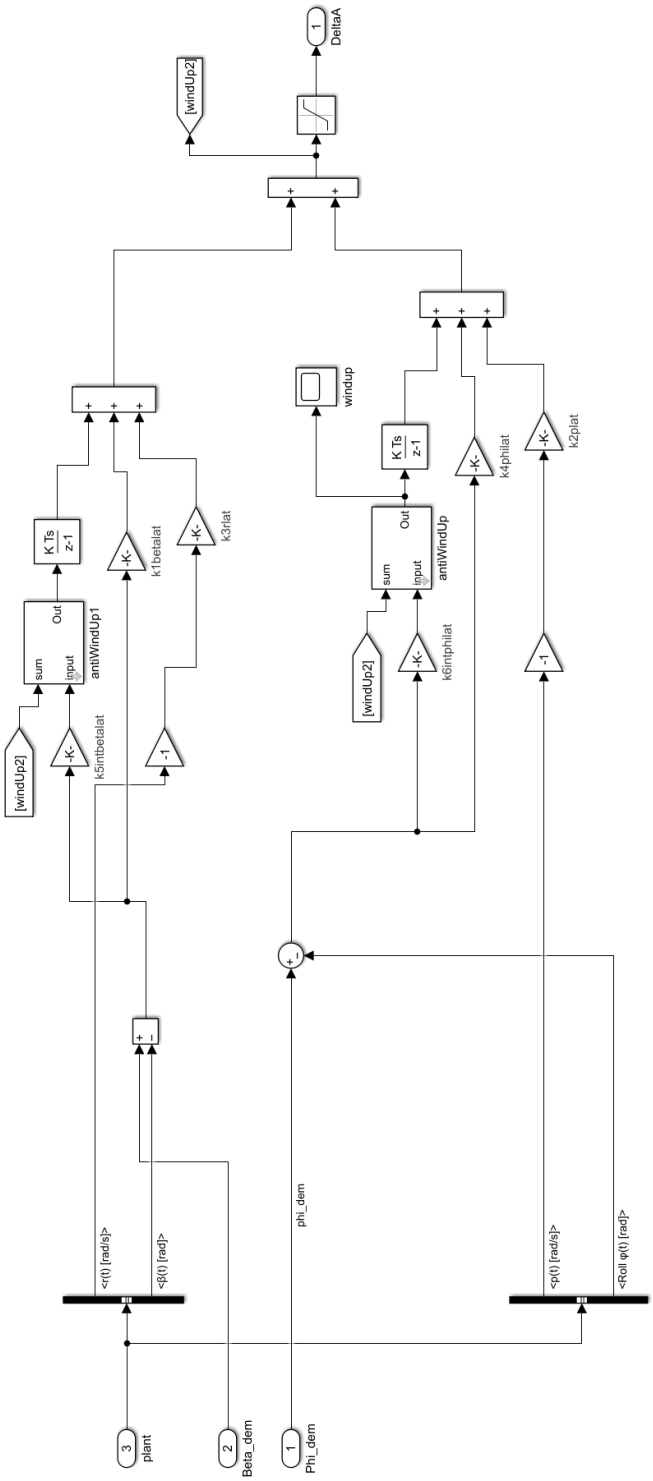


Figure A.3: Ailerons LQR block - Simulink model

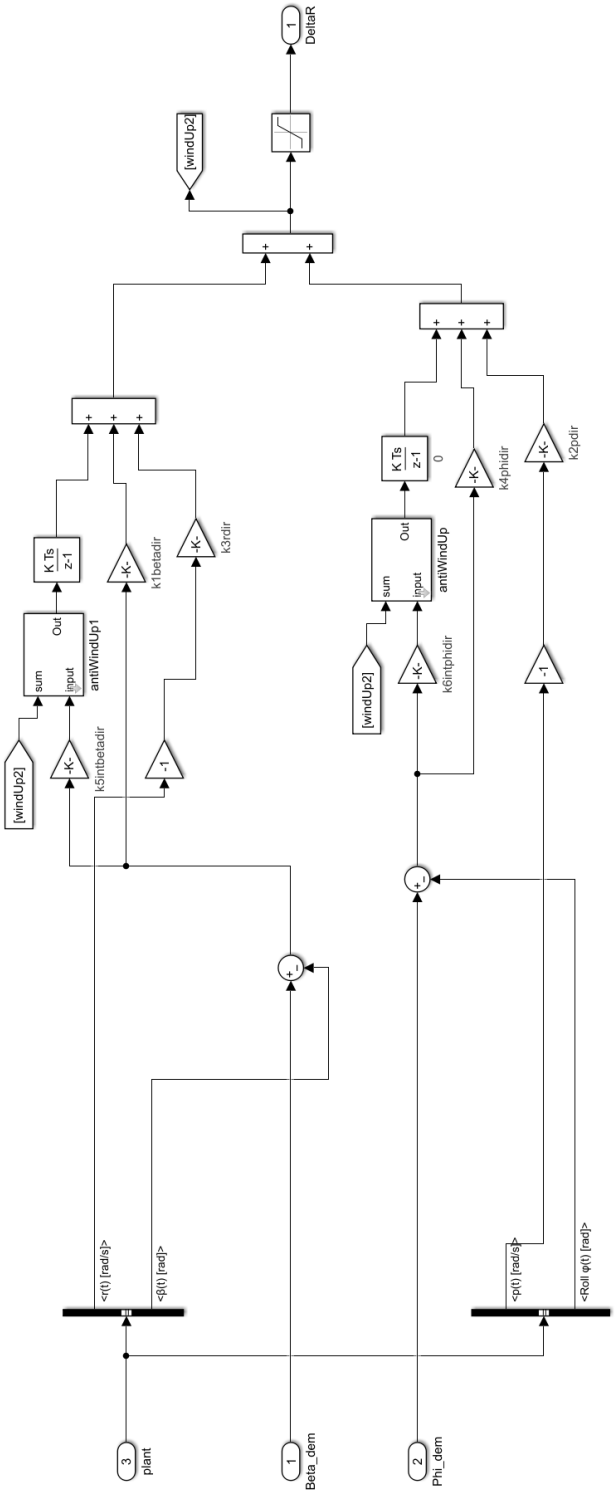


Figure A.4: Rudder LQR block - Simulink model

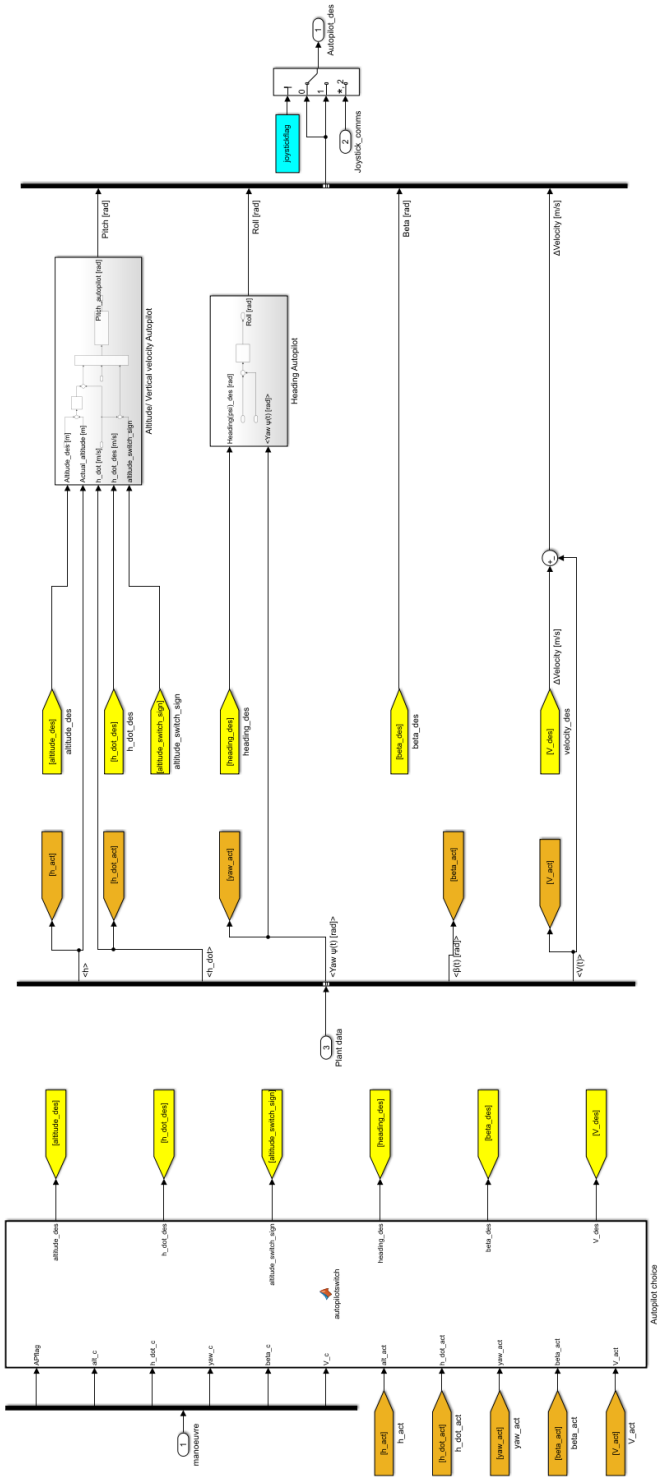


Figure A.5: *Autopilot* block for *Formation PID* - Simulink model

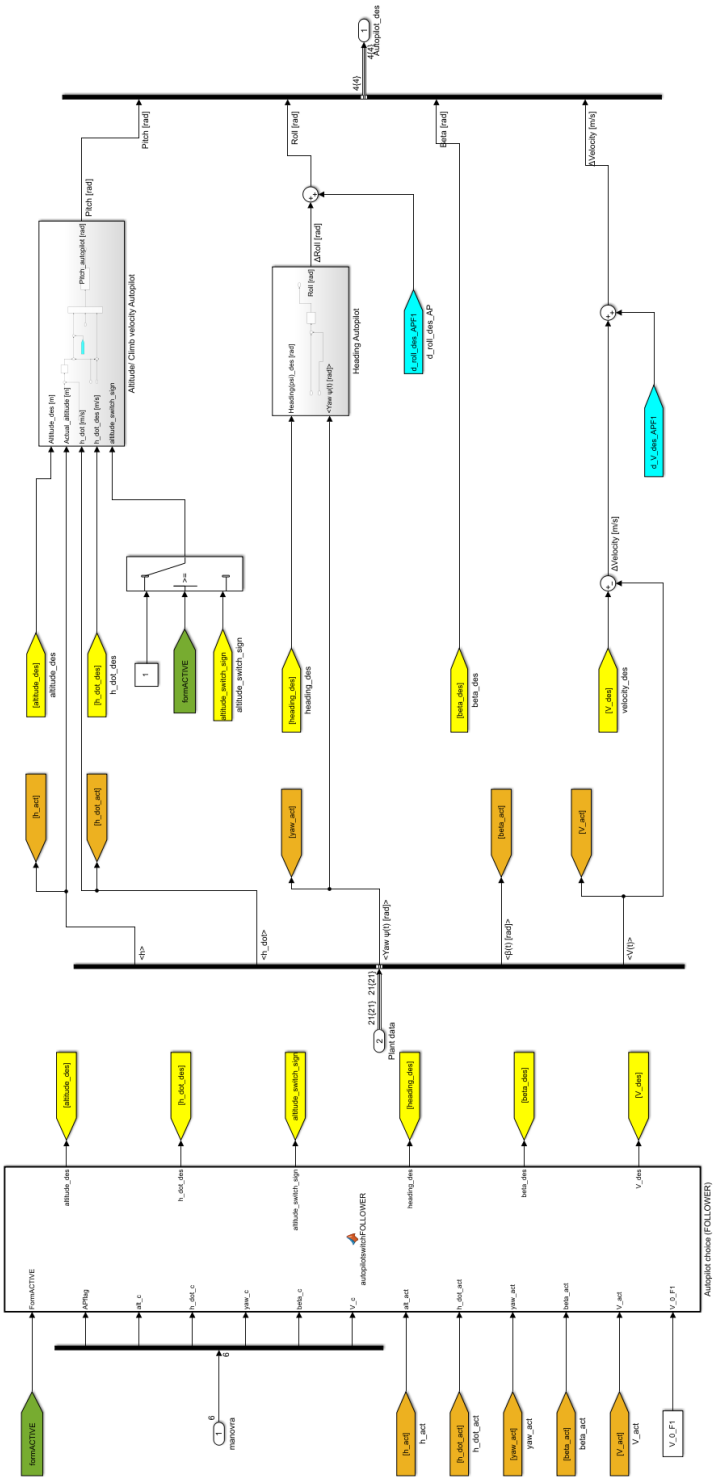


Figure A.6: *Autopilot* block for *Formation APF* - Simulink model

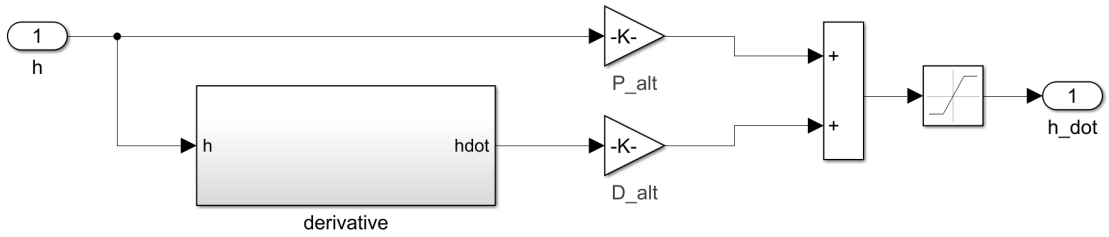


Figure A.7: *PD altitude* block

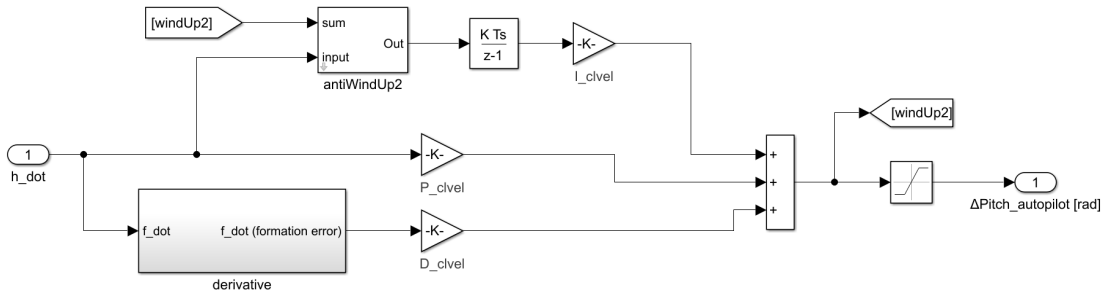


Figure A.8: *PID vertical velocity* block

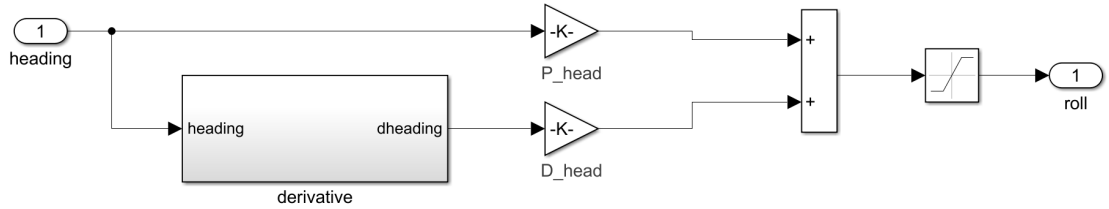


Figure A.9: *PD roll* block

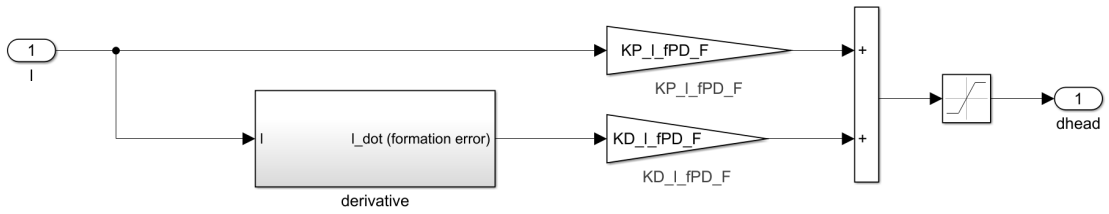


Figure A.10: Lateral formation PD (position) block

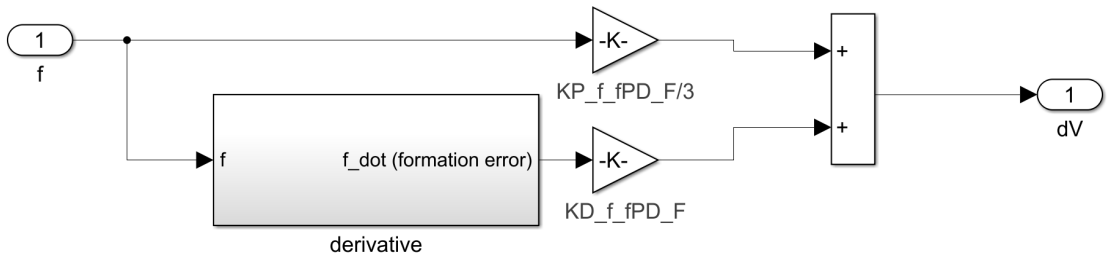


Figure A.11: Forward formation PD (position) block

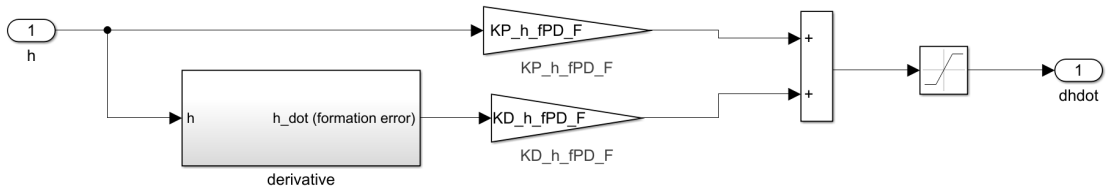


Figure A.12: Height (dot) formation PD (position) block

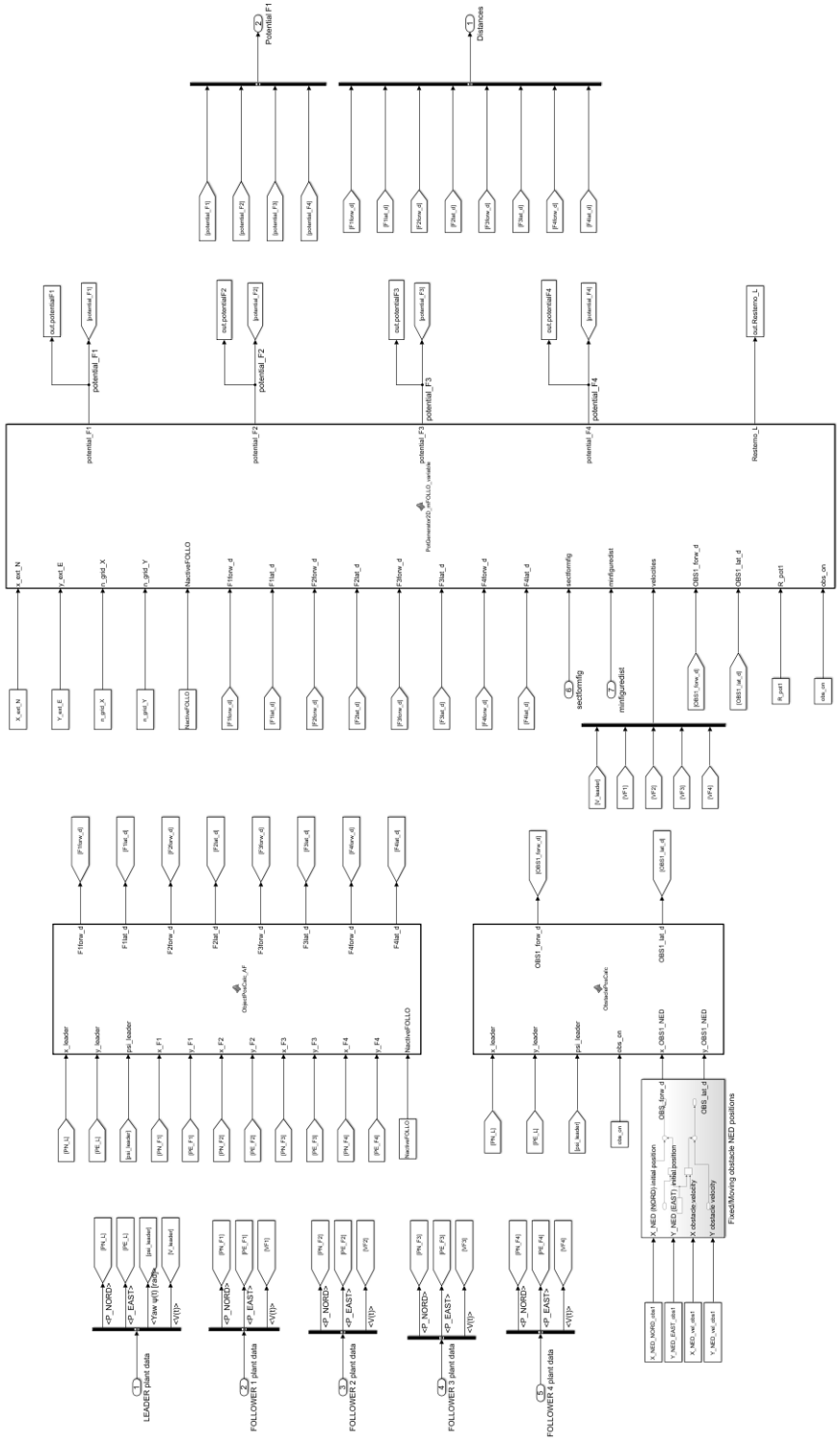


Figure A.13: Two-dimensional APF - leader block overview

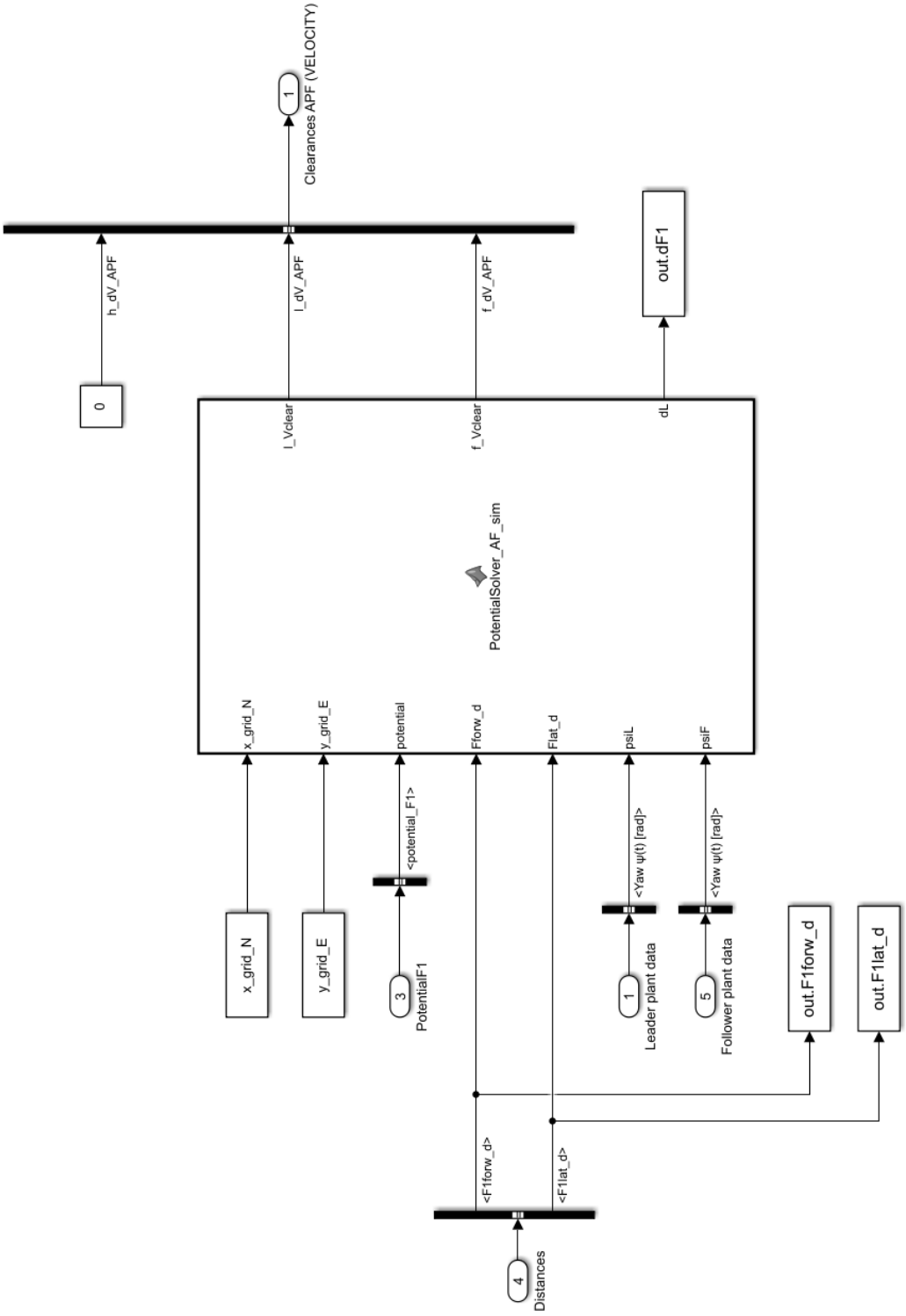


Figure A.14: Two-dimensional APF - follower block overview

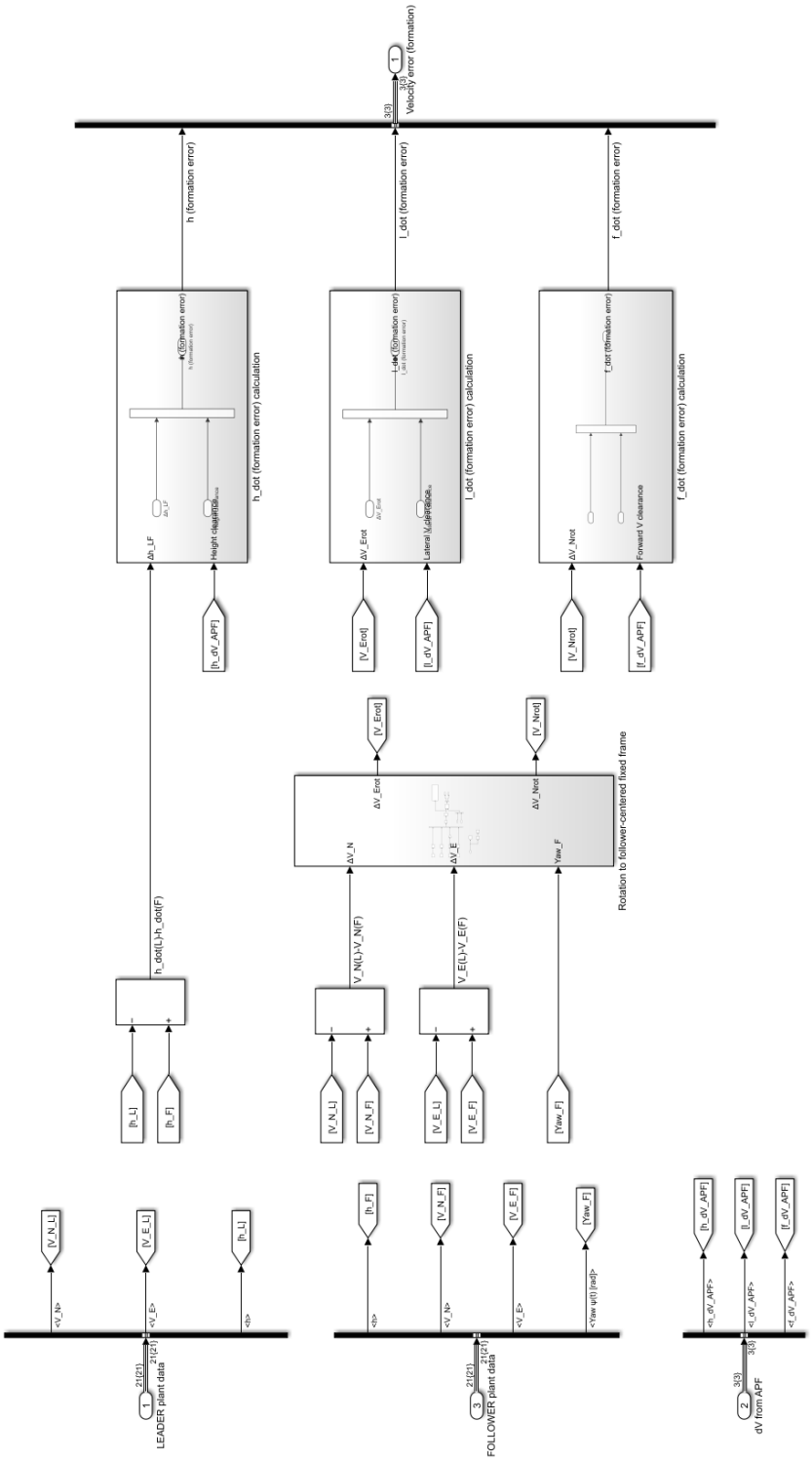


Figure A.15: Two-dimensional APF - velocity errors calculation

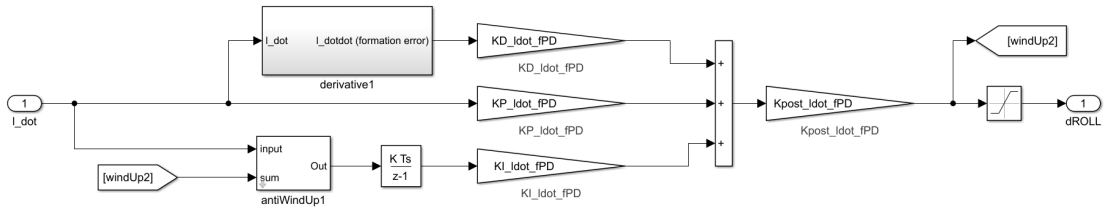


Figure A.16: Lateral formation PID (velocity) block

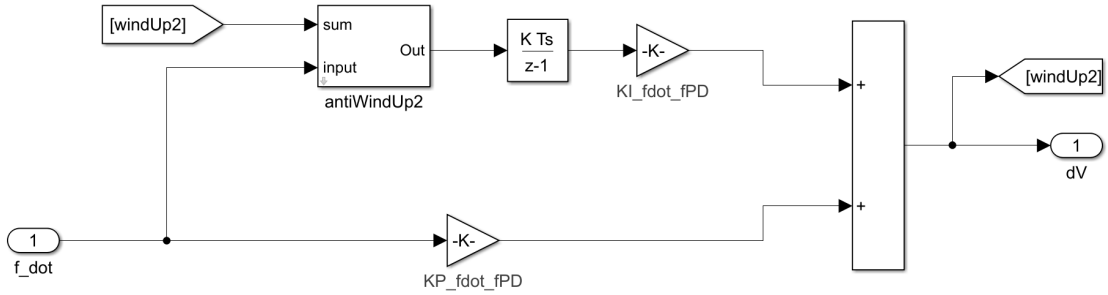


Figure A.17: Forward formation PI (velocity) block

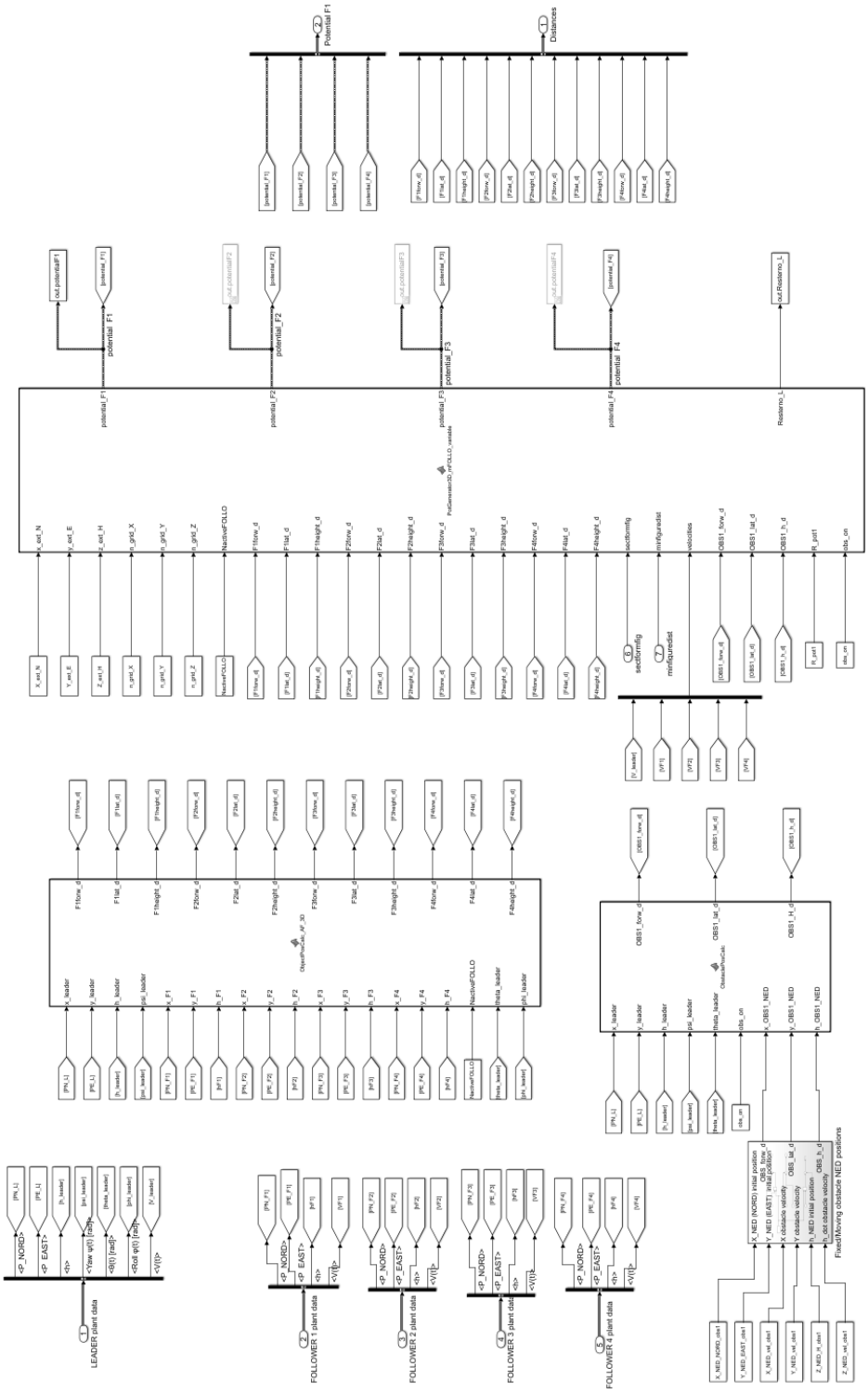


Figure A.18: Three-dimensional APF - leader block overview

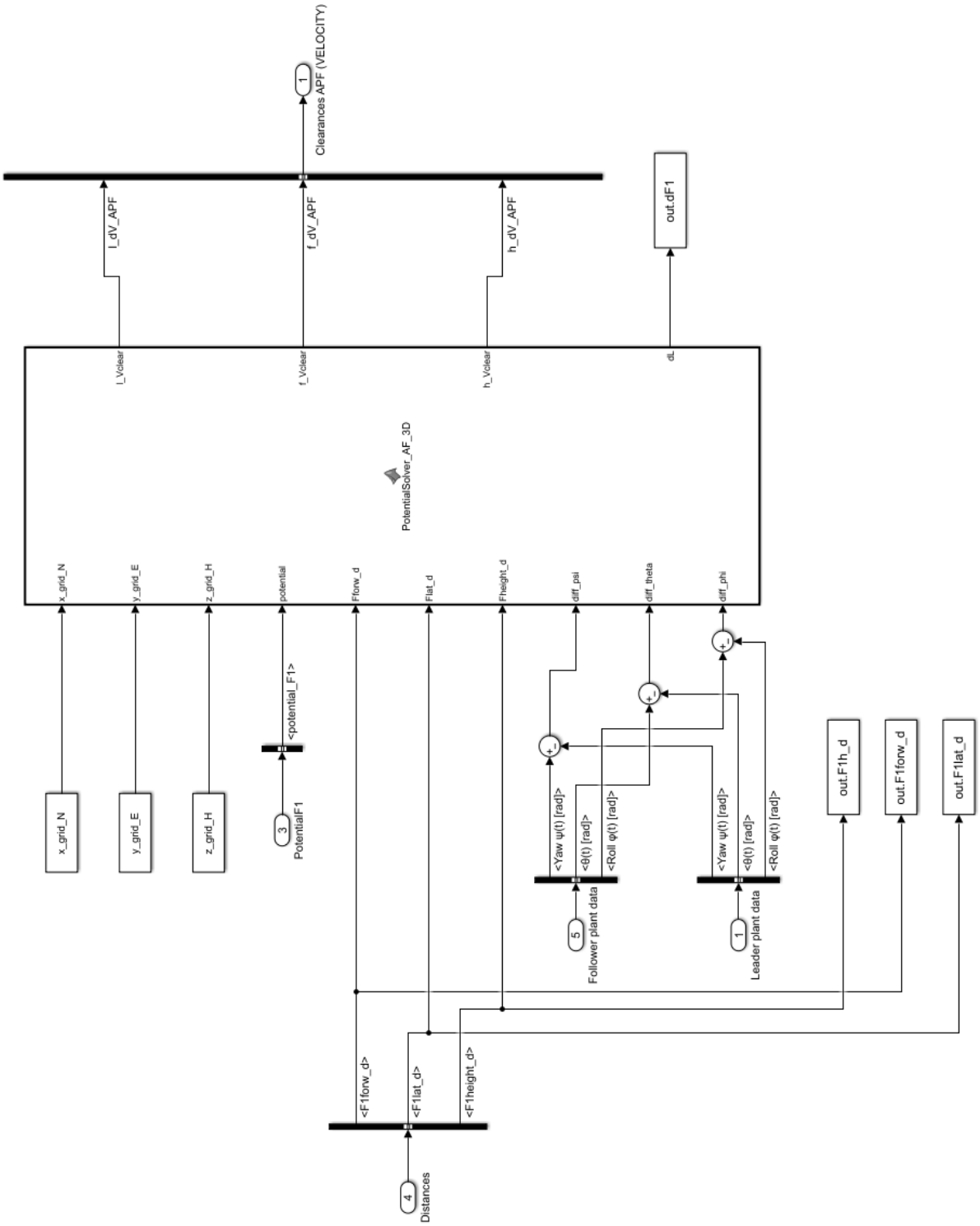


Figure A.19: Three-dimensional APF - follower block overview

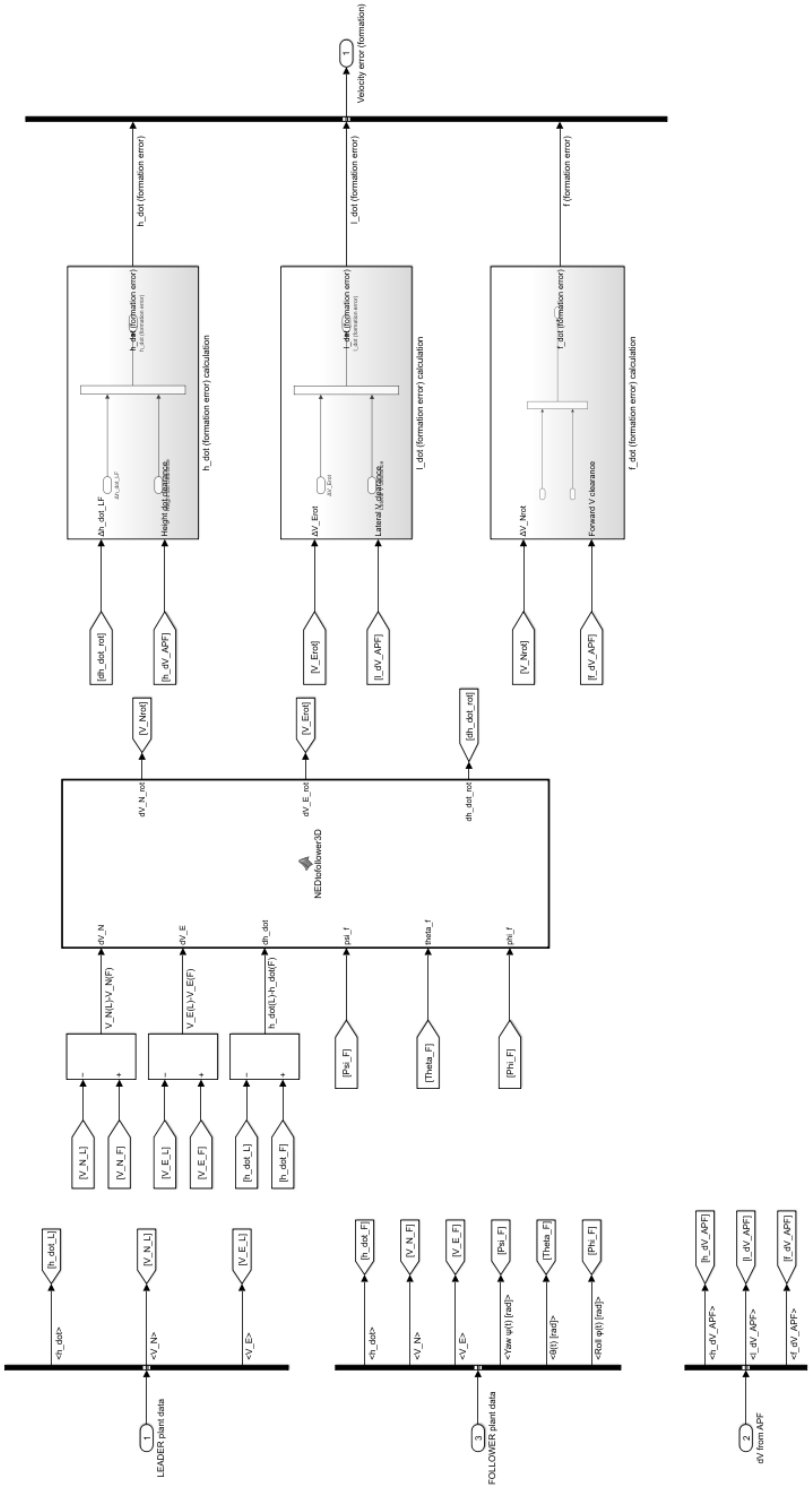


Figure A.20: Three-dimensional APF - errors block overview

Appendix B

Matlab and Simulink code listings

Listing B.1: *"Autopilot choice"* follower code

```
function [altitude_des,h_dot_des,altitude_switch_sign,heading_des,
    beta_des,V_des] =autopilotswitchFOLLOWER(FormACTIVE,APflag,alt_c,
    h_dot_c,yaw_c,beta_c,V_c,alt_act,h_dot_act,yaw_act,beta_act,V_act,
    V_0_F1)

% AUTOPILOT SWITCH - Alessandro Favia
altitude_des=0;
h_dot_des=0;
altitude_switch_sign=0;
heading_des=0;
beta_des=0;
V_des=0;

if FormACTIVE==0 % i.e. formation control not enabled

    switch APflag
        case 1 % Altitude autopilot
            altitude_des=alt_c;
            h_dot_des=h_dot_act; % useless
            altitude_switch_sign=1;
            heading_des=yaw_act;
            beta_des=beta_act;
            V_des=V_act;
        case 15 % Altitude HOLD autopilot
            altitude_des=alt_act;
            h_dot_des=0;
            altitude_switch_sign=-1;
            heading_des=yaw_act;
            beta_des=beta_act;
            V_des=V_act;
        case 2 % Vertical velocity autopilot
            altitude_des=0; % useless
```

```
30              h_dot_des= h_dot_c;
31              altitude_switch_sign=-1;
32              heading_des=yaw_act;
33              beta_des=beta_act;
34              V_des=V_act;
35          case 3 % Heading Autopilot (same H - with "Altitude Hold
                autopilot")
36              altitude_des=alt_act;
37              h_dot_des=h_dot_act; % useless
38              altitude_switch_sign=1;
39              heading_des=yaw_c;
40              beta_des=beta_act;
41              V_des=V_act;
42          case 4 % Heading Autopilot (with climb/descent through "Vertical
                velocity autopilot")
43              altitude_des= alt_act; % useless
44              h_dot_des=h_dot_c;
45              altitude_switch_sign=-1;
46              heading_des=yaw_c;
47              beta_des=beta_act;
48              V_des=V_act;
49          case 5 % Beta change (same H - with "Altitude Hold autopilot")
50              altitude_des=alt_act;
51              h_dot_des=h_dot_act; % useless
52              altitude_switch_sign=1;
53              heading_des=yaw_act;
54              beta_des=beta_c;
55              V_des=V_act;
56          case 6 % Beta change (with climb/descent through "Vertical
                velocity autopilot")
57              altitude_des=alt_act; % useless
58              h_dot_des=h_dot_c;
59              altitude_switch_sign=-1;
60              heading_des=yaw_act;
61              beta_des=beta_c;
62              V_des=V_act;
63          case 7 % Velocity change (same H - with "Altitude Hold autopilot
                ")
64              altitude_des=alt_act;
65              h_dot_des=h_dot_act; % useless
66              altitude_switch_sign=1;
67              heading_des=yaw_act;
68              beta_des=beta_act;
69              V_des=V_c;
70          case 8 % Velocity change (with climb/descent through "Vertical
                velocity autopilot")
71              altitude_des=alt_act; % useless
72              h_dot_des=h_dot_c;
73              altitude_switch_sign=-1;
74              heading_des=yaw_act;
75              beta_des=beta_act;
76              V_des=V_c;
77      end
```

```
else
    altitude_des=alt_act;
    h_dot_des=h_dot_act;
    altitude_switch_sign=-1;
    heading_des=yaw_act;
    beta_des=beta_act;
    V_des=V_0_F1;

end
end
```

Listing B.2: PID formation control - simulation code

```
% MANOEUVRE vector
    case 'manovra5'

        % Straight flight
        APflag=1;
        alt_c=H0; % [m]
        h_dot_c=0; % useless
        yaw_c=0; % useless
        beta_c=0; % useless
        V_c=0; % useless
        c1=[APflag alt_c h_dot_c yaw_c beta_c V_c];
        t1=0; % [s]
        m1=[t1 c1];

        % Turn with altitude increase
        APflag=4;
        alt_c=0; % useless
        h_dot_c=2;
        yaw_c=deg2rad(180);
        beta_c=0; % useless
        V_c=0; % useless
        c2=[APflag alt_c h_dot_c yaw_c beta_c V_c];
        t2=20; % [s]
        m2=[t2 c2];

        manoeuvre=[m1;m2];


% SECTFORM vector
    case 'in_cond + line_astern + echelon'

        xF1=PN_integr_in_cond_F1; xF2=PN_integr_in_cond_F2;
        xF3=PN_integr_in_cond_F3; xF4=PN_integr_in_cond_F4;
        yF1=PE_integr_in_cond_F1; yF2=PE_integr_in_cond_F2;
        yF3=PE_integr_in_cond_F3; yF4=PE_integr_in_cond_F4;
        hF1=0; hF2=0; hF3=0;hF4=0;
        c1=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
        t1=0; % [s]
```

```
39          f1=[t1 c1];
40          sectform=f1;
41          minfiguredist1=[t1 norm([xF1,yF1])]; % Max dist in form used to
                limit repulsive radius
42
43          % LINE ASTERN
44          xF1=-35; %[m] to modify
45          xF2=xF1*2; xF3=xF1*3; xF4=xF1*4;
46          yF1=0; yF2=0; yF3=0; yF4=0;
47          hF1=0; hF2=0; hF3=0;hF4=0;
48          c2=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
49          t2=50; % [s]
50          f2=[t2 c2];
51          minfiguredist2=[t1 norm([xF1,yF1])];
52
53           % ECHELON
54          xF1=-35; %[m] to modify
55          ang=45; %[deg] to modify
56          xF2=2*xF1; xF3=3*xF1; xF4=4*xF1;
57          yF1=xF1*tand(ang); yF2=2*yF1; yF3=3*yF1; yF4=4*yF1;
58          hF1=-10; hF2=-20; hF3=-30;hF4=-40;
59          c3=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
60          t3=100; % [s]
61          f3=[t3 c3];
62          minfiguredist3=[t3 norm([xF1,yF1])];
63
64          sectform=[f1;f2;f3];
65          minfiguredist=[minfiguredist1;minfiguredist2;minfiguredist3];
```

Listing B.3: APF 2D formation control - simulation #1 code

```
1  % SECTFORM vector
2      case 'sect sim1 APF 2D'
3
4          xF1=PN_integr_in_cond_F1; xF2=PN_integr_in_cond_F2;
5          xF3=PN_integr_in_cond_F3; xF4=PN_integr_in_cond_F4;
6          yF1=PE_integr_in_cond_F1; yF2=PE_integr_in_cond_F2;
7          yF3=PE_integr_in_cond_F3; yF4=PE_integr_in_cond_F4;
8          hF1=0; hF2=0; hF3=0;hF4=0;
9          targetpointF1=[xF1,yF1,hF1];
10         targetpointF2=[xF2,yF2,hF2];
11         targetpointF3=[xF3,yF3,hF3];
12         targetpointF4=[xF4,yF4,hF4];
13         c1=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
14         t1=0; % [s]
15         f1=[t1 c1];
16         minfiguredist1=[t1 norm([xF1,yF1])];
17
18         % VIC
19         xF1=-60; %[m] to modify
20         ang=60; %[deg] to modify
```

```
21          xF2=xF1; xF3=2*xF1; xF4=xF3;
22          yF1=xF1*tand(ang); yF4=2*yF1; yF2=-xF2*tand(ang); yF3=2*yF2;
23          hF1=-20; hF2=-20; hF3=-40;hF4=-40;
24          targetpointF1=[xF1,yF1,hF1];
25          targetpointF2=[xF2,yF2,hF2];
26          targetpointF3=[xF3,yF3,hF3];
27          targetpointF4=[xF4,yF4,hF4];
28          c2=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
29          t2=5; % [s]
30          f2=[t2 c2];
31          minfiguredist2=[t2 norm([xF1,yF1])];
32
33          % BOX
34          xF1=50; %[m] to modify
35          xF2=0; xF4=0; xF3=-xF1;
36          yF1=0; yF3=0; yF2=xF1; yF4=-yF2;
37          hF1=0; hF2=0; hF3=0;hF4=0;
38          targetpointF1=[xF1,yF1,hF1];
39          targetpointF2=[xF2,yF2,hF2];
40          targetpointF3=[xF3,yF3,hF3];
41          targetpointF4=[xF4,yF4,hF4];
42          c3=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
43          t3=120; % [s]
44          f3=[t3 c3];
45          minfiguredist3=[t3 norm([xF1,yF1])];
46
47          sectform=[f1;f2;f3];
48          minfiguredist=[minfiguredist1;minfiguredist2;minfiguredist3];
49
50  % MANOEUVRE vector
51      case 'manovra sim1 APF 2D'
52          % Straight flight
53          APflag=1;
54          alt_c=H0; % [m]
55          h_dot_c=0; % useless
56          yaw_c=0; % useless
57          beta_c=0; % useless
58          V_c=0; % useless
59          c1=[APflag alt_c h_dot_c yaw_c beta_c V_c];
60          t1=0; % [s]
61          m1=[t1 c1];
62
63          % First turn with  constant altitude
64          APflag=3;
65          alt_c=0; % useless
66          h_dot_c=0; % useless
67          yaw_c=deg2rad(-90);
68          beta_c=0; % useless
69          V_c=0; % useless
70          c2=[APflag alt_c h_dot_c yaw_c beta_c V_c];
71          t2=40; % [s]
72          m2=[t2 c2];
73
```

```
74          % Second turn with constant altitude
75          APflag=3;
76          alt_c=0; % useless
77          h_dot_c=0; % useless
78          yaw_c=deg2rad(-180);
79          beta_c=0; % useless
80          V_c=0; % useless
81          c3=[APflag alt_c h_dot_c yaw_c beta_c V_c];
82          t3=120; % [s]
83          m3=[t3 c3];
84
85          manoeuvre=[m1;m2;m3];
86
87  % OBSTACLE creation
88      case 'no-flyzone sim1 APF 2D'
89          obs_on=1;
90          X_NED_NORD_obs1=4000;
91          Y_NED_EAST_obs1=-2000;
92          Z_NED_H_obs1=4000;
93          X_NED_vel_obs1=0;
94          Y_NED_vel_obs1=0;
95          Z_NED_vel_obs1=0;
96          R_obs1=2800/2;
97          R_pot1=R_obs1*2; %=R_obs1+toll
```

Listing B.4: APF 2D formation control - simulation #2 code

```
1   % SECTFORM vector
2       case 'sect sim2 APF 2D'
3
4           xF1=PN_integr_in_cond_F1; xF2=PN_integr_in_cond_F2;
5           xF3=PN_integr_in_cond_F3; xF4=PN_integr_in_cond_F4;
6           yF1=PE_integr_in_cond_F1; yF2=PE_integr_in_cond_F2;
7           yF3=PE_integr_in_cond_F3; yF4=PE_integr_in_cond_F4;
8           hF1=0; hF2=0; hF3=0;hF4=0;
9           targetpointF1=[xF1,yF1,hF1];
10          targetpointF2=[xF2,yF2,hF2];
11          targetpointF3=[xF3,yF3,hF3];
12          targetpointF4=[xF4,yF4,hF4];
13          c1=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
14          t1=0; % [s]
15          f1=[t1 c1];
16          sectform=f1;
17          minfiguredist1=[t1 norm([xF1,yF1])]; % Max dist in form used to
                limit repulsive radius
18          minfiguredist=minfiguredist1;
19
20          % BOX
21          xF1=75; %[m] to modify
22          xF2=0; xF4=0; xF3=-xF1;
23          yF1=0; yF3=0; yF2=xF1; yF4=-yF2;
24          hF1=0; hF2=0; hF3=0;hF4=0;
25          targetpointF1=[xF1,yF1,hF1];
26          targetpointF2=[xF2,yF2,hF2];
```

```
27          targetpointF3=[xF3,yF3,hF3];
28          targetpointF4=[xF4,yF4,hF4];
29          c2=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
30          t2=50; % [s]
31          f2=[t2 c2];
32          minfiguredist2=[t2 norm([xF1,yF1])]; % Max dist in form used to
                limit repulsive radius
33
34         % ECHELON
35          xF1=-35; %[m] to modify
36          ang=45; %[deg] to modify
37          xF2=2*xF1; xF3=3*xF1; xF4=4*xF1;
38          yF1=xF1*tand(ang); yF2=2*yF1; yF3=3*yF1; yF4=4*yF1;
39          hF1=-10; hF2=-20; hF3=-30;hF4=-40;
40          targetpointF1=[xF1,yF1,hF1];
41          targetpointF2=[xF2,yF2,hF2];
42          targetpointF3=[xF3,yF3,hF3];
43          targetpointF4=[xF4,yF4,hF4];
44          c3=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
45          t3=100; % [s]
46          f3=[t3 c3];
47          minfiguredist3=[t3 norm([xF1,yF1])]; % Max dist in form used to
                limit repulsive radius
48
49          sectform=[f1;f2;f3];
50          minfiguredist=[minfiguredist1;minfiguredist2;minfiguredist3];
51
52  % OBSTACLE creation
53      case 'no-flyzone sim2 APF 2D'
54          obs_on=1;
55          X_NED_NORD_obs1=4000;
56          Y_NED_EAST_obs1=-2000;
57          Z_NED_H_obs1=4000;
58          X_NED_vel_obs1=0;
59          Y_NED_vel_obs1=0;
60          Z_NED_vel_obs1=0;
61          R_obs1=2800;
62          R_pot1=R_obs1*2; %=R_obs1+toll
```

Listing B.5: APF 3D formation control - simulation #1 code

```
1   % SECTFORM vector
2       case 'sect sim1 APF 3D'
3
4           xF1=PN_integr_in_cond_F1; xF2=PN_integr_in_cond_F2;
5           xF3=PN_integr_in_cond_F3; xF4=PN_integr_in_cond_F4;
6           yF1=PE_integr_in_cond_F1; yF2=PE_integr_in_cond_F2;
7           yF3=PE_integr_in_cond_F3; yF4=PE_integr_in_cond_F4;
8           hF1=0; hF2=0; hF3=0;hF4=0;
9           targetpointF1=[xF1,yF1,hF1];
10          targetpointF2=[xF2,yF2,hF2];
```

```
11          targetpointF3=[xF3,yF3,hF3];
12          targetpointF4=[xF4,yF4,hF4];
13          c1=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
14          t1=0; % [s]
15          f1=[t1 c1];
16          sectform=f1;
17          minfiguredist1=[t1 norm([xF1,yF1])]; % Max dist in form used to
                limit repulsive radius
18
19          % ARROW
20          xF1=75; %[m] to modify
21          xF2=-xF1; xF3=-xF1; xF4=-xF1;
22          yF1=0; yF2=0; yF3=-xF1;yF4=xF1;
23          hF1=0; hF2=0; hF3=0;hF4=0;
24          targetpointF1=[xF1,yF1,hF1];
25          targetpointF2=[xF2,yF2,hF2];
26          targetpointF3=[xF3,yF3,hF3];
27          targetpointF4=[xF4,yF4,hF4];
28          c2=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
29          t2=25; % [s]
30          f2=[t2 c2];
31          minfiguredist2=[t2 norm([xF1,yF1])];
32
33          % LINE ASTERN
34          xF1=-30; %[m] to modify
35          xF2=xF1*2; xF3=xF1*3; xF4=xF1*4;
36          yF1=0; yF2=0; yF3=0; yF4=0;
37          hF1=-20; hF2=-40; hF3=-60;hF4=-80;
38          targetpointF1=[xF1,yF1,hF1];
39          targetpointF2=[xF2,yF2,hF2];
40          targetpointF3=[xF3,yF3,hF3];
41          targetpointF4=[xF4,yF4,hF4];
42          c3=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
43          t3=70; % [s]
44          f3=[t3 c3];
45          minfiguredist3=[t3 norm([xF1,yF1])]; % Max dist in form used to
                limit repulsive radius
46
47          sectform=[f1;f2;f3];
48          minfiguredist=[minfiguredist1;minfiguredist2;minfiguredist3];
49
50  % MANOEUVRE vector
51      case 'manovra sim1 APF 3D'
52          % Volo rettilineo
53          APflag=1;
54          alt_c=H0; % [m]
55          h_dot_c=0; % useless
56          yaw_c=0; % useless
57          beta_c=0; % useless
58          V_c=0; % useless
59          c1=[APflag alt_c h_dot_c yaw_c beta_c V_c];
60          t1=0; % [s]
61          m1=[t1 c1];
```

```
62
63          % Virata con aumento di quota
64          APflag=4;
65          alt_c=0; % useless
66          h_dot_c=2;
67          yaw_c=deg2rad(60);
68          beta_c=0; % useless
69          V_c=0; % useless
70          c2=[APflag alt_c h_dot_c yaw_c beta_c V_c];
71          t2=20; % [s]
72          m2=[t2 c2];
73
74          % Richiamata
75          m3=zeros(1,7);
76          n_rich=100;
77          hdotdes=linspace(-10,10,n_rich);
78          trich=linspace(90,140,n_rich);
79
80
81          for i=1:n_rich
82              APflag=2;
83              alt_c=0;% useless
84              h_dot_c=hdotdes(1,i); % [m/s]
85              yaw_c=0; % useless
86              beta_c=0; % useless
87              V_c=0; % useless
88              c=[APflag alt_c h_dot_c yaw_c beta_c V_c];
89              t=trich(1,i); % [s]
90              m3(i,:)=[t c];
91          end
92
93          manoeuvre=[m1;m2;m3];
94
95  % OBSTACLE creation
96      case 'sim1 APF 3D'
97          obs_on=1;
98          X_NED_NORD_obs1=-200;
99          Y_NED_EAST_obs1=70;
100         Z_NED_H_obs1=H0;
101         X_NED_vel_obs1=V_0+10;
102         Y_NED_vel_obs1=0;
103         Z_NED_vel_obs1=0;
104         R_obs1=30;
105         R_pot1=R_obs1*2; %=R_obs1+toll
```

Listing B.6: *sim3tesi2D* function - simulation #3

```
1  function [sectformfig,minfiguredist] = sim3tesi2D(orologio,psi_leader,
       x_leader,y_leader)
2
3  t2=5;
```

```

% FOLLOWER 1
PN_integr_in_cond_F1=-400; % [m]
PE_integr_in_cond_F1=280.083; % [m]

% FOLLOWER 2
PN_integr_in_cond_F2=-400;% [m]
PE_integr_in_cond_F2=-280.083; % [m]

% FOLLOWER 3
PN_integr_in_cond_F3=-800; % [m]
PE_integr_in_cond_F3=560.166; % [m]

% FOLLOWER 4
PN_integr_in_cond_F4=-800; % [m]
PE_integr_in_cond_F4=-560.166; % [m]


if orologio<t2
    % INITIAL POSITIONS
    xF1=PN_integr_in_cond_F1; xF2=PN_integr_in_cond_F2;
    xF3=PN_integr_in_cond_F3; xF4=PN_integr_in_cond_F4;
    yF1=PE_integr_in_cond_F1; yF2=PE_integr_in_cond_F2;
    yF3=PE_integr_in_cond_F3; yF4=PE_integr_in_cond_F4;
    hF1=0; hF2=0; hF3=0;hF4=0;
    targetpointF1=[xF1,yF1,hF1];
    targetpointF2=[xF2,yF2,hF2];
    targetpointF3=[xF3,yF3,hF3];
    targetpointF4=[xF4,yF4,hF4];
    sectformfig=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
    minfiguredist=norm([xF1,yF1]);

elseif (orologio>t2) && ((8000-x_leader)>100 && (1250-y_leader)>100)

    xF2=PN_integr_in_cond_F2;
    xF3=PN_integr_in_cond_F3; xF4=PN_integr_in_cond_F4;
    yF2=PE_integr_in_cond_F2;
    yF3=PE_integr_in_cond_F3; yF4=PE_integr_in_cond_F4;
    hF1=0; hF2=0; hF3=0;hF4=0;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    xF11=8000-x_leader; yF11=1150-y_leader;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    R=[cos(psi_leader) sin(psi_leader);-sin(psi_leader) cos(psi_leader)];
    dpos_rot=(R*[xF11;yF11]);
    xF1=dpos_rot(1);
    yF1=dpos_rot(2);

    targetpointF1=[xF1,yF1,hF1];
    targetpointF2=[xF2,yF2,hF2];
    targetpointF3=[xF3,yF3,hF3];
    targetpointF4=[xF4,yF4,hF4];
```

```
57      sectformfig=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
58      minfiguredist=norm([xF2,yF2]);
59
60  else
61      % ECHELON
62      xF1=-250; %[m] to modify
63      ang=-40; %[deg] to modify
64      xF2=2*xF1; xF3=3*xF1; xF4=4*xF1;
65      yF1=xF1*tand(ang); yF2=2*yF1; yF3=3*yF1; yF4=4*yF1;
66      hF1=0; hF2=0; hF3=0;hF4=0;
67      targetpointF1=[xF1,yF1,hF1];
68      targetpointF2=[xF2,yF2,hF2];
69      targetpointF3=[xF3,yF3,hF3];
70      targetpointF4=[xF4,yF4,hF4];
71      sectformfig=[xF1,yF1,hF1,xF2,yF2,hF2,xF3,yF3,hF3,xF4,yF4,hF4];
72      minfiguredist=norm([xF1,yF1]);
73  end
74  end
```

Bibliography

[1] Encyclopaedia Britannica. Formation flying. https://www.britannica.com/technology/formation-flying.

[2] professionalaviation.it. Le frecce tricolori: la storia della pattuglia acrobatica nazionale. https://www.professionalaviation.it/le-frecce-tricolori-la-storia-della-pattuglia-acrobatica-nazionale/.

[3] Formation and Safety Team (FAST). The formation pilots' knowledge guide - version 2.0. http://flyfast.org/sites/all/docs/FAST_FKG_2.0.pdf.

[4] Lieutenant Colonel L. Rossetti. Manned-unmanned teaming: A great opportunity or mission overload? *The Journal of the JAPCC*, 29, 2020.

[5] Leonardo Company. A hero for our times. https://www.leonardo.com/en/news-and-stories-detail/-/detail/a-hero-for-our-times.

[6] Mário Monteiro Marques. Stanag 4586 - standard interfaces of uav control system (ucs) for nato uav interoperability.

[7] NASA. Past projects: Autonomous formation flight (aff). https://www.nasa.gov/centers/dryden/history/pastprojects/AFF/index.html.

[8] W. B. Blake et al. Surfing aircraft vortices for energy. *Journal of Defence Modeling and Simulation: Applications, Methodology, Technology*, 12:31–39, 2015.

[9] G. S. Schkolnik and B. Cobleigh. Autonomous formation flight: a primary goal is to reduce fuel consumption during cruise by 10 percent. *NASA Tech Briefs*, pages 20–21, 2004.

[10] D. Rhodes L. Jenkinson, R. Caves. Automatic formation flight - a preliminary investigation into the application to civil operations. *AIAA*, 2012.

[11] NASA. Aard - autonomous airborne refueling demonstration. https://ntrs.nasa.gov/api/citations/20070025036/downloads/20070025036.pdf.

[12] A. Tsukerman et al. Optimal rendezvous guidance laws with application to civil autonomous aerial refueling. *Journal of Guidance,Control and Dynamics*, 41, 2018.

[13] Bradley Perret. Moving fast. *AviationWeek.com*, April-May 2022.

[14] R. Bucknall Y. Liu. A survey of formation control and motion planning of multiple unmanned vehicles. *Robotica*, 36:1019 –1047, 2018.

[15] Kwang-Kyo Oha et al. A survey of multi-agent formation control. *Automatica*, 53:424–440, 2015.

[16] M. Tillerson et al. J.S. Bellingham. Cooperative path planning for multiple uavs in dynamic and uncertain environments. *Proceedings of the 41st IEEE Conference on Decision and Control*, 2002.

[17] P. K. C. Wang. Navigation strategies for multiple autonomous mobile robots moving in formation. *Journal of Robotic Systems*, 8, 1991.

[18] R.Napolitano et al. Y. Gu, G.Campa. Design and flight-testing evaluation of formation control laws. *IEEE Transactions on Control Systems Technology*, 14, 2006.

[19] J.P.Desai et al. Controlling formations of multiple mobile robots. *IEEE International Conference on Robotics and Automation*, 1998.

[20] J. Horn A. Dang. Formation control of leader-following uavs to track a moving target in a dynamic environment. *Journal of Automation and Control Engineering*, 3, 2014.

[21] K. Tan M.A. Lewis. High precision formation control of mobile robots using virtual structures. *Autonomous Robots*, 4:387–403, 1997.

[22] R. C. Arkin T. Balch. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 55:926–939, 1998.

[23] Hui Liu. *Robot Systems for Rail Transit Applications*. Elsevier, 2020.

[24] M. Shanmugavel et al. Co-operative path planning of multiple uavs using dubins paths with clothoid arcs. *Control Engineering Practice*, 18:1084–1092, 2010.

[25] P. U. Lima S. Garrido, L. Moreno. Robot formation motion planning using fast marching. *Robotics and Autonomous Systems*, 59:675–683, 2011.

[26] E. Feron T. Schouwenaars, J. How. Receding horizon path planning with implicit safety guarantees. *Proceedings of the American Control Conference*, 6:5576–5581, 2004.

[27] I. Petrović M. Dakulović. Two-way d star algorithm for path planning and replanning. *Robotics and Autonomous Systems*, 59:329–342, 2011.

[28] Y. Ayaz A. H. Qureshi. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robotics and Autonomous Systems*, 68:1–11, 2015.

[29] P. M. Morse. Diatomic molecules according to the wave mechanics. *Physical Review*, 34:57–64, 1929.

[30] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5:396–404, 1986.

[31] Motion Imagery Standard Board. Misb st 0601.8. Standard - UAS Datalink Local Set.

[32] R.Nelson. *Flight stability and automatic control - second edition*. McGraw-Hill, 1998.

[33] Matematical Association of America (MAA). Historical activities for calculus - module 3: Optimization – galileo and the brachistochrone problem.

[34] W.F. Denham A.E. Bryson. A steepest -ascent method for solving optimum programming problems. *Journal of Applied Mechanics*, 1962.

[35] N. Minorsky. Directional stability of automatically steered bodies. *Journal of the American Society for Naval Engineers*, 34:280–309, 1922.

[36] the Free Encyclopedia Arturo Urquizo Wikipedia. Pid controller. `https://en.wikipedia.org/wiki/PID_controller` (accessed on 17 July 2022).

[37] F. Dabbene. L2 - control specifications and pid control, 2021. Lessons for the *Dynamics and control of space vehicles* course.

[38] G. Campa. "3dscope" block - matlab file exchange. `it.mathworks.com/matlabcentral/fileexchange/4915-3dscope` (accessed on 17 July 2022).

[39] R.Napolitano et al. G.Campa. Development of formation flight control algorithms using 3 yf-22 flying models. 2007.

[40] ECSM Wing Air Training Corps. Aircraft handling manual. `http://www.967atc.co.uk/wordpress/wp-content/uploads/2011/09/AircraftHandlingManualECSM.pdf` (accessed on 20 July 2022).

[41] L. Sciavicco et al. B. Siciliano. *Robotics - Modelling, planning and control*. Springer, 2009.

[42] G.Punzo. Verifiable swarm engineering with limited communication. 2013.

[43] et al. H. Choset. *Principles of Robot Motion - Theory, Algorithms and Implementation*. MIT Press, 2005.

[44] X. Zhu et al. A flexible collision avoidance strategy for the formation of multiple unmanned aerial vehicles. 2019.

[45] MATLAB Documentation. Visualizing four-dimensional data. `https://it.mathworks.com/help/matlab/visualize/visualizing-four-dimensional-data.html`.