Politecnico di Torino

Department of Mechanical and Aerospace Engineering

Master's Degree Thesis in Aerospace Engineering

Integration of a Tilt-Rotor Flight Simulation Platform

Supervisors
Giorgio **Guglieri**
Federico **Barra**

Candidate
Federico **Veronese**

A.A. 2020-2021

To my mother, for the affection

To my father, for the passion

To Ambra, for the love

A mia madre, per l'affetto

A mio padre, per la passione

Ad Ambra, per l'amore

ABSTRACT

The main purpose of this thesis was to implement the real-time simulation model of the Bell Aircraft Corporation XV-15 aircraft to give a face to the mathematical model describing the flight mechanics of the tilt-rotor. The mathematical model has been developed thanks to several thesis works carried out at the Politecnico di Torino in collaboration with the ZHAW University located in Winterthur, Switzerland, and includes simplified mathematical models for the aerodynamics of the whole aircraft, rotors, and engine dynamics.

The author's task was therefore to take the model in question and make it work with a graphic environment that reproduces the aircraft following the commands given by the pilot. The visualisation environment chosen is FlightGear, an open-source and cross-platform software that is widely used in research for its characteristics.

The simulator is therefore made up of a portable workstation running the simulation model, written in MATLAB/Simulink® language, and a hardware input consisting of a USB flight stick and a pedal board. The hardware then communicates with the Simulink software, which evaluates the status of the aircraft, including position and attitude, but also including the rotational speed of the rotors, the position of the moving surfaces and the status of the landing gears, which are sent to the FlightGear environment, that reproduces an animated 3D version of the aircraft.

The FlightGear simulation environment also has the task of reproducing a simplified view of the portion of planet Earth on which the aircraft is located, including aerial infrastructures but also vegetation and various obstacles.

Fundamental work was then to synchronise the processor clock with that of the physical simulation to make the model truly real-time, thanks to the use of the Simulink® Desktop Real-Time™ library and special cares in the use of the software to speed up the simulation.

CONTENTS

Abstract III

1 Introduction 1

1.1 Flight Simulators 1
1.1.1 Purposes of Flight Simulators 4
1.1.2 The importance of Flight Simulators 5
1.1.3 Flight Simulator Standards 6
1.1.4 Real-Time simulation 7
1.2 Tilt-Rotors 8
1.2.1 Growing interest in Tilt-Rotors 9
1.2.2 Bell XV-3 10
1.2.3 Bell XV-15 11
1.3 Tilt-Rotors - Military and Civil developments 12
1.3.1 V-22 Osprey 12
1.3.2 Bell V-280 Valor 13
1.3.3 AgustaWestland AW609 14
1.4 Electric Tilt-Rotors 15

2 Mathematical Model 16

2.1 Tilt-Rotor model 16
2.1.1 Inputs 17
2.1.2 Flight Control Computer – FCC 17
2.1.3 Actuators 17
2.1.4 Physics 18
2.1.5 Equations of Motion – EoM 20
2.1.6 Data Exchange 20
2.2 Reference systems 20
2.2.1 Geodesy and Coordinate Systems 22
2.2.2 Latitude and Longitude 24
2.2.3 Geocentric and Geodetic coordinates 25

2.2.4 North-East-Down reference system 26
2.2.5 Earth's radii of curvature 26
2.2.6 Flat Earth to LLA reference transformation 27
2.3 Solving Differential Equations 28
2.3.1 Laplace Transform Properties 28
2.3.2 Laplace and steady-state 30
2.3.3 Z-transform 31
2.4 Communication Protocols 33
2.4.1 UDP – Universal Datagram Protocol 34
2.4.2 UDP Packet structure 35

3 Simulation Environment and Architecture 36

3.1 FlightGear 37
3.1.1 Aircraft Reproduction 37
3.1.2 TerraSync & TerraMaster 39
3.1.3 How to launch FlightGear 40
3.1.4 Simulink Interface 41
3.2 Simulink 43
3.2.1 Simulink Acceleration 43
3.2.2 Real Time Simulation 45
3.2.3 SLDRT - Simulink® Desktop Real-Time™ 46
Real-Time Sync block 47
Input blocks 47
Packet Input and Packet Output blocks 49
3.2.4 Hardware Interface in Simulink 51
3.2.5 FlightGear Interface 53
3.3 Flight controls 54
3.3.1 Controls Normalization 55
3.4 Hardware 59
3.4.1 Computer 59
3.4.2 Flight Stick 60
3.4.3 Rudder Pedals 61

4 Simulations and Results 62

4.1 Longitudinal Translation 63

4.2 Hovering Near Ground 67
4.3 Take-Off + Conversion + Landing 72

5 Conclusions 77

5.1 Achievements 77
5.2 Future developments 77

List of Figures 84

List of Tables 86

Bibliography 87

1

Introduction

1.1 Flight Simulators

Flight simulators are devices that allows to reproduce a virtual environment, in which an aircraft is reproduced according to the mathematical equations that govern the dynamics of flight, the dynamics of on-board systems, but also meteorology and much more.

Fig. 1.1 shows the main components of a flight simulator. The equations of motion are the most important part of all flight simulators. Their function is to determine the states of the simulator by taking all inputs, including pilot controls, winds, aerodynamic terms, and engine terms to calculate the variables that represent the state of the simulated aircraft, specifically forces, moments, attitude, altitude, direction, and speed. The equations of motion, in most flight simulators, are updated 50 to 60 times per second.

The aerodynamic model allows aerodynamic forces and moments to be calculated. The aerodynamic data is provided as a large database obtained from a combination of flight tests, wind tunnel tests, and possibly computational fluid dynamics (CFD) analysis. It will also include a large amount of validation data to allow the simulator developer to compare the simulator's dynamics and performance to actual aircraft data. The aerodynamic model is the most critical element of a flight simulator: in fact, an error in modelling the aerodynamics of the aircraft can lead to a simulation that may fail in the qualification process or be unacceptable to pilots who have experience in this type of aircraft.

Development of the engine requires a model of the engine dynamics that will be used in any control system design activities. The engine manufacturer will undertake extensive testing in the development of the engine. The engine data is dependent on the state of the aircraft; implementation of the engine model requires access to variables calculated in the flight model.

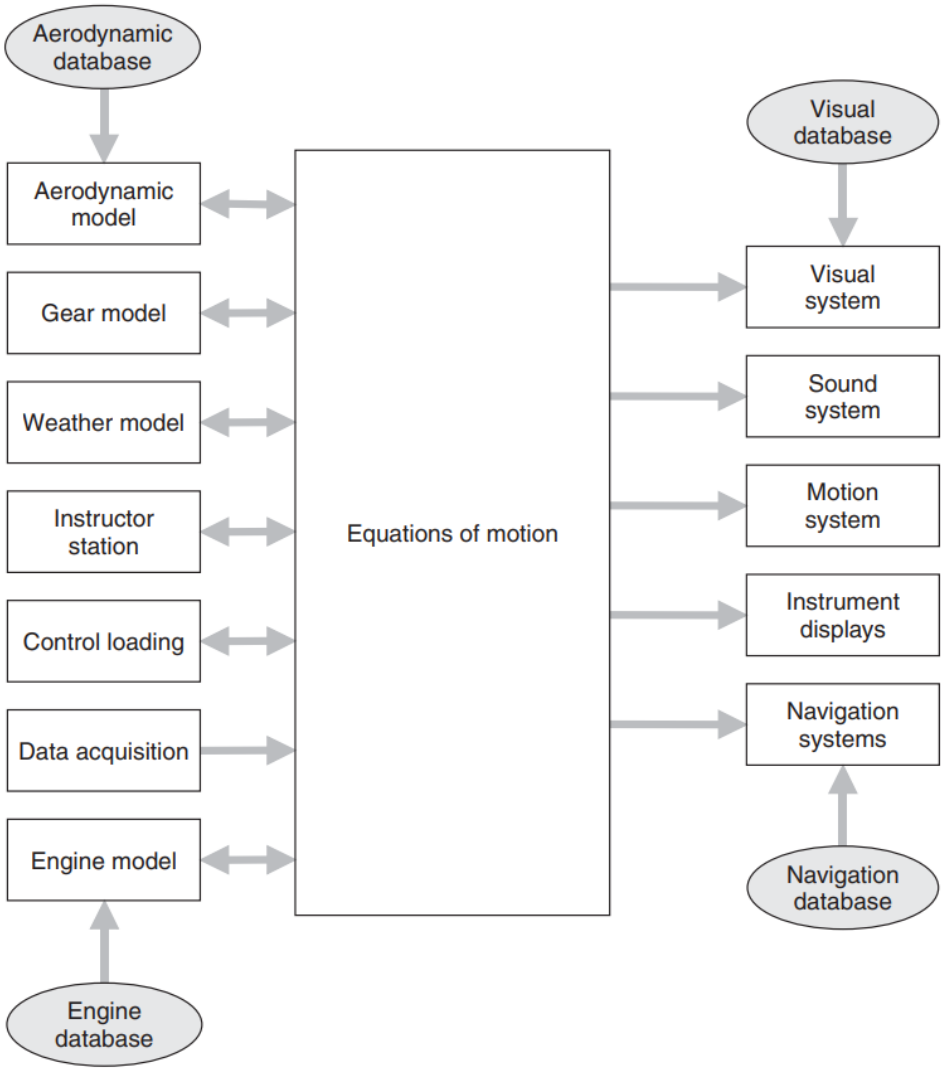

Figure 1.1: Organization of a Flight Simulator [1]

The inputs can provide discrete data (0 or 1) or analogue data. They must be sampled, converted to an appropriate value, and passed to the simulator module.

When dealing with many inputs, there are multiple processors dedicated to the data acquisition function. In the case of analogue data, the data acquisition software is responsible for minimizing any delay in data acquisition, ensuring that the data resolution is as high as possible, and that any conditioning or filtering of the data signal is properly applied.

An additional model is usually provided for ground handling to include the effects of the tires and undercarriage assembly, which are responsible for different dynamics than the aircraft in flight. In practice, there are additional state transitions, just before take-off and just after landing, where the aerodynamic contribution to motion is combined with the dynamics of the undercarriage assembly.

Aircraft performance is, of course, affected by the atmosphere. In fact, equations are implemented to calculate air pressure, air density, and air temperature. Wind influences both navigation and aircraft handling, for example, in crosswind or

turbulence landings. The wind model must be three-dimensional and time varying as it plays an important role in flight planning. The model must account for pressure fronts, altitude, and position. An alternative approach is to use wind data acquired from agencies that monitor global wind currents.

The visual system provides a series of real-time image channels viewed from the position of the pilot's eye. A database is loaded into the memory of the visual system, which may contain fields, airfields, roads, lakes, coastlines, vehicles, buildings, trees, forests, vegetation, and aircraft. Each object is reduced to coloured polygons (usually triangles), defined in the database coordinate system. As the aircraft manoeuvres, the position and orientation of the pilot's eye are computed in the equations of motion and the scene is rendered at each frame (typically 60 Hz). There is a delay between the acquisition of a new eye position and the pilot seeing the projected image. It is called visual latency and must be kept to a minimum but can extend to three or four frames.

Noisy environments are present in aircraft, such as the cockpit or flight deck. In addition, other noises from the engine, subsystems, wake, actuators, or alarms are present. Although sounds are provided in a simulator to increase fidelity, they are also important cues and should be consistent with the sounds heard in an aircraft. Some sounds, such as airspeed or engine revolutions per minute (RPM), vary with flight conditions, while other sounds provide a consistent tone. Usually, a separate sound system is provided that takes input from other modules.

As the simulated aircraft is maneuvered, the pilot expects to feel the accelerations that would be experienced in real flight. The accelerations are calculated in the flight model and are passed to the motion system. For the standard motion platform that includes six linear hydraulic actuators, each actuator is moved to a new position to try to replicate the accelerations on the pilot's body. The motion system contains computers to derive the actuation equations to move the platform to the desired position, along with filters to optimize the platform trajectory and provide appropriate motion for the pilot's balance sensors. However, motion evaluation is a very important aspect of simulator qualification and is controlled for critical phases of flight, particularly take-off, landing, and engine failures.

When an aircraft flies, the wake that passes over the control surfaces changes the primary flight controls: the elevator, aileron, and rudder control surfaces. To account for these effects, control loading is used, provided by attaching actuators to the flight controls in the simulator so that the actuator provides resistance to motion, typically varying with airspeed. The control load is also an important part of simulator qualification. With both hydraulic and electrical control load systems, the trimming function is simply implemented as an offset of the null datum for the zero-load position.

Civilian and military aircraft take advantage of electronic flight instruments, known as EFIS displays, based on computer graphics with 8 in. ruggedized monitors,

typically with the displays refreshing at least 20 times per second (20 Hz). The graphics hardware in the display has a very fast drawing speed to support the frame rate and includes anti-aliasing algorithms to smooth out any jagged lines or edges as characters and lines are rendered.

A significant part of flight training involves navigation training. Flight simulation offers two advantages: first, an in-flight navigation exercise consumes fuel, and second, navigation errors in training can be dangerous. Consequently, simulators provide varying degrees of navigation skills.

The simulator should be checked regularly to ensure that it is operating within limits. In fact, for qualified simulators, the operator must keep records of all scheduled and unscheduled maintenance and repeat diagnostic tests to confirm that the simulator's characteristics have not changed significantly after any maintenance procedures.

1.1.1 Purposes of Flight Simulators

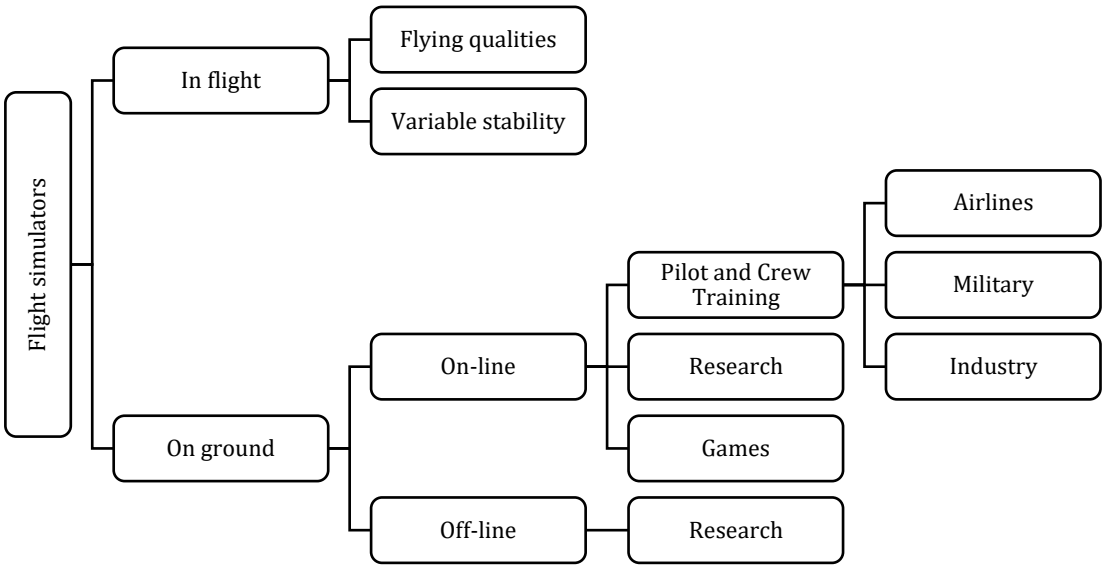

Figure 1.2: Main purposes of flight simulators

Interest in the development and construction of modern flight simulators is driven by several purposes:

- Pilot Training: modern flight simulators have become a fundamental part of pilot and cockpit crew training.
- Research: flight simulators can also be used for research purposes, to produce the mathematical models or to validate them, providing an experimental result without preparing physical tests of the system.
- Games: aviation has always aroused a lot of interest from pilots for recreational purposes, for this reason the market has always presented an

offer of video games that produce a millionaire turnover for the companies Tilt-Rotors concept of operations.

- In the case of "In flight" simulators, a simulator is placed on a real aircraft to understand the operation of the control system.

1.1.2 The importance of Flight Simulators

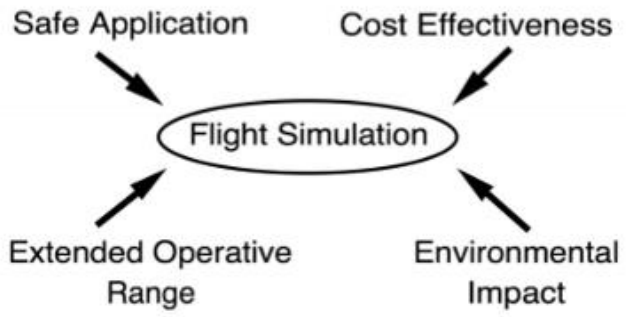

Figure 1.3: Main Advantages of flight simulators

- Simulators on an industrial level are useful as they ensure safety for training. In addition, they are essential for certifying aircraft and verifying the envelope diagram in a more realistic manner and prior to flight test, and thus verifying the feasibility of the manoeuvre.
- Flying a simulator compared to a real aircraft is, of course, less expensive.
- There are no problems in the operating ranges. It is possible also to explore unforeseen cases in order to understand, for example, the dynamics of an accident.
- They are very important from an environmental point of view for flight tests to reduce the number of pilot flight hours.

There are project and training simulators. The former does not operate in real time and are used for engineering. They use complex calculation tools based on very complex data and can take structure elasticities into account. A correlation is created between the simulation model and the engineering model to understand how to vary the mathematical simulation model and reapply it to the physical project. Interdisciplinary models are used that cannot be used for real-time simulators because the computers are too slow, even if the fidelity is very high. Thus, real-time simulator models are simpler parametric models derived from statistics that do not follow the real physical model, but faithfully reproduce the dynamics useful for training.

Military training uses simulators at different levels, from flight familiarisation to combat. Simulators focus on military risks and tend to recreate conditions that are

not even easily reproducible during training. These trainers are based on a synthetic scenario in which there are also other infrastructures, military dynamics, even hostile ones (radar, enemies, etc.), and the pilot is trained in combat and countermeasures for survival. In civil training, the full flight simulator is often used for airliners, but training for PPL is also done on the simulator, which has proved to be very valid. There are also maintenance training aircrafts that are even in VR. They allow to learn maintenance procedures consistently (not force actions) and to direct information giving warnings. In the environment there can also be a virtual instructor who may not be physically present. It is also useful for safety because there can be risks for the maintainer even during training. These environments remove the risk and allow the maintainer to become familiar with the procedures.

1.1.3 Flight Simulator Standards

Standards and normative references are also important for flight simulation. At the beginning of simulation development, standards were imposed by the manufacturers of flight simulators. The reproducibility of the simulators was low because there was no standard. Simulators were bought by companies in good faith thinking that the manufacturer could produce a sensible simulator. This made it difficult to think of using simulators for systematic training or skills validation, so flight experience was always required and in fact subjective considerations were made that lacked normative validation of the simulator.

At the beginning of the 1970s, the possibility of standardisation bodies began to arise with manufacturers to standardise the methods for the design of training simulators, and companies immediately joined in through IATA because they understood the usefulness. A technical committee was set up to lay the foundations for the development of standards and they joined forces.

In 1995, the ICAO, which is a third party with respect to certification and companies, produced a technical manual for flight simulators that is still used today. This manual led to the creation of working groups within FAA and EASA that implemented the ICAO guidelines. EASA calls it CS-FSTD (Flight Simulation Training Device) declined in Aircraft and Helicopters and contains all the simulation specifications that the designer must follow for certification. In Italy, ENAC follows these standards and enforces them.

The simulator must be certified via software and hardware. The software must also be recognized by regulatory bodies to be certified for training; the hardware must be consistent with the requirements and the flight controls must be compatible with the aircraft to be simulated. Their combination must also be validated.

In the civilian sector, no one certifies a simulator except for the training of pilots. The Full Flight Simulator is the best case and allows to train the pilot in accelerations. The motion system is not consistent with the dynamics in flight

because, if it were consistent with the angular excursions and linear accelerations, enormous excursions would be needed. What is admitted as feedback is the transitory part of the dynamics that the pilot perceives during the flight. FFS can also be used to maintain the license, such as EASA which requires minimum hours to maintain the rating.

Flight Training Devices (FTD) and Flight Navigation Procedures Trainers (FNPT) differ in motion feedback since FFS has motion system. The commercial pilot can use the FTD and FNPT for some training, but the validation of the pilot rating is done only in the FFS. The FTDs have a cockpit equal to the real aircraft, but the visual system is simpler and allows the use of cheaper technological solutions. FNPT has an open cockpit and with simplified and reduced controls, not equal to the real aircraft.

There is a lower category, called Basic Instrument Training Devices (BITD), which is used for instrumental training and to give the pilot the opportunity to familiarize himself with the instruments and the cockpit.

Integrated Procedures Trainers (IPT) is an anomalous training simulator in which the flight dynamics disappear and the pilot flies with the instruments and does not receive visual or sensory perceptions. These tools are important to familiarize with the procedures connected to them and detached from the perceptions of piloting. This is to train pilots and gain confidence in air traffic management.

1.1.4 Real-Time simulation

In normal everyday life, everything seems to be continuous and instantaneous. Computation is a very different world in which a computer executes the instructions of a given program. Each of these instructions requires a finite number of machine cycles, and each of these cycles is synchronized to the speed of the processor. In other words, for a given computer, a small snippet of code may take several microseconds to execute. What the operating system does is discretize time, in very small-time steps, such that a seemingly instantaneous and continuous response is guaranteed.

Exactly the same situation occurs in flight simulation, where the position of the stick is sampled, the elevator deflection is calculated, a new pitch attitude is calculated, and an image is displayed by the visual system with the new pitch attitude, allowing the pilot to correct the attitude of the aircraft. The important point is that the overall time for this calculation must be short enough so that it appears instantaneous to the pilot. In a modern simulator, these calculations must be completed within 1/50th of a second or 20 ms.

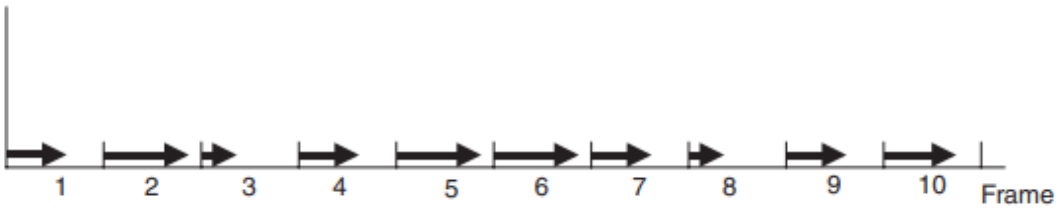

Figure 1.4: Real-Time frames [1]

This concept is illustrated in Figure 1.4, which represents 10 frames of a simulation. The arrow for each frame shows the proportion of the frame used in the simulation calculation. If the frame time is small enough, say 1/50th of a second, and if the computation in each frame never exceeds the frame time, then the simulation is in real time.

A real-time simulation requires a fast computer, but all calculations must be completed within the time limit, unlike a fast simulation where the only metric is overall time. The operating system must guarantee execution of the simulation task every frame and never introduce delays that cause the simulation task to exceed its frame limit. Ensuring real-time performance, especially for the worst conditions, is an essential part of system validation and acceptance testing. [1]

1.2 Tilt-Rotors

Since the beginning of aviation, due to technological and engineering limitations, aircraft were divided into those that and those that require movement itself to generate lift, and those which can maintain stationary flight, which at first were the lighter-than-air aircraft, and then helicopters,

Both configurations have always been used simultaneously to exploit the most of their peculiarities.

- **Aircrafts:** thanks to innovations in aircraft design, manufacture and operation, aircraft today can achieve ever higher cruise speeds, range, cruise altitude and payload capacity. However, at low speeds there are significant stability, control, and performance issues for fixed-wing aircraft. The latter over time has been exploited for its improved top speed characteristic. For these reasons, given the commercial interest in developing fast aircraft that could cover great distances, the minimum speed has also been progressively increased. This has led to an increase in the space required for aircraft take-off and landing and, therefore, a reduced ability of aircraft to integrate with civil transport in densely populated areas. It follows that airports are located quite far from urban centres, thus reducing the time gain offered by air

transport compared to other means such as rail and road that are more easily integrated into the urban fabric.

- **Helicopters:** helicopters have emerged as a solution particularly suitable for a wide variety of missions, both civil and military, thanks to their low- speed performance and their ability to take off and land vertically. In fact, traditional airplanes are hardly able to complete missions such as: search and rescue, surveying large areas, supporting troops and police forces, point-to-point transportation in isolated sites or difficult environments, aerial work, and many others. However, helicopters suffer from severe limitations on their maximum speed, range, altitude, and achievable payload capacity. This, along with the fuel consumption and noise and vibration levels typical of these machines, negatively impacts their productivity.

The aerospace industry has always tried to combine the two worlds to create a new type of aircraft that could travel at high speed with large payloads over long distances, arriving at successful hybrid configurations.

Figure 1.5: XV-22 Osprey in different Flying Modes [2]

- *helicopter mode* allows to generate lift with vertically oriented rotors and then perform Vertical Take-Off and Landing (VTOL) and hovering like a helicopter
- *airplane mode*, on the other hand, allows to produce lift from the wing, using thrust to overcome drag. This is done by rotating the engine nacelles by means of actuators, which direct the rotors towards the bow of the aircraft.

1.2.1 Growing interest in Tilt-Rotors

In the category of V/STOL aircraft, which seek to minimize take-off distance while maintaining the typical characteristics of a fixed-wing aircraft, the most promising configuration that the industry is focusing on is tilt-rotor.

Its success is dictated by the ability to take off vertically as a traditional helicopter would, then rotate the propulsion system and perform an airplane mode conversion.

This allows to cover great distances and therefore to have a great operative range while preserving the peculiar characteristics of the helicopter.

Figure 1.6: Range Comparison Between Bell 525 and Boeing V-22 [3]

Despite efforts, the tilt-rotor configuration does not come without problems that limit its performance. Right from the start, the biggest issue for this type of aircraft was in-flight stability and manoeuvrability, as well as the ability to perform flight mode conversion. In addition, the mechanical complexity of the tilt-rotor design greatly increases operational and maintenance costs, reducing its employability.

1.2.2 Bell XV-3

Figure 1.7: Bell XV-3 in hovering near ground

In the 50s the concepts of tilt-rotor and tilt-wing were explored, leading to the birth of the U.S. Army and U.S. Air Force Convertiplane Program. The first prototype, developed by Bell Helicopter Company, was the XV-3 tilt-rotor which represented the first step in demonstrating the capabilities and potential of this type of

configuration. After several iterations, the final prototype featured a pair of rotors mounted on the wingtips that could rotate to perform the conversion from helicopter to airplane. The wingtips were of conventional design and allowed for stability both in hover and in straight flight. The controls in the cabin remain similar to those of a helicopter even if there are some convertiplane models with the throttle lever instead of the collective one. The controls are mixed independently during the conversion to switch transparently to the pilot from one flight mode to another.

1.2.3 Bell XV-15

In order to make the tilt-rotor configuration finally usable for military or commercial purposes, in the early 1970s, a joint program between the U.S. Army and NASA developed the XV-15 model produced by Bell Helicopter Textron. This represented a significant evolution with respect to the previous XV-3 thanks to a different shape in the nacelles, an innovative transmission with herringbone gears and above all a more powerful gas turbine propeller. In addition, a flight control system (SCAS) was integrated to solve the problems of manoeuvrability and stability.

Figure 1.8: Bell XV-15 in hovering near ground

To reduce the loss of performance in stationary flight caused by the downward force due to the rotor wake on the wing, the flaps can be lowered to three pre-sets deflection positions. At the same time, the ailerons can also be deflected downward when the flaps are set, although the displacement is limited to two-thirds of the flap position. Such surfaces are called flaperons and are activated at high speed.

One of the major issues lies in the mathematical modelling of the aerodynamics of the rotors and their interaction with the underlying wing.

The development of a reliable model would require a lot of onerous experimental tests. Therefore, the model was developed based on the accessible evidence of the

XV-15 and taking advantage of the generic model for tilt-rotors described by Ferguson S. W. in *"A Mathematical Model for Real Time Flight Simulation of a Generic Tilt-Rotor Aircraft"* [4].

1.3 Tilt-Rotors - Military and Civil developments

Nowadays, the characteristics of Tilt-Rotors have led to various solutions for commercial and military use. As always in the aviation industry, innovation starts in the military field, because a less proven configuration is associated with a higher risk in operations and often not socially tolerable except in the military field.

For this reason, the Bell V-22 Osprey and Bell V-280 aircraft are currently operational for the military field, deriving from the legacy of experimental tests conducted on the XV-15. Also worth mentioning is the AgustaWestland AW609 aircraft produced by the Italian company Leonardo S.p.A. for civil use.

1.3.1 V-22 Osprey

Figure 1.9: Bell V22 Osprey in airplane mode [3]

The Bell V-22 Osprey (Figure 1.9) is a tiltrotor aircraft, manufactured by Boeing and Bell Helicopter Textron, and used by the US Army and Navy primarily for troop transport. In addition, as a multi-role aircraft, it can conduct air assault, special operations during night or in critical conditions, carrier onboard delivery, evacuation and recovery operations and VIP transport. Regardless of its purpose, the Tilt-Rotor is widely used due to its 860 nm range and 266 kts speed, which are superior to those of a conventional helicopter.

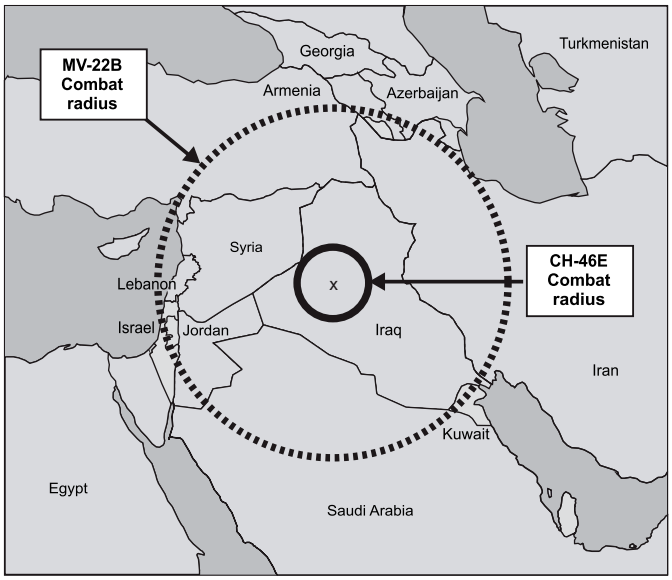

Figure 1.10: MV-22 and CH-46 Combat Radius Comparison [5]

Figure 1.7 compares the flight radius of the MV-22 to that of legacy CH-46. The MV-22 proved very advantageous for transporting people and cargo during missions in Iraq. Its advantages consisted of greater speed and range than legacy helicopters. In addition, due to its ability to fly at very high altitudes, it was possible to avoid small arms fire during missions. *"According to users and troop commanders of the MV-22, its speed and range 'cut the battlefield in half', expanding the coverage of the battlefield with less resource use and allowing it to do two to three times as much as legacy helicopters could in the same flight time."* [5]

1.3.2 Bell V-280 Valor

Bell V-280, shown in Fig. 1.8, is a Tilt-Rotor developed by Bell Helicopter Textron and Lockheed Martin.

Figure 1.11: Bell V-280 in airplane mode (wiki)

Its name derives from the fact that it is designed for a cruising speed of 280 knots. In addition, it can reach a maximum speed of 300 knots, a range of 2,100 nautical miles, and an effective combat radius of 500-800 nmi. The maximum expected take-off weight is approximately 30,000 pounds (14,000 kg). Unlike the previous V-22 Osprey tiltrotor, the engines remain in place while the rotors and drive shafts tilt. A drive shaft runs across the straight wing, allowing both propeller rotors to be driven by a single engine in the event of engine loss. The V-280 will have retractable landing gear, a three-redundant fly-by-wire control system and a V-tail configuration. The aircraft's production costs, and weight are reduced by using a carbon-fibre reinforced polymer composite material for the wing construction. The V-280 will have a crew of four and will be able to carry up to 14 troops.

1.3.3 AgustaWestland AW609

The AW609 is a Tilt-Rotor capable of making vertical landings, unlike conventional fixed-wing aircraft, allowing the type to serve locations such as heliports or very small airports, while possessing twice the speed and range of any available helicopter.

Figure 1.12: AgustaWestland AW609 in airplane mode [6]

The AW609 appears to be outwardly similar to the military-oriented V-22 Osprey; however, the two aircraft share few components. Unlike the V-22, the AW609 has a pressurised cabin with soundproofing to increase passenger comfort. Access to the cabin is via a two-piece, 89 cm wide clamshell door set into the centre of the fuselage under the wings.

When flying in airplane mode, as can be seen in Fig. 1.9, most of the lift is produced by the AW609's wings, which are slightly angled forward. Both the wings and the main fuselage are made largely of composite materials. The wings are 34 ft long and are equipped with flaperon control surfaces, which are normally controlled automatically. In vertical flight, the flaperons drop to a 66-degree angle downwards to reduce the area of the wing affected by downwash from the thrusters. A rudderless vertical stabiliser is mounted at the rear of the fuselage to stabilise flight in airplane

mode. The AW609 has been designed to develop full transport/class 1 performance to operate safely even in single engine flight conditions. It is equipped with an anti-icing system and must be certified to fly in known icing conditions. The avionics include a three-redundancy digital fly-by-wire flight control system, a head-up display system, and Full Authority Digital Engine Controls (FADEC).

1.4 Electric Tilt-Rotors

One of the challenges today is to reduce environmental pollution and to transfer goods, people, and information more quickly. Due to their operational flexibility, tilt-rotors are certainly a good option. Therefore, the need to develop fully electrically powered aircraft has arisen. An increasing number of aerospace companies are investing time and resources in developing new fully electric tilt-wing and tilt-rotor concepts. The best-known tilt-rotor, developed by Airbus, is the A3 Vahana (Fig. 1.10), an all-electric, self-piloted tilt-wing, the design of which began in 2016. In 2018, thanks to flight tests, its major merits, such as quietness, time savings, autonomy, and absence of emissions, were evaluated.

Figure 1.13: A3 Vahana in airplane mode

2
Mathematical Model

The study and validation of the mathematical model have not been conducted by the author and is not the main purpose of this thesis. Therefore, they are reproposed from the work previously performed.

A mathematical background is however necessary to better understand the dynamics that drives the motion of the aircraft and to continue the implementation of the model on a real-time platform with hardware controls. The mathematical model is thus based on the theory in [4], which describes how to generate a Generic Model FOR Real-Time Tilt-Rotor Simulation (GTRS), even though it was initially developed by NASA for the development of the XV-15.

2.1 Tilt-Rotor model

The mathematical model of the aircraft considers various elements that enable it to adequately represent the mechanics of flight.

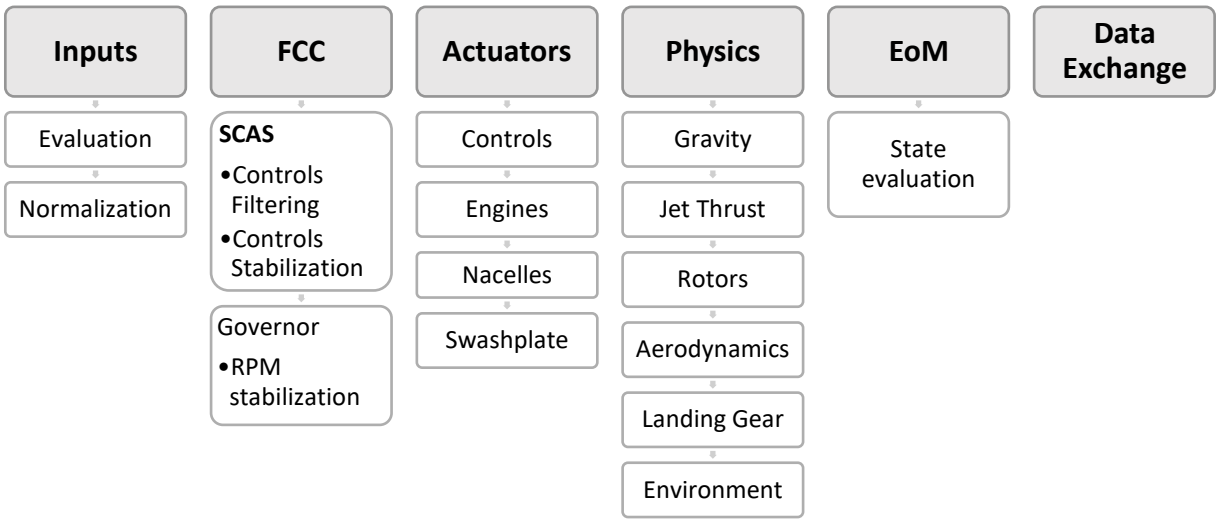

Figure 2.1: XV15 Tilt-Rotor Model Structure

The model starts with the evaluation of command inputs and controls stabilization by an integrated SCAS system, and then moves on to flight dynamics which depends on the response of the actuators that move the controls. The physics then takes into account Aerodynamics, Rotor and Engine Thrust, Landing Gear, but also Environmental Conditions. Then the contributions of forces and moments are summed together to integrate the equations of motion over time and evaluate the evolution of the aircraft's instantaneous state. The last step is to communicate the state and thus the current conditions with the simulation environment for on-screen display.

The simplified structure is shown in Figure 2.1, in which can be observed the main activities that the model carries out to get from the input to the animation of the aircraft.

2.1.1 Inputs

Flight commands are issued through USB peripherals, which then communicate axis positions and button presses with digital data (more information in Sect. 3.4).
These commands are then normalised to interface with the flight commands of the XV-15, as the model bases its calculations on the movement of the controls in inches (more information in Sect. 3.3).

2.1.2 Flight Control Computer – FCC

The flight control computer consists of two main systems, the SCAS and the Collective Governor.

- The *SCAS* system allows the roll, pitch, and yaw rates to be stopped once the flight controls are released and increases the stability of the system. The angular rates in the system are first filtered with a washout, which selects only those rates not imposed by the pilot, and then produces a command that stabilises the aircraft and adds to the pilot's commands.
- The *Collective Governor*, on the other hand, is a system that adjusts the collective pitch to maintain a constant number of revolutions depending on the deflection angle of the nacelles. Compensation gain is either proportional or integral.

2.1.3 Actuators

The hydraulic or electric actuators are very complex systems that can introduce in the physics of the problem of the strong non-linearities to be taken into account often. For a simulator of this type these systems are modelled with very simple

transfer functions, because an accurate description of them is beyond the scope of the project and clashes with the need for real-time execution.

- *Hydraulic actuators* for flight controls always represent a system that needs to be properly modelled in an aircraft, because they can induce non-negligible delays in the control loops of both the pilot and the stabilization systems. With this in mind, a first-order transfer function has been introduced to account for the delay.
- *Engines* also have a dynamic, although the number of revolutions is almost always constant, however the dynamics introduces a delay in the response of the collective or throttle command. It is also modelled as a first-order system that does not introduce a lot of unnecessary numerical complexity.
- The *Nacelles Actuators* have a certain actuation speed that is fundamental for the correct simulation of the tiltrotor. In a tiltrotor in fact the nacelles command becomes a primary command to move longitudinally and therefore a delay in its actuation is critical. It is modelled with a constant actuation speed.
- The *Swashplate* is that device which allows to translate the pilot's inputs into cyclic or collective blade variations by means of a mechanic present on the rotor. Their implementation is provided in the helicopter flight mode and is essential for maintaining position in hover. This implementation is also modelled with a first-order transfer function.

2.1.4 Physics

The physical model of the aircraft consists of evaluating and calculating all the force and momentum contributions and adding them together to find the resultant to be integrated into the equations of motion. As shown in Figure 2.2, the major contributions are:

- *Gravity:* must obviously be considered and is treated as a constant force that depends on the weight of the medium and rotated according to the Euler angles of the aircraft.
- *Engines:* produce force and momentum because they are turboprops, so part of the thrust is produced by the exhaust.
- *Rotors:* both rotors, according to the conditions of the swashplate, of aircraft movement and of the environment, produce a variable force and moment which is fundamental for maintaining position and attitude in helicopter mode. A complex mathematical model based on the blade element theory coupled with the Pitt-Peters theory for the computation of the inflow, allows to evaluate forces and moments.

- *Aerodynamics:* is a very complex part of the model that allows the evaluation of forces and moments due to 4 main effects: fuselage, vertical stabilizer, horizontal stabilizer and wing pylons interaction with rotors inflow and air stream.
- *Landing gear:* for the landing gear it is necessary to consider the effect of aerodynamic resistance that it produces in movement, but also the interaction with the ground. The landing gear also provides damping due to the relative compression with dissipation of the landing gear stem and the friction they produce with the ground. In addition, depending on the pressure of the brakes, the longitudinal resistant force on the ground varies.
- *Environment:* based on the aircraft position and user modifiable parameters, the environment model calculates parameters such as air density, sound speed and wind that affect all the forces and moments described above.

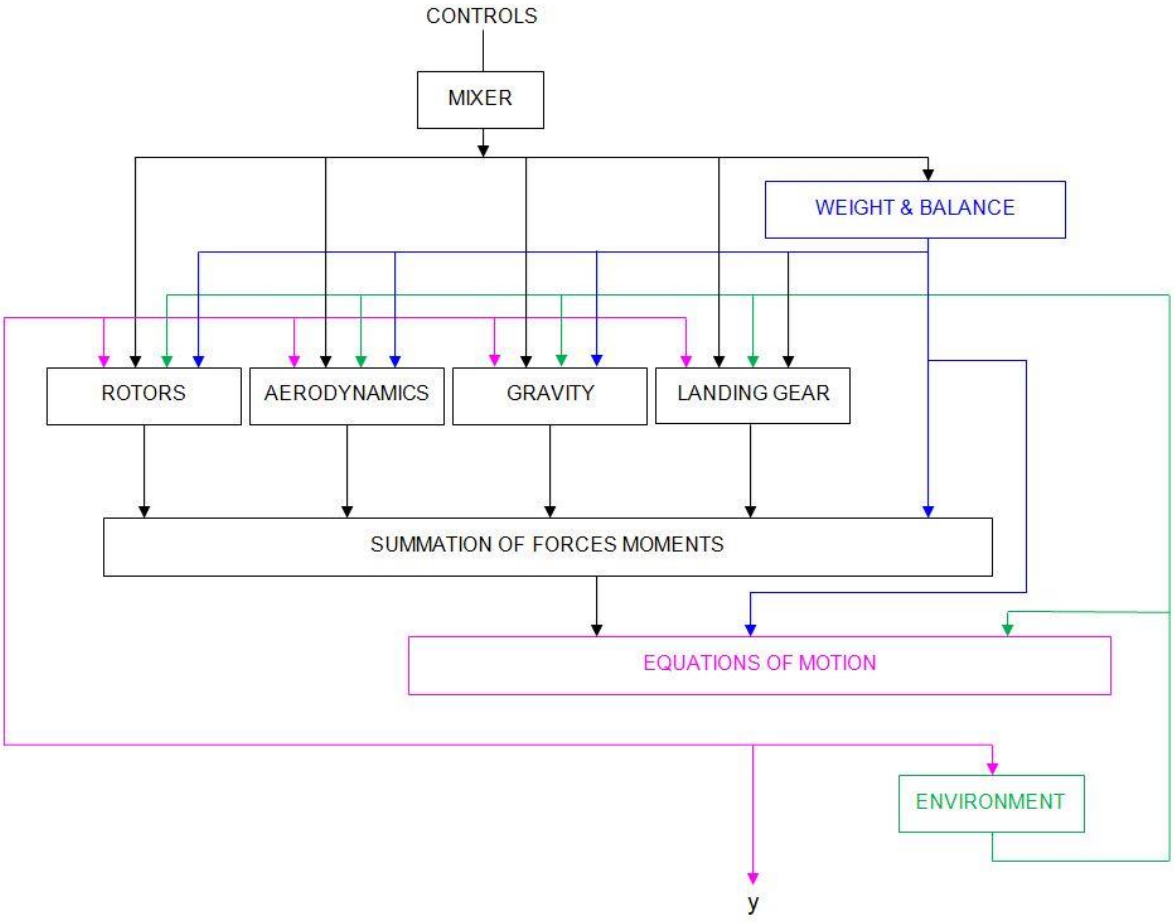

Figure 2.2: Mathematical Model complete Structure [7]

2.1.5 Equations of Motion – EoM

The equations of motion are equations derived from the balance of forces and momentums to calculate linear and angular accelerations. Based on the increments of accelerations, the model integrates the velocities and thus the displacements or rotations that the aircraft performs over time. The rotations are evaluated with respect to the body reference of the aircraft (see Sect. 2.2), but then they must be transferred and rotated for the calculation of the variations of the Euler angles useful for the simulation.

2.1.6 Data Exchange

The last part of the Tilt-Rotor Model is the exchange of the calculated data with the simulation environment (more info at Par. 3.1.4 and Par. 3.2.5). Simulink in fact is only a software of physical simulation, and the visual representation of the numerical results can be executed with the integration of add-on for Simulink or from external software. The communication happens usually with protocols that concur the transmission between more applications in simple and standardized way (more info at Sect. 2.4).

2.2 Reference systems

From the model, instantaneous forces and moments are calculated by individually summing the total contributions of each subsystem that produces them. It is critical to be able to report the calculated forces and moments in a single aircraft reference system. This is done by using different reference systems [8] that are more suitable depending on the subsystem and then transferring them to the aircraft reference. The main ones are:

- To allow the evaluation of translational, rotational speeds and Euler's angles a reference system for aircraft body must be defined as in Figure 2.3.

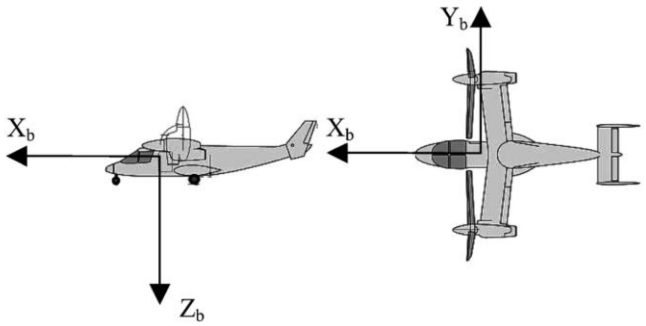

Figure 2.3: Aircraft Body Reference System

- The choice of rotor model is very important since the wake of the rotors impacts the wing producing an important aerodynamic effect. The rotor model used was developed at ZHAW in the work described in [7] and uses the reference system shown in Figure 2.4.

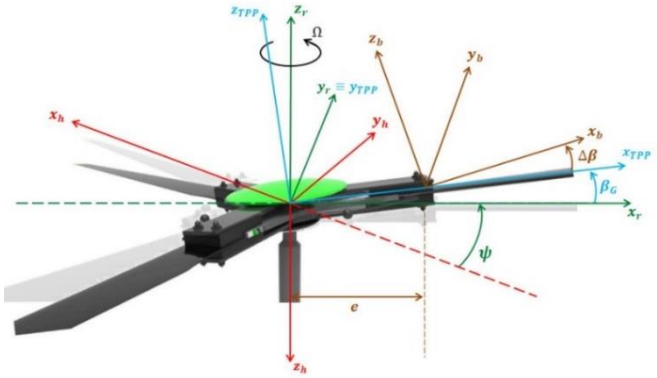

Figure 2.4: Rotor Axes System

- To allow the calculation of moments, a reference system for geometric distances must be defined, as in Figure 2.5.

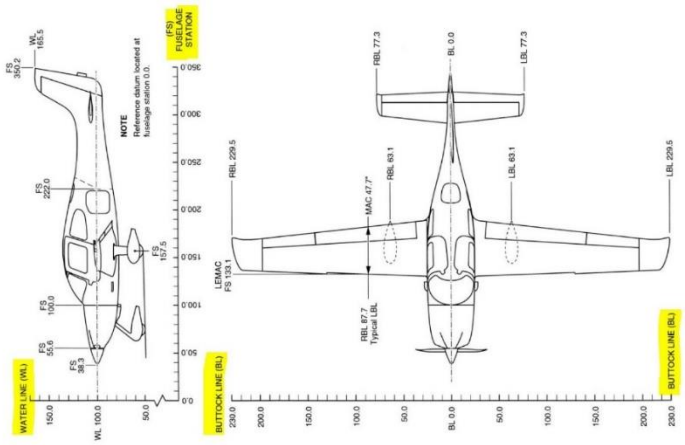

Figure 2.5: Geometric Reference System

- The reference system of the aerodynamic part to allow the use of the data present in [4].

2.2.1 Geodesy and Coordinate Systems

Geodesy is a branch of mathematics that deals with the shape and area of the Earth. It's necessary to use an accurate model of Earth's shape, rotation, and gravity in order to simulate high-speed flight over large areas of Earth's surface.

Meridional cross sections of Earth are approximately elliptical, and the polar radius of Earth is about 21 km less than the equatorial radius, so the solid figure generated by rotating an appropriately scaled ellipse about its minor axis will provide a model of Earth's shape.

The mostly widely used model in the world is the WGS (World Geodetic System), that is a standard for cartography, geodesy, and satellite navigation including GPS. This model defines an Earth reference system centred in the Earth's centre of mass (derived from satellite orbits), the x and y axes on the equator plane and the z axes pointing the north. The gravitation is defined with the Earth Gravitational Model (EGM), associated with the World Magnetic Model (WMM).

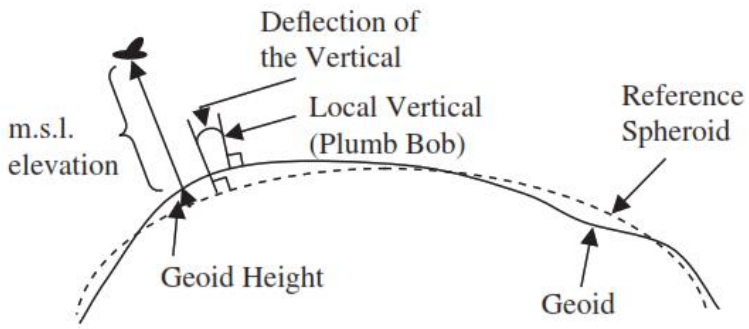

Figure 2.6: The Geoid definitions

The *geoid* (Figure 2.6) is defined as the equipotential surface of gravity field which coincides with the undisturbed mean sea level extending continuously below the land surface. The irregular shape of the Earth causes the mass distribution to be uneven, thus the Plumb-Bob define an angle, called deflection of local vertical, with normal to the spheroid.

The geoide is approximated in the WGS model with a spherical harmonics series of multiple degrees depending on different version of the model (the last WGS 1984 uses 2160 harmonics).

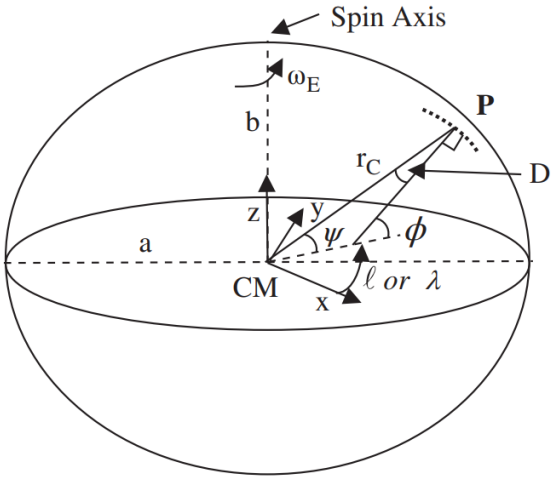

Figure 2.7: Oblate spheroidal model of the Earth

The Earth shape is an oblate *spheroid*, thus an ellipsoid obtained by rotating an ellipse about one its principal axes, in this case the z axis (Figure 2.7). The spheroid equation is so defined as:

$$\frac{x^2 + y^2}{a^2} + \frac{z^2}{b^2} = 1 \tag{2.1}$$

With a the semimajor axis and b the semiminor axes of the generating ellipse. Two other fundamental parameters to describe the Earth shape and its reference system are its *flattening*, f, which accounts the oblateness at the poles of the spheroid, and the *eccentricity*, e, which accounts the deviation of the generating ellipse from a circumference.

The WGS-84 (stands for 1984) defines these values to generate the Earth Spheroid based on the least-squares best fit. a and f are defined from the model, instead b and e are derived:

$$\begin{aligned} a &= 6\,378\,137.0\ m \\ f &= \frac{a-b}{a} = 1 - \frac{b}{a} = \frac{1}{298.257\,223\,563} \\ b &= 6\,356\,752\ m \\ e &= \frac{(a^2 - b^2)^{\frac{1}{2}}}{a} = 0.0818\,191\,908\,426 \end{aligned} \tag{2.2}$$

To complete the WGS-84 reference frame are necessary two additional parameters related to the Earth: the *Earth's gravitational constant, GM or* $\mu_\oplus$ (mass times the universal gravitational constant), and the *angular rotational speed,* ω_E:

$$\begin{aligned} GM &= 3\,986\,004.418 \cdot 10^8\ m^3/s^2 \\ \omega_E &= 7.292\,115 \cdot 10^{-5}\ rad/s \end{aligned} \tag{2.3}$$

where the angular speed is also defined as *sidereal rate of rotation*, so the speed relative to the fixed stars.

2.2.2 Latitude and Longitude

The reference frames used typically are the Earth and an inertial frame centred in the Earth's centre of mass, considered as a fixed point. Several polar and Cartesian coordinate systems could be defined in the frames, where the equatorial plane and the spin axis of the planet are used for reference.

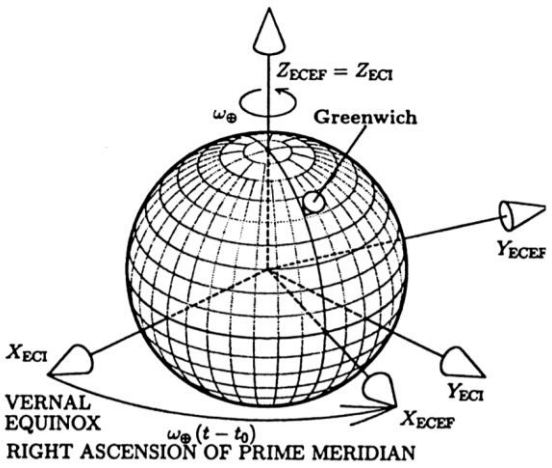

Figure 2.8: ECI and ECEF reference systems [9]

The two main reference systems based on these frames and shown in Figure 2.8 are:

- Earth-centred inertial (ECI), with the origin the Earth's centre of mass, the z axis as the spin axis and the x axis parallel to the line linking the Sun and the Earth.
- Earth-centred Earth-Fixed (ECEF), with the origin the Earth's centre of mass, the z axis as the spin axis and the x axis passing through the Greenwich Meridian.

To define the position of a point P on the spheroid, as shown in Figure 2.6, it's necessary to define the *Terrestrial longitude,* ℓ, and the *celestial longitude,* λ, which differ from the reference system chosen. The relation between the two is:

$$\lambda - \lambda_0 = l - l_0 + \omega_E t \tag{2.4}$$

with λ_0 and ℓ_0 the values at $t = 0$. Moreover the *Latitude angles,* Ψ, does not suffer from this difference, because the equatorial plane remains with a good approximation the same during time, and is positive in the Northern Hemisphere.

2.2.3 Geocentric and Geodetic coordinates

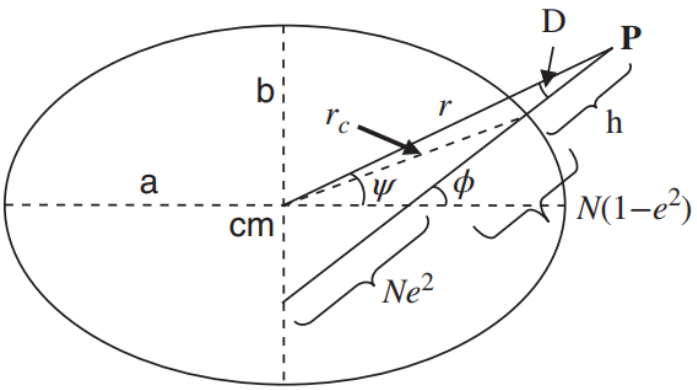

Figure 2.9: Geodetic coordinates of a Point

Looking at Figure 2.9, it's clear that at a latitude different from 0, the angle between the normal to the spheroid with the equatorial plane is different from *Latitude*. For these two coordinates could be defined for the same point P:

- Geocentric Coordinates, referenced to the ECI/ECEF system and defined by:
  - *Geocentric latitude* of P, Ψ
  - *Geocentric radius* of P, r
- Geodetic Coordinates, referenced to the normal to the spheroid form point P, is used for maps and navigation:
  - *Geodetic latitude*, the angle of the normal with the equatorial plane, ϕ
  - *Geodetic height*: the height above the spheroid, along the normal, h

The triangle composed by the line linking P and the Earth's centre of mass and the normal to the spheroid surface relates the geocentric latitude and geodetic latitude with the formula:

$$\phi = \Psi + D \tag{2.5}$$

With the small D angle that varies from 0 to 11.5 arc-min when the latitude is 45° and is called *deviation from the normal.*

2.2.4 North-East-Down reference system

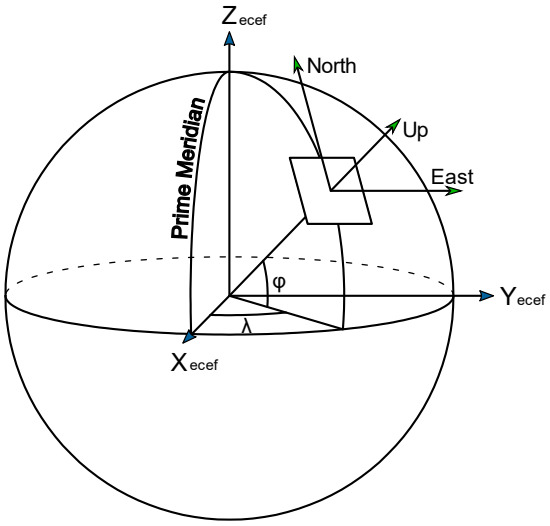

Figure 2.10: NED and ECEF reference system

The *local tangent plane system* (LTP) is a geographical coordinate system based on the tangent plane defined by the local normal direction and the Earth's spin axis. It is composed with three coordinates: the position on the North axis along the meridian; the position on the East axis, along the parallel; the vertical position, which is usually pointed towards down in the aerospace field, the same direction of the z body axis.

2.2.5 Earth's radii of curvature

The radius of curvature in differential geometry is, for a curve, equal to the radius of the circular arc which best approximates the curve at that point, the osculating circle. The concept can be extended to a surface in which the radius of curvature is the radius of the osculating circle that best fits a normal section or a combination of sections.

In the Earth's spheroidal model, two radii of curvature should be defined to estimate distances and speeds over the real Earth starting from a NED reference system.

- *Meridian radius of curvature, R_M or M*, is the radius of curvature in a meridian plane that relates the increments in geodetic **latitude** starting from North-South distances. It clearly depends on the shape of the spheroid, in terms of axes length a and b, eccentricity e, and flattening f, and on the geodetic latitude ϕ. Applying the differential geometry to the ellipse equation:

$$R_M = \frac{a(1-e^2)}{(1-e^2\,sin^2\,\phi)^{\frac{3}{2}}}, \qquad \frac{b^2}{a} \leq M \leq \frac{a^2}{b} \tag{2.6}$$

The arc length travelled by an airplane could be evaluated integrating the radius of curvature respect to angle, but the integral can not be found in a closed form in this case. It's much easier to use spherical triangles to approximate the distances.

The Meridian radius remains useful to evaluate the velocity component directed to North, v_n at a geodetic height h, starting from the geodetic latitude rate, $\dot{\phi}$:

$$v_n = (M + h)\,\dot{\phi} \tag{2.7}$$

- *Prime vertical radius of curvature, R_N or N,* is the radius of curvature in a perpendicular plane to the meridian plane and containing the *prime vertical,* that relates the increments in **longitude** starting from East-West distances along a parallel. It depends on the shape of the spheroid, in terms of axis length a, but not b, eccentricity e, and on the geodetic latitude ϕ, which determines the dimension of the parallel. Applying the differential geometry to the ellipse equation:

$$R_N = \frac{a}{(1 - e^2 \sin^2 \phi)^{\frac{1}{2}}}, \qquad a \leq N \leq \frac{a^2}{b} \tag{2.8}$$

As for the meridian radius of curvature, also the prime vertical radius of curvature is useful to evaluate the East component of the velocity, while the arc of parallel travelled should be integrated with osculating spherical triangle:

$$v_E = (N + h)\cos(\phi)\,\dot{\ell} \tag{2.9}$$

2.2.6 Flat Earth to LLA reference transformation

Having defined the reference systems, it is clear that the displacements of the aircraft are evaluated in a flat Earth reference system. The equations, therefore, work by understanding the aircraft as being in flat space, but a simulation environment needs longitude, latitude, and altitude data to place the aircraft model in the Earth environment.

The position is defined from the equation of motion in terms of the position vector $\boldsymbol{p} = [x, y, h]^T$. First, a rotation from the flat Earth reference system to the NED system must be followed with the following rotation matrix:

$$\begin{bmatrix} N \\ E \\ D \end{bmatrix} = \begin{bmatrix} \cos(\Psi) & -\sin(\Psi) & 0 \\ \sin(\Psi) & \cos(\Psi) & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ h \end{bmatrix}_{Flat} \tag{2.10}$$

where Ψ is the angle in degrees clockwise between the x-axis and north axes of the NED system.

Small changes in the North and East positions could be used to approximate small changes in longitude and latitude as follows:

$$\begin{aligned} d\phi &= \operatorname{atan}\left(\frac{1}{R_M}\right) dN \\ d\ell &= \operatorname{atan}\left(\frac{1}{R_N \cos(\phi)}\right) dE \end{aligned} \tag{2.11}$$

The output latitude and longitude are simply evaluated as the initial value summed to the small changes:

$$\begin{aligned} \phi &= \phi_0 + d\phi \\ \ell &= \ell_0 + d\ell \end{aligned} \tag{2.12}$$

The altitude is directly related to the Down distance where only the sign has to be changed:

$$h = -D = p_z \tag{2.13}$$

2.3 Solving Differential Equations

The Laplace transform is an integral transform that converts a function of real time $t \geq 0$, to a function of a complex variable $s = \sigma + j\omega$:

$$\mathcal{L}\{f\}(s) = F(s) = \int_0^{+\infty} f(t)e^{-st}dt \tag{2.14}$$

The transform exists if the f function is locally integrable on the interval $[0, +\infty)$.

Laplace transform finds many applications in engineering and science because it provides a convenient aid in the analysis and resolution of ordinary differential equations (ODE) with constant coefficients as Eq. (2.15).

$$a_1 x(t) + a_2 x'(t) + a_3 x''(t) + \cdots + a_n x^{(n)}(t) = h(t) \tag{2.15}$$

2.3.1 Laplace Transform Properties

The Laplace transform has important properties that allow it to solve differential equations:

1. *Linearity*, which leads to the superposition of effects:

$$\mathcal{L}[C \cdot f(t)] = C \int_0^\infty f(t)e^{-st}dt = C \cdot F(s)$$
$$\mathcal{L}[f_1(t) + f_2(t)] = \mathcal{L}[f_1(t)] + \mathcal{L}[f_2(t)] \tag{2.16}$$

2. *Transform of integral:*

$$\mathcal{L}\left\{\int_0^\infty f(t)dt\right\} = \frac{1}{s}\mathcal{L}\{f(t)\} \tag{2.17}$$

3. *Transform of derivatives*, which is fundamental to resolve ODEs:

$$\mathcal{L}\{f'(t)\} = sF(s) - f(t=0) \tag{2.18}$$

These two properties can be used to transform an ordinary differential equation as follows (with initial conditions equal to 0):

$$\mathcal{L}\left\{a\frac{d^2x}{dt^2} + b\frac{dx}{dt} + cx\right\} = \mathcal{L}\{h(t)\}$$
$$s^2\, a\, x(s) + b\, s\, x(s) + c\, x(s) = h(s) \tag{2.19}$$

At this point the differential equation can be resolved as an algebraic equation:

$$x(s) = \frac{h(s)}{as^2 + bs + c} \tag{2.20}$$

$x(s)/h(s)$ is also called the *transfer function* of the system examined.

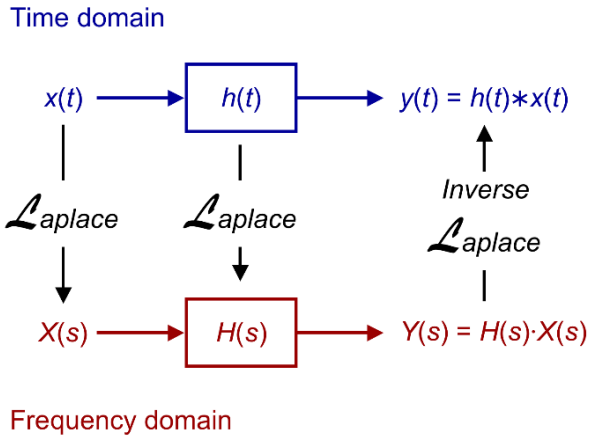

Figure 2.11: Relationship between the time domain and the frequency domain

To return to the time domain from the Laplace domain (Figure 2.11), the inverse Laplace transform operation can be applied:

$$f(t) = \mathcal{L}^{-1}\{F(s)\}(t) = \frac{1}{2\pi i}\lim_{T\to\infty}\int_{\gamma-iT}^{\gamma+iT} e^{st}F(s)ds \tag{2.21}$$

where the integration is often difficult to achieve. If the ODE has constant coefficients, the complex function $F(s)$ is a rational function $N(s)/D(s)$, where $N(s)$ and $D(s)$ are polynomials, and the function $F(s)$ can be written as the sum of rational functions, called partial functions, as visible in Eq. (2.22).

$$F(s) = \frac{N(s)}{D(s)} = \frac{\prod_{k=1}^{n}(s - z_i)}{\prod_{k=1}^{n}(s - p_i)} \tag{2.22}$$

Where z_i and p_i are the roots respectively of $N(s)$ and $D(s)$. If the $F(s)$ function is descriptable as above, the *Heaviside expansion formula* could be used to return to the time domain. Supposing that $N(s)$ has degree less than that of $D(s)$, $Q(s)$ has n distinct roots $p_k, k = 1,2,3, \dots, n$, then:

$$\mathcal{L}^{-1}\left\{\frac{N(s)}{D(s)}\right\} = f(t) = \sum_{k-1}^{n}\left[\frac{N(s)}{D(s)}s(p_i)\right]_{s=p_i} e^{p_i t}$$

4. *Initial value theorem*

$$\lim_{t\to 0} f(t) = \lim_{s\to\infty} sF(s) \tag{2.23}$$

5. *Final value theorem*

$$\lim_{t\to\infty} f(t) = \lim_{s\to 0} sF(s) \tag{2.24}$$

2.3.2 Laplace and steady-state

The Laplace transform also provides information on the steady-state condition of a system. The steady state of a system given by $f(s)$ is defined by $sf(s)$ when s tends to zero. If the system is described with an ODE or a set of ODE differential equations with constant coefficients, if all the roots of D(s), the poles, are negative, the system is stable and the final value is a constant, as in the example that follows:

$$f(t) = 1 - e^{-pt}, \qquad f(t) = 1 \; as \; t \to \infty \tag{2.25}$$

In other words, from the analysis of the final value of transfer function, an insight of the steady state system can be studied, without passing from the inverse transform. Each physic system could be modelled as a steady-state system, meaning that the coefficients of the ODE equations are constant, and it's called a Linear Time Invariant (LTI) system.

This approach is valid and can be applied by linearizing the model of the physical system around an equilibrium point. In any chosen point the derivates coefficients could be treat as constant and the Laplace theory remains valid.

The state vector, $\boldsymbol{x} \in \mathbb{R}^n$, is the instantaneous state of the system, while the command vector, $\boldsymbol{u} \in \mathbb{R}^m$, provides inputs to the system (in an airplane model could be the pilot commands). The output to be analysed is the vector $\boldsymbol{y} \in \mathbb{R}^l$.

To study the evolution of the system in the time domain, the derivates of the vector $\boldsymbol{x}$ can be find through a set of matrices as follows:

$$\begin{cases} \dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t) \\ \boldsymbol{y}(t) = C\boldsymbol{x}(t) + D\boldsymbol{u}(t) \\ \boldsymbol{x}(0) = 0 \end{cases}, \quad \begin{matrix} A \in R^{n\,x\,n} & B \in \mathbb{R}^{n\,x\,m} \\ C \in \mathbb{R}^{l\,x\,n} & D \in \mathbb{R}^{l\,x\,m} \end{matrix} \tag{2.26}$$

Where D is typical the null matrix for a mechanical system. Applying the Laplace transform to the Equations (2.26:

$$\begin{aligned} s \cdot \boldsymbol{x}(s) &= A\boldsymbol{x}(s) + B\boldsymbol{u}(s) \\ \boldsymbol{x}(s) \cdot [s - A] &= B\boldsymbol{u}(s) \\ \boldsymbol{x}(s) &= [s - A]^{-1} B\boldsymbol{u}(s) \end{aligned} \tag{2.27}$$

The output $\boldsymbol{y}$ is then:

$$\begin{aligned} \boldsymbol{y}(s) &= C[s - A]^{-1} B\boldsymbol{u}(s) + D\boldsymbol{u}(s) \\ \boldsymbol{y}(s) &= \{C[s - A]^{-1} B + D\}\boldsymbol{u}(s) \end{aligned} \tag{2.28}$$

The equations system is so resolvable with a bunch of matrix operations to find the state vector derivates that can be integrated for small step in time. So, starting from a state, x_k, the next state, x_{k+1}, can be evaluated from the derivates, $\dot{x}_k$, if they are multiplicated for a small step of time dt using different methods (Euler, Heun, Runge-Kutta, etc.). The simplest but still useful one is the explicit Euler method described in the following equation:

$$x_{k+1} = x_k + \dot{x}_k \cdot dt \tag{2.29}$$

2.3.3 Z-transform

When dealing with linear systems, Laplace theory can be applied because the domain of variation of a real system is considered continuous. For digital systems, such as that of a simulator that reproduces a physical system. A digital system, such as that of a computer that implements simulators, works in steps in a discrete time domain. The frequency with which the digital system executes instructions is called the sample rate. This type of implementation can lead in a real-time system to aliasing and latency phenomena that can lead to instability and numerical problems.

Aliasing is an effect that causes different signals to become indistinguishable when sampled. It could generate distortion, visual artifacts or, in general, numerical instabilities.

The Laplace transform is no longer adequate to describe digital systems, so to analyse systems governed by continuous laws but described by digital systems that apply sampling to the relations, the z-transform is used.

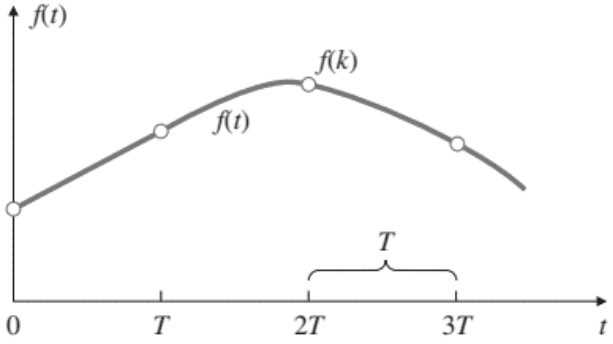

Figure 2.12: A continuous function f(t) and its sampling f(k) [10]

The z-transform is the mathematical tool for the analysis of linear discrete systems. It plays the same role for discrete time systems that the Laplace transform does for continuous time systems and is defined as follows:

$$Z\{f(k)\} = F(z) = \sum_{k=0}^{\infty} f(k)z^{-k} \tag{2.30}$$

where $f(k)$ the sampled function of f(t), as shown in Figure 2.12, and $k = 0,1,2, \ldots$ refers to the discrete sample times $t_0, t_1, t_2, \ldots$. The Laplace transform of derivates (Eq. (2.18)) property is similar with the z-transform:

$$Z\{f(k-1)\} = z^{-1}F(z) \tag{2.31}$$

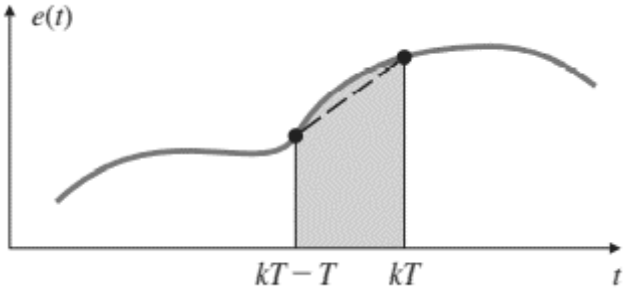

Figure 2.13: Tustin method – trapezoidal integration [10]

As described in [1] and [10], an alternative approach to digitization is the Tustin's method, that treats the problem as one of numerical integration. Supposing to have an input function f(s) and an output function y(s) in the Laplace domain that:

$$\frac{y(s)}{f(s)} = \frac{1}{s} \tag{2.32}$$

where $1/s$ is an integration as shown in Eq. (2.17). Hence:

$$y(kT) = \int_0^{kT-T} f(t)dt + \int_{kT-T}^{kT} f(t)dt \tag{2.33}$$

where T is the sample time period of the digital system. The integral could be rewritten as:

$$y(kT) = y(kT - T) + area\ under\ f(t)\ over\ last\ period\ T \tag{2.34}$$

At each step the trapezoidal integration can be used, as shown in Figure 2.13, to approximate the f(t) with a straight line between two samples divided by a sample period T. Rewriting $y(kT)$ as $y(k)$ and $y(kT - T)$ as $y(k - 1)$, the Equation (2.34) converts to:

$$y(k) = y(k - 1) + \frac{T}{2}[f(k - 1) + f(k)] \tag{2.35}$$

applying the z-transform:

$$\frac{y(z)}{f(z)} = \frac{T}{2}\left(\frac{1 + z^{-1}}{1 - z^{-1}}\right) = \frac{1}{\frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right)} \tag{2.36}$$

it means that the Laplace variable s can be transferred to the *zeta* domain as:

$$s = \frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right) \tag{2.37}$$

With this method all the transfer function in the Laplace domain can be easily transformed to transfer functions in *zeta* domain.
This formulation also requires initial conditions for y_k at the time $t = 0$, but the methodo does not requires a numerical integration method, because it's already based on the past values of the variable y, but a special care has to be spent on the choice of the sampling interval T, to avoid aliasing phenomena.

2.4 Communication Protocols

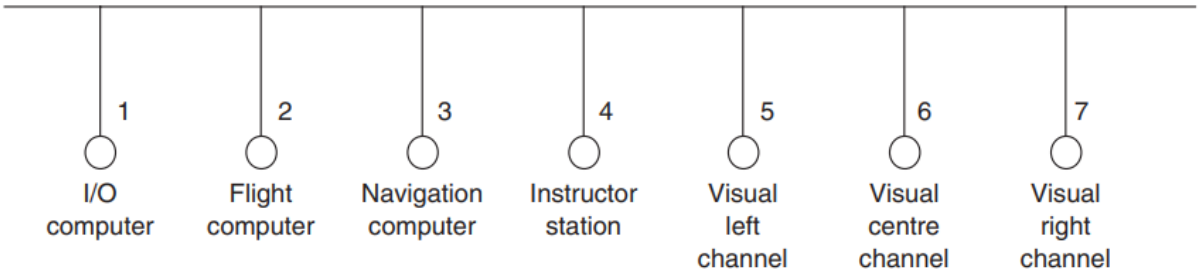

Figure 2.14: Simulator architectures nodes

As shown in Figure 2.14 a typical flight simulator architecture contains one or more logical units performing different operations that contribute to the simulation.

Information must thus be exchanged from one to several points in the architecture which form the so-called *nodes* of the communication network.

To guarantee deterministic transfers in a network in real time, all nodes must respect a protocol, that is, the set of rules to which all nodes in the network adhere. If the protocol is followed, it transmits only one node at a time, avoiding collisions and ensuring that the delays incurred in the transfer depend only on the characteristics of the bus.

2.4.1 UDP – Universal Datagram Protocol

UDP provides a simple method of node addressing based on the Internet Protocol (IP) address. Each node in the network is assigned a unique IP address of the form a.b.c.d where a, b, c and d denote a value in the range 0-255.

For a dedicated real-time system, the use of IP addresses is important. Nodes on the network are likely to be addressed in some logical way in the simulator algorithms. It is necessary to map between the IP addresses assigned to the network nodes and the logical addresses used in the simulation. However, if the designer of the simulator can allocate the IP addresses, they can be assigned numerical values corresponding to the logical addresses. Most operating systems allow static allocation of node IP addresses. A node can inspect the IP address of an incoming packet and effectively route the packet to the appropriate software that requests the specific packet. Of course, this convention assumes a dedicated network, specific to a real-time application.

UDP is a connectionless Transport protocol, meaning that no connection has to be established between the sending and the receiving segment. The data packets in the UDP protocol are processed and sent to the network as soon as any application request it This approach does not include an error checking and correction and avoids any communication congestion. On the other hand, e.g. Transmission Control Protocol (TCP) protocol needs a direct link between the source and destination hosts, and the packet are resent until the source host has been acknowledged by the destination of the receipt of the segment, without taking into account the time needed for the operation.

For there reasons, the UDP connection is preferable in time-sensitive applications, where the data packets dropping is better than the delay generated by the protocol transmission.

So in real-time application UDP transfers are often used for several advantages:

- *Speed*: The bandwidth overhead (excess time spent in computation, memory, networking etc.) is reduced compared to a connection-oriented protocol such as TCP

- *Implementation:* The interface between application software and operating systems are often standardized and the implementation is straightforward

2.4.2 UDP Packet structure

The UDP packet structure is composed by 2 main parts: the header and the data section. The header consists of 4 fields with a size of 2 bytes each (16 bits). The checksum and the source fields are optional.

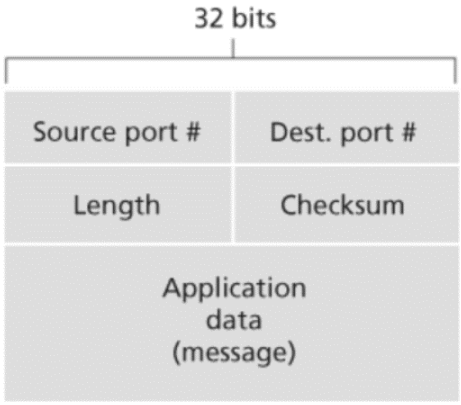

Figure 2.15: UDP Packet structure

- *Source port number* identifies the sender's port number in an IP format (a.b.c.d)
- *Destination port number* identifies the receiver's port number and is always required for the transfer.
- *Length* specifies the size in bytes of the UDP header and UDP data, whereas the minimum size is 8 bytes or 64 bits (just the size of the header if no data are sent), and the maximum size is 8 bytes + 65.527 bytes, almost 64 kB.
- *Checksum* is a field that may be used for error-checking of the header and data

3

Simulation Environment and Architecture

The main purpose of the work is presented in the remaining chapters, in which an attempt is made to link the previously described mathematical model, which represents the flight dynamics, including the dynamics of actuators and engines, the stabilization logic of the SCAS system, with the actual simulation environment. This allows, first of all, to visualize the dynamics in a simple and intuitive way, but also to verify the response, thanks to the commands given by the pilot with the connected hardware.

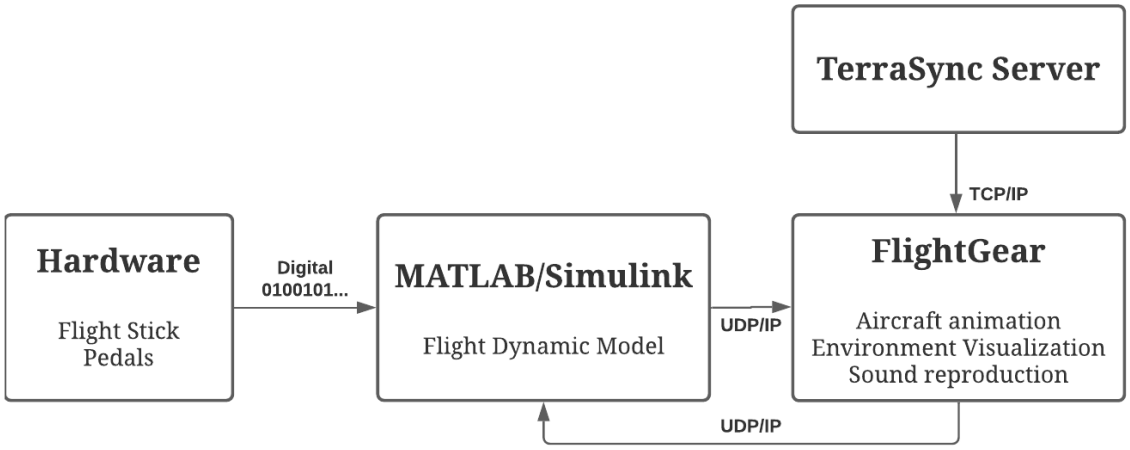

Figure 3.1: Software architecture

As it is predictable the software part remains the most delicate for a simulator with a didactic and research purpose. All the model of simulation is based on a code made on blocks and produced with the language Simulink® and MATLAB®, that simplify the writing and reading, guaranteeing at the same time the optimal performances in execution.
Simulink is only a numerical calculator on which, therefore, the data of attitude and position of the aircraft are evaluated, also in answer to the supplied commands. In

order to realize the simulator in a complete form, it is necessary to have additional software that is able to visualize on screen this information in a virtual environment that reproduces the aircraft object and the surrounding terrain, according to the instantaneous position in the globe.

3.1 FlightGear

The software identified as the flight environment is FlightGear, a full-featured, open-source and cross-platform flight simulator, designed also for research purposes like this one [11].

The simulator allows to reproduce the terrestrial environment including terrain, airports, cities, vegetation, all over the globe and to reproduce the aircraft in 3D with its attitude and position provided by Simulink, but also to reproduce sounds and movements of moving parts and engines.

3.1.1 Aircraft Reproduction

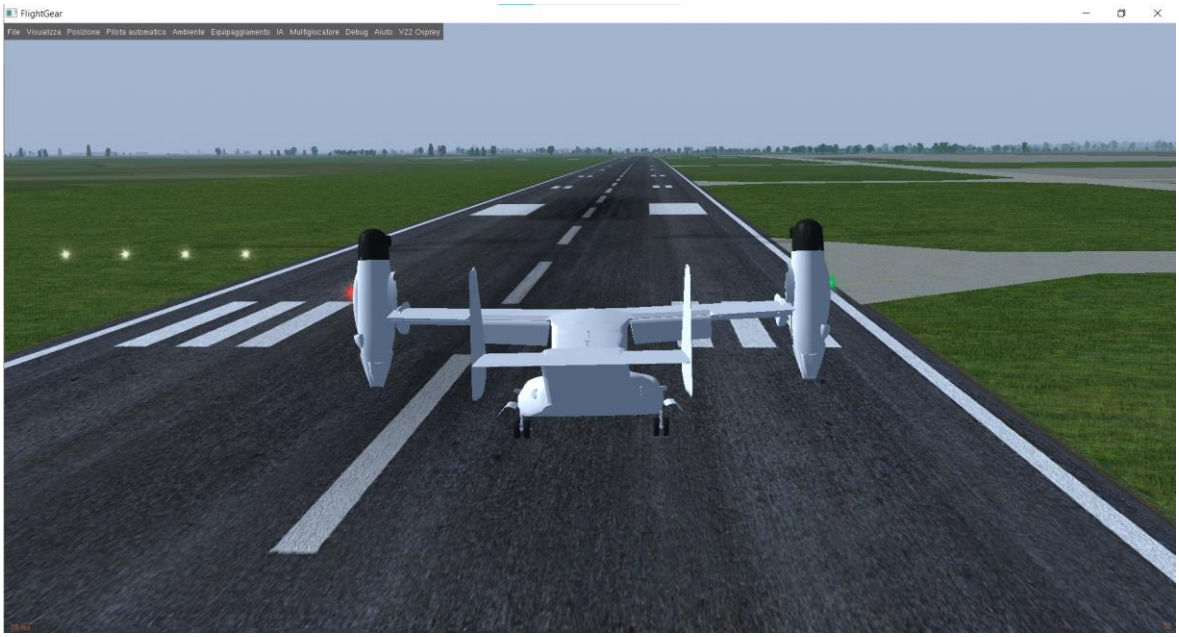

Figure 3.2: FlightGear running screenshot

To display a realistic aircraft with the appropriate animations and sounds, has been chosen to start with an aircraft already available in the official FlightGear database. The aircraft in question is the Boeing V-22 Osprey, which was provided under the GNU GPL licence by Mr. Baranger E.

The necessary modifications were then applied to disable the flight mechanics that used the open-source physical simulator JSBSim, in order to use the physical model based on Simulink.

Various changes were also made to the animations, sound and to make it compatible with the new communication protocol between Simulink and FlightGear. The result is an aircraft very similar to the XV-15 with the same flight controls and dynamics.

Figure 3.3: Aircraft Instrumentation

In order to be aware of the condition of the aircraft and to check the flight status, a Primary Flight Display (Figure 3.3) has been added, with the following instruments:

- Artificial Horizon
- Compass
- Altimeter
- Vertical Speed Indicator
- CAS Speed Indicator
- RPM gauge
- Nacelles Position Gauge
- Flaps Position Indicator
- Gear Position Indicator
-

Figure 3.4: Nacelles angle gauge texture

Each instrument on board must be designed with a CAD software that can then export the 3D model in ".ac" format. The most suitable free software is Blender, an open-source program that allows editing of 3D files. Subsequently it must be set up a texture (Figure 3.4) in ".rgb" format to the model and to connect a property (a variable) of the simulator with the animation of rotation or translation of the 3D object, like visible in the code in Figure 3.5.

```
<PropertyList>

    <path>NAC.ac</path>

    <animation>
        <type>material</type>
        <object-name>NACneedle</object-name>
        <object-name>NACface</object-name>
        <emission>
            <red>1</red>
            <green>1</green>
            <blue>1</blue>
        </emission>
    </animation>

    <animation>
        <type>rotate</type>
        <object-name>NACneedle</object-name>
        <property>/surface-positions/tilt</property>
        <interpolation>
            <entry><ind>-5</ind><dep>-25</dep></entry>
            <entry><ind>0</ind><dep>0</dep></entry>
            <entry><ind>5</ind><dep>24.25</dep></entry>
            <entry><ind>20</ind><dep>48.5</dep></entry>
            <entry><ind>50</ind><dep>72.75</dep></entry>
            <entry><ind>75</ind><dep>97</dep></entry>
            <entry><ind>90</ind><dep>121.25</dep></entry>
        </interpolation>
        <axis>
            <x>-1.0</x>
        </axis>
    </animation>

</PropertyList>
```

Figure 3.5: Nacelles gauge XML code

3.1.2 TerraSync & TerraMaster

TerraSync is a utility within FlightGear that allows automatic downloading of flight scenarios during simulation based on the instantaneous position of the aircraft. It uses an internet connection to update or obtain the environment from the FlightGear Scenery Database or portions of the globe can be installed in advance thanks to TerraMaster. The Figure 3.6 shows a graphical interface written in Java that allows to select and download any geographic area with some simple clicks.

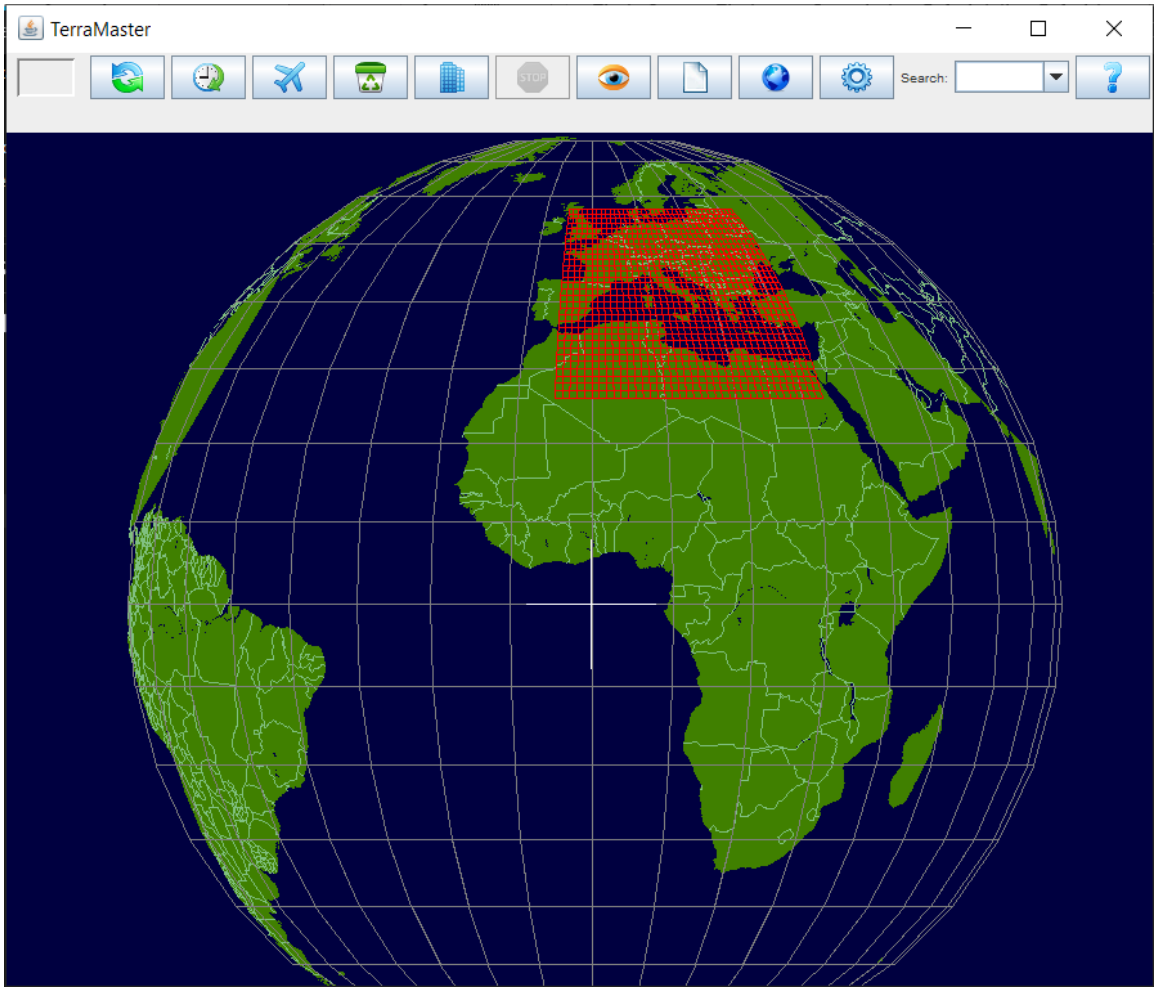

Figure 3.6: TerraMaster interface

3.1.3 How to launch FlightGear

To launch FlightGear it is necessary to generate a Windows script that adds the appropriate arguments to the executable. The file to generate has *.bat* extension (Figure 3.7) and has the main purpose of setting the correct data transmission protocol, choose the aircraft model and disable the flight dynamics inside the simulator.

Arguments are passed via a double dash followed by variable to set, and the most important ones in order are:

- *aircraft:* allows to set the displayed aircraft.
- *fdm:* allows to modify the flight dynamics.
- *generic:* allows to set a generic transmission protocol complete with update frequency, IP address, port, and protocol type. The protocol described below in Par. 3.1.4 is set here.

The other topics allow to intuitively select the runway, activate TerraSync and much more.

```
C:
cd C:\Program Files\FlightGear

SET FG_ROOT=C:\Program Files\FlightGear\data
.\\bin\fgfs --aircraft=vmx22 ^
--fdm = null ^
--generic = socket,in,100,127.0.0.1,1509,udp,SimulinkProtocolBinUDP ^
--generic = socket,out,25,127.0.0.2,1510,udp,SimulinkProtocolBinUDP ^
--fog-fastest ^
--disable-clouds ^
--disable-ai-traffic ^
--start-date-lat = 2004:06:01:09:00:00 ^
--airport = LIMF ^
--runway = 18 ^
--in-air ^
--heading=180 ^
--prop:position/altitude-ft=980 ^
--prop:sim/current-view/view-number-raw=2 ^
--offset-distance = 0 ^
--offset-azimuth = 0 ^
--enable-terrasync
--enable-fullscreen


--disable-sound
--enable-clock-freeze
```

Figure 3.7: FlightGear Launcher Script

3.1.4 Simulink Interface

The interface with Simulink permits to produce a TCP/UDP protocol for the exchange of information regarding the position, attitude of the aircraft and all those necessary to the simulation environment. This can be done by creating an *.xml* file in the FlightGear\data\Protocol folder, indicating the position and type of data within the exchanged packet.

As shown in Figure 3.8, the file is divided into input and output data, where first is defined if the data is sent in binary or text (ASCII) form, and the endianness of the data. Then chunks are inserted for each variable within the packet with its type and the variable within FlightGear that it will command.

The <factor> word allows the input variable to be multiplied by a fixed factor useful for converting the units of measure if necessary.

The input portion includes information about geographical position (longitude, latitude, and altitude), attitude, surface positions, engines RPM and landing gear. The

output portion, instead, returns to Simulink only the altitude information of the underlying terrain with respect to sea level, as shown in Figure A.2.

```
<?xml version="1.0"?>
<PropertyList>
<generic>

<input>

<binary_mode>true</binary_mode>
<!-- byte_order>host</byte_order --> <!-- host is default -->

<!-- Position -->
<chunk>
<name>Longitude</name>
<node>/position/longitude-deg</node>
<type>double</type>
</chunk>

<chunk>
<name>Latitude</name>
<node>/position/latitude-deg</node>
<type>double</type>
</chunk>

<chunk>
<name>Altitude</name>
<node>/position/altitude-ft</node>
<factor>3.280839895</factor> <!-- converts from meters to fts -->
<type>double</type>
</chunk>
```

Figure 3.8: FlightGear UDP Protocol (1)

3.2 Simulink

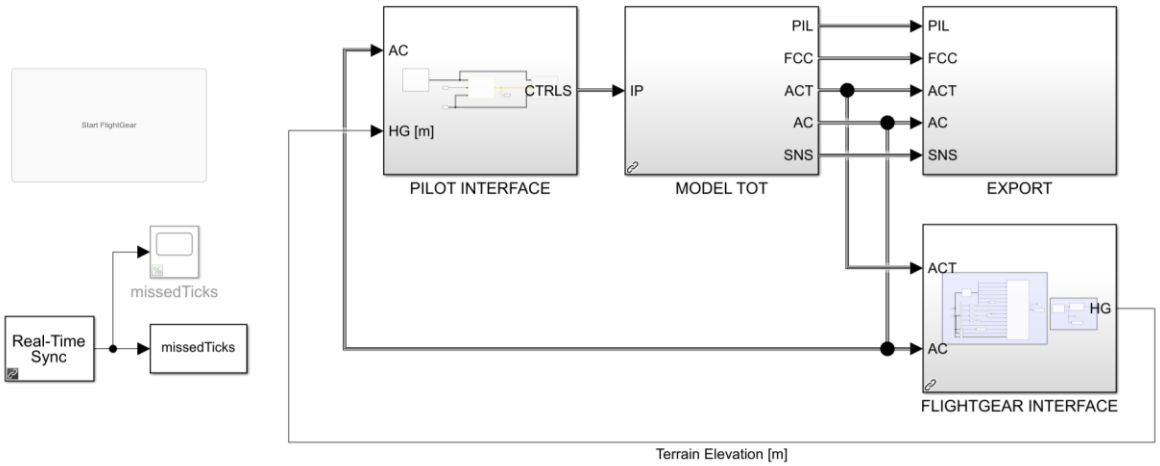

Figure 3.9: Overall Simulink Model

Simulink is the software that allows to write and to modify the code and to compile it, in order to execute the simulation model. It is introduced like an extension of MATLAB that, in a graphical way, concurs to connect the several blocks to execute the wished calculations. The mathematical model so produced is executed in a fast and intuitive way, without problems of allocation of memory or syntax. Several libraries are available for download that simplify the integration of different functions with the flight simulator.

The mathematical model consists of a simple block that can be treated as a system that receives inputs and produces outputs. In this work, the key is to interpret its operation and add the correct systems and libraries to connect the hardware and the flight environment to Simulink.

Next to the mathematical model there are the *Pilot Interface block*, which connect the hardware and therefore the commands given by the pilot to the mathematical model, and the *FlightGear Interface block*, which takes the output data from the model and sends it to FlightGear for display.

An export block also saves all the data output from the model in the workspace to graph the results and perform diagnostic actions.

In addition, through the functions present in Simulink, it is necessary to ensure that the simulation is performed in real time, i.e., code is needed to synchronize the simulation time with the processor clock time.

3.2.1 Simulink Acceleration

Acceleration is an operating mode of Simulink that allows to speed up the execution of the model by changing the way the code is compiled and replacing the interpreted code in the simulation.

There are 3 modes available:

- *Normal mode* is the basic mode in which Simulink controls the solver and model methods used during the simulation, and all run in a process (Figure 3.10)

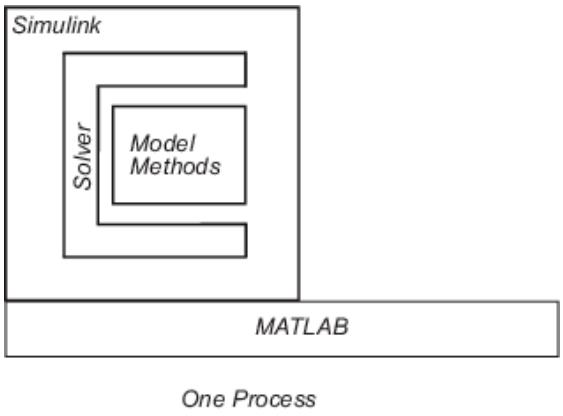

Figure 3.10: Normal Mode processes scheme

- *Acceleration mode* uses Just-in-Time (JIT) acceleration to generate an execution engine in memory instead of generating C code or MEX files. Methods in this mode are separate from Simulink and are part of the acceleration target code. Simulink generates an in-memory execution engine only for the top-level model and not for referenced models. Consequently, a C compiler is not required during simulation (Figure 3.11).

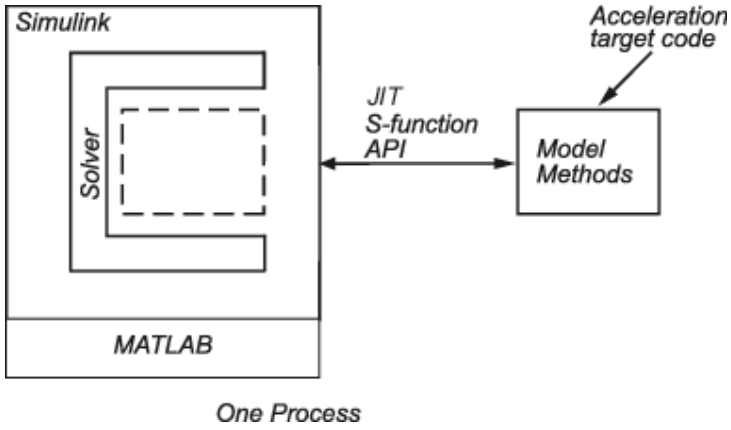

Figure 3.11: Accelerator Mode processes scheme

- *"Rapid accelerator mode* creates a Rapid Accelerator standalone executable from your model. This executable includes the solver and model methods, but it resides outside of MATLAB and Simulink. It uses external mode to communicate with Simulink". This mode is designed with the aim of outsourcing calculation processes to external machines specialised in real-time calculation and certified by MathWorks. (Figure 3.12)

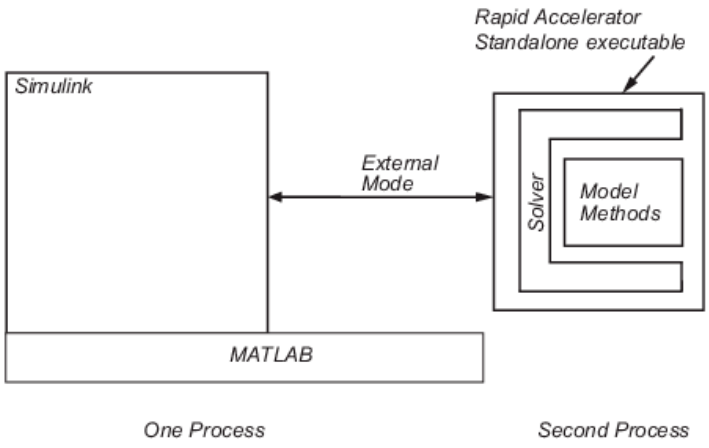

Figure 3.12: Rapid Accelerator Mode processes scheme

3.2.2 Real Time Simulation

The first step to make the simulation executable in real time is to synchronize the physical simulation time with the real time measured by the processor clock. This can be done in different ways through various libraries available within Simulink.

TESTS	TOLERANCE	FLIGHT CONDITIONS	FSTD LEVEL										COMMENTS
			FFS				FTD		FNPT			BITD	
			A	B	C	D	Init	Rec	I	II	MCC		
4. VISUAL SYSTEM													
a. SYSTEM RESPONSE TIME													
(1) Transport delay.	- 150 ms or less after controller movement. - 300 ms or less after controller movement.	Pitch, roll and yaw	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	One separate test is required in each axis. See Appendix 5 to AMC1 FSTD(A).300. For FNPT I and BITD, only the instrument response time applies.
-- or --													
(2) Latency	- 150 ms or less after controller movement. - 300 ms or less after controller movement.	Take-off, cruise, and approach or landing	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	One test is required in each axis (pitch, roll, yaw) for each of the three conditions compared with aeroplane data for a similar input. The visual scene or test pattern used during the response testing should be representative of the required system capacities to meet the daylight, twilight (dusk/dawn) and night visual capability as applicable. FFS only: Response tests should be confirmed in daylight, twilight and night settings as applicable. For FNPT I and BITD, only the instrument response time applies.

Figure 3.13: CS-FSTD Specifications for Flight Simulator Training Devices [12]

The assurance that the simulation can be defined as "Real Time" and therefore is performed at a realistic speed and with low latency is necessary for the correct interpretation of the flight dynamics by the pilot and for the accuracy of the simulation. The maximum latency with respect to current time is, in addition, defined by the CS-FSTD(A) standard which states: *"Latency: the visual system, flight deck instruments and initial motion system response shall respond to abrupt pitch, roll and yaw inputs from the pilot's position within 150 ms of the time, but not before the time, when the airplane would respond under the same conditions."*

Figure 3.14: Simulation Pace Block from Aerospace Toolbox [13]

The simplest approach may be to use the Simulation Pace block found within the "Aerospace Blockset" library, shown in Figure 3.14.

The Simulation Pace block lets to run model simulation at a slower pace so that connected animations views can be comfortably seen and understood to observe the system behaviour. Visualizing simulations at a slower rate makes it easier to understand underlying system design, identify design issues and demonstrate near real-time behaviour. The results can be viewed, and the system inspected while the simulation is in progress.

However, the blocking is not able to ensure a real synchronization with the processor clock, but simply slows down the simulation. The most correct method is to use the library "Simulink® Desktop Real-Time™" (SLDRT).

3.2.3 SLDRT - Simulink® Desktop Real-Time™

Simulink® Desktop Real-Time™ provides a real-time kernel for running on the computer and a set of blocks that can connect I/O devices to generate simulations with hardware inputs.

The first step is to install the Real-Time kernel that interfaces the library with the operating system. The kernel assigns the highest priority to the execution of the Simulink model, allowing it to run at the desired sample rate without interference. The kernel can be simply installed in the MATLAB Command Window using the command *"sldrtkernel -install"*. Usage is transparent to the user and the kernel will run automatically if library blocks are present.

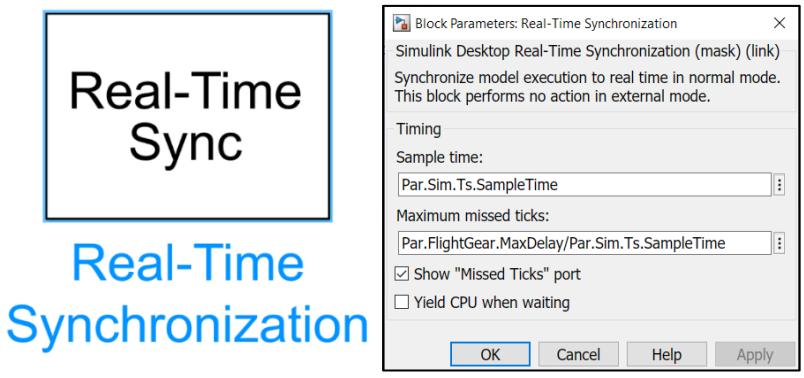

Figure 3.15: Real-Time Sync Block and its Parameters

Real-Time Sync block

The real-time sync block (Figure 3.15) allows to synchronize the real-time kernel clock with every time the block requires it. The time step to adjust the sync check is the sample rate, which is set to the minimum value of the model, i.e. every 0.0025 s, an update rate of 400 Hz.

An important data point to consider for library blocks are Missed Ticks. Missed Ticks are defined as the instants of time that the model lags with respect to the clock time calculated by the real-time kernel. The information is useful to monitor the latency of the model and report it on graphs. The actual latency time in seconds is calculated by multiplying the Missed Ticks in the block output with the block sample rate.

Input blocks

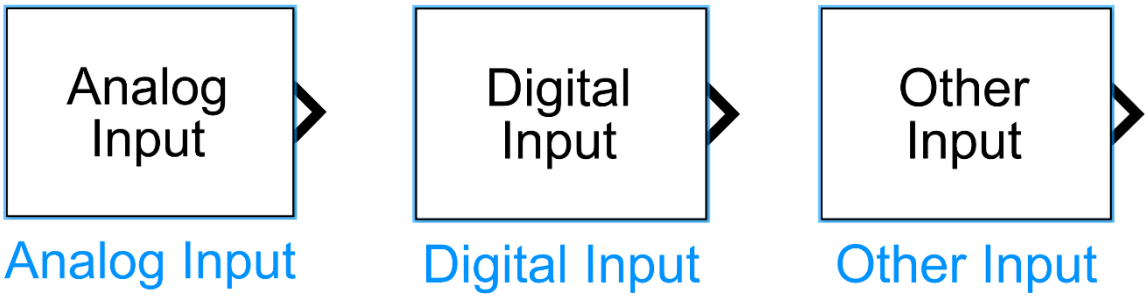

Figure 3.16: Analog/Digital/Other Input blocks form SLDRT library

To guarantee the priority in the reading of the inputs coming from the input hardware, the SLDRT library foresees various blocks of input and output (Figure 3.16) that let to synchronize the reading with a sample rate of choice. This allows to improve the speed of answer to the commands of the simulator, guaranteeing a true real-time simulation. The blocks used are 3:

- *Analog Input:* to read the input values for the proportional axes of the joystick and therefore for the roll, yaw, and pitch commands.
- *Digital Input:* to read the input values of the digital commands, i.e. the buttons that allow to change the function of brake, flaps, landing gear and SCAS.
- *Other Input:* to read the input values of the secondary controls, in this case the POV hat of the joystick, to move the nacelles and adjust the aircraft trim.

For all the blocks in question it was necessary to set the update sample rate, equal to that of the model execution. For the Analog Input block, it is also possible to select the signal output mode from the 4 available:

- *Volts:* returns the analogue voltage value

- *Normalized bipolar:* returns a value between -1 and +1 depending on the voltage
- *Normalized unipolar:* returns a value between 0 and +1 depending on the voltage
- *Raw:* returns a value between 0 and 2n-1, depending on the resolution in bits of the joystick

The most useful way and the one that best interfaces with the mathematical model is the *normalized bipolar*, which allows the joystick to be used in the simulator without any modifications, also with a view to modularity and subsequent changes to the simulator regarding the hardware (Figure 3.17).

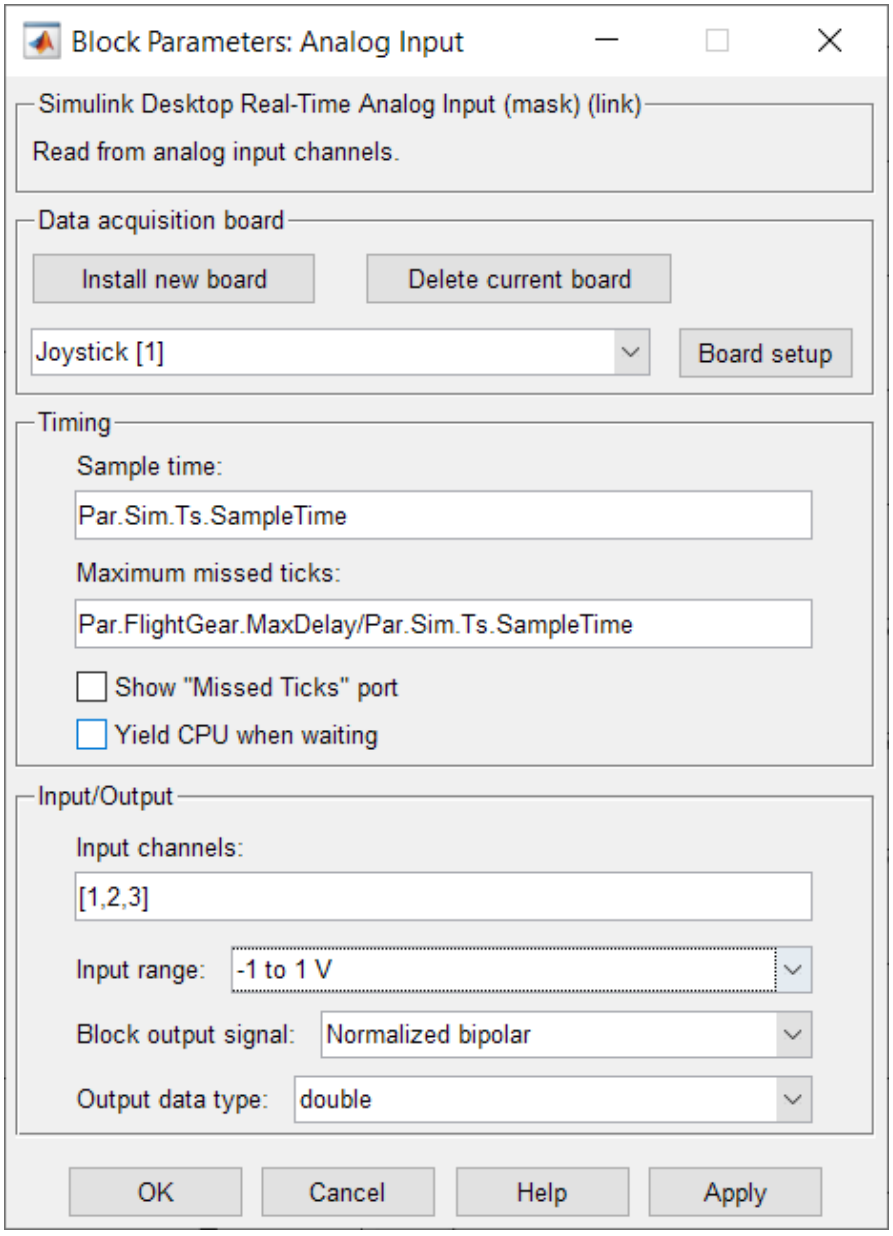

Figure 3.17: Analog Input Block parameters

Packet Input and Packet Output blocks

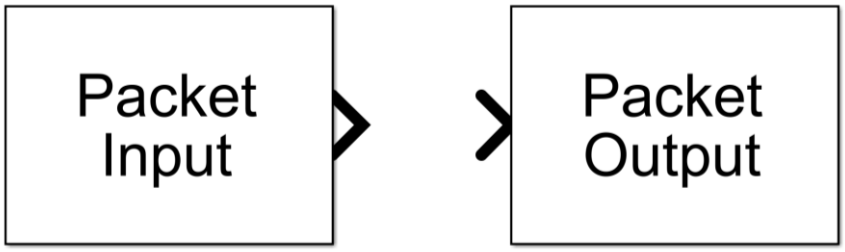

Figure 3.18: Packet data interface blocks from SLDRT library

Input and output packet blocks of the SLDRT library (Figure 3.18) allow to set up a packet and data transfer between two nodes in a network.

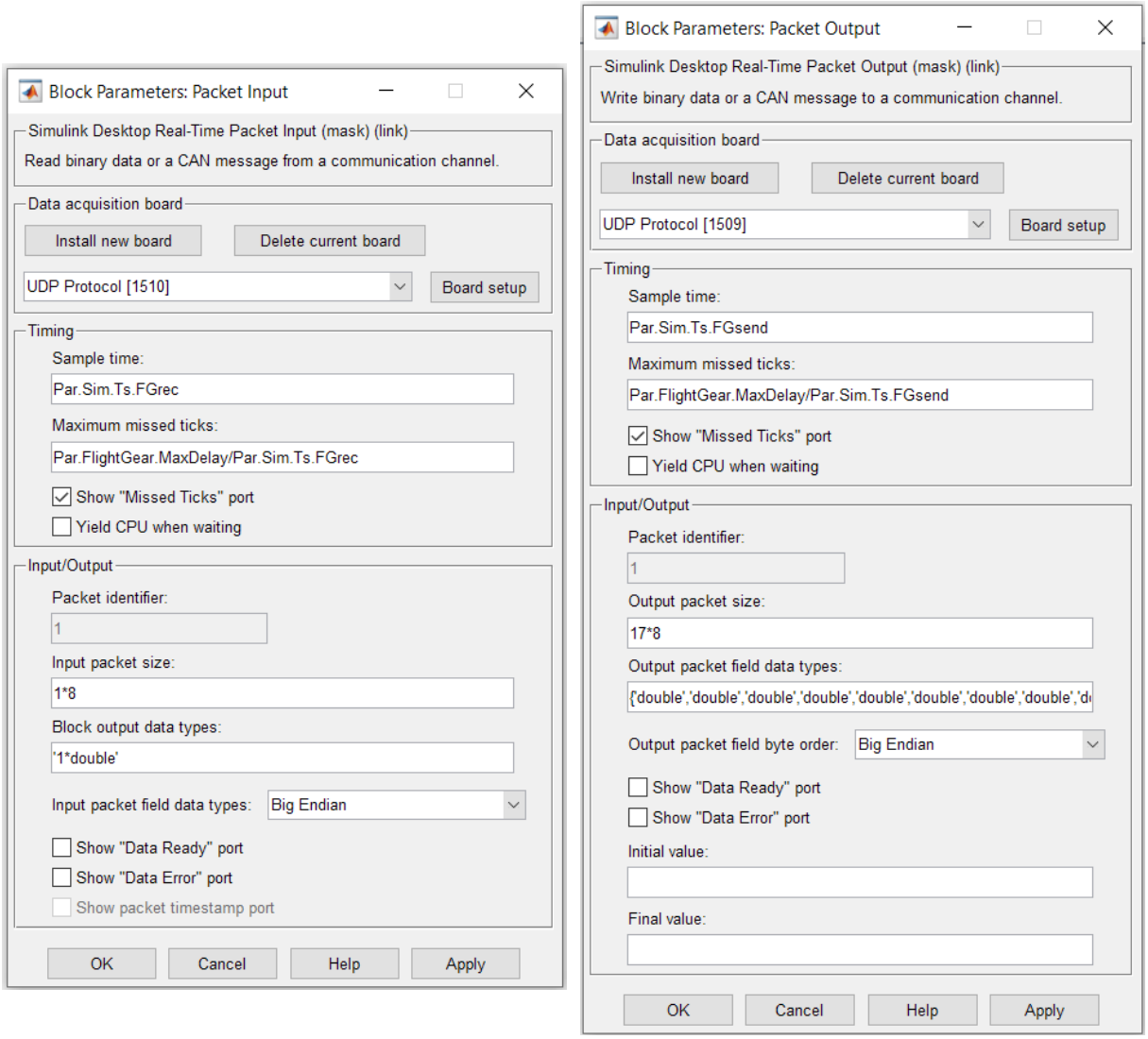

Figure 3.19: Packet Output and Packet Input blocks Parameters

Through the block settings, as shown in Figure 3.19, it is possible to choose between different communication protocols, including Serial, TCP, UDP, CAN. As explained in

Par. 2.4.1, the communication protocol chosen is the UDP protocol, which is best suited to a real-time application such as this.

The blocks, according to the settings provided in the mask, prepare the packet in terms of header and data set, and send it to the specified destination port at the set sample rate.

The size of the packet and the sequence of data to be sent via the block, their type, and endianness must be defined in accordance with the FlightGear software protocol described in Par 3.1.4.

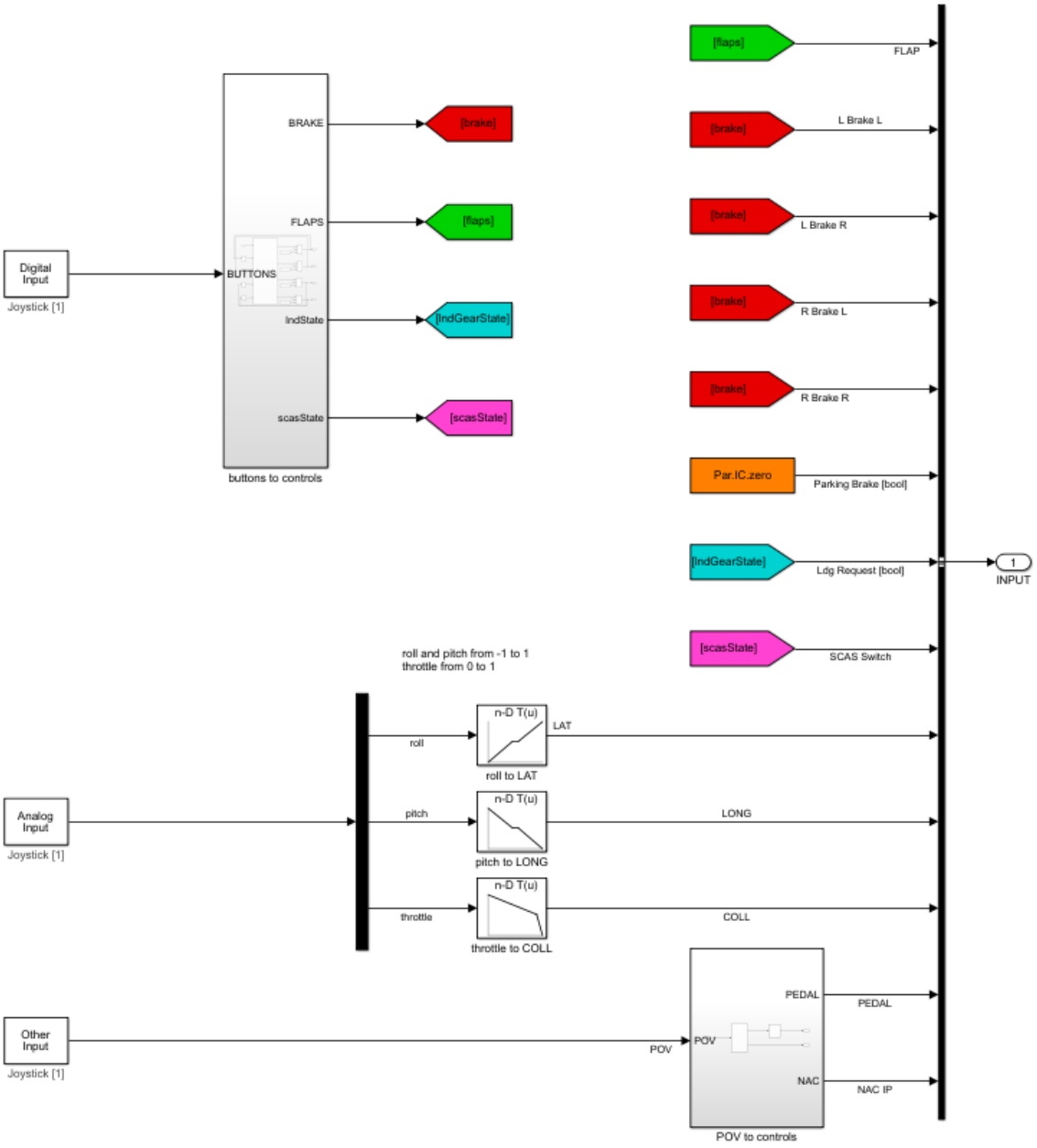

Figure 3.20: ThrustMaster USB Joystick Simulink Interface

3.2.4 Hardware Interface in Simulink

In the first iteration of the simulator, to set up the simulation model and its interface with FlightGear, it was decided to use an inexpensive stick (ThrustMaster USB Joystick) for the first few starts. This made it possible to immediately generate the programming logic useful for managing the signals to be sent to the mathematical model (Figure 3.20).

First of all, it was decided to insert a lookup table for the roll and pitch commands, introducing a dead band in the central position, to reduce unintentional commands and simplify hovering in helicopter mode.

In addition, a control logic was introduced for the buttons, which alternates the ON/OFF status of the signal with each press. This is done thanks to a series of blocks as shown in Figure 3.22, which allow to vary the signals according to the current state and the previous instant of time of the model. Simulink is a non-sequential language and can therefore lead to algebraic loops, i.e. structures for which the priority of calculation execution is important but undefined [14].

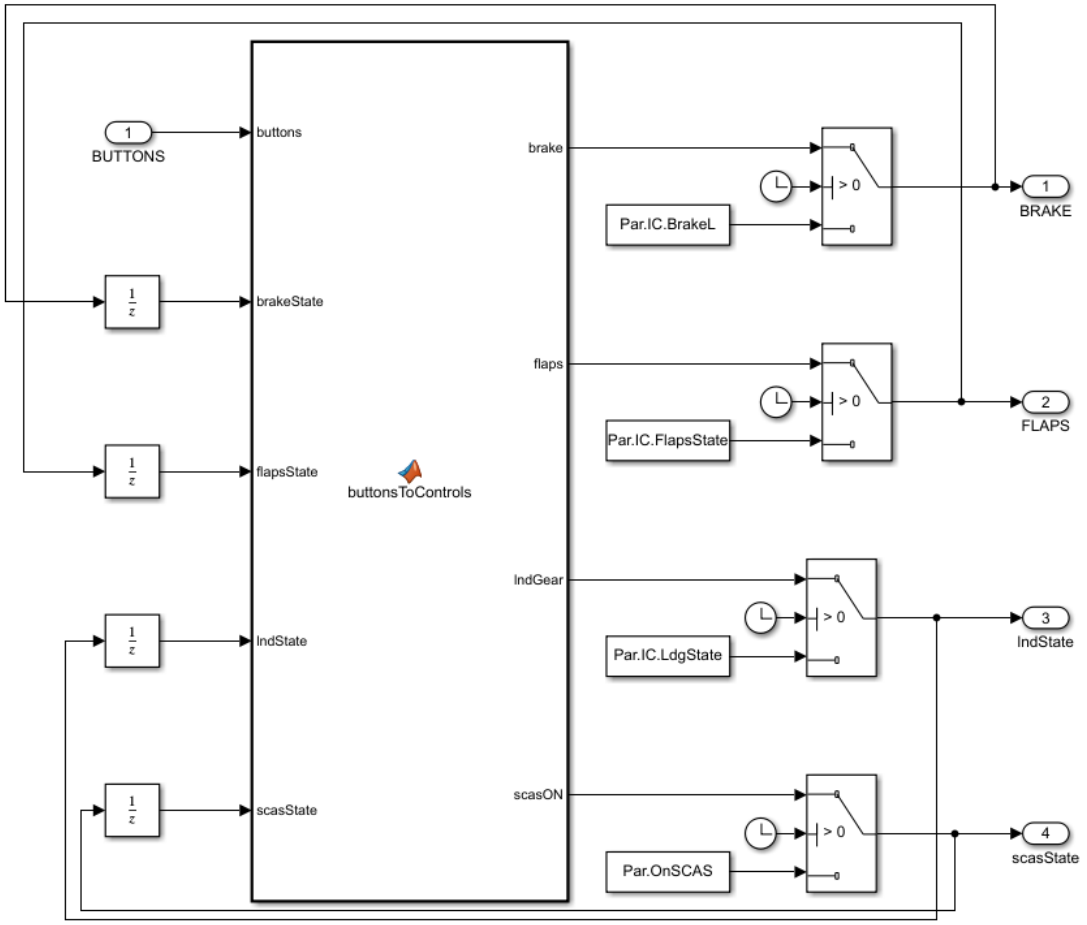

Figure 3.21: “buttons to controls” function detail

To break the loop, it was decided to introduce a delay equal to one unit of time for the feedback signal, obtained with integrators (1/z), as shown in Figure 3.21:

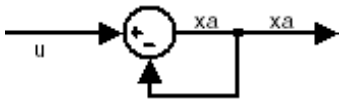

Figure 3.22: Algebraic Loop example in Simulink

The MATLAB function "buttonsToControls" built into Simulink then uses this information to toggle the output value and finally send it to the simulation model. (For the code see Appendix A - Figure A.3)

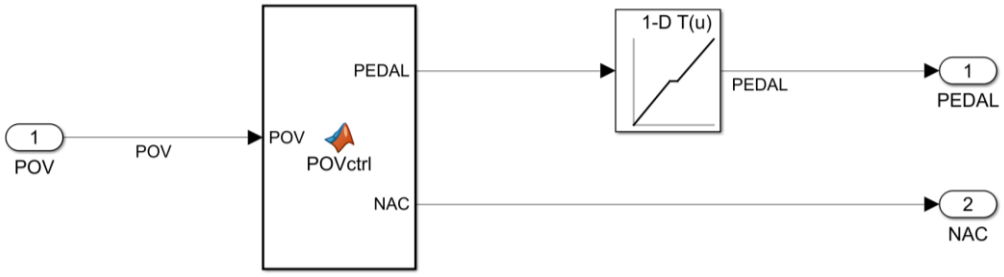

Figure 3.23: "POV to controls" function detail

A similar procedure was carried out for the POV hat, on which the control of the nacelles and, in the first iteration, the rudder were mapped in the absence of a specific axis for this command. The logic is similar to that for the buttons, but since they are momentary commands, it is not necessary to feedback the signal output from the MATLAB function (Figure 3.23).

```
1   function [PEDAL, NAC] = POVctrl(POV)

2   PEDAL = 0;
3   if POV == 90 %POV dx
4   PEDAL = 1;
5   elseif POV == 270 %POV sx
6   PEDAL = -1;
7   end

8   NAC = 0;
9   if POV == 180 %POV down
10  NAC = 1;
11  elseif POV == 0 %POV up
12  NAC = -1;
13  end

14  end
```

Figure 3.24: "POVctrl" MATLAB code

3.2.5 FlightGear Interface

The interface with FlightGear within Simulink, as introduced in chapter 3.1 consists of two parts of code, one for receiving and one for sending data, as shown in Figure 3.25.

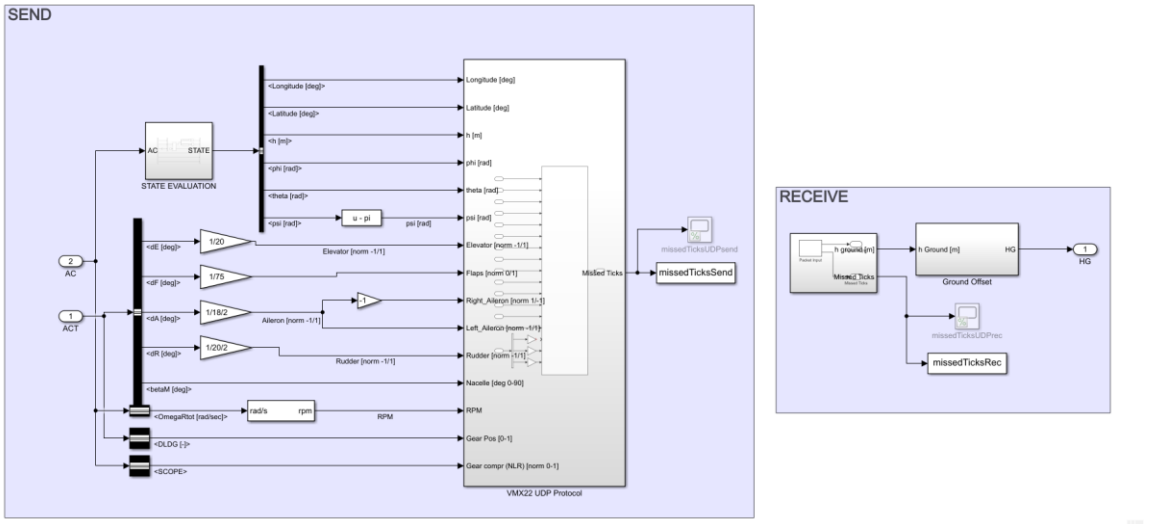

Figure 3.25: Flight Gear Interface in Simulink

The signals generated by the Simulink code are sent from the packet exchange blocks according to the set transmission protocol. The order in which the data are sent, and their size must be the same as set in the protocol used in FlightGear, as shown in Paragraph Simulink Interface 3.1.4.

To reproduce the correct position of the aircraft, FlightGear requires positions in terms of longitude, latitude, and altitude in relation to sea level. However, the mathematical model is independent of the geographical position and is designed as an offline simulator. For this reason, the output values for the position are displacements on the x, y, and z axes integral to the ground (insert custom fig.), which define the displacement from the starting point of the simulation.

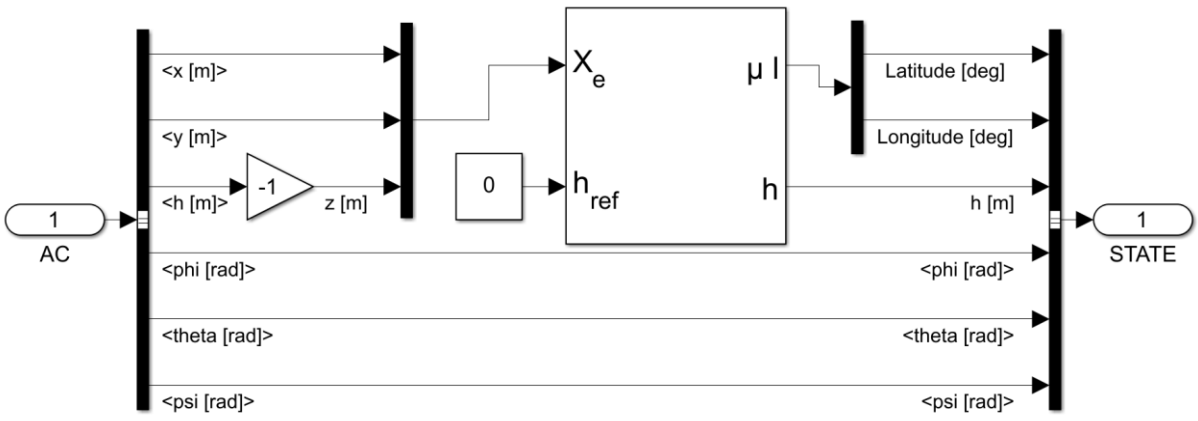

Figure 3.26: State Evaluation block detail

Therefore, it is necessary to calculate the latitude and longitude by changing from a Flat Earth reference system to one based on the 3D geographical position, as shown in Figure 3.26.

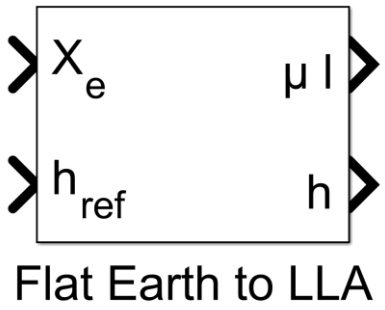

Figure 3.27: Flat Earth to LLA block from Aerospace Toolbox [15]

In the Simulink library "Aerospace Blockset" there is a block (Figure 3.27) that performs the operation of transforming the position coordinates from a flat model to the geographical position of the aircraft. As input, it receives the x, y and z signals as a vector and a possible offset altitude and returns the coordinates in LLA (Latitude, Longitude, Altitude) form that can be used directly by FlightGear (See Par 2.2.6 for further information).

On the other hand, for the reception side, the only useful data for the simulation coming from FlightGear is the altitude of the ground below. The simulation environment knows this data from the position in terms of instantaneous latitude and longitude, from which it derives the altitude of the terrain thanks to the database that is constantly updated live with the TerraSync utility.

The information is then sent back to the mathematical model, which uses it to translate the altitude with respect to the terrain and thus correctly calculate the environmental data and represent the correct interaction of the landing gear with the terrain.

3.3 Flight controls

The flight controls foreseen by the mathematical model are the standard ones of an airplane, with rudder, aileron, and balancer controls, which are however mixed in a transparent way to the pilot according to the flight mode. Moreover, there are the classical flaps, brakes, and landing gear, but also the more particular collective and nacelle position control commands.

The controls are specialized within the model for the XV-15 aircraft, with the respective stroke values to be given in inches. In Table 3.1 can be read the values for the main input controls, while in

Table 3.2 for secondary input controls.

Table 3.1: Main Controls Ranges

Direction	Longitudinal [in]	Lateral [in]	Pedal [in]	Collective	Brake (L, R)
-	-4.8	-4.8	-2.5	0	0
+	+4.8	+4.8	+2.5	10	1

Table 3.2: Secondary Controls Ranges

Control	Range Values
Flap	4 positions (equispaced 0÷1)
Parking Brake	ON/OFF (Bool [0/1])
Landing Gear	ON/OFF (Bool [0/1])
SCAS Switch	ON/OFF (Bool [0/1])

3.3.1 Controls Normalization

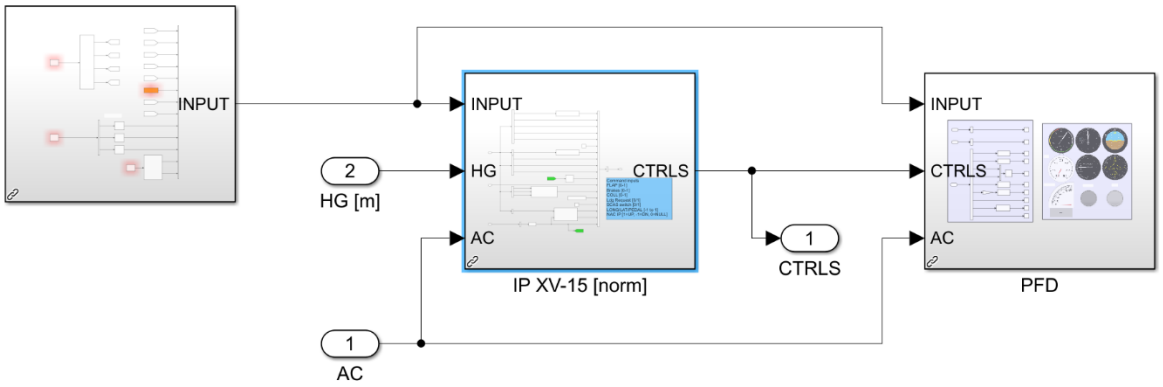

Figure 3.28: Pilot Interface Block

To simplify the implementation of the simulator, to allow future upgrades or to change the control peripherals, a block was added to the Simulink code to normalize the command input values. Moreover, the normal peripherals return through the Simulink blocks variable signals from -1 to 1 for the control axes, simplifying the implementation. This normalization is carried out inside the block "Pilot Interface" (visible in Figure 3.28) that contains all the code to produce the signals of input for the mathematical model of simulation (Figure 3.29).

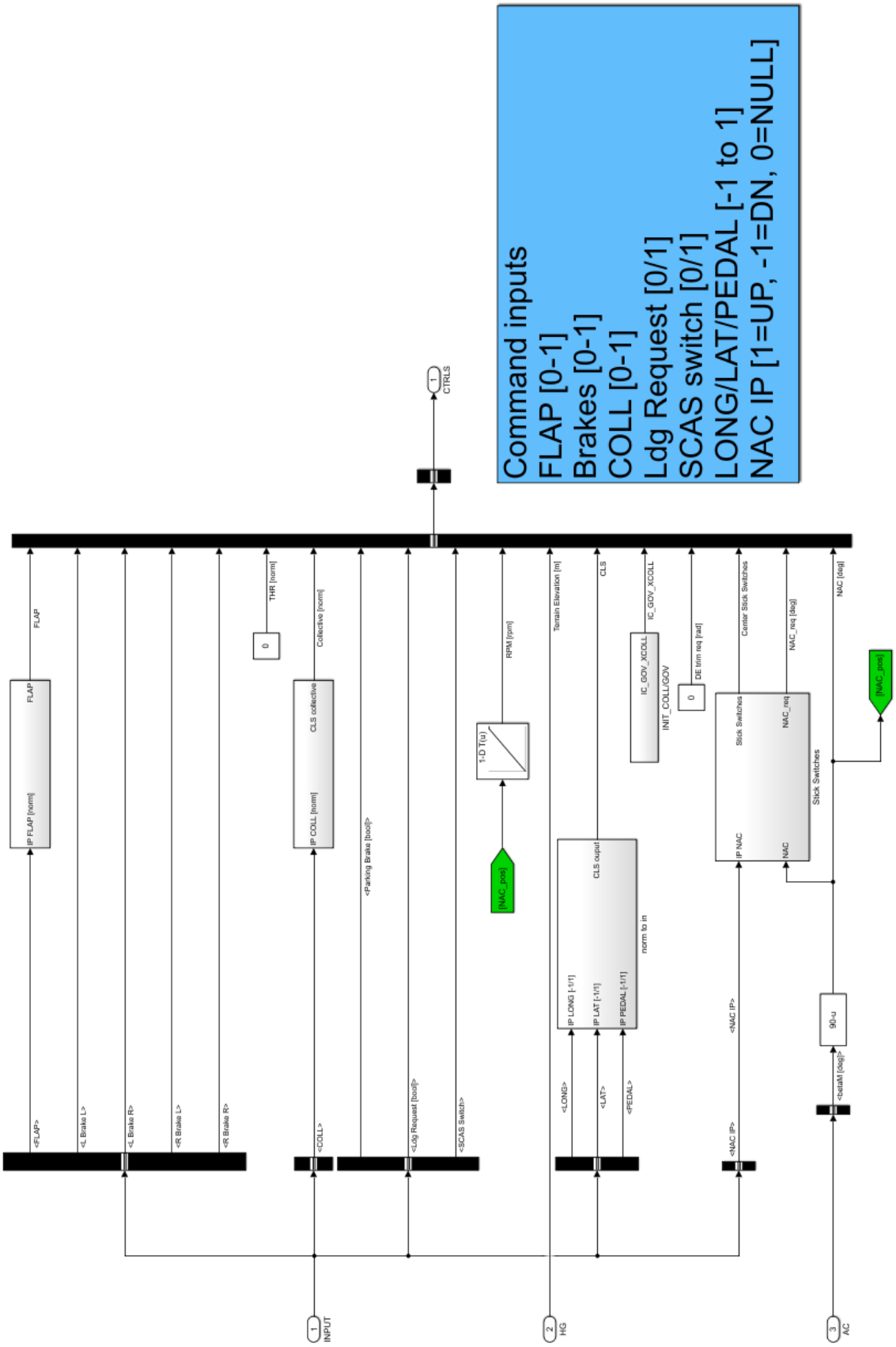

Figure 3.29: Normalization block - “IP XV-15 [norm]” detail

A separate discussion should be made for the throttle and engine nacelle position controls:

- For the throttle, the model must be provided with the RPM value required from the governor. In the XV-15 aircraft this command is transparent to the pilot and mixed with the position of the nacelles. The value therefore remains fixed for each position angle and varies from a value of 589 RPM when the rotors are vertical, up to 517 RPM in horizontal position (Figure 3.30).

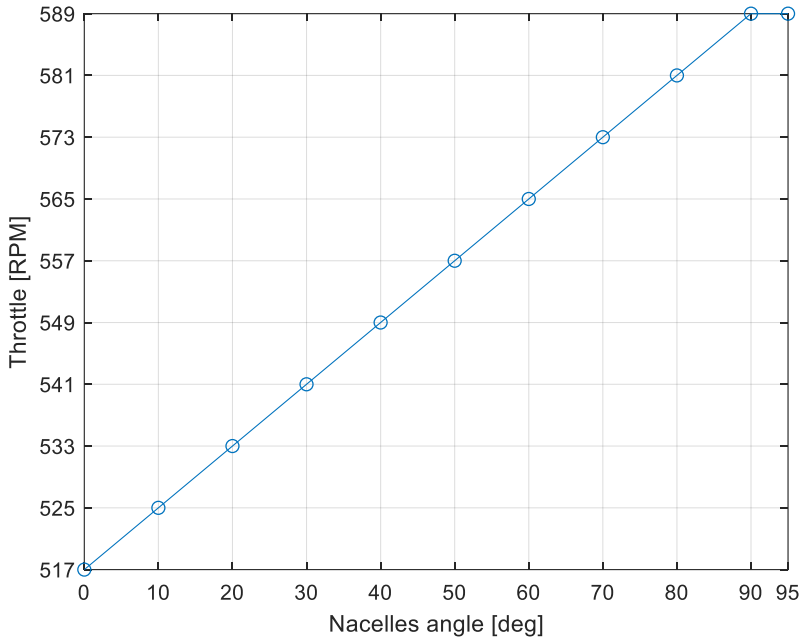

Figure 3.30: NAC to RPM logic

- For the nacelles, the simulation model receives as input only a command to move up or down. Given the hardware configuration of the simulator, it was chosen, in the normalization block, to include a logic that allows the stroke to be divided into steps, an easier method to not be forced to hold the buttons down (Figure 3.31). The steps size can then be modified within the code at will (15° for each step is the value chosen by the author).

  With the same logic expressed in the function "buttonsToControls" (Par. 3.2.4), the actual value requested to the actuators to move the nacelles is fed back to the MATLAB function "nac_IP2req", to compare the pilot input to the actual position (NAC input number 2 in Figure 3.31). At the last the error signal is read by the function "nac_DIFF2act" to produce a movement input for nacelles actuators (UP or DOWN) (full code functions respectively in Figure A.5 and Figure A.6)

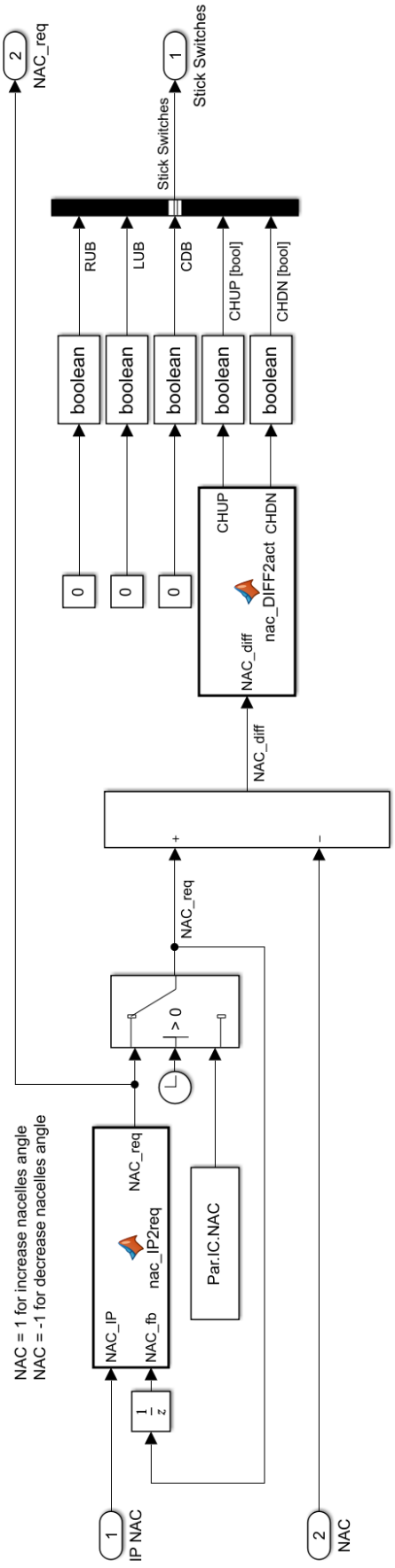

Figure 3.31: NAC control logic – “Stick Switches” block detail

Figure 3.32: Simulink Primary Flight Display (PFD) Block detail

In the Pilot Interface Block there is also a block that integrates a simple Primary Flight Display useful for displaying basic flight data, for command diagnostics and aircraft response (Figure 3.32).

3.4 Hardware

The hardware is composed from a laptop on which it runs the simulation software and the graphical environment at the same time. Given the modest computational complexity of the model and the graphical environment, it has been chosen to use a portable workstation, more than sufficient to run everything respecting the requirements of latency provided by legislation for the input from hardware and the updating frequency of the model and the graphical environment.

In addition to the laptop there are peripherals for the pilot to control the flight commands. They consist of 2 commercial hardware components: the flight stick and the rudder pedals.

3.4.1 Computer

As shown in Figure 3.33, the device on which run MATLAB and the simulation environment is a Dell Precision 7550 laptop. This device is a workstation with excellent computing power that provides the right performance to run the simulator.

Figure 3.33: Dell® Precision 7550

3.4.2 Flight Stick

The flight stick is a ThrustMaster® product that consists of a lever that controls the lateral and the longitudinal axes, and also has a longitudinal rotation axis that could be used for rudder control, which however is delegated to the pedals. There are 15 buttons set as in the Figure 3.34 to set the positions of the undercarriage, flaps, longitudinal trims, position of the nacelles. At the bottom is located the collective lever, which allows to set the pitch of the blades, because the throttle is automatically managed by the governor that sets the speed of the engine based on the position of nacelles.

The main axes output is visible in Figure 3.35, in which can be seen that the signals values follow the bipolar normalized logic explained in Par. 3.2.4.

Figure 3.34: ThrustMaster® T-16000

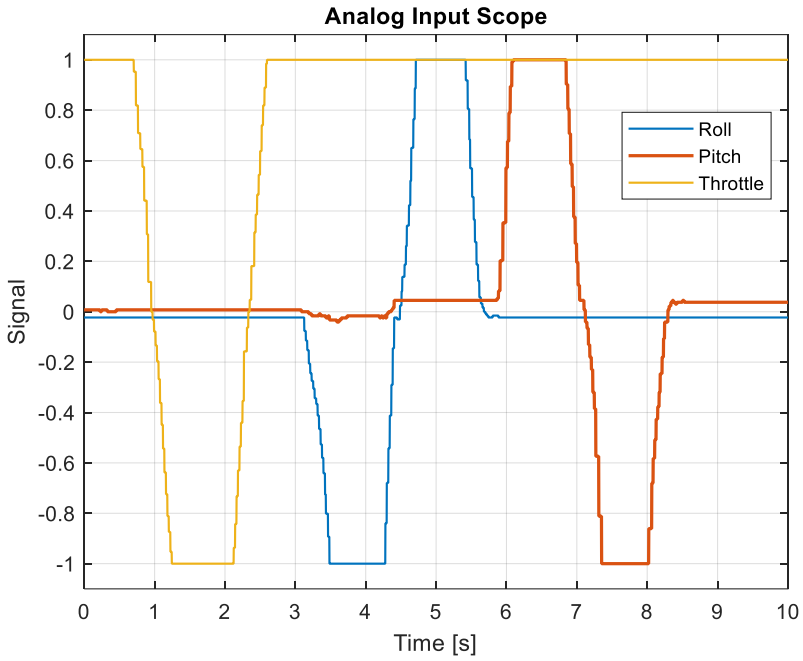

Figure 3.35: Analog Input values demo

3.4.3 Rudder Pedals

Figure 3.36: CH Products Pro Pedals

The pedals allow precise control of the yaw command, which consists of a rudder command or a differential cyclic command for the two rotors in helicopter mode. In addition, individual pedals can be pressed to generate a separate brake command located on the left and right landing gear.

4

Simulations and Results

In order to validate the quality of the simulation and demonstrate the real truthfulness of the adjective real-time, many tests were carried out by the author, in which the latency in terms of execution of the activities carried out by the blocks of the SLDRT library was mainly evaluated.

The investigations were carried out by graphing and evaluating the results for the values of "Missed Ticks" (1 Tick = 1 sample rate [s]) of the blocks sending and receiving data with FlightGear and reading the hardware data. To retrieve the latency information in terms of seconds, it was necessary to multiply the number of Missed Ticks of each block by the sample rate set in the Simulink mask of that particular block.

The next paragraphs then show graphs for the instantaneous latencies during the simulation (latency over time) and the frequency of occurrence of the latency values. In order for each test:

- Attitude in time
- 3D trajectory in time
- Latency [ms] + Occurrences from Sync Block
- Latency [ms] + Occurrences from UDP Receiving Block
- Latency [ms] + Occurrences from UDP Sending Block

4.1 Longitudinal Translation

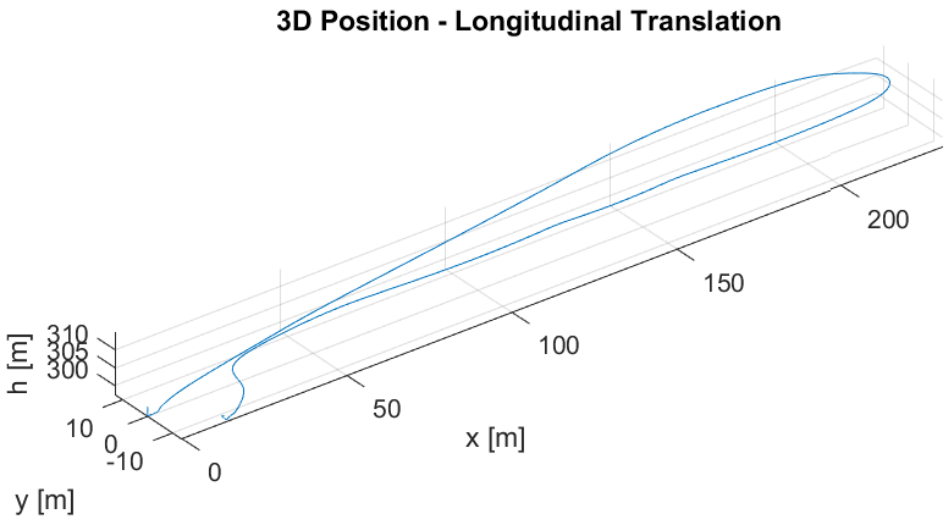

Figure 4.1: Longitudinal Translation Test – Position in time

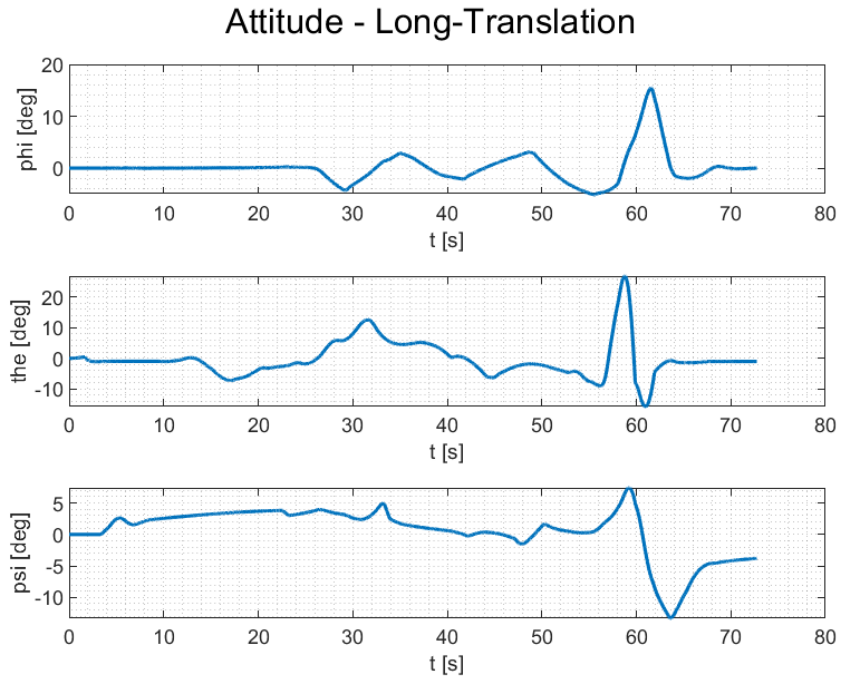

Figure 4.2: Longitudinal Translation Test – Attitude in time

From Figure 4.2 and Figure 4.1 it is possible to observe the manoeuvres and trajectories carried out in the longitudinal translation test. The test consists of a stationary take-off with the nacelles in a vertical position, which are then rotated to perform a longitudinal translation, first forwards and then backwards. Once the aircraft is stopped, a landing is performed with the nacelles upright near the starting point.

In the figures below there are the results of the latencies and provided by the Simulink blocks.

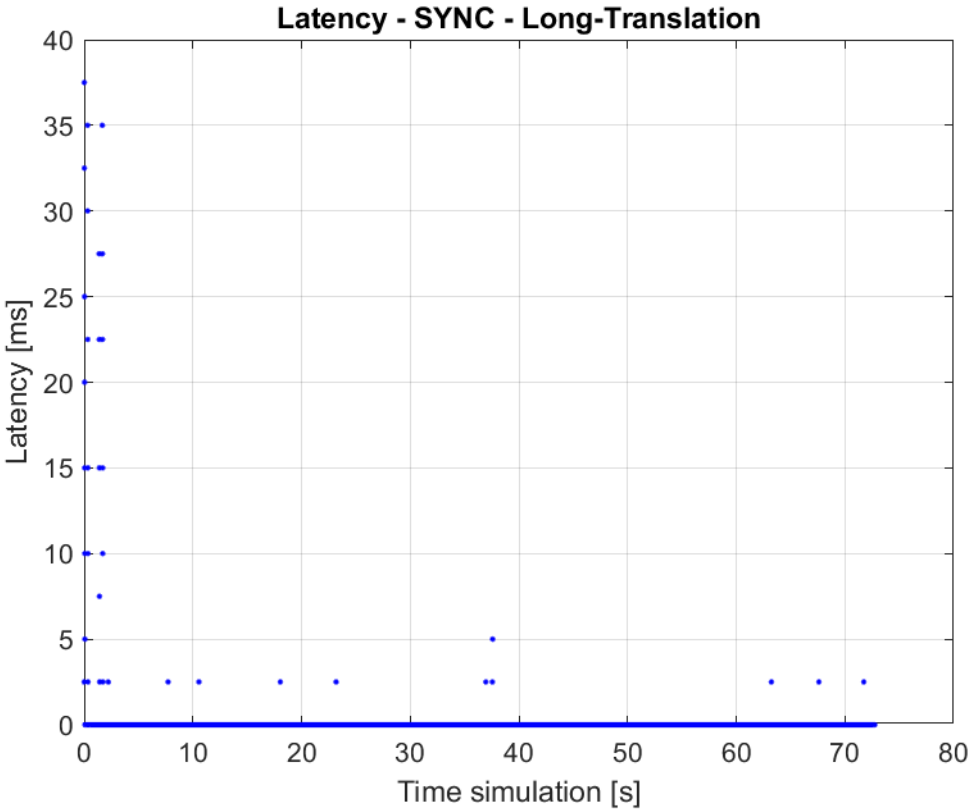

Figure 4.3: Longitudinal Translation Test – SYNC Block Latency

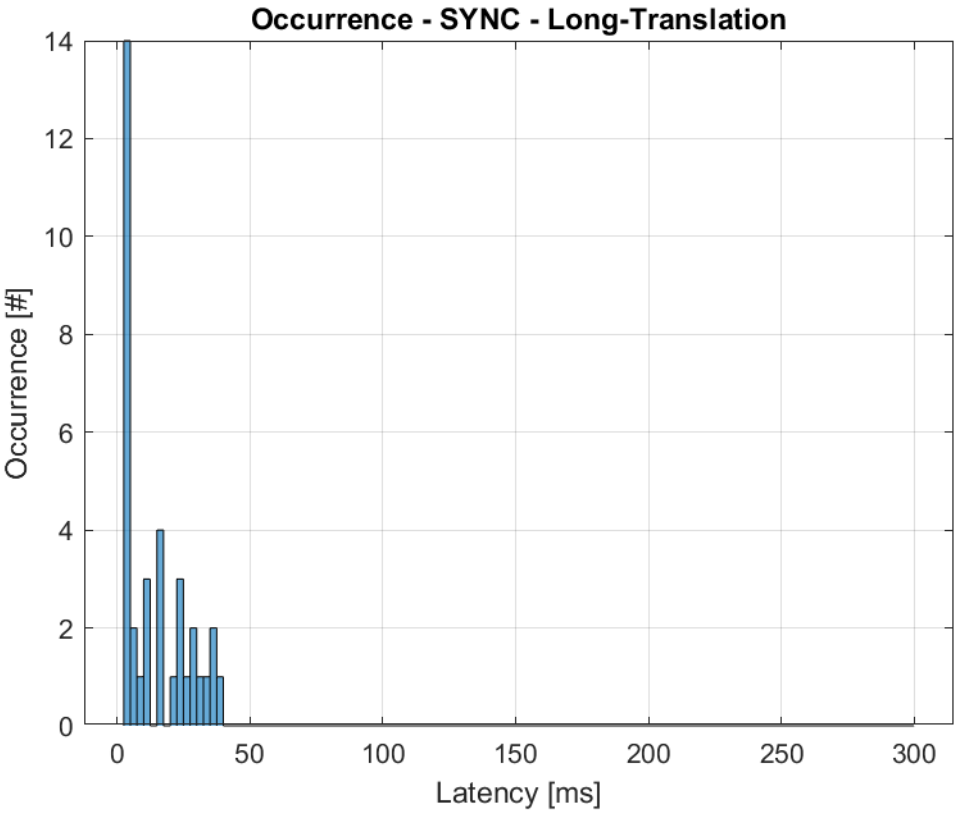

Figure 4.4: Longitudinal Translation Test – SYNC Block Occurrence

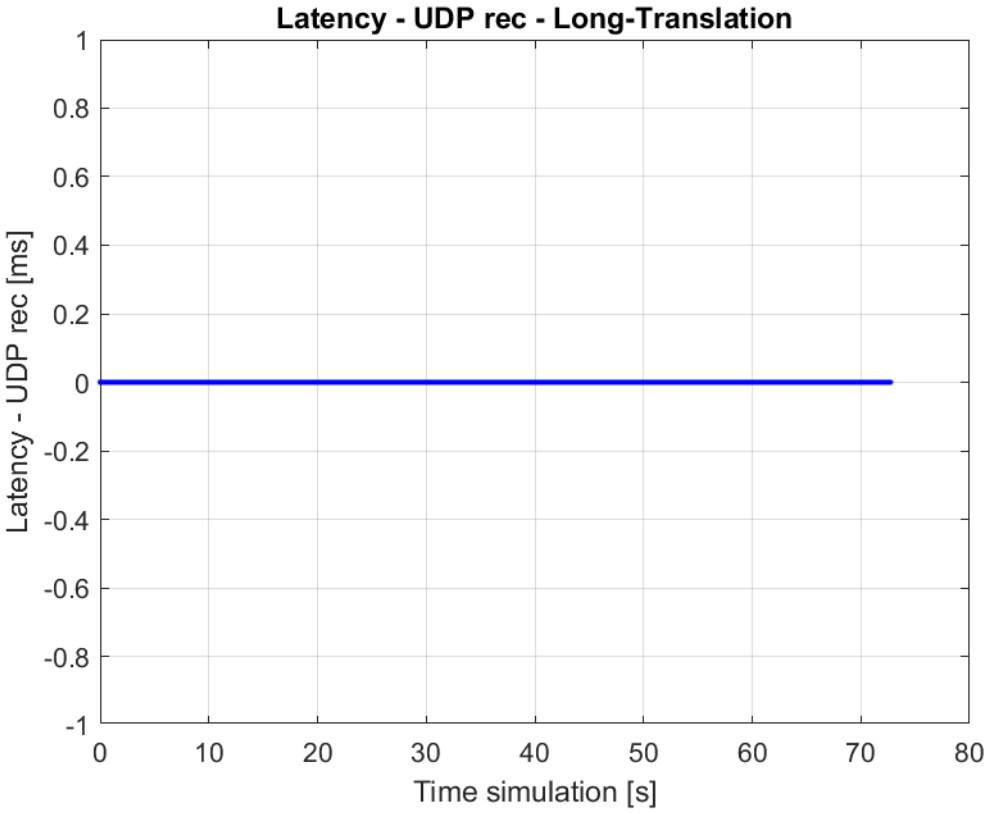

Figure 4.5: Longitudinal Translation Test – UDP rec Latency

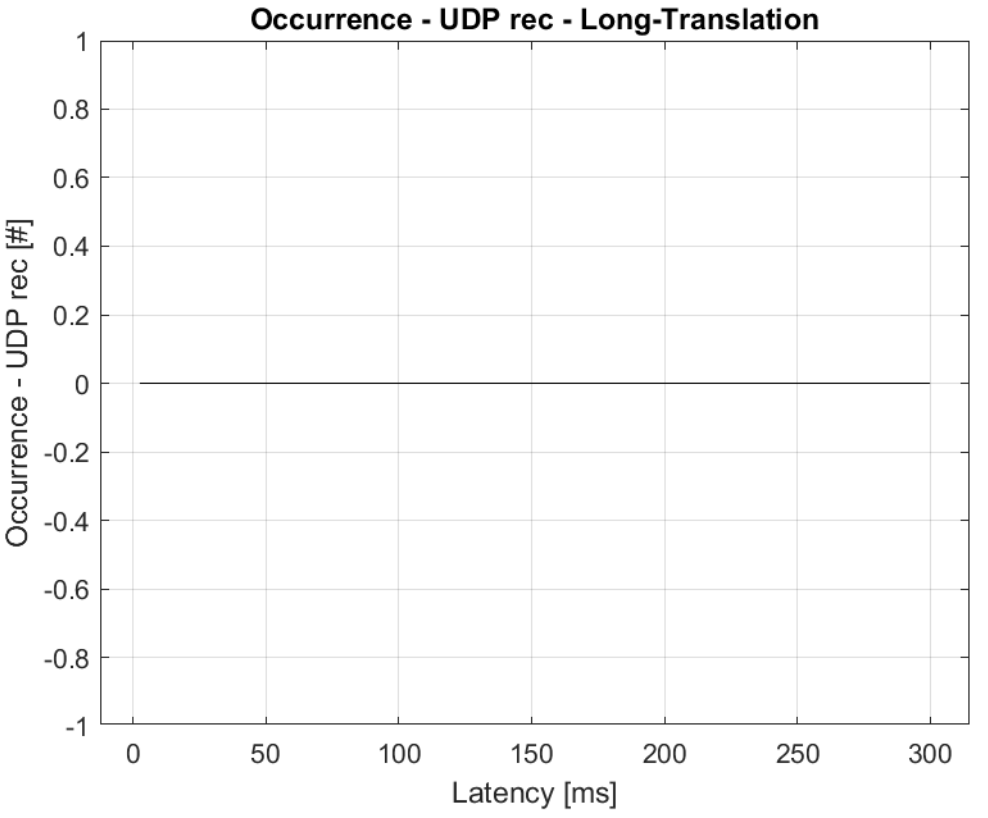

Figure 4.6: Longitudinal Translation Test – UDP rec Occurrence

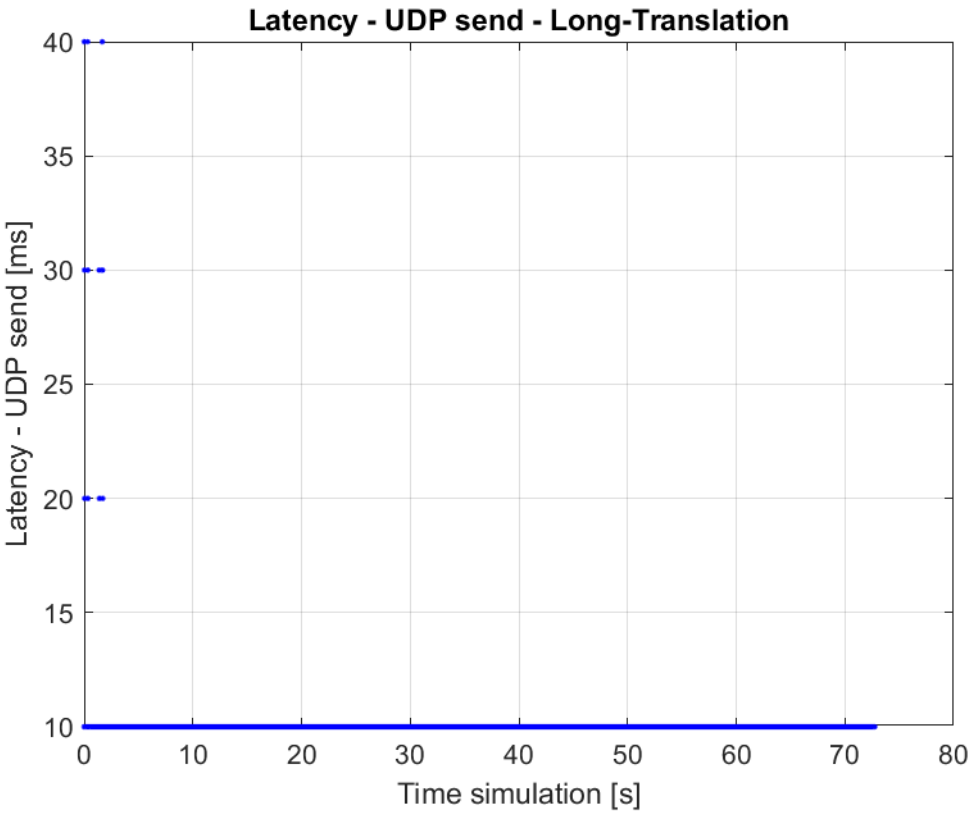

Figure 4.7: Longitudinal Translation Test – UDP send Latency

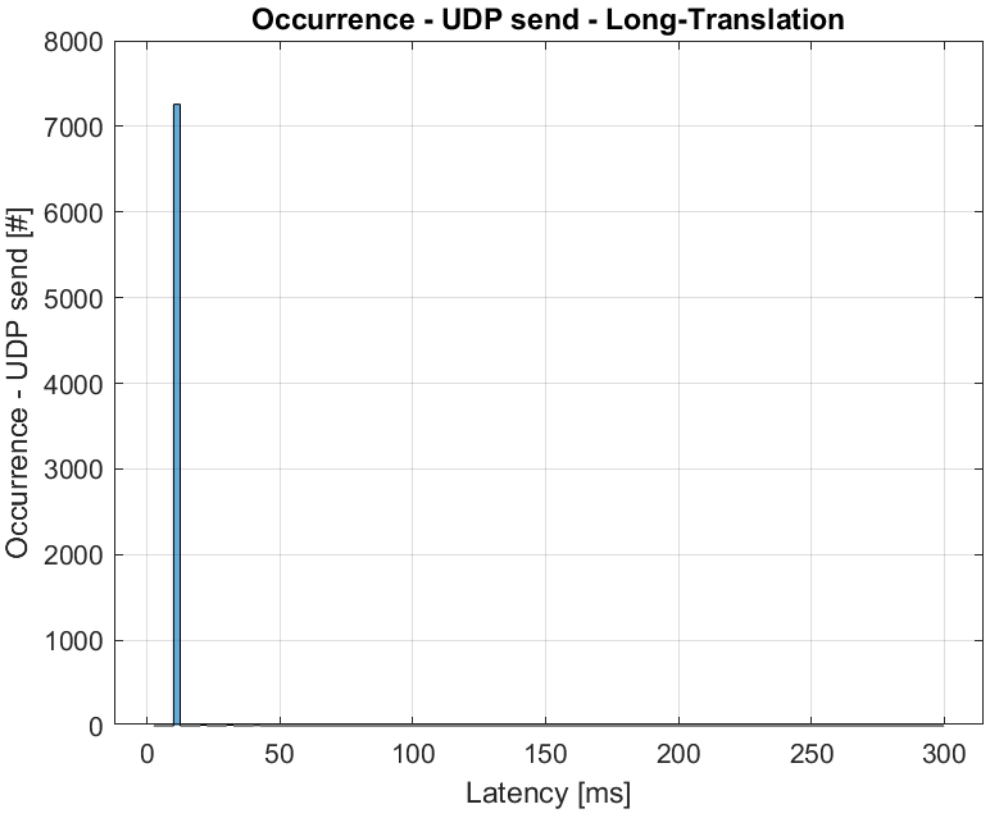

Figure 4.8: Longitudinal Translation Test – UDP send Occurrence

From Figure 4.3 and Figure 4.4 on the latencies of the SYNC block, it can be seen that in the first few seconds of simulation there are small delays due to the start of the simulation. After the simulation has settled down, it remains stable with some sporadic ticks (here equal to 2.5 ms) in the synchronization of the model with the processor clock.

From Figure 4.5 and Figure 4.6 could be appreciated the delay that the UDP reception block presents when it is called to read the only input information, the altitude of the terrain. As can be seen from the simulation, this process does not involve any delay, and Simulink always manages to perform the operation in the expected sampling period.

Finally, in Figure 4.7 and Figure 4.8 there are small delays at the beginning of the simulation, as already indicated for the SYNC block. Once the simulation is started, the delay is not zero, but 10 ms. This value corresponds to the delay induced by the sampling rate of 100 Hz set for the UDP send block, since the operation is performed at the end of the sampling period.

4.2 Hovering Near Ground

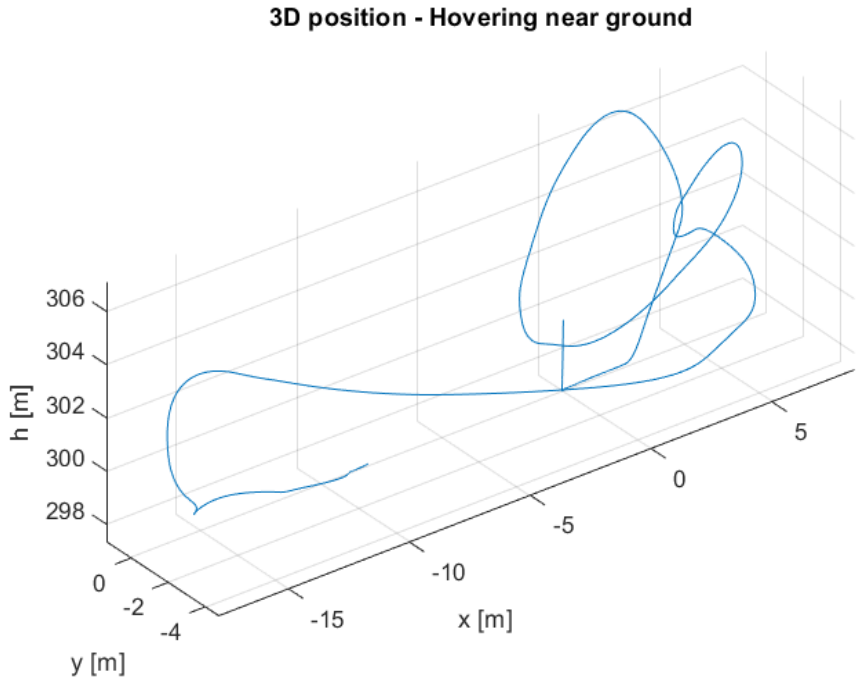

Figure 4.9: Hovering Test – Position in time

From Figure 4.8 and Figure 4.9 it is possible to observe the manoeuvres and trajectories carried out in the hovering near ground test.

The test consists of a take-off with the positions of the nacelles vertical, which remain so throughout the flight. The combined use of the cyclic and collective commands allows the position to be maintained from a few metres above the ground.

In this test, ground effects are more evident and complicate the computational cost, making the test interesting in terms of real-time maintenance.

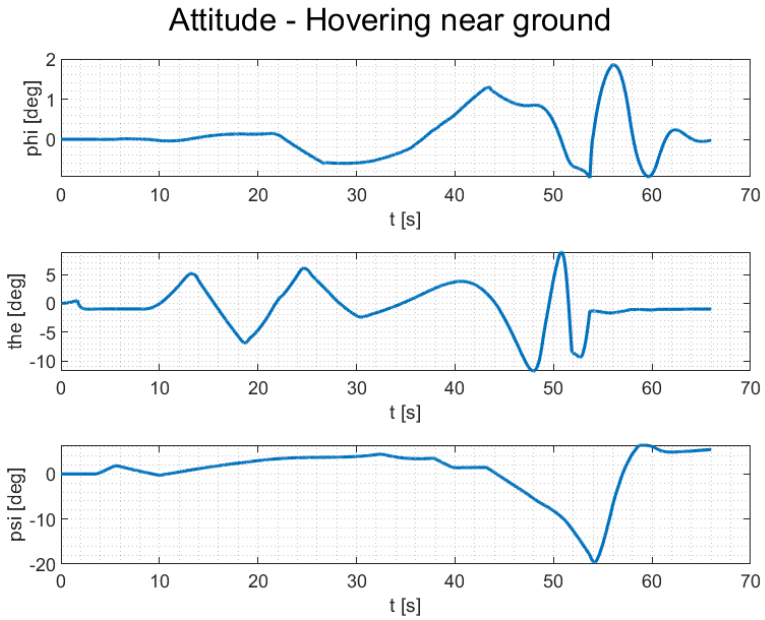

Figure 4.10: Hovering Test – Attitude in time

In the figures below there are the results of the latencies and provided by the Simulink blocks.

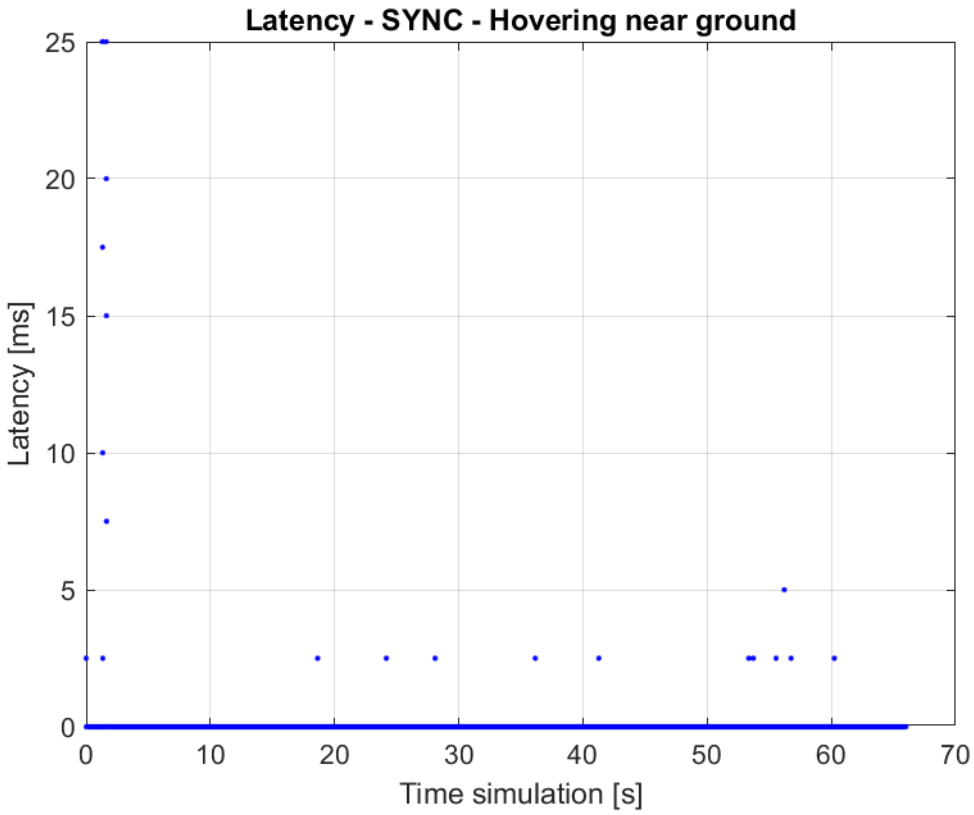

Figure 4.11: Hovering Test – SYNC Block Latency

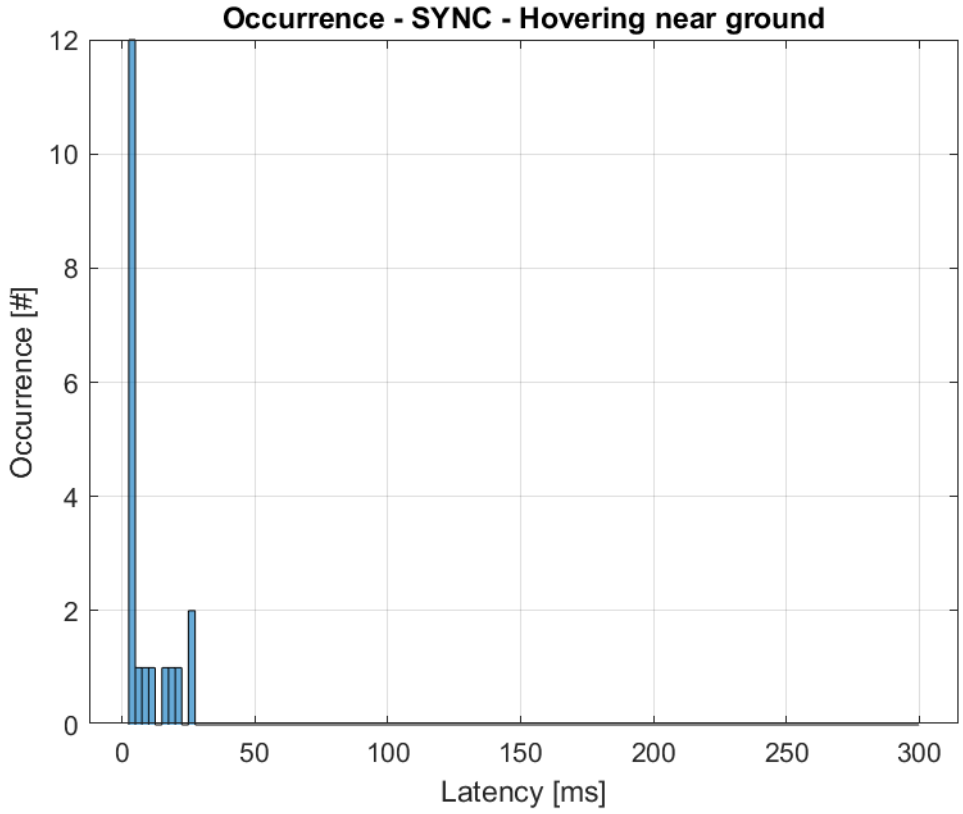

Figure 4.12: Hovering Test – SYNC Block Occurrence

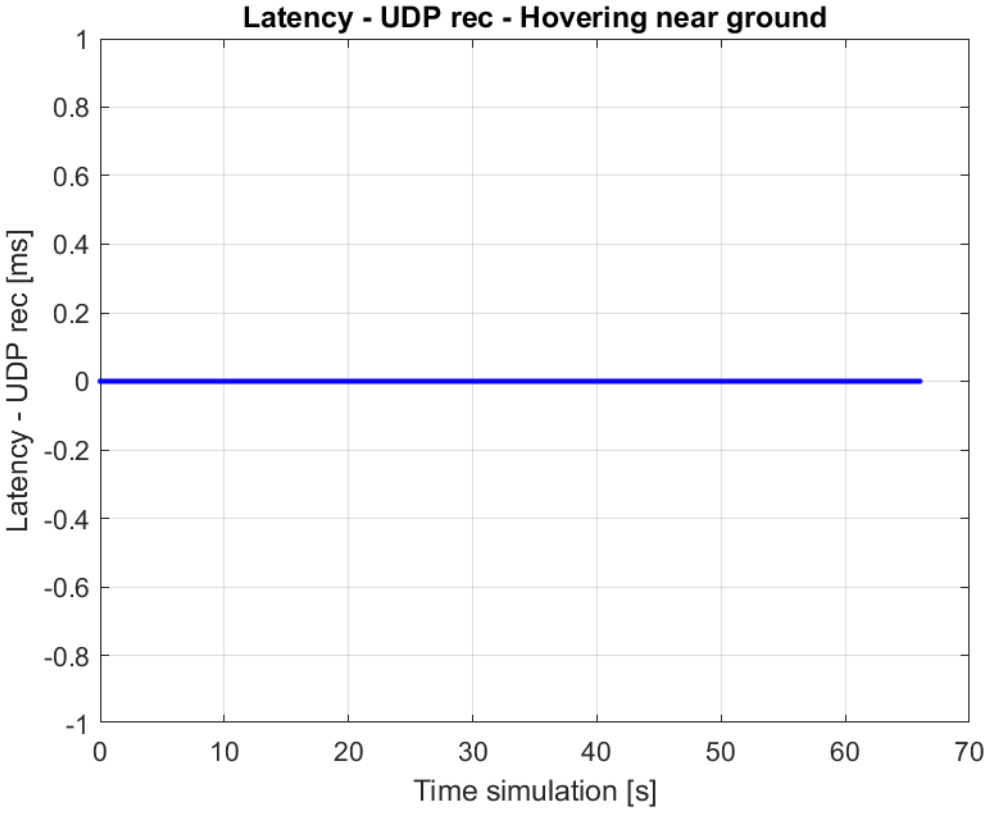

Figure 4.13: Hovering Test – UDP rec Block Latency

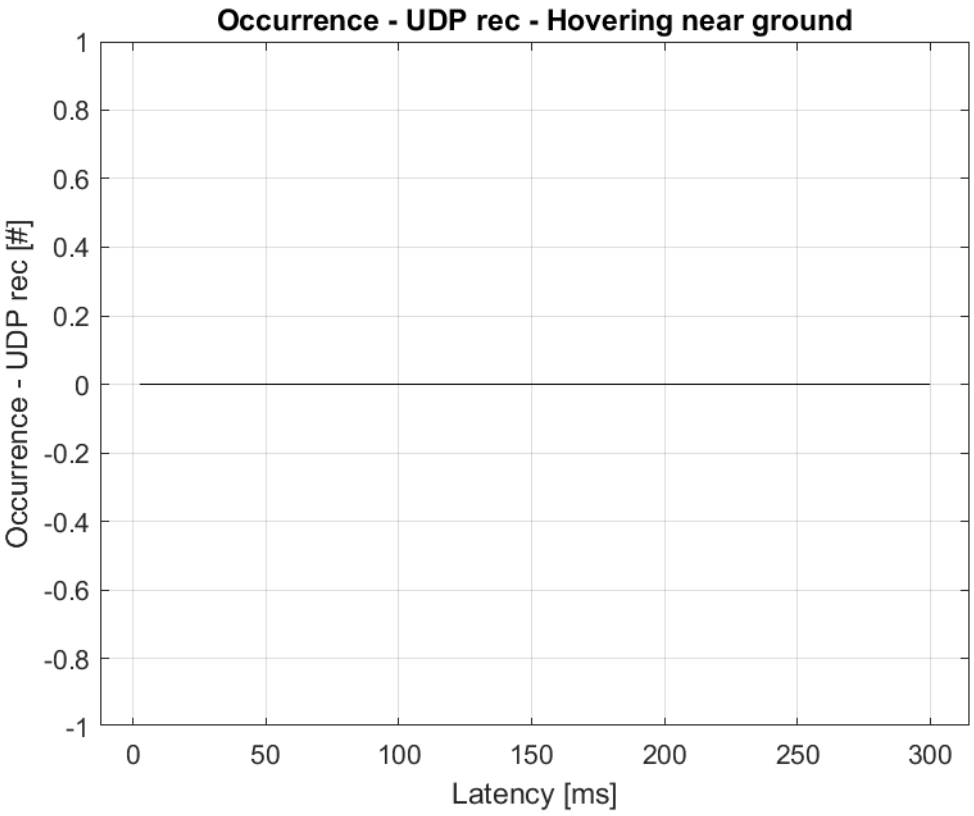

Figure 4.14: Hovering Test – UDP rec Block Occurrence

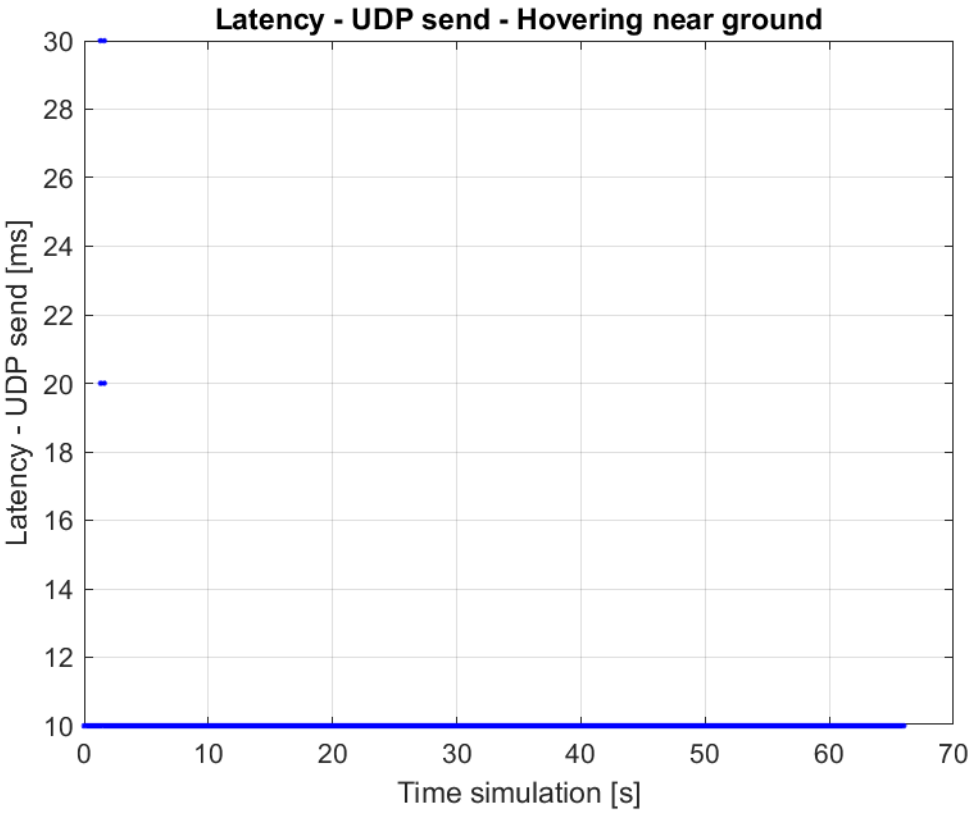

Figure 4.15: Hovering Test – UDP send Block Latency

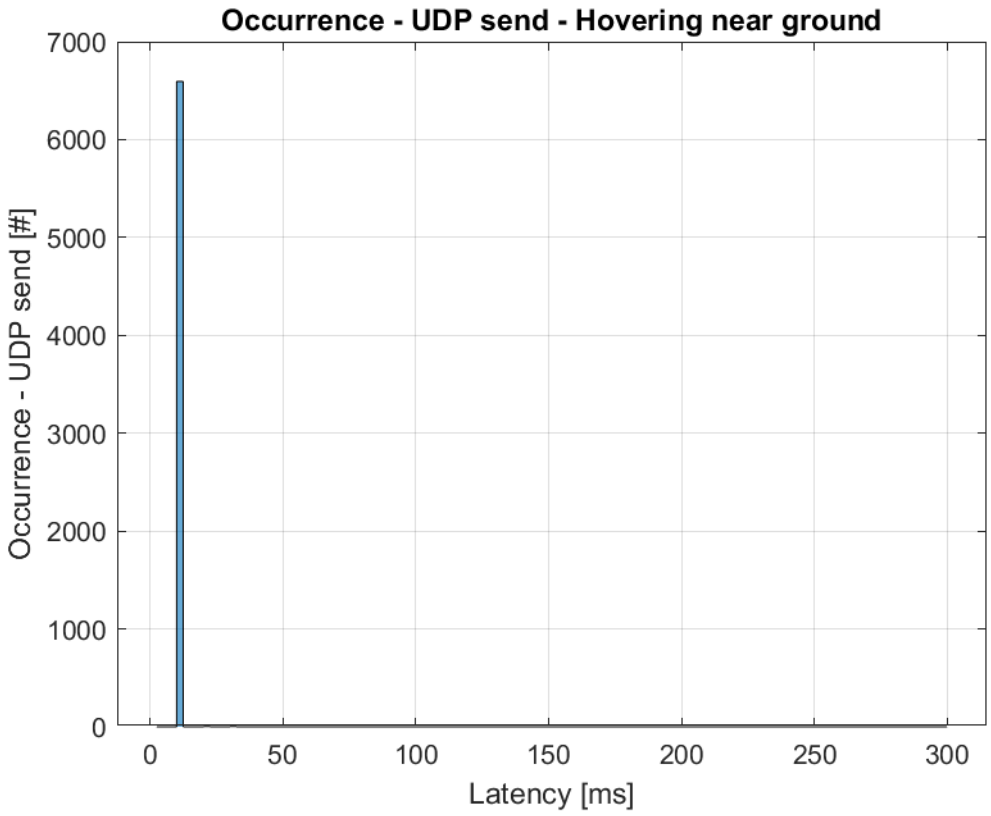

Figure 4.16: Hovering Test – UDP send Block Occurrence

The considerations for the above figures remain the same as those made in Sect. 4.1. This test was carried out to verify the maintenance of the simulation in real-time also for situations in which the computational cost of the mathematical model can be higher. This is because the rotors are close to the ground and the inflow evaluation is more complex.

However the difficulty of added calculation does not seem to induce of the delays much different from the previous test, index of the fact that the simulation model has got an adequate computational cost to a modern portable.

Again for the send and receive blocks (Figure 4.13 and Figure 4.15), delays are concentrated only at the beginning of the simulation, when Simulink has to start the communication via UDP protocol and FlightGear requires more resources to start the animation of the aircraft.

After the brief initial transient, however restricted in a few seconds, the simulation proceeds without delays, if not sporadic peaks as visible in Figure 4.11, which can be traced to a higher computational cost due to the fast manoeuvres that are performed during the landing phase.

4.3 Take-Off + Conversion + Landing

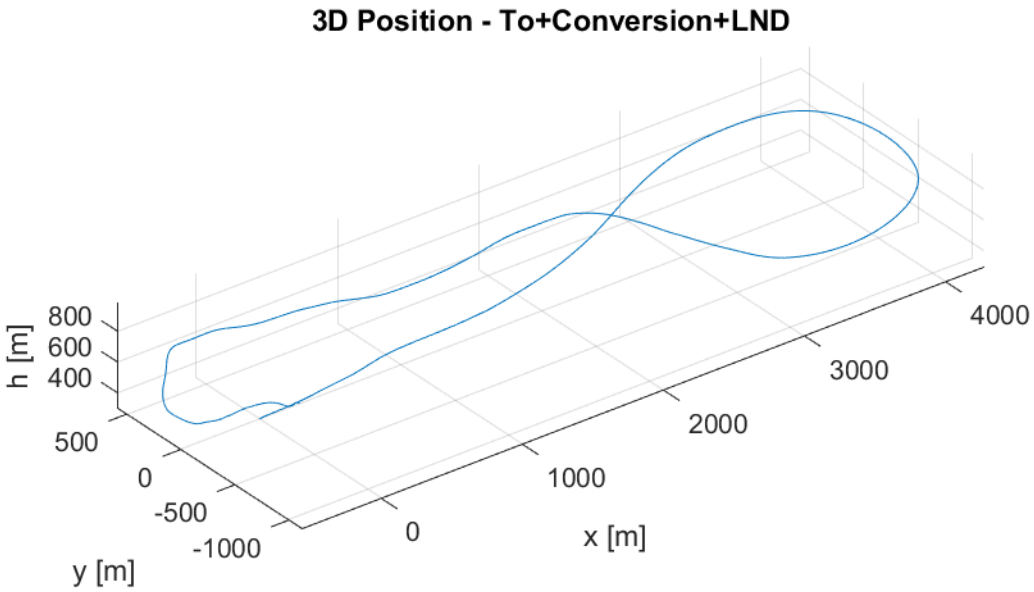

Figure 4.17: Conversion Test – Position in time

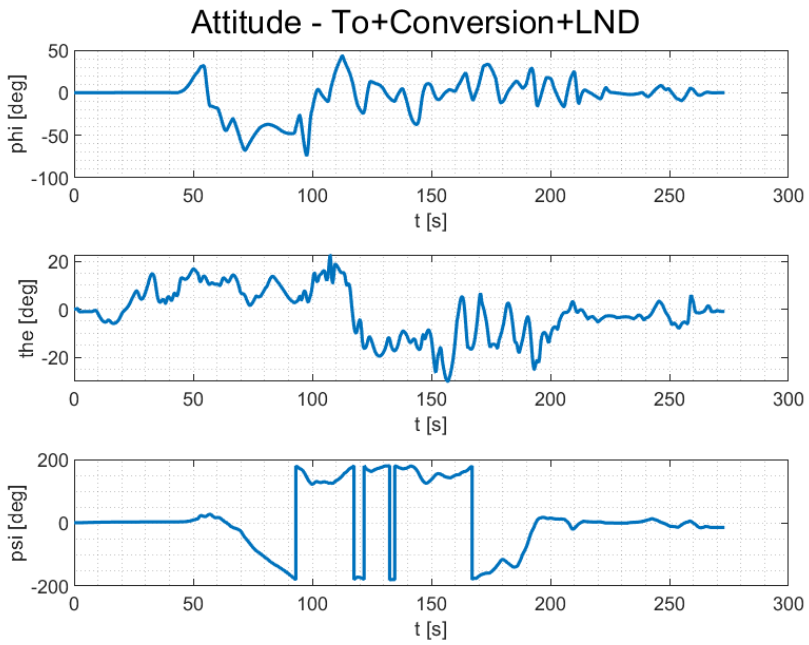

Figure 4.18: Conversion Test – Attitude in time

Figure 4.17 and Figure 4.18 show the trajectory and thus the overall flight performed during the test. This consists of a vertical take-off followed by a conversion to airplane mode by progressively turning the nacelles forward. After a turn to the right the aircraft returns close to the initial position to land in helicopter mode.

In the figures below can be seen the results of the latencies and provided by the Simulink blocks.

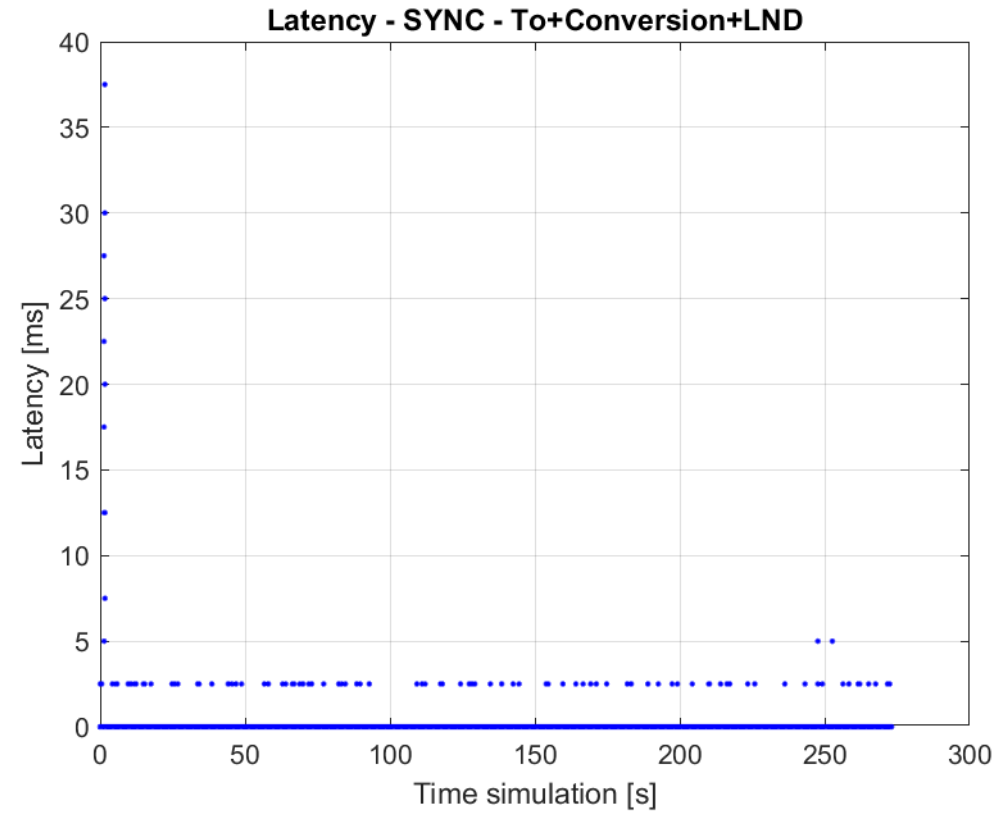

Figure 4.19: Conversion Test – SYNC Block Latency

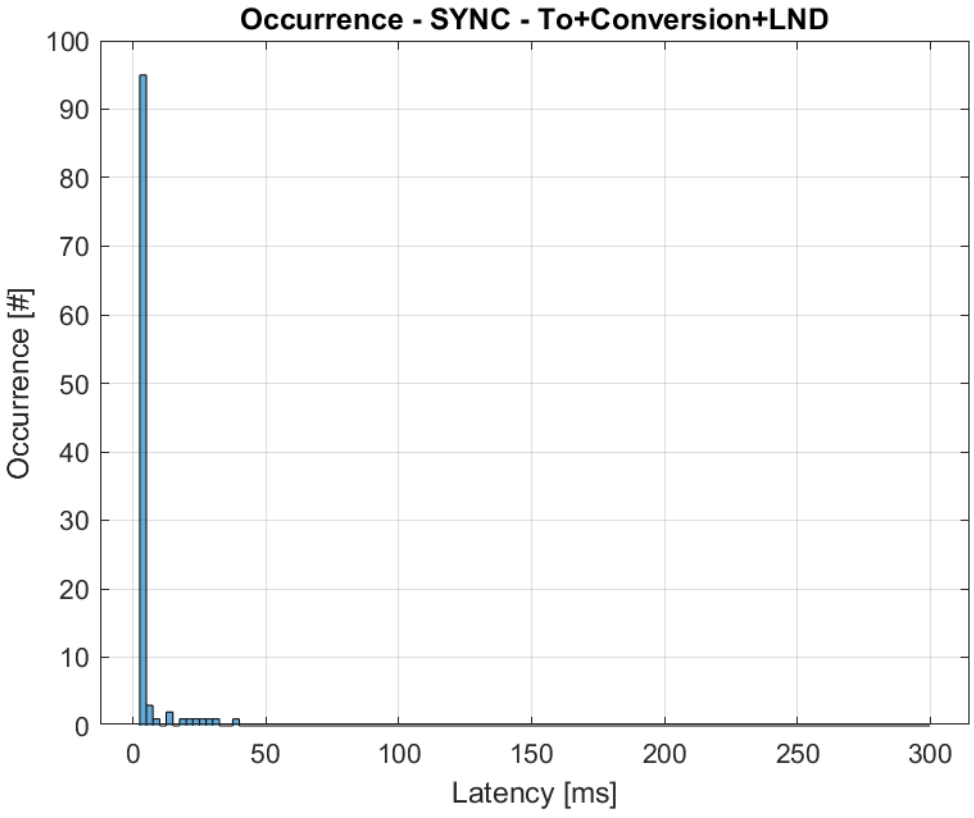

Figure 4.20: Conversion Test – SYNC Block Occurrence

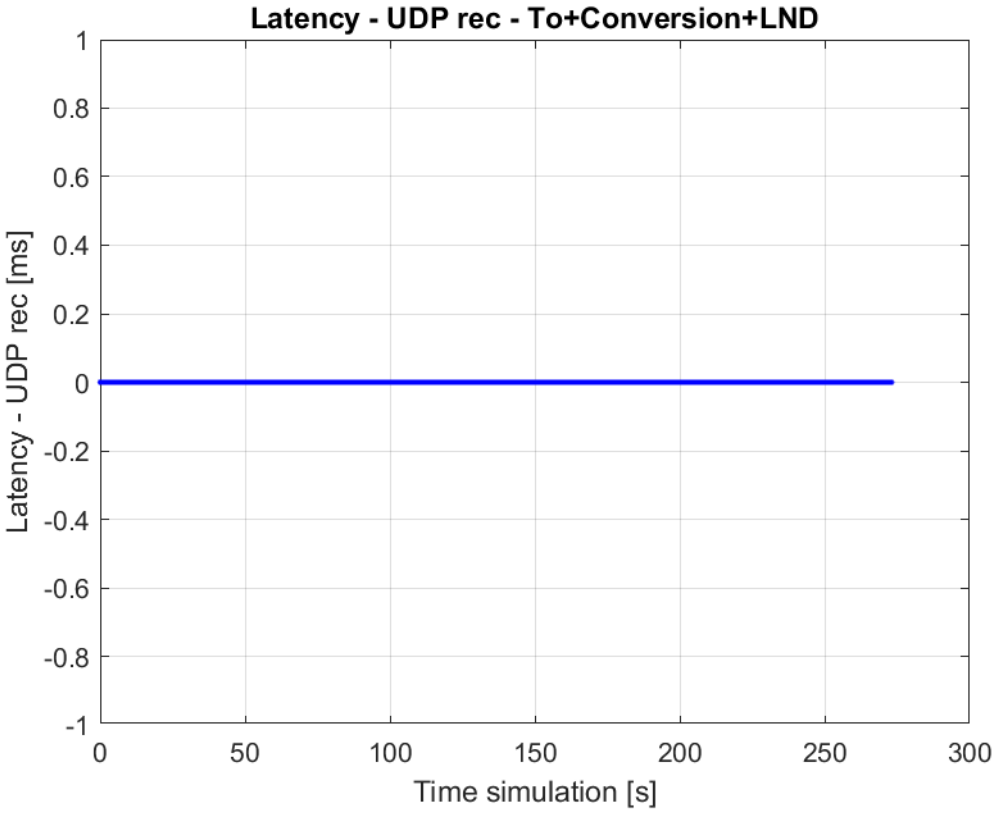

Figure 4.21: Conversion Test – UDP rec Block Latency

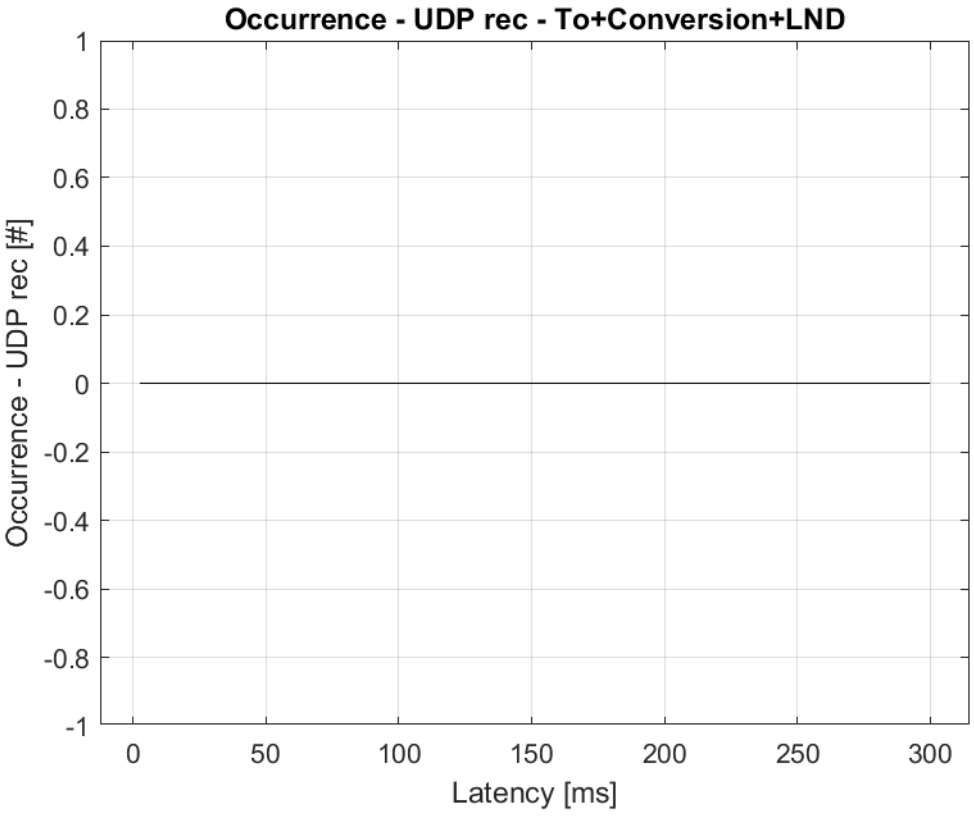

Figure 4.22: Conversion Test – UDP rec Block Occurrence

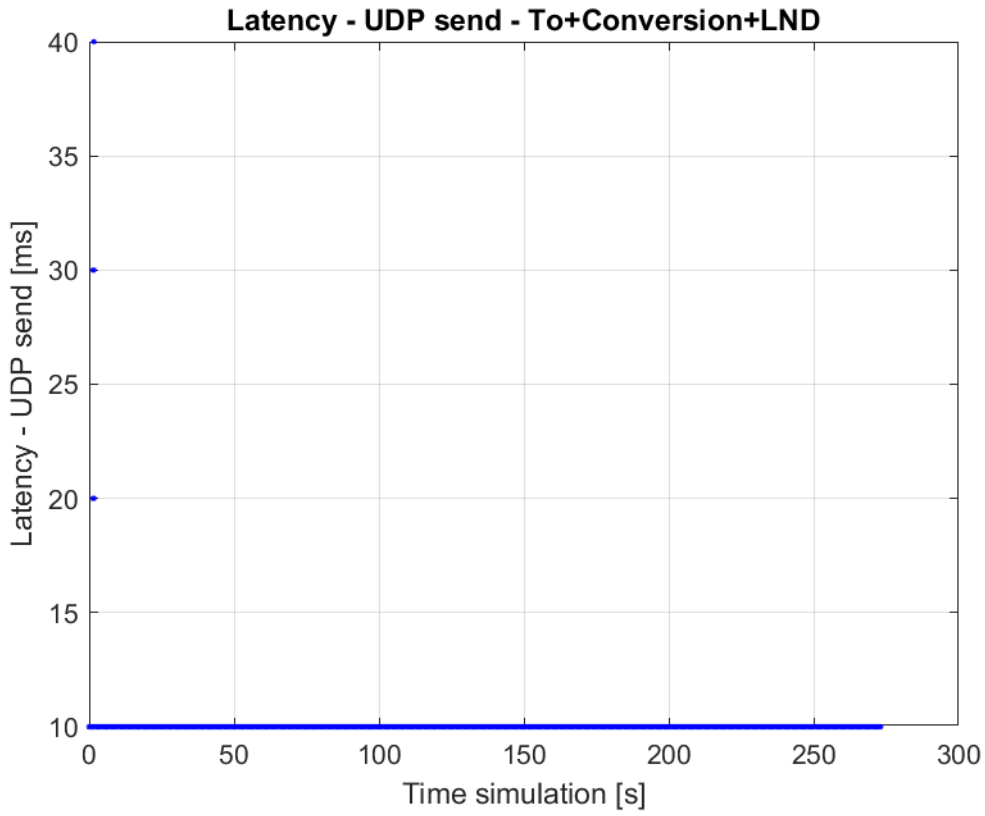

Figure 4.23: Conversion Test – UDP send Block Latency

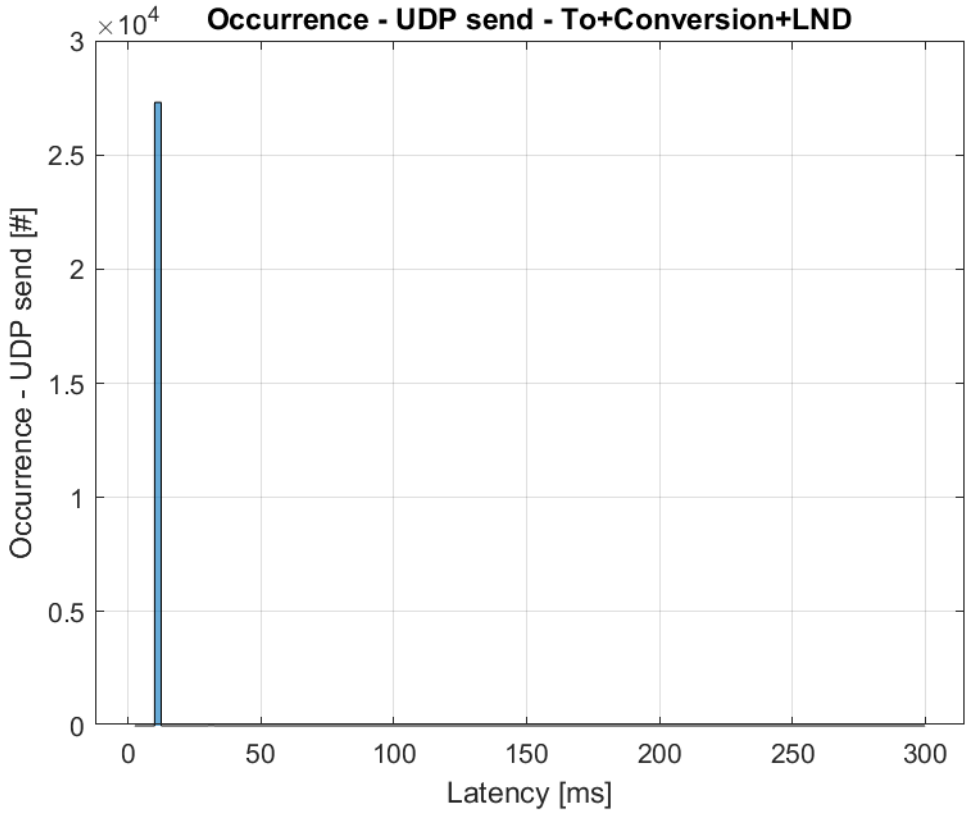

Figure 4.24: Conversion Test – UDP send Block Occurrence

From Fig. 4.19 and 4.20 it is possible to observe how, after a delay due to the start of the simulation, the average delay is null with sporadic peaks of 2.5 ms, equal to the integration step. These peaks present during the simulation can have different triggering factors, such as system processes that run in the background and require higher priority resources than the real-time kernel, or the loading of the scenario within FlightGear by means of the TerraSync utility. The utility in fact updates the terrain database dynamically during the simulation, requiring system resources to download data via the network and save them in memory. Constant access to the computer's hard disk would explain the more distributed delays in the latter test where, being a larger flight, a larger portion of the simulation environment is exploited and may require intensive use of TerraSync.

From Fig. 4.21 and 4.22 can be seen that the delay is constantly null, a sign that Simulink is able to read the information coming from FlightGear without problems.

Finally, Fig. 4.22 and 4.23 show that the delay in sending the information packet to FlightGear is constantly 10 ms, i.e. at the end of the sampling period set in the block.

5
Conclusions

5.1 Achievements

The purpose of this thesis work was to implement the mathematical model of simulation, previously produced on Simulink, in an graphical environment of simulation that consent the interaction with the system.

First of all the first obstacle has been to understand the operation of the mathematical model in its entirety, identifying the main logical connections and the assumptions and simplifications introduced in the models.

Once the Simulink model was studied, the attention moved to the understanding of FlightGear software with regard to the XML and Nasal programming languages that allow its operation and the transmission protocols that consent to interface with the external simulation models. In addition, the use of open-source software has allowed the learning and understanding of the methodologies of 3D reproduction of a graphic environment, the use of textures applied to three-dimensional models and the operation of a database for the reproduction of the terrain.

Being a real-time simulator, the main goal was to keep the latency of the simulation low. This has been possible thanks to the Simulink libraries correctly set up, that have allowed to maintain the maximum values of latency within those required by the normative for a certified simulator (see chapter 4).

The completeness of the Simulink libraries combined with the versatility of the FlightGear simulation environment has made this possible, not without difficulties related to the implementation of the 3D model of the aircraft, its animations, and interfaces between the two software including transmission protocols.

In conclusion the result is more than satisfactory and fulfills the main purpose of having a flight simulator of a tiltrotor ready to use fore the aerospace deprtment of Politecnico di Torino, for didactic and research purposes.

The writing of the thesis in a difficult moment like the global COVID19 pandemic has limited the author's possibilities, but in spite of everything a very important result has been achieved from a didactic point of view.

5.2 Future developments

This work provides a basis for further improvements that can be made to the simulation model, as it allows the Simulink code to be tested with in-the-loop pilot tests. In fact, performing tests with the pilot allows for standardized maneuvers and tests to be made, that empirically assess the validity and accuracy of the simulation, as well as the handling qualities and operation of the systems.

Several improvements could also be made in FlightGear, such as the actual reproduction of the XV-15 aircraft with a 3D model designed ad hoc, the addition of a realistic instrumentation that reproduces the one actually on board the XV15, but also the simulation of the internal systems of the aircraft.

As far as hardware is concerned, it would be necessary to integrate a lever collective control, if the aim was to better represent the XV15 aircraft operation, but if the simulation model remains generic for any type of tiltrotor, even a throttle collective control is correct for the simulator.

A Software Code

```
24    <!-- Attitude -->
25    <chunk>
26    <name>Roll</name>
27    <node>/orientation/roll-deg</node>
28    <factor>57.29577951</factor> <!-- converts from radians to deg -->
29    <type>double</type>
30    </chunk>

31    <chunk>
32    <name>Pitch</name>
33    <node>/orientation/pitch-deg</node>
34    <factor>57.29577951</factor> <!-- converts from radians to deg -->
35    <type>double</type>
36    </chunk>

37    <chunk>
38    <name>Yaw</name>
39    <node>/orientation/true-heading-deg</node>
40    <factor>57.29577951</factor> <!-- converts from radians to deg -->
41    <type>double</type>
42    </chunk>

43    <!-- Surface positions -->
44    <chunk>
45    <name>elevator-pos-norm</name>
46    <node>/surface-positions/elevator-pos-norm</node>
47    <type>double</type>
48    </chunk>

49    <chunk>
50    <name>Flap Pos [norm -1/1]</name>
51    <node>surface-positions/flap-pos-norm</node>
52    <type>double</type>
53    </chunk>

54    <chunk>
55    <name>right-aileron-pos-norm</name>
56    <node>surface-positions/right-aileron-pos-norm</node>
57    <type>double</type>
58    </chunk>

59    <chunk>
60    <name>left-aileron-pos-norm</name>
61    <node>surface-positions/left-aileron-pos-norm</node>
62    <type>double</type>
63    </chunk>

64    <chunk>
65    <name>rudder-pos-norm</name>
66    <node>/surface-positions/rudder-pos-norm</node>
67    <type>double</type>
68    </chunk>

69    <chunk>
70    <name>Tilt</name>
71    <node>/surface-positions/tilt</node>
72    <type>double</type>
73    </chunk>
```

Figure A.1: FlightGear UDP Protocol (2)

```
69    <!-- Engines -->
70    <chunk>
71    <name>RPM</name>
72    <node>/engines/engine[0]/rpm</node>
73    <type>double</type>
74    </chunk>

75    <!-- Landing Gear -->
76    <chunk>
77    <name>Gear Position</name>
78    <node>gear/gear[0]/position-norm</node>
79    <type>double</type>
80    </chunk>

81    <chunk>
82    <name>Gear compression front</name>
83    <node>gear/gear[0]/compression-norm</node>
84    <type>double</type>
85    </chunk>

86    <chunk>
87    <name>Gear compression left</name>
88    <node>gear/gear[1]/compression-norm</node>
89    <type>double</type>
90    </chunk>

91    <chunk>
92    <name>Gear compression right</name>
93    <node>gear/gear[2]/compression-norm</node>
94    <type>double</type>
95    </chunk>

96    <!-- Blade for sound -->
97    <!-- chunk>
98    <name>Blade flapping</name>
99    <node>/rotor/flapping</node>
100   <type>double</type>
101   </chunk -->

102   </input>


103   <output>

104   <binary_mode>true</binary_mode>
105   <!-- byte_order>host</byte_order --> <!-- host is default -->

106   <chunk>
107   <name>Ground elevation [m]</name>
108   <node>/position/ground-elev-m</node>
109   <type>double</type>
110   </chunk>

111   </output>

112   </generic>
113   </PropertyList>
```

Figure A.2: FlightGear UDP Protocol (3)

```
1  function [brake, flaps, lndGear, scasON] =
   buttonsToControls(buttons, brakeState, flapsState,
   lndState, scasState)
2  % Buttons
3  % buttons(1) - grilletto
4  % buttons(2) - zebrato
5  % buttons(3) - dorsale
6  % buttons(4) - bottone destro

7  % Blocks input until the button is released
8  persistent toggleFLAPS
9  if isempty(toggleFLAPS)
10 toggleFLAPS = 0;
11 end

12 persistent toggleGEAR
13 if isempty(toggleGEAR)
14 toggleGEAR = 0;
15 end

16 persistent toggleSCAS
17 if isempty(toggleSCAS)
18 toggleSCAS = 0;
19 end

20 %Use this if brake is ON/OFF
21 % brake = brakeState;
22 % if(buttons(1) == 1 && toggleBRAKE == 0)
23 %     if brakeState == 1
24 %         brake = 0;
25 %     elseif brakeState == 0
26 %         brake = 1;
27 %     end
28 %     toggleBRAKE = 1;
29 % end

30 %Use this if brake has to kept pressed
31 if(buttons(1) == 1)
32 brake = 1;
33 else
34 brake = 0;
35 end
```

Figure A.3: “buttonsToControls” MATLAB code (1)

```
flaps = flapsState;
if(buttons(4) == 1 && toggleFLAPS == 0)
flaps = flapsState + 0.2;
toggleFLAPS = 1;
end
if flaps > 1
flaps = 0;
end


lndGear = lndState;
if (buttons(2) == 1 && toggleGEAR == 0)
if lndState == 1
        i. lndGear = 0;
elseif lndState == 0
        i. lndGear = 1;
end
toggleGEAR = 1;
end


scasON = scasState;
if (buttons(3) == 1 && toggleSCAS == 0)
if scasState == 1
        i. scasON = 0;
elseif scasState == 0
        i. scasON = 1;
end
toggleSCAS = 1;
end


if buttons(4) == 0
toggleFLAPS = 0;
end
if buttons(2) == 0
toggleGEAR = 0;
end
if buttons(3) == 0
toggleSCAS = 0;
end


end
```

Figure A.4: “buttonsToControls” MATLAB code (2)

```
function NAC_req = nac_IP2req(NAC_IP, NAC_fb)

persistent toggle
if isempty(toggle)
toggle = 0;
end

NAC_req = NAC_fb;
if (NAC_IP == 1 && toggle==0) %NAC_IP up
NAC_req = NAC_fb + Par.FlightGear.NACstepLength; % +15;
toggle = 1;
elseif (NAC_IP == -1 && toggle==0 && NAC_fb == 95) %NAC_IP down
from 95
NAC_req = NAC_fb - 5;
toggle = 1;
elseif (NAC_IP == -1 && toggle==0) %NAC_IP down
NAC_req = NAC_fb - Par.FlightGear.NACstepLength; % -15;
toggle = 1;
end

if (NAC_IP ~= 1 && NAC_IP ~= -1)
toggle = 0;
end

if NAC_req < 0
NAC_req = 0;
elseif NAC_req > 95
NAC_req = 95;
end

end
```

Figure A.5: NAC input angle request function

```
function [CHUP, CHDN] = nac_DIFF2act(NAC_diff)

res = 0.1;
if NAC_diff < -res
CHUP = 1;
CHDN = 0;
elseif NAC_diff > res
CHUP = 0;
CHDN = 1;
else
CHUP = 0;
CHDN = 0;
end
```

Figure A.6: NAC difference to actuators signal function

LIST OF FIGURES

Figure 1.1: Organization of a Flight Simulator [1] 2
Figure 1.2: Main purposes of flight simulators 4
Figure 1.3: Main Advantages of flight simulators 5
Figure 1.4: Real-Time frames [1] 8
Figure 1.5: XV-22 Osprey in different Flying Modes [2] 9
Figure 1.6: Range Comparison Between Bell 525 and Boeing V-22 [3] 10
Figure 1.7: Bell XV-3 in hovering near ground 10
Figure 1.8: Bell XV-15 in hovering near ground 11
Figure 1.9: Bell V22 Osprey in airplane mode [3] 12
Figure 1.10: MV-22 and CH-46 Combat Radius Comparison [5] 13
Figure 1.11: Bell V-280 in airplane mode (wiki) 13
Figure 1.12: AgustaWestland AW609 in airplane mode [6] 14
Figure 1.13: A3 Vahana in airplane mode 15
Figure 2.1: XV15 Tilt-Rotor Model Structure 16
Figure 2.2: Mathematical Model complete Structure [7] 19
Figure 2.3: Aircraft Body Reference System 20
Figure 2.4: Rotor Axes System 21
Figure 2.5: Geometric Reference System 21
Figure 2.6: The Geoid definitions 22
Figure 2.7: Oblate spheroidal model of the Earth 23
Figure 2.8: ECI and ECEF reference systems [9] 24
Figure 2.9: Geodetic coordinates of a Point 25
Figure 2.10: NED and ECEF reference system 26
Figure 2.11: Relationship between the time domain and the frequency domain 29
Figure 2.12: A continuous function f(t) and its sampling f(k) [10] 32
Figure 2.13: Tustin method – trapezoidal integration [10] 32
Figure 2.14: Simulator architectures nodes 33
Figure 2.15: UDP Packet structure 35
Figure 3.1: Software architecture 36
Figure 3.2: FlightGear running screenshot 37
Figure 3.3: Aircraft Instrumentation 38
Figure 3.4: Nacelles angle gauge texture 38
Figure 3.5: Nacelles gauge XML code 39
Figure 3.6: TerraMaster interface 40
Figure 3.7: FlightGear Launcher Script 41

Figure 3.8: FlightGear UDP Protocol (1) 42
Figure 3.9: Overall Simulink Model 43
Figure 3.10: Normal Mode processes scheme 44
Figure 3.11: Accelerator Mode processes scheme 44
Figure 3.12: Rapid Accelerator Mode processes scheme 45
Figure 3.13: CS-FSTD Specifications for Flight Simulator Training Devices [12] 45
Figure 3.14: Simulation Pace Block from Aerospace Toolbox [13] 46
Figure 3.15: Real-Time Sync Block and its Parameters 46
Figure 3.16: Analog/Digital/Other Input blocks form SLDRT library 47
Figure 3.17: Analog Input Block parameters 48
Figure 3.18: Packet data interface blocks from SLDRT library 49
Figure 3.19: Packet Output and Packet Input blocks Parameters 49
Figure 3.20: ThrustMaster USB Joystick Simulink Interface 50
Figure 3.21: “buttons to controls” function detail 51
Figure 3.22: Algebraic Loop example in Simulink 52
Figure 3.23: “POV to controls” function detail 52
Figure 3.24: “POVctrl” MATLAB code 52
Figure 3.25: Flight Gear Interface in Simulink 53
Figure 3.26: State Evaluation block detail 53
Figure 3.27: Flat Earth to LLA block from Aerospace Toolbox [15] 54
Figure 3.28: Pilot Interface Block 55
Figure 3.29: Normalization block - “IP XV-15 [norm]” detail 56
Figure 3.30: NAC to RPM logic 57
Figure 3.31: NAC control logic – “Stick Switches” block detail 58
Figure 3.32: Simulink Primary Flight Display (PFD) Block detail 59
Figure 3.33: Dell® Precision 7550 60
Figure 3.34: Thrustmaster® T-16000 60
Figure 3.35: Analog Input values demo 61
Figure 3.36: CH Products Pro Pedals 61
Figure 4.1: Longitudinal Translation Test – Position in time 63
Figure 4.2: Longitudinal Translation Test – Attitude in time 63
Figure 4.3: Longitudinal Translation Test – SYNC Block Latency 64
Figure 4.4: Longitudinal Translation Test – SYNC Block Occurrence 64
Figure 4.5: Longitudinal Translation Test – UDP rec Latency 65
Figure 4.6: Longitudinal Translation Test – UDP rec Occurrence 65
Figure 4.7: Longitudinal Translation Test – UDP send Latency 66
Figure 4.8: Longitudinal Translation Test – UDP send Occurrence 66
Figure 4.9: Hovering Test – Position in time 67
Figure 4.10: Hovering Test – Attitude in time 68
Figure 4.11: Hovering Test – SYNC Block Latency 68
Figure 4.12: Hovering Test – SYNC Block Occurrence 69

Figure 4.13: Hovering Test – UDP rec Block Latency 69
Figure 4.14: Hovering Test – UDP rec Block Occurrence 70
Figure 4.15: Hovering Test – UDP send Block Latency 70
Figure 4.16: Hovering Test – UDP send Block Occurrence 71
Figure 4.17: Conversion Test – Position in time 72
Figure 4.18: Conversion Test – Attitude in time 72
Figure 4.19: Conversion Test – SYNC Block Latency 73
Figure 4.20: Conversion Test – SYNC Block Occurrence 73
Figure 4.21: Conversion Test – UDP rec Block Latency 74
Figure 4.22: Conversion Test – UDP rec Block Occurrence 74
Figure 4.23: Conversion Test – UDP send Block Latency 75
Figure 4.24: Conversion Test – UDP send Block Occurrence 75

Figure A.1: FlightGear UDP Protocol (2) 79
Figure A.2: FlightGear UDP Protocol (3) 80
Figure A.3: “buttonsToControls” MATLAB code (1) 81
Figure A.4: “buttonsToControls” MATLAB code (2) 82
Figure A.5: NAC input angle request function 83
Figure A.6: NAC difference to actuators signal function 83

LIST OF TABLES

Table 3.1: Main Controls Ranges 55
Table 3.2: Secondary Controls Ranges 55

BIBLIOGRAPHY

[1] D. Allerton, Principles of Flight Simulation, John Wiley & Sons, 2009.

[2] D. Harris Franklin, Introduction to Autogyros, Helicopters, and Other V/STOL, NASA, 2015.

[3] Bell Textron Inc., [Online]. Available: https://www.bellflight.com/. [Accessed 9 September 2021].

[4] S. W. Ferguson, *A Mathematical Model for Real Time Flight Simulation of a Generic Tilt-Rotor Aircraft,* 1988.

[5] GAO, «Assessments Needed to Address V-22 Aircraft Operational and Cost Concerns to Define Future Investments,» *Report to Congressional Requesters,* 2009.

[6] Leonardo S.p.A., "Leonardo Products," [Online]. Available: https://www.leonardocompany.com/it/products/aw609. [Accessed 11 September 2021].

[7] A. Abà, *MA Thesis, Implementation of a comprehensive real-time simulation model of a tilt-rotor aircraft.*

[8] F. Lleshi, *Master Degree Thesis, Validation of a XV-15 Tilt Rotor Aerodynamic Database.*

[9] M. S. Grewal, L. R. Weill and A. Andrews, "Global Positioning Systems, Inertial Navigation, and Integration," II ed., Wiley-Interscience, 2007.

[10] G. F. Franklin, J. D. Powell and A. Emami-Naeini, Feedback Control of Dynamic Systems, VII ed., Pearson, 2014.

[11] "Flight Gear Info," [Online]. Available: https://www.flightgear.org/. [Accessed 26 August 2021].

[12] EASA, *Certification Specifications for Airplane Flight Simulation Training Devices, CS-FSTD(A).*

[13] "MathWorks Help Center, Aerospace Blockset," [Online]. Available: https://it.mathworks.com/help/aeroblks/simulationpace.html?searchHighlight=simulation%20pace&s_tid=srchtitle. [Accessed 27 August 2021].

[14] "MathWorks Help Center, Algebraic Loops," [Online]. Available: https://it.mathworks.com/help/simulink/ug/algebraic-loops.html. [Accessed 06 09 2021].

[15] "MathWorks Help Center, Flat Earth to LLA," [Online]. Available: https://it.mathworks.com/help/aeroblks/flatearthtolla.html. [Accessed 6 September 2021].

[16] MathWorks, *Simulink® Desktop Real-Time™ User's Guide 2021a.*

[17] “MathWorks Help Center, How Acceleration Modes Work,” [Online]. Available: https://it.mathworks.com/help/simulink/ug/how-the-acceleration-modes-work.html. [Accessed 6 September 2021].

[18] MathWorks, *Simulink® Desktop Real-Time™ Reference 2021a.*

[19] NASA, *XV-15 Manual,* Moffet Field, California, 1975.

[20] D. M. Maisel, J. D. Giulianetti and C. D. Dugan, The History of the XV-15 Tilt Rotor Research Aircraft, National Aeronautics and Space Administration, 2000.

[21] S. Godio, *MA Thesis, Multi-purpose rotor model for a real-time flight simulator,* 2019.

[22] F. Barra, *MA Thesis, Development of a tilt-rotor model for real-time flight simulation,* 2018.

[23] M. R. Spiegel, Schaum's Outline of Laplace Transforms, McGraw-Hill, 1965.

[24] B. L. Stevens and F. L. Lewis, Aircraft Control and Simulation, Hoboken, New Jersey: Jhon Wiley & Sons, 2003.

[25] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach, VII ed., Pearson, 2017.

[26] G. Seeber, Satellite Geodesy, II ed., New York: Walter de Gruyter, 2003.

ACKNOWLEDGMENTS

Il percorso per il conseguimento della laurea è stato lungo faticoso, fatto di grandi soddisfazioni, ma anche grandi difficoltà. La distanza, lo studio, le responsabilità personali sono stati scogli da superare per la mia crescita professionale ma soprattutto per la mia crescita umana.

Sarebbe stato impossibile portarla a termine senza il supporto della mia famiglia, dei miei genitori e fratelli, degli amici tutti, che in questi anni sono diventati la mia famiglia e il mio punto di riferimento.

Un sentito ringraziamento va alla mia fidanzata Ambra, che mi ha trovato alla deriva nel mare della vita e mi ha insegnato a guardarmi intorno per capire che non ero solo. Mi ha sempre dato l'affetto di cui avevo bisogno, nonostante le mie paure, le mie preoccupazioni, ma anche la forza e la determinazione per non fermarmi mai.

Ringrazio il professor Giorgio Guglieri, per la possibilità di aver approfondito un argomento come la simulazione del volo che spero possa diventare il mio lavoro. Un ringraziamento va anche al dottor Federico Barra, per la disponibilità e l'aiuto nella stesura di questa tesi.

The path to graduation was long and tiring, made of great satisfaction, but also great difficulties. The distance, the study, the personal responsibilities have been obstacles to overcome for my professional growth but especially for my human growth.

It would have been impossible to complete it without the support of my family, my parents, and brothers, and all my friends, who over the years have become my family and my point of reference.

A heartfelt thanks goes to my fiancée Ambra, who found me adrift in the sea of life and taught me to look around me to understand that I was not alone. She always gave me the affection I needed, despite my fears, my worries, but also strength and determination to never stop.

I would like to thank Professor Giorgio Guglieri, for the opportunity to have deepened a topic such as flight simulation that I hope will become my job. Thanks also to Dr. Federico Barra, for his kindness and help in writing this thesis.