MASTER THESIS

TITLE: Realistic VTOL simulator

MASTER DEGREE: Master's degree in Applications and Technologies for Unmanned Aircraft Systems (Drones) (MED)

AUTHOR: Sergi Lucas Millan

PROFESSIONAL ADVISOR: Miquel Mulet Autonell

ACADEMIC ADVISOR: Jose Ignacio Rojas Gregorio

DATE: October 9, 2020

Abstract

This master's thesis is focused on the development of a VTOL drone flight simulator. Two main objectives have been set. The first one is that the simulator must simulate all the flight phases of a VTOL. To make it possible, the simulator software is integrated with the simulation of the drone flight controller, which can be ArduPilot or PX4. The second objective is that it should be possible to control the simulated UAV via radio control, in the same way that we would do with a real drone. The thesis is structured in four chapters.

In the first chapter, we do a study of the different types of VTOL drones. There are mainly three types: tailsitters, tiltrotors, and QuadPlanes. The three flight phases of a VTOL (vertical take-off and landing, transition and horizontal flight) are also studied. The advantages of a VTOL over a fixed-wing and a multirotor are studied. Finally, we analyse the characteristics of the VTOL that Venturi is developing, called V1.

The second chapter summarizes the European laws that affect drones. There are currently two laws: Delegated Regulation 2019/945 and Implementing Regulation 2019/947. The first regulation classifies drones into five classes, according to their capabilities and characteristics. The second regulation deals with the rules and procedures that drones must fulfil. Operations are classified into three categories: open, specific and certified.

In the third chapter, a study of the software that is being used for simulation of UAVs is done. The pros and cons of each option are analysed. In view of this study, Gazebo, a robot simulation environment, is chosen for this project.

Finally, the last chapter explains the structure of the software that has been developed to carry out the desired simulation. Then, to test this simulator, tests are done with two different QuadPlane models: a model designed by Gazebo and a model of the Venturi V1. Three tests are performed for each of these QuadPlane models: 1) a test of the rotation of the engines and movement of the control surfaces; 2) a test in which the UAV must follow a pre-planned mission; and 3) a test in which we try to control the drone with a joystick.

For the first UAV model, the three tests are satisfactory; in particular, the computed average error when following the planned mission is 1.9 m. Moreover, in the joystick control test, the drone responds perfectly to the controls, just like a real VTOL. For the Venturi V1, the first test is satisfactory, but unfortunately the second and third tests cannot be carried out, likely due to an error in the Gazebo model of the V1.

As a result of this project, the developed simulator is being integrated in Venturi with a computer vision system for detection of pedestrians, to make safer landings, and for detection of the power lines, so that the UAV can follow them autonomously during power line inspection missions.

INDEX

FIGURE INDEX

INTRODUCTION

A Vertical Take-Off and Landing (VTOL) drone is a drone that is capable of taking off and landing vertically, while in the other flight phases it evolves as planes do. So, we could say that a VTOL drone combines the operation of a quadrotor, for take-off and landing, with the operation of a conventional airplane, for the other flight phases.

Venturi Unmanned Technologies is a company that was born in 2015. Since then, it has been dedicated to the construction of a VTOL drone to carry out inspection tasks of electrical networks and gas pipelines. This Master's Thesis has been developed during an internship of the author at Venturi.

1. Motivation

The origin of this thesis is caused by one of the needs that Venturi has from the first years of work. The main drawback of all companies related to the world of unmanned aerial vehicles (UAV) is the European legislation on airspace in relation to drones.

The European legislation is currently very restrictive regarding the use of UAV, because they have to share airspace with aircraft and the technology needed to ensure the separation between drones and aircraft is not yet sufficiently developed, nor the protocol to be followed in order to fly a drone in controlled airspace. In addition, the legislation is more restrictive the larger and heavier the UAV. Obviously, work is being done to integrate drones into this airspace. The Single European Sky Air Traffic Management Research Joint Undertaking (SESAR JU) has created the U-Space proposal, which is a European Union (EU) project for the integration of drones in all European airspace.

Venturi’s drone is a VTOL with a 3 m wingspan and a 25 kg Maximum Take-Off Weight (MTOW). Given these characteristics and the current legislation, it is very difficult to fly this drone in Spain, you have to be given special conditions and fly in a segregated airspace. These conditions are far from airports and areas with dense air traffic. Therefore, every time Venturi wants to fly the drone, they have to move far away from the offices in Castelldefels, since these are close to Barcelona-El Prat airport, which generates a very high temporal and economic cost.

Because of this, the motivation to develop a VTOL drone flight simulator in Venturi arose. In this way, it will be possible to implement new flight configurations, simulate missions and see the behaviour of the aircraft in a simulated environment without having the expenses of transporting the drone to the actual flight site and renting the segregated airspace in which drone flights are performed.

2. Goals

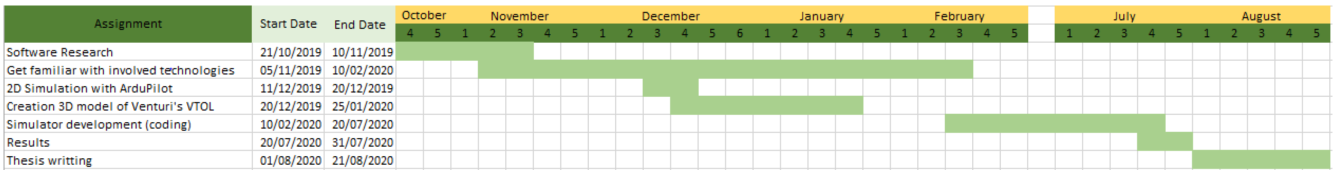
Once we know the motivation for developing a drone simulator, we need to be clear about what the simulator's goals will be.

The objectives of the simulator will mainly be two. The first and most important goal is that the simulator is able to simulate the flight dynamics of a VTOL during vertical take-off, transition vertical-to-horizontal flight, horizontal flight, transition horizontal-to-vertical flight, and finally vertical landing. Along with the flight dynamics, the simulator must also allow integration with the actual aircraft flight controller software. In the case of Venturi, at the moment, they are using ArduPilot. Therefore, the simulator must be able to understand the commands generated by ArduPilot [1] and above all, understand and execute the missions that can be created through a ground station.

The second goal of the simulator is to be able to understand the commands generated by a radio control. In this way, if the simulated drone is able to execute the commands sent to it through a radio control, it will be possible to carry out trainings for future pilots of the company. Thus, novice pilots will be able to have a first contact with a VTOL in a simulated environment. They will be able to learn the different modes of flight, without endangering the actual aircraft.

3. Time planning

Due to the scale of the project, a Gantt chart was developed to establish efficient and orderly time planning, as shown in Fig. 0.1.



Assignment	Start Date	End Date	October		November					December						January					February						July					August				
			4	5	1	2	3	4	5	1	2	3	4	5	6	1	2	3	4	5	1	2	3	4	5		1	2	3	4	5	1	2	3	4	5
Software Research	21/10/2019	10/11/2019	🟩	🟩	🟩	🟩	🟩																													
Get familiar with involved technologies	05/11/2019	10/02/2020				🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩													
2D Simulation with ArduPilot	11/12/2019	20/12/2019										🟩	🟩																							
Creation 3D model of Venturi's VTOL	20/12/2019	25/01/2020											🟩	🟩	🟩	🟩	🟩	🟩	🟩																	
Simulator development (coding)	10/02/2020	20/07/2020																					🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩							
Results	20/07/2020	31/07/2020																												🟩	🟩					
Thesis writting	01/08/2020	21/08/2020																														🟩	🟩	🟩	🟩	🟩

Fig. 0.1 Gant chart showing the time planning of the Master's Thesis project

As can be seen in the Gantt Chart, there was extensive work to do prior to actually starting to develop the simulator, as the technologies used in the work have a very long learning curve and therefore it was necessary to learn how to use these tools well.

Once this learning time passed, almost 5 months were spent developing the designed code.

CHAPTER 1. VTOL

VTOL is the acronym for Vertical Take Off and Landing. Thus, a VTOL drone arises from the combination of a multirotor with a fixed wing aircraft, combining the best features of each of these types of aircraft: it takes off and lands like a helicopter and flies like a plane.

VTOLs use rotors to generate lift (the force that balances weight and keeps the aircraft flying in the air) and thrust (the force that propels the aircraft forward) during the take-off and landing phases. During the horizontal flight phase, the rotors only generate thrust, while the lift is generated by the vehicle's wings.

For a better understanding of VTOLs, we will classify the different types of VTOLs that exist today and the different phases of flight for each type of VTOLs, and then we will analyse the advantages and disadvantages of using a VTOL.

Finally, we will analyse the VTOL used by Venturi and its characteristics.

1.1 Types of VTOLs

There are currently three types of VTOLs: tailsitters, tiltrotors and QuadPlanes. Each type has the same common denominator, which is vertical take-off and landing capability, but the configuration and operation of the engines are different in each case [2].

1.1.1 Tailsitter

Tailsitters, as their name suggests, are drones that usually sit on their tail for take-off and landing (see Fig. 1.1). Not all tailsitters land on their tail, as it is a very tricky manoeuvre, and thus there are some that land on the belly of the vehicle. Once they take off, the entire drone tilts forward to begin horizontal flight. You can see a video of the operation of this type of drone at the following link. This is the only VTOL type that has a vertical orientation, the rest have a horizontal orientation.

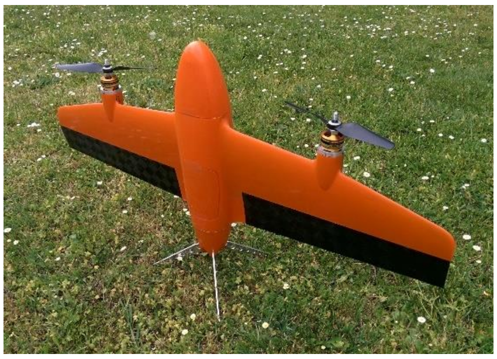


Fig. 1.1 Tailsitter drone

Typically, tailsitters can have two, three, or four rotors, and have two control surfaces. These control surfaces are elevons, which are responsible for changing the orientation of the drone during take-off and landing of the vehicle, and manoeuvrability during flight. Elevons are control surfaces that combine the functions of ailerons and elevators. Ailerons are used for roll control and elevators are used for pitch control. So, elevons are used for roll and pitch control. Fig. 1.2 shows the Euler angles and axes around which aircraft can rotate.

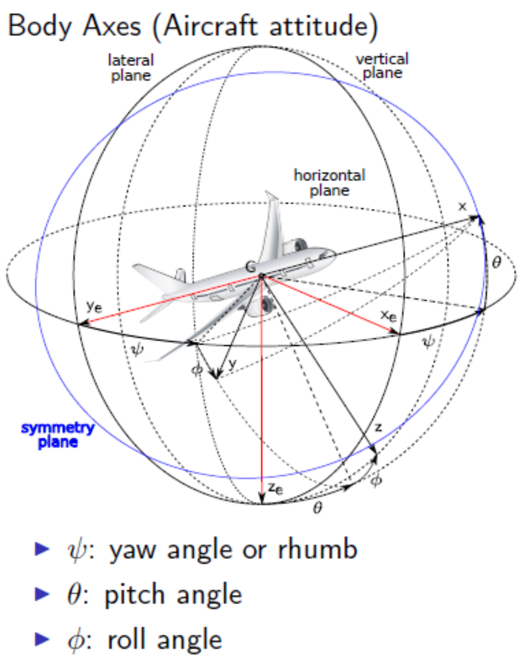


Fig. 1.2 Earth fixed axes, body axes and Euler angles for aircraft

In addition to having different numbers of engines, tailsitters may have different configurations; for instance, having four engines can vary their configuration with respect to the fuselage: they can have an x or + configuration, in the same way as multicopters (see Fig. 1.3).

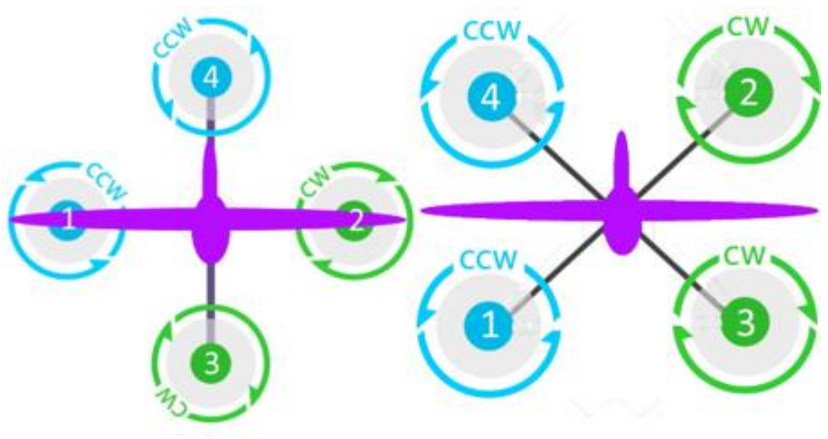


Fig. 1.3 Different frame configuration for tailsitters: + configuration (left) and x configuration (right)

Lastly, tailsitters may have a directional thrust or not; this will depend on whether the rotors can be tilted or not. On one hand, tailsitters with directional thrust will be better as they will be more stable, that is, they will be able to find balance more easily. On the other hand, those tailsitters which do not have directional thrust must have the centre of mass well located, as in the event that the centre is displaced to the back or shoulder of the vehicle, the drone might have stability problems during take-off [3].

The stability of a body depends on the position of its centre of gravity and the size of its base. If the centre of gravity falls outside the base of the body, the object becomes unstable. Normally all tailsitters have a very narrow body, therefore it is very easy for the centre of gravity to fall outside its base and so become unstable. In the case of a tailsitter with four rotors or with vectored thrust, there are no problems as they can compensate the swinging caused by instability. In the case of a tailsitter with two non-vectored throttles, it is not able to cope with this instability. For this reason, it is necessary that the centre of gravity of a tailsitter is well positioned.

1.1.2 Tiltrotors

For the tiltrotors, like for the tailsitters, their name indicates their operation: the rotors tilt to move from take-off configuration to horizontal flight configuration, and to move from horizontal flight configuration to landing configuration. You can see a video of the operation of this type of drone at the following link. Tiltrotors are vehicles that land and take off horizontally, on their legs or belly (see Fig. 1.4).

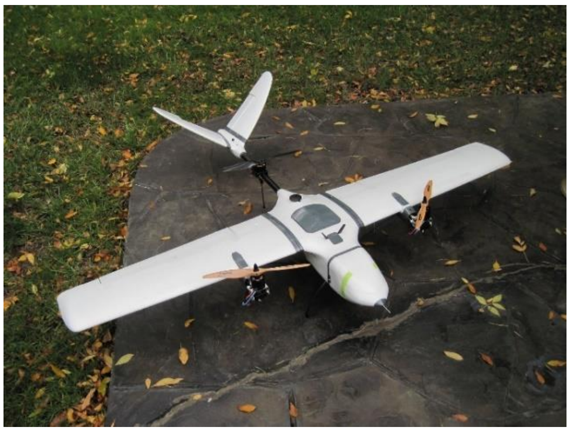


Fig. 1.4 Tilt rotor drone

Tilt rotors have a wide variety of configurations, in terms of the number of engines, their position, the number of engines that are tilted and whether the wings are tilted or not. They can usually have three, four or six engines. As for quadrotors, in the case that the drone has four rotors, the distribution can be in the form + or x. Also, depending on the number of rotors and their position, not all of them may have the option of being tilted. There are some configurations in which the engines are placed on the wings and therefore these also tilt. This type of VTOLs is called tilt wing. An example of tilt wing is shown in Fig. 1.5.

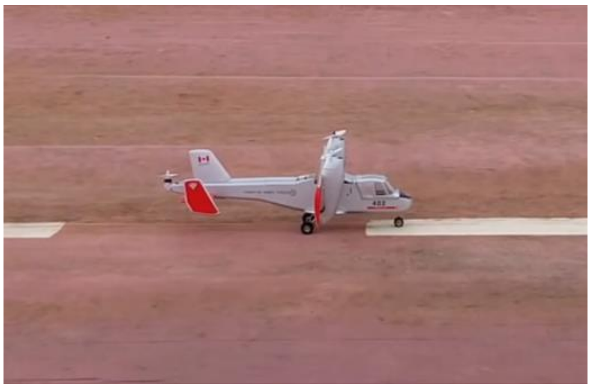


Fig. 1.5 Tilt wing drone

In order to control the direction of the rotors, servos are used, which can be continuous or binary. If the servos are binary, the rotors will only have vertical or horizontal position, that is, they cannot remain in an intermediate position. Obviously, continuous servos are better because they allow tilting the rotors to any position in a given range of positions. There are some configurations in which the servos allow the thrust to be directional during horizontal flight; this allows for better control of the rotations of drones around their yaw axis [4].

Depending on the configuration, tiltrotors can have different control surfaces to control the flight (see Table 1.1). In the case of a tiltrotor drone without tail, the control surfaces are two elevons. In the case that the tiltrotor has a tail, there are more control surfaces. The primary control surfaces in the wings are ailerons. The tail can be V-shaped, A-shaped, or the standard T-shaped tail. In the case of a standard tail, the drone has an elevator and a yaw rudder. In the case of a V- or A-shaped tail, the drone has two control surfaces called ruddervators, which perform the combined functions of both a rudder and an elevator.

Table 1.1 Movement of the control surfaces to control tiltrotors

	Tilt rotor without tail	**Tilt rotor with tail**
Pitch up	Both elevons move up	Both tail surfaces move up
Pitch down	Both elevons move down	Both tail surfaces move down
Roll right	Right elevon moves down and left elevon moves up	Right aileron moves up and left aileron moves down
Roll left	Right elevon moves up and left elevon moves down	Right aileron moves down and left aileron moves up
Yaw right		Both tail surfaces move right
Yaw left		Both tail surfaces move left

1.1.3 QuadPlanes

A QuadPlane is a combination of a fixed wing aircraft with a multirotor (see Fig. 1.6). It combines vertically oriented engines, a horizontally oriented engine and the typical control surfaces of fixed wing aircraft. Like tiltrotors, QuadPlanes take off and land in horizontal orientation, on the belly or legs of the vehicle.

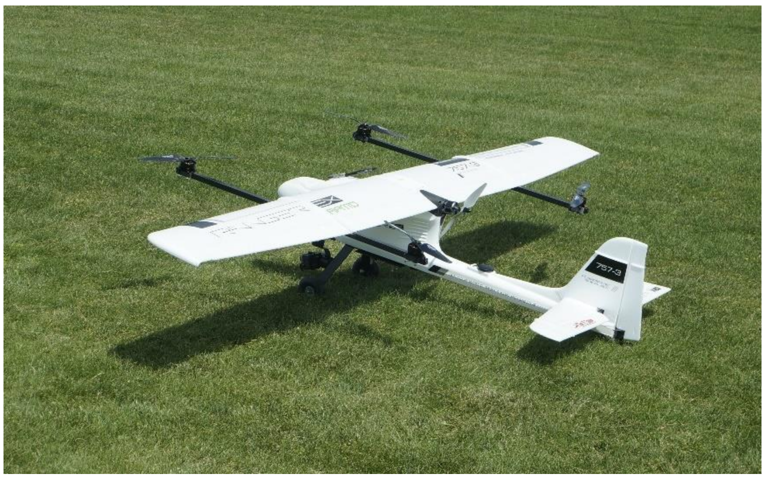


Fig. 1.6 QuadPlane

QuadPlanes have a simpler configuration than tiltrotors in terms of the number of engines and their position: four or eight engines are used for vertical propulsion. The distribution of the engines in both cases can be H, x or + with respect to the fuselage. For horizontal flight, QuadPlanes only have one engine for propulsion, as they have wings, which are responsible for the lift.

As for tiltrotors, there are many configurations in terms of control and manoeuvrability of the vehicle and therefore different control surfaces can be selected. If the UAV has a tail, then it has two ailerons on the wings and two control surfaces on the tail. The control surfaces of the tail depend on the shape. In the case of a V- or A-shaped tail, the control surfaces are called ruddervators. In the case of a standard T-shaped tail, the contour surfaces are rudder and elevator. If the QuadPlane has no tail, the wing control surfaces are elevons.

1.2 Flight phases

Unlike other UAVs, VTOLs fly in three very different flight phases, where the rotors and control surfaces behave very differently. These phases of flight are, take-off and landing, transition and horizontal flight.

1.2.1 Take-off and landing

In this flight phase, the UAV operates in the same way as a multicopter. It generates thrust and lift with its engines, and maintains stability only thanks to them. In the case of a tailsitter, this phase can be more delicate than for tiltrotors and quadplanes if it has only two rotors, since it will have to have the centre of gravity very well positioned. Otherwise, it will be impossible to maintain the stability with only two engines.

It is one of the flight phases where the vehicle consumes most energy, as all or most of the vehicle’s engines are working at full throttle, or close, to ascend or descend vertically, and/or maintain position.

1.2.2 Transition

The transition is the most important and critical phase of the flight of a VTOL. It is the phase that allows the VTOL to move from vertical flight to horizontal flight and vice versa. Most VTOL accidents happen at this phase due to its difficulty. The transition from horizontal to vertical flight is more critical due to the inertia of the vehicle.

The behaviour and operation of the drone varies depending on the drone type. In the case of a tailsitter, at this stage, thanks to the control surfaces of the vehicle, it tilts to move from the vertical to the horizontal position, or in the case of the landing, from horizontal to vertical.

In the case of a tiltrotor, the transition involves the tilt of the engines to move from a vertical flight to a horizontal flight, or vice versa, for the landing. In order to shift from vertical to horizontal flight, the servos move the rotors such that their axes are in horizontal position and the speed of the rotors decreases. The reason is that, since horizontal speed is gained, the wing of the vehicle begins to generate lift and therefore engines do not need to generate lift anymore. For the landing, the procedure is the opposite: the rotors begin to tilt their axes towards vertical position and to spin faster, since the horizontal speed of the vehicle is decreasing and therefore the wing does not generate as much lift and the rotors must compensate for this.

In the case of a QuadPlane, the operation of the engines is simpler: during the take-off, the rotors are used with vertical orientation; to shift to horizontal flight, the propulsive rotor begins to turn and the vehicle begins to gain speed; with this speed, the wings and body of the VTOL begin to generate lift and therefore the vertical rotors do not need to spin so fast. When the wings generate enough lift, the vertical rotors stop rotating and only the propulsion rotor works.

The transition phase must be very well set up. Due to its difficulty many vehicles lose some height in the transition.

1.2.3 Horizontal flight

In this flight phase, the drone flies in the same way as a fixed-wing UAV (see Fig. 1.7). In horizontal flight, the UAV is controlled by its control surfaces. In the case of a tailsitter, it is controlled with the elevons. In the case of a tiltrotor or a

QuadPlane, they usually have more control surfaces: elevons on tail-less UAVs and ailerons and tail control surfaces for UAVs with tail.

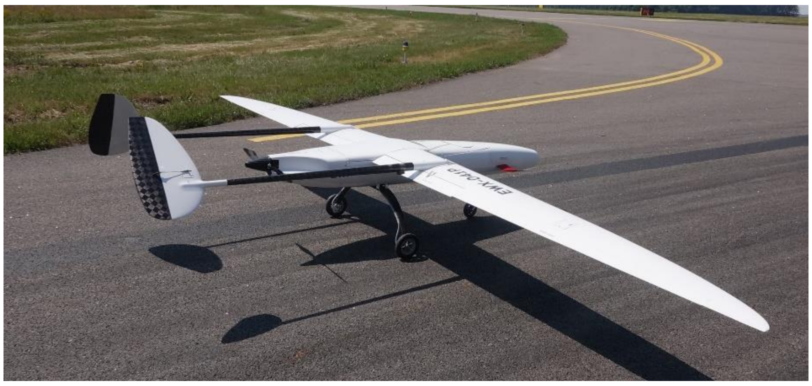


Fig. 1.7 Fixed-wing UAV

During the horizontal flight phase, depending on the type of VTOL, not all the engines might work. In the case of a tailsitter, as the engines go in the same direction as the fuselage, they will all work. In the case of a tiltrotor, only those engines that are inclined to the horizontal position will work. Depending on their configuration, there may be engines that are only used for take-off and landing. In the case of a QuadPlane, only the propulsion engine will work, as the rest are oriented vertically.

1.3 Advantages and disadvantages

Because a VTOL is a combination of a multicopter and a fixed wing aircraft, it has the advantages of each vehicle and is therefore more competent when compared against a pure multirotor or a pure fixed wing aircraft. To see the advantages and disadvantages, we will compare a VTOL with the latter vehicles. We will start by analysing the advantages and disadvantages of VTOLs with respect to a fixed wing aircraft and then with respect to a multirotor.

The biggest advantage of a VTOL with respect to a fixed wing aircraft is of course that it has vertical take-off and landing capabilities, just like multirotors do, so this makes it easy to start flying or landing anywhere, without the need of a runway. In addition, these types of take-offs and landings are safer, as the inertia and horizontal speed of the vehicle is not as large as for a fixed wing.

Because it is capable of operating as a multirotor, a VTOL can make hovering, while a fixed wing aircraft cannot, so a VTOL will be able to perform inspection and monitoring tasks where it is necessary to maintain a steady position in the air, that is, where hovering is necessary, as hovering refers to the action of staying in the same position in the air, in location and height.

In addition, due to the vertical movement, the VTOL will be able to adapt better to a change in the environment. A fixed wing aircraft will not be able to land if the conditions of the environment in which it flies vary greatly, whereas a VTOL could be able, as it needs little space to land. Finally, a VTOL is more manoeuvrable than a fixed wing thanks to the ability to vary the relative speed of each rotor, which will create changes in thrust and torque.

The advantages of a VTOL over a multirotor are three: the larger payload capacity, range (maximum flight distance) and/or endurance (maximum flight time). This is thanks to the lift generated by the wings of the VTOL during horizontal flight. Because in horizontal flight the lift is generated with the wings of the UAV, some VTOL engines will stop working, so the batteries will last longer and the range and/or of operation will be greater.

The only downside to VTOLs is that they tend to be less manoeuvrable than multirotors in air hovering.

With respect to a fixed wing aircraft, the VTOLs have less range of operation, since in the phases of take-off, landing and transition, the wings of the VTOL do not generate all the lift and, therefore, the rotors must do it. This causes a significant consumption of energy.

Another drawback of a VTOL in front of multirotors and fixed wing aircraft is that VTOLs are less reliable, because the configuration of a VTOL is much more complex, as many more movable and mechanical elements are involved (combining rotors, servos and control surfaces), and therefore the probability that a component fails is greater.

1.4 Venturi VTOL

Venturi intends to inspect power lines and pipelines with its drone. To do so, it must carry a set of cameras and sensors safely. Due to the size of the network of power lines and pipelines, a UAV is needed that has the largest possible operating range. To achieve this goal, the UAV must fly at a high altitude constantly and at a high speed. Due to these conditions, Venturi opted for the construction of a VTOL, specifically of the QuadPlane type.

Venturi’s QuadPlane (see Fig. 1.8) is designed entirely by the company, using 3D modelling software. The commercial computer-aided design (CAD) software used to design the model from scratch is SolidWorks from Dassault Systèmes SolidWorks Corp., Waltham, MA (USA).

This first VTOL designed by Venturi is called V1. It is built of carbon fibre reinforced epoxy (CFRE) as this composite material has very good mechanical properties. CFRE weighs little and has very high strength, compared with some aluminium alloys, for example.

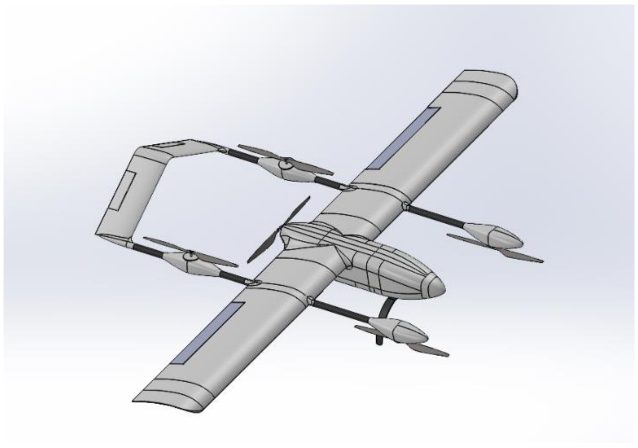


Fig. 1.8 V1 model on SolidWorks

As a QuadPlane, Venturi's V1 has four engines in charge of vertical propulsion and one engine in charge of providing thrust for horizontal flight. It also has four control surfaces, two ailerons in the wings for balance/bank control and two ruddervators in the tail for the pitch control. The tail of the V1 has an inverted V shape; this type of tail is called A-tail (it has an A shape). The configuration of the fuselage and engines for vertical propulsion has the H-shape.

The V1 has a special feature and this is that it has engines that are used for slightly inclined take-off and landing. With this inclination, the aim is to have more control over the yaw of the UAV [5]. Because the distribution of the rotors is H-shaped, the inclination of the motors is only two or three degrees.

It has been designed with a 3 m wing span configuration.

In order to see that the design of the V1 has been transformed into a real UAV, an image of the V1 is shown in Fig. 1.9, where the V1 can be seen in the airfield before performing an operation.

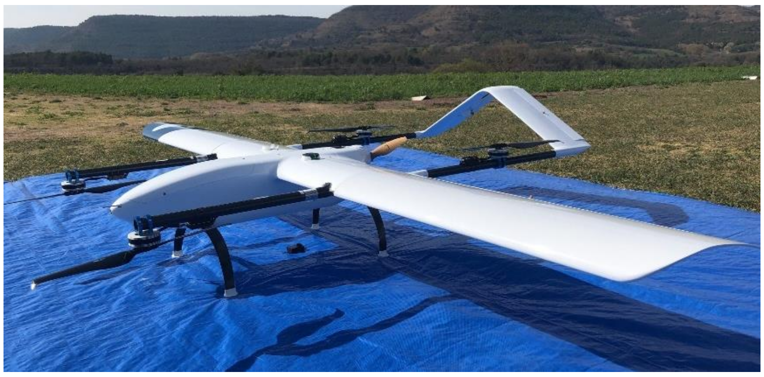


Fig. 1.9 Real V1 before operation

The total mass of the V1 is around 21-22 kg, so it is estimated that the payload it can carry is around 3-4 kg. It has a cruising speed of up to 70 km/h (19.44 m/s) and a coverage range of up to 110 km, but of course the maximum speed and range will depend on the payload carried by the UAV.

The V1 can work in conditions of wind speeds up to 36 km/h (10 m/s) in all directions, in quad and cruise mode and in a temperature range between 0 and 40ºC. Work is currently underway to make the UAV hydrogen-powered, which will increase the distance it can travel.

CHAPTER 2. EUROPEAN LEGISLATION

As explained in the introduction, one of the motivations for making a VTOL simulator for Venturi is the current EU legislation on drones. In order to understand how restrictive is the current legislation, and where stands the drone designed by Venturi, I will explain the current legislation.

The European Commission has currently published two delegated regulations on unmanned aerial systems (UAS) or drones. These are the “Delegated Regulation (EU) 2019/945” of the Commission of 12 March 2019 and the “Implementing Regulation (EU) 2019/947” of the Commission of 24 May 2019.

2.1 Delegated Regulation 2019/945

The European Commission published on March 12, 2019, this law that deals with the requirements of the UAS and the requirements that must be met by UAS manufacturers, designers, importers and distributors to obtain a mark of conformity and to control the security of the market and competitiveness of the same [6].

This law divides the UAS into five different classes, from C0 to C4, according to the characteristics of each UAS. If a manufacturer wants to build a UAS of a specific class, it must meet all the characteristics set for that class.

2.1.1 C0 Class

This class encompasses the smallest drones. The Maximum Take Off Mass (MTOM) must be less than 250 g, including the payload that the UAS can carry. Their maximum speed at level flight must be 19 m/s. They can reach a maximum height of 120 m from the point of origin of take-off.

C0 class UAS must be powered by electric batteries that may not exceed 24 V direct current (DC) or equivalent alternate current (AC) power. The UAS must ensure a safe flight in terms of stability, manoeuvrability and performance of the data link with a pilot radio control, which will be used following the UAS manufacturer's instructions. The C0 class UAS must be built without elements that could cause damage to people during their operation, so sharp shapes must be avoided.

In case the Remotely Piloted Aircraft System (RPAS) has the follow-me flight mode, the UAV must never be able to move beyond 50 m of the pilot, and this will always maintain the controllability of the drone above the follow-me mode.

To pilot a C0 class drone, it will not be necessary for the pilot to have any type of license.

A C0 class UAS must display the badge shown in Fig. 2.1 in order to identify the class it belongs to.

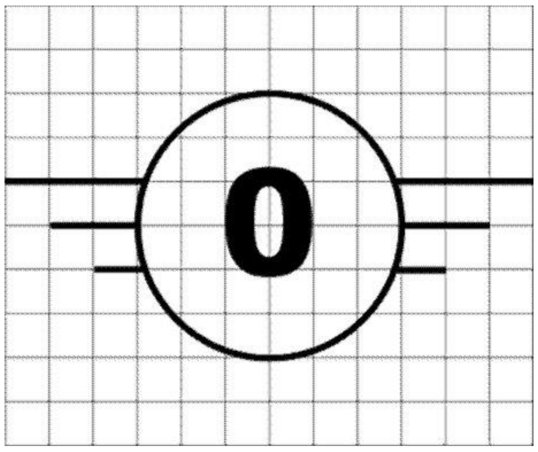


Fig. 2.1 C0 class UAS identification label

2.1.2 C1 Class

C1 class UAV must not have an MTOM higher than 900 g and must not exceed a flight speed of 19 m/s. The speed and weight limits are key to limiting the kinematic energy of the UAS. In addition to energy, the impact of the drone in the event of a possible fall or crash would be greater and cause more damage.

These RPAS will not be allowed to fly above 120 m from the point of origin of take-off. As in C0 class, C1 class must be safe in terms of stability, manoeuvrability and performance of the data link with the remote control following the manufacturer's instructions. In this class, in case the data link is lost, the UAS must be able to resume the data link or end the flight.

The UAS must not have sharp shapes to avoid causing harm to people during operation. As in the previous class, the RPAS must be powered by DC batteries that do not exceed 24 V or the equivalent AC. The drone must have a unique physical serial number. It must be equipped with lights in order to improve manoeuvrability, to be able to follow the drone at night and to distinguish the RPAS from an airplane.

The UAV can be identified remotely through a documented transmission protocol, where real-time UAV information such as serial number, position, speed, height, etc. can be viewed in real time.

Finally, these drones must incorporate geo-conscious systems that prevent them from flying into restricted areas such as airports, nuclear power plants, etc. There must also be clear indication to the pilot when either the drone battery or the radio control battery is low. In the same way as C0 class UAS, if the C0 class UAS has a follow-me flight mode, it must not be able to move beyond 50 m from the pilot, and the pilot must always maintain the drone's controllability above the tracking mode.

Like for C0 class drones, the pilot does not need a license to fly a C1 class UAS.

C1 class drones must display the badge shown in Fig. 2.2.

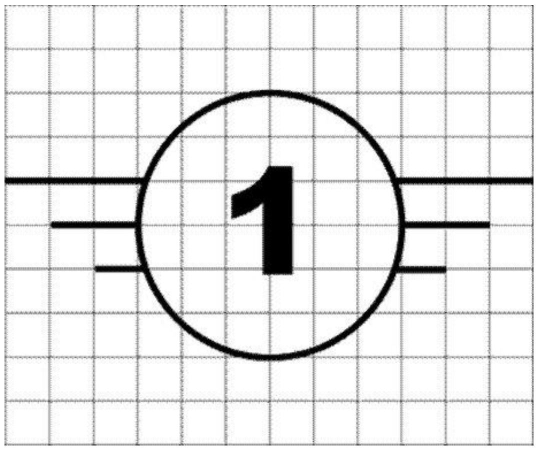


Fig. 2.2 C1 class UAS identification label

2.1.3 C2 Class

C2 class UAS must not have an MTOM higher than 4 kg, including payload. As in the first two classes, the maximum flight height will be 120 m from the take-off point. It must be safe in terms of stability, manoeuvrability and performance of the data link, following those defined in the Implementing Regulation (EU) 2019/947 and following the UAS manufacturer’s instructions.

It must be built in such a way as to avoid damage to persons during the operation, so sharp shapes must be avoided. In case the UAS is tied by a tether, the cable must not be longer than 50 m.

If the UAV loses the data link, it must be able to retrieve the link on its own or end the flight, unless the drone is tied up. The data link must be protected against unauthorized connections for commands and control functions, unless the UAV is tied.

Unless it is a fixed wing, the RPAS must have a slow flight mode, which can be activated from the pilot's remote control and which must limit the cruising speed to 3 m/s. C2 class drones must be powered by batteries that do not exceed 48 V DC or equivalent AC power.

As in the previous class, they must have a unique serial number, be equipped with lights to improve controllability, night tracking and to distinguish them from an airplane and have a geo-conscious system. There must also be clear indications to the pilot when either the drone battery or the radio control battery is low. The UAV information must be remotely accessible, showing the serial number, position, height, speed, pilot position if possible, etc.

From C2 class drones the pilot must have a license to operate with them.

C2 class drones must display the badge shown in Fig. 2.3.

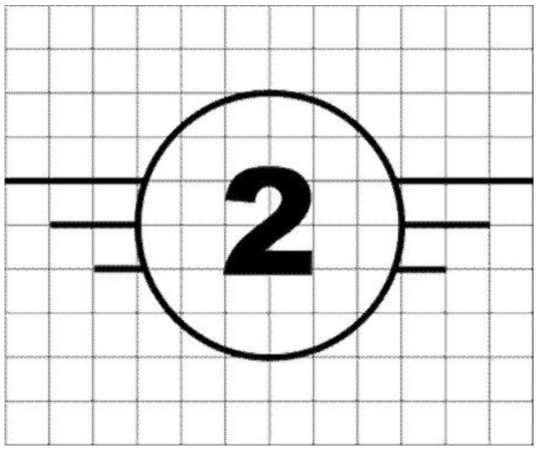


Fig. 2.3 C2 class UAS identification label

2.1.4 C3 Class

C3 class UAV must not have an MTOM higher than 25 kg, including payload, and cannot exceed 3 m in characteristic dimension. They may not exceed 120 m in height from the point of origin of take-off. Like the C2 class, it must be secure in terms of data link stability, manoeuvrability and performance in accordance with Implemented Regulation (EU) 2019/947 and the UAS manufacturer's instructions.

If the rope is tied, it cannot exceed 50 m in length. In the event that they are unattached and the data link is lost, you must have a secure method to recover the data link or end the flight. The data link must be protected against unauthorized connections, unless it is linked.

As in the previous class, the battery may not exceed 48 V DC or equivalent AC power. It will also need to have lights to improve the controllability, tracking and distinguishing it from an airplane. It must be equipped with a geo-conscious system.

It must have a unique serial number. Unless linked, it must be possible to access transaction information remotely and in real time.

The pilot will need a license to fly C3 class drones.

C3 class drones must display the badge shown in Fig. 2.4.

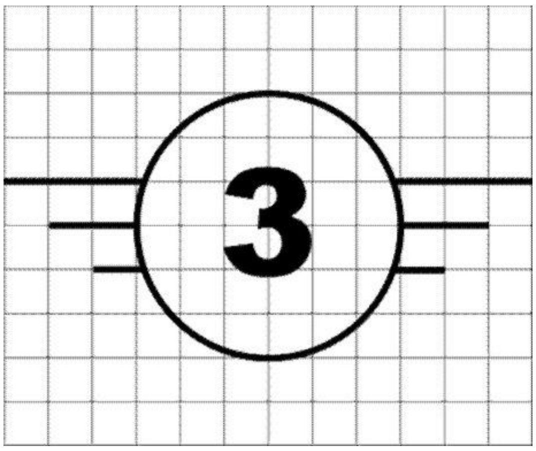


Fig. 2.4 C3 class UAS identification label

2.1.5 C4 Class

C4 class is very similar to C3 class. The main difference is that in the C4 class it corresponds to model airplanes.

C4 class vehicles will not have an automatic control mode except for stabilized flight assistance. For the C4 class it will also not be necessary for the pilots to be licensed.

C4 class UAS must display the badge shown in Fig. 2.5.

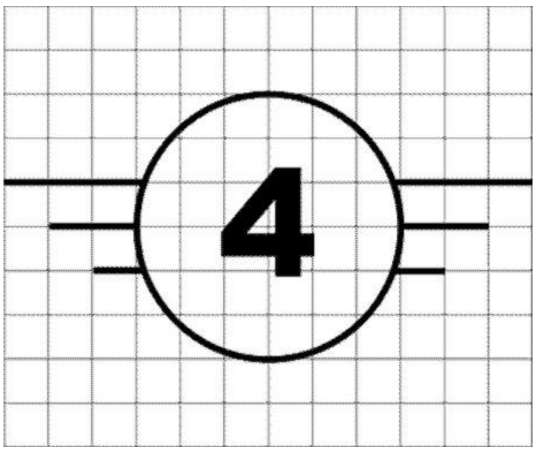


Fig. 2.5 C4 class UAS identification label

2.2 Implementing Regulation 2019/947

The European Commission published on May 24, 2019, this law, which deals with the rules and procedures for the use of drones by pilots and operators, defining categories of use and a series of requirements for their use [7].

UAS operations must be performed under either the open, specific or certified categories.

2.2.1 Open category

Open category UAS operations are not subject to any prior operational authorization or to an operational statement from the UAS operator prior to the operation.

For an operation to be recognized within an open category, it must meet a number of requirements. First of all, the UAS must belong to one of the classes marked in Delegated Regulation (EU) 2019/947 that we have explained before, or be built privately. The MTOM must be less than 25 kg.

The pilot must ensure a safe distance between the drone and people and at no time he/she will be allowed to fly over a group of people. The pilot must always fly the drone in Visual Line of Sight (VLOS), unless he/she is using the follow-me mode or there is some observer controlling the UAV who is in constant communication with the remote pilot.

During the operation, it will not be possible to fly the UAS further than 120 m from the point of origin of the flight unless it has to overcome some obstacle. The UAS is not allowed to carry any dangerous load or drop any material.

Open category UAS operations are divided into three subcategories according to Part A of the Annex to this law. These categories are called A1, A2 and A3.

2.2.1.1 Subcategory A1

Subcategory A1 must meet two requirements. If the drone is C0 class, it must be conducted in such a way that the remote pilot will not be able to fly over groups of people, but will be allowed to fly over people not involved in the operation. If it is a C1 class UAS, it must be flown in such a way by the pilot that he/she may overfly uninvolved persons but shall never overfly assemblies of people.

2.2.1.2 Subcategory *A2*

Only C2 class drones fall into subcategory A2. The pilot must ensure that the UAS never flies over people who are not involved in the operation and maintain a horizontal distance of at least 30 m from them. The pilot will be allowed to reduce the minimum distance to 5 m if the UAV is flying in low speed mode.

Finally, the operation must be carried out by a pilot who knows the UAS manufacturer's flight manual and has the corresponding remote pilot certificate.

2.2.1.3 Subcategory A3

In the subcategory A3, the UAS of classes C2, C3 and C4 are involved. The UAS must be piloted by the remote pilot in an area where the persons not involved will not be in danger, throughout the area of the operation for the total time of the operation. The operation will take place at a minimum distance of 150 from residential, commercial, industrial or recreational areas. Finally, the pilot must have successfully passed an online training course and a theoretical knowledge test.

2.2.2 Specific category

An operation will be within a specific category if only one of the requirements of the open category is no longer met. If one of the requirements of the open category is not met, the operator must obtain an operating authorization from the competent authority of the Member State where he/she is registered.

When the operator requests an operating authorization from a competent authority, the operator shall carry out a risk assessment in accordance with Article 11 of the law and submit it together with the application, including the appropriate mitigation measures. The competent authority shall issue an operational authorization if it considers that the operational risks are sufficiently mitigated in accordance with Article 12, which sets out the points necessary to authorize an operation.

2.2.2.1 Article 11

Article 11 of Implementing Regulation 2019/947 sets out the requirements that the operational risk assessment to be done by the operator requesting authorization must have in order for the competent authority to give permission to carry out an operation.

The operational risk assessment must describe the characteristics of the operation to be carried out by the UAS. It must adequately propose operational safety objectives. The operator requesting authorization must identify the hazards on the ground and in the air, and must also identify a range of measures to mitigate the risks and, finally, determine the necessary level of robustness of the mitigation measures, so that the operation can be carried out safely.

The description of the operation must include at least explanation of:

- the purposes/objectives of the activities to be carried out
- the environment and geographical area of the operation
- the complexity of the operation

- the preparation and execution of the operation
- the composition of the team who will participate in the operation
- the skills, experience, competencies, training and role of each of the members of the staff in charge of the operation
- the technical means to carry out the operation
- the technical characteristics of the UAS, including its performance in the face of the conditions encountered during the operation

The assessment must propose a security level objective that must be equivalent to the aviation security level, in view of the specific characteristics of the operation.

All risks both on land and in the air that cannot be avoided during the operation must also be included. Ground hazards take into account the type of operation and the conditions under which the operation will be carried out.

As the risks are analysed, the necessary mitigation measures will also need to be analysed in order to achieve the proposed level of security. These mitigation measures shall consider the following possibilities:

- containment measures for people on the ground
- strategic operational limitations for the operation of the UAS
- strategic mitigation according to the general rules of aviation and shared airspace and services
- capability to withstand adverse weather conditions
- factors such as operation and maintenance procedures can also be organized by the operator following the UAS manufacturer's manual
- the level of competency and expertise of the personnel involved in the safety of the operation
- the risk of human error
- the design and performance characteristics of the UAS

The soundness of the proposed mitigation measures will be assessed to ensure that the targeted level of security is achieved, to ensure that all stages of the operation are safe.

2.2.3 Certified category

UAS operations will be classified within the certified category if during the operation the UAS has to fly over people, or if the operation involves the transport of people or some dangerous object, which can be a high risk for third parties in case of accident.

In addition, UAS operations may also fall into this category if, based on the operational risk assessment provided by the operator, the competent authority considers that the risk of the operation cannot be adequately mitigated without certification of the UAS, the operator or when applicable without a license from the UAS pilot.

2.3 Venturi VTOL

Once we have seen the current European legislation for drones and the characteristics of the Venturi V1 drone in Chapter 1, now we can know how the legislation affects the V1 drone.

Regarding the Delegated Regulation 2019/945, due to the its wing dimensions of 3 m and its MTOM lower than 25 kg, the Venturi V1 belongs to the C3 class.

With respect to the type of operation carried out by the V1 drone, we have already explained previously that the objective of the drone is the inspection of power lines and gas pipelines. In order to perform these tasks, the drone must travel long distances in a single operation, moving far away from the pilot. This type of operation is called Beyond Line of Sight (BVLOS). In addition, depending on the characteristics of the cameras and the power lines, the drone can exceed 120 m in height at certain times. These two factors mean that the operation cannot be of the open category and therefore the operation is of the specific category.

As the operation of the drone belongs to the specific category, Venturi must carry out an operational risk assessment, present it to the competent authority and have it approved before flying the V1.

CHAPTER 3. STATE OF THE ART OF DRONE SIMULATORS

In this chapter, the fundamentals and the state of the art of drone simulators are presented. Most drones that use free software technology use ArduPilot or PX4 from DroneCode Foundation, San Francisco, CA (USA), as flight controller software. ArduPilot was born in 2007, but the developers did not manufacture the first autopilot plate for UAV until 2009. Since then, this software has grown continuously and is currently capable of controlling up to seven different vehicles or systems: multi-rotor drones, fixed-wing and VTOL aircraft, helicopters, rovers, submarines, ships and antennas.

In 2011, DroneCode created PX4 [8] from ArduPilot. DroneCode is a non-profit Linux company that began the project of professionalizing the development of open source software for flight controllers for UAV.

ArduPilot and PX4 have developed a simulation so that their software can be tested in a simulated environment. There are two simulation options: Software In the Loop (SITL) and Hardware In the Loop (HITL) [9]. These simulations have a very poor visual representation, only the icon of the simulated vehicle is shown on a map. The simulation does not show the movements of the rotors or the control surfaces of the UAV.

SITL is test software that simulates a vehicle's hardware and its behaviour during a mission. HITL is a bit different from SITL in the sense that it needs a flight controller to do the simulations, because HITL simulates all the hardware components except the flight controller. The tool commonly used is SITL due to its simplicity, so we will also work with SITL.

Venturi is currently using ArduPilot, but in the short-term future they plan to use PX4, so our simulation software will have to be combined with the current ArduPilot and PX4 simulation. Our simulator must show the simulated vehicle, the rotation of the rotors, the movement of the control surfaces and the movement of the UAV in a 3D environment.

The simulation must be in 3D because it will be used to train future pilots of the company and in this way, we will have a good approximation of a real flight. To choose the best software for performing our 3D simulation (our simulation will be carried out within an external simulation software), in the following subsections we will study different options, analysing their advantages and disadvantages.

3.1 X-Plane

X-Plane from Laminar Research, Columbia, SC (USA) [10], is a commercial flight simulator, which has very good graphics. It can simulate the behaviour of airplanes and helicopters in a very good way. In addition, it is able to connect with ArduPilot's SITL, but not with PX4. It has a wide variety of airplane and helicopter models.

The big drawback of X-Plane is that it is commercial software, and therefore, be able to use it completely (that is, with all the full capabilities and functionalities), it will be necessary to acquire it. Another major drawback is that it does not allow simulating models other than those incorporated by the official software distributor. In particular, it is only capable of simulating airplanes and helicopters, so it cannot simulate all types of multirotor drones or VTOL vehicles.

Moreover, because it is commercial software, its code is not accessible and its configuration to work with SITL is not entirely intuitive and nothing can be modified. X-Plane can run on Windows, Linux and Mac.

3.2 RealFlight

RealFlight from Knife Edge Software, Corvallis, OR (USA) [11], is also a commercial flight simulator, so it would also be necessary to acquire it in order to combine SITL with RealFlight. RealFlight is designed for drone pilots and for aircraft controlled with a radio control.

With RealFlight we can use our own models and also simulate different environments. This software has different versions and not all of them work with SITL. In fact, only ArduPilot can use SITL. PX4 cannot use SITL, but HITL can.

The big downside to RealFlight is that it is not easy to add new vehicle models and that it only works on Windows computers.

3.3 Gazebo

Gazebo [12] is a simulator that allows the user to simulate the dynamics of any robot, create outdoor and indoor environments, and simulate all kinds of sensors and design Plugins that control these robots. Moreover, it is able to simulate any type of vehicle, designed from scratch by the user.

Gazebo can be combined with both ArduPilot and PX4 SITL, and since it can represent any model, it is capable of simulating all SITL vehicles. Another great advantage of Gazebo is that it is open source code and therefore it is for free.

The big downside to Gazebo is the big learning curve, which is needed to be able to work with it, as knowledge of C++ and Robot Operating System (ROS) is needed. Works on Windows and Linux computers.

3.4 jMAVSim

jMAVSim [13] is a simple quadrotor simulator that allows the user to simulate with PX4. It is very easy to set up and use due to its simplicity.

The big downside is that it only works with PX4's SITL, as it has been developed by the same developers. On the other side, it is capable of simulating multiple vehicles at once, while keeping the simulation environment very simple. Therefore, it does not require large computational resources, compared to other simulators. jMAVSim works on Windows, Linux and Mac computers.

3.5 JSBSim

JSBSim [14] is a flight dynamics model (FDM) software that is capable of simulating the flight of any air vehicle. FDM are the physical and mathematical models that define the movement of a vehicle under the forces and moments acting on the vehicle thanks to the control systems that can use the simulator and the forces of nature that apply on the vehicle.

The strengths of JSBSim are the accuracy of its simulation calculations, and the fact that the flight control systems, aerodynamics and propulsion of the vehicles to be simulated can be configured. In addition, it takes into account the effects of the rotation of the Earth.

The big downside to JSBSim is that it has no graphical user interface (GUI), so the user needs the combination of another software package to be able to visualize the results. JSBSim can be used with SITL, but we will still not have 3D visualization. For this purpose, it is usually combined with FlightGear [15], which is a free code simulator that aims to create a sophisticated and open flight simulator environment for use in research, academic environments, pilot training, in industry, etc.

Apart from all this, the learning curve for being able to create your own model and make it realistic is very long, as there is not a large users community that shares its experiments, which makes it difficult to learn.

Finally, JSBSim only works with ArduPilot's SITL; currently it cannot work with PX4. It can be used on Windows, Linux and Mac computers.

3.6 AirSim

AirSim from Microsoft, Redmon, WA (USA) [16], is a free code simulator from Microsoft, capable of simulating vehicles such as drones and cars. The simulator has been built with Unreal Engine [17], which is the most powerful 3D rendering tool today, used in many video games, architecture, television, etc.

This simulator is designed for use in deep learning projects, computer vision and learning algorithms for autonomous vehicles. The simulation is very good in terms both of visualization and modelling of physic phenomena, that is, many laws of physics are properly introduced into the simulation such that more realistic effects and results are achieved. The physics simulated is a close approximation to real physics, although discrete values are used.

The big downside to using AirSim is that it only works for one quadrotor model. Thus, so far, user customary models cannot be incorporated into the simulation, therefore it is not possible to simulate any VTOL. Another major drawback of AirSim compared to other simulators is that it works on Unreal Engine, and this tool has a very high computational cost, which means that not all computers are able to use this simulator.

In addition, it works with both ArduPilot and PX4 SITL. AirSim works on Windows, Linux and Mac computers.

3.7 Choice of technology

Table 3.1 summarises the main simulators that exist in the market, their main advantages, disadvantages, and characteristics, as found in the previous analysis, to see in perspective the strengths and weaknesses of each simulator, and to be able to choose the technology that we will finally use in this project.

Table 3.1. Comparison between the studied simulator software packages

Simulator	Pros	Cons
X-Plane	- Good graphics - Good physics modelling	- Commercial software (750 $) - You cannot create your own models - It does not simulate VTOL - Does not work with PX4
RealFlight	- Good graphics	- Commercial software (110 $) - Difficult for the users to create their own models - It does not simulate VTOL - Does not work with PX4
Gazebo	- Allow the users create their own models - Simulates worlds, forces, sensors, etc. - Works with ArduPilot and PX4 - Free code	- Very long learning curve
jMAVSim	- Simple simulation - Free code	- Does not work with ArduPilot - It does not simulate VTOL
JSBSim	- Free code - Very precise physics - Allows the users create their own models	- Needs extra software for 3D visualization - Long learning curve - Does not work with PX4
AirSim	- Amazing graphics (uses Unreal Engine) - Good physics modelling - Free code - ArduPilot and PX4	- It does not simulate VTOL - Very high computational cost

In order to make the right choice, we need to know the requirements of the simulator we will be using. There are several conditions that must be met, and those are that it must be a free code simulator and that it must be able to simulate any type of vehicle, that is, we must be able to create our own models.

In addition, the simulator must be able to work with at least ArduPilot and, if possible, with PX4, as in the future the simulator will also have to work with PX4.

Once we know the minimum requirements of the simulator software, and we also know the strengths and weaknesses of the main simulators that exist in the market, we can choose the most suitable software for our project and purposes. Finally, the chosen simulator is Gazebo.

The choice of Gazebo is basically due to the fact that it is open source software and that it allows simulating any type of vehicle and environment. In addition, with Gazebo the user can simulate sensors and physics. However, the development of the simulation with Gazebo must be done with ROS and C++, which are two programming languages with a very long learning curve, and therefore the development of the code will be difficult.

CHAPTER 4. SIMULATION

As explained in the first chapter, after a thorough analysis of the alternatives to make our simulations, the chosen technology is Gazebo, which is a robot simulation environment. In order to develop our own simulation in Gazebo, we will have to combine different elements. First of all, we will need the model to be simulated. Then, we will have to design the program that will be responsible for giving the commands to the Gazebo model. In addition, these commands will have to come from somewhere, which will be the SITL of ArduPilot.

In order to understand how the simulation works, all the elements involved will be explained in detail below. Once the elements involved have been explained, the necessary configuration will be explained in the SITL and in Gazebo. Then, the development that has been done and its operation will be explained. Finally, the results will be presented.

To make sure the simulator works well, we will test the simulation first of all with a standard VTOL created by Gazebo. In case it works correctly, we will use the simulator to fly the UAV designed by Venturi, if possible.

4.1 Elements involved

4.1.1 Gazebo model

In Gazebo, a model is a physical entity with dynamic, kinematic and visual properties. Everything represented in a simulation in Gazebo is a model, even the ground is a model. A model can represent any type of shape, from the simplest to the most complex. Plugins are available to control the behaviour of a model in Gazebo. In our simulation, we will focus only on the drone model, which is the main protagonist of our simulation.

Models in Gazebo are usually defined in Simulation Description Format, usually abbreviated as SDF. The SDF format is an XML format that describes objects and environments for robot simulation, visualization, and control. It was born from the hand of Gazebo, as it was designed for scientific applications with robots. Over time, it has become very stable and robust, and is being used to describe more phenomena, such as the statics and dynamics of an object, terrain, physics, light, sensors, and so on.

To perform a simulation, the user can create his/her own model or use those that are already created. In either case, all robots have the same parts. The robot model has three main parts: links, joints and Plugins [18]:

1. **links:** links contain the physical properties of a model body; there are different types of links:
    a. collision: encapsulates the geometry used to calculate a collision with the model/robot

b. visual: used to display parts of a link, and the link may or may not be visible
c. inertia: describes the physical properties of the link, such as the mass and rotation/inertia matrix
d. sensor: collects information about the link, which may be needed in Plugins
e. light: describes the light source that corresponds to it

2. **joints**: joints connect two links, between which there should always be a parent-child relationship. In addition, other parameters are set in a joint, such as the axis of rotation, the limits of the movement, etc.
3. **Plugins**: Plugins are the libraries that are created in order to be able to control the robots

With these three elements, we can define any type of robot. In our case, we want to define in SDF format the Venturi V1. So far, the UAV is designed in SolidWorks, so first it must be converted from a SolidWorks model to an SDF model. This step is not at all easy as there is no well-known method for doing so.

In fact, there is no software that does the conversion from SolidWorks to SDF, but luckily there is one that does the conversion to Unified Robotic Description Format (UDRF). This is another XML format for describing robots. The mentioned software is called "SW2URDF". Once the model is converted to URDF format, it can be transformed to SDF format. ROS can automatically transform a URDF file to SDF. The difference between URDF and SDF is that URDF only describes the robot, while SDF can define many more things, such as the world where the robot is located. Therefore, an SDF file includes the information found in a URDF file.

Although it is possible to create an SDF file from a SolidWorks model, if this model does not have the characteristics well defined in SolidWorks, its simulation will not work well in Gazebo.

Fig. 4.1 shows the V1 in the Gazebo simulation environment.

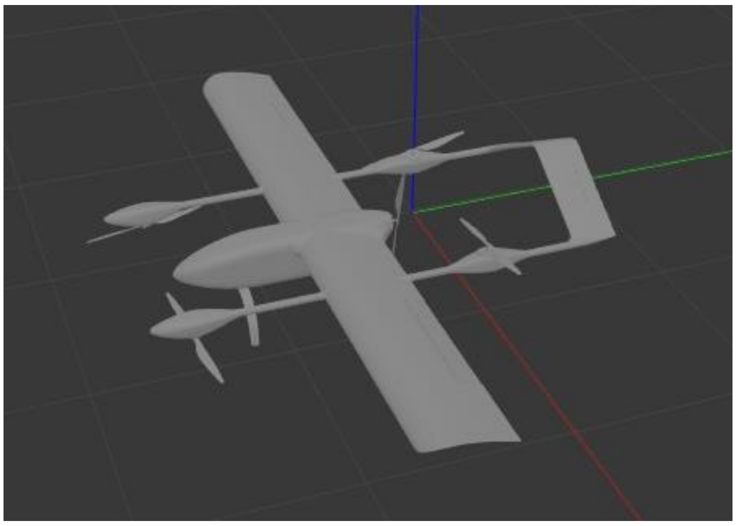


Fig. 4.1 Venturi V1 UAV model in Gazebo

While we want to simulate the UAV designed by Venturi, we will also simulate a Gazebo standard VTOL model, which was designed from scratch by the developers of Gazebo. This ensures with high degree of confidence that the model has no bugs. We will first test the simulation with the Gazebo standard model and, if it works well, we will do it with the Venturi V1.

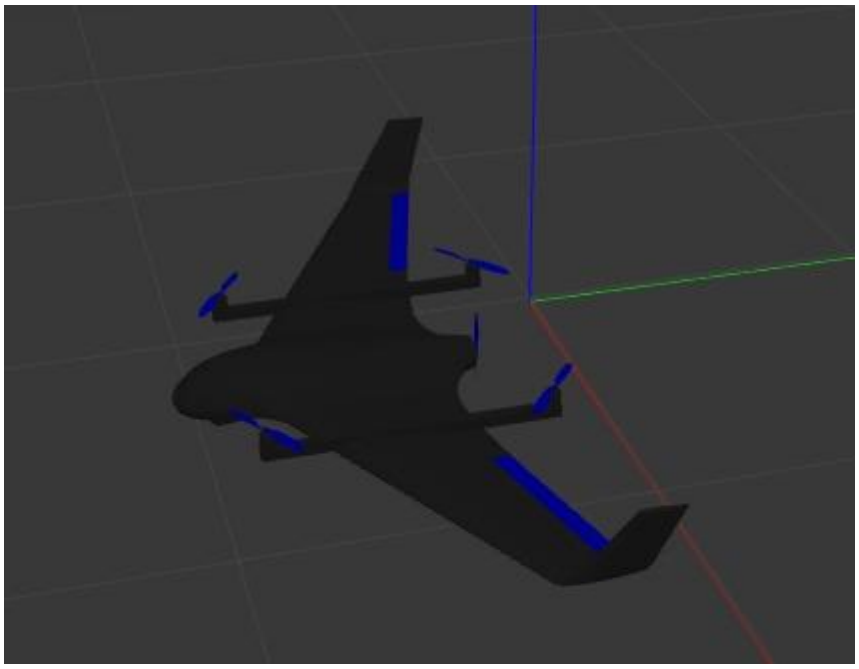


Fig. 4.2 Standard VTOL UAV model in Gazebo

As can be seen in Fig. 4.2, unlike Venturi's V1, this UAV has no tail, it only has two control surfaces on the wings. However, it has four engines for vertical take-off and landing and one engine for horizontal propulsion, like Venturi's V1.

Although the two VTOL models that will be simulated do not have the same characteristics, they do have the same operation and a similar flight controller configuration. Therefore, the two models will be used to see the flight dynamics of a VTOL (its flight phases) and to train future pilots.

4.1.2 Plugin

As we explained in the previous section, every model in Gazebo can be controlled by a Plugin. First of all, we need to understand what a Plugin means [19]. A Plugin is basically a piece of code compiled like a library, which is inserted into the simulation and with which we are able to control any aspect of Gazebo.

There are many types of Plugins, but since we are interested in controlling a UAV, the relevant type of Plugin to be used is called Model Plugin. Depending on the type of Plugin, we can control different aspects of Gazebo.

The development of the Plugin that will control the entire simulation must be done with the C++ programming language [20]. This programming language was designed in the 1980s. Its main goal was to extend the C programming language and make it object-oriented. It is one of the most widely used languages today due to its great performance.

In addition to using C++ as a programming language, we will also need ROS. ROS [21] is a robotic middleware, i.e., a collection of frameworks for the development of robot software.

ROS is a system based on nodes. A node is any software or hardware with which it can interact. ROS has defined a standard for communication between the nodes that can form the system. There are two communication mechanisms: the first and most used is subscription/publication, while the second is services. The first is based on sending and receiving messages. These messages will be posted or received based on topics. Publisher continually broadcasts a message to all connected subscribers. Services, unlike the subscription/publishing method, only send the message when the client requests it.

What nodes basically do is to receive messages from the topics the user is subscribed to, and/or post messages on the topics the user creates and can also act with the data the user has. If a node posts a message in a particular topic, in order for another node to receive it, it must be subscribed to that topic. Then, it will receive the message. Fig. 4.3 shows a sketch for a better understanding of how a ROS node system works.

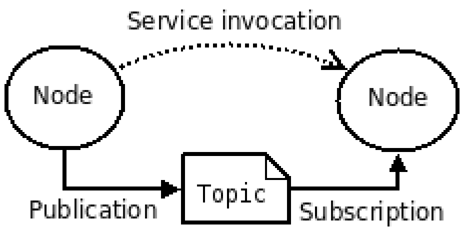


Fig. 4.3 Communication between nodes in a ROS system

In order to use a Plugin with a Gazebo model, the user must call this Plugin from the SDF file that defines the model. Each Plugin has some variables, which will vary depending on the used model. These variables must be described in the file that defines the simulated vehicle. Then, the user should identify what settings the chosen Plugin needs to work.

4.1.3 Ground Control Station

A Ground Control Station (GCS) is a ground control centre that allows the operator to control a UAV. In our case, even though it is a simulation, the simulated drone also has the same operating system as a real UAV, so we need this GCS to be able to send commands to the drone to control it.

Since the flight controller of the simulated UAV uses ArduPilot, we will use the GCS most used by ArduPilot, i.e., Mission Planner. The Mission Planner can be used as a utility for drone configuration and as a dynamic complement to the UAV.

Communication between the GCS and the simulated drone occurs through the Micro Aerial Vehicle Link (MAVLink) communication protocol. This protocol is designed for communication with small UAVs. Communication between the GCS and the UAV flight controller is done through packets that follow this protocol.

For the simulation we need the GCS because through the GCS we can connect a control radio, with which we will be able to control the drone of the simulation. In addition, through the GCS we will read the values of the drone sensors, such as the output generated by each engine and control surface. Thanks to a GCS we can plan missions to be carried out by the UAV, these missions can be sent to the UAV through the MAVLink protocol and once the drone has received the instructions it can execute the planned mission.

4.2 Pre-configuration

Once the elements involved have been explained, it is time to explain the configuration to be done in the SITL and what parameters the Plugin needs to work.

4.2.1 SITL configuration

As we have explained on more than one occasion in this work, the SITL is an ArduPilot simulator. ArduPilot is the software that many UAV flight controllers use. As we know, not all UAVs are the same, there can be multirotors, fixed wings, VTOLs, helicopters, etc. Therefore, since the ArduPilot software works for many types of vehicles, the SITL must also be able to simulate all these types of vehicles. This is why the configuration of the SITL is very important, as the configuration must be appropriate to be able to achieve a good simulation.

As we saw in a previous section, the two models to be simulated (the Gazebo standard VTOL model and the V1 model) do not have the same number of control surfaces, so they will have to be configured differently. What really needs to be configured in the SITL is the type of UAV that is being simulated (in our case, a QuadPlane), the shape of the fuselage and engines configuration (in our case, the H shape) and, finally, the number of rotors and control surfaces of the UAV, and which outputs correspond to each of them, as will be explained in the following sections.

4.2.1.1 Standard VTOL configuration

Let us start with the configuration of the Gazebo standard VTOL. As we saw in Fig. 4.2, this UAV has two control surfaces and five rotors.

According to ArduPilot's documentation, we have up to sixteen engine outputs [22]. Of these sixteen outputs, in the case of a QuadPlane, outputs five through eight (both included) are reserved for engines that produce vertical thrust [23]. The control surfaces of the wings will be considered elevons.

To tell SITL what each output is, we need to look for the conversion of these values in the ArduPilot parameter list. The conversion will be a number from 0 to 133, so there are 134 options to set as output of the engines.

Since we have 2 control surfaces and 5 rotors, the motor distribution will be as follows (next to each element, the corresponding number from the list of 134 parameters is indicated):

$$\begin{cases} \text{Output 1: Left elevon} \rightarrow 77 \\ \text{Output 2: Right elevon} \rightarrow 78 \\ \text{Output 3: Throttle} \rightarrow 70 \\ \text{Output 4: Motor 1} \rightarrow 33 \\ \text{Output 5: Motor 2} \rightarrow 34 \\ \text{Output 6: Motor 3} \rightarrow 35 \\ \text{Output 7: Motor 4} \rightarrow 36 \end{cases}$$

4.2.1.2 V1 configuration

In the same way we configured the Gazebo standard VTOL, we now configure the V1 designed by Venturi. Since the V1 is a real UAV, we can use the actual UAV configuration for the simulation.

In this model, we have 4 control surfaces and 5 rotors, so we will have 9 outputs:

$$\begin{cases} \text{Output 1: VTail Right} \rightarrow 80 \\ \text{Output 2: Throttle} \rightarrow 70 \\ \text{Output 3: Left aileron} \rightarrow 4 \\ \text{Output 4: Right aileron} \rightarrow 4 \\ \text{Output 5: Motor 1} \rightarrow 33 \\ \text{Output 6: Motor 2} \rightarrow 34 \\ \text{Output 7: Motor 3} \rightarrow 35 \\ \text{Output 8: Motor 4} \rightarrow 36 \\ \text{Output 9: VTail Left} \rightarrow 79 \end{cases}$$

4.2.2 Plugin configuration

Once we have completed the configuration of the SITL, to simulate the two different models, it is now the time to set the parameters of the Plugin. As we explained before, to use a Plugin in a model, it must be called in the SDF file of the UAV, which is the file that defines our model.

Plugin is designed to be used with different types of UAVs, which have a number of different engines as well as control surfaces. To avoid having to modify the code each time we work with a different model, we can make the software know the model configuration through the parameters we define in the SDF file.

The definition of the parameters must be done through an XML structure. In Fig. 4.4, you can see which parameters the Plugin needs and the explanation of each parameter.

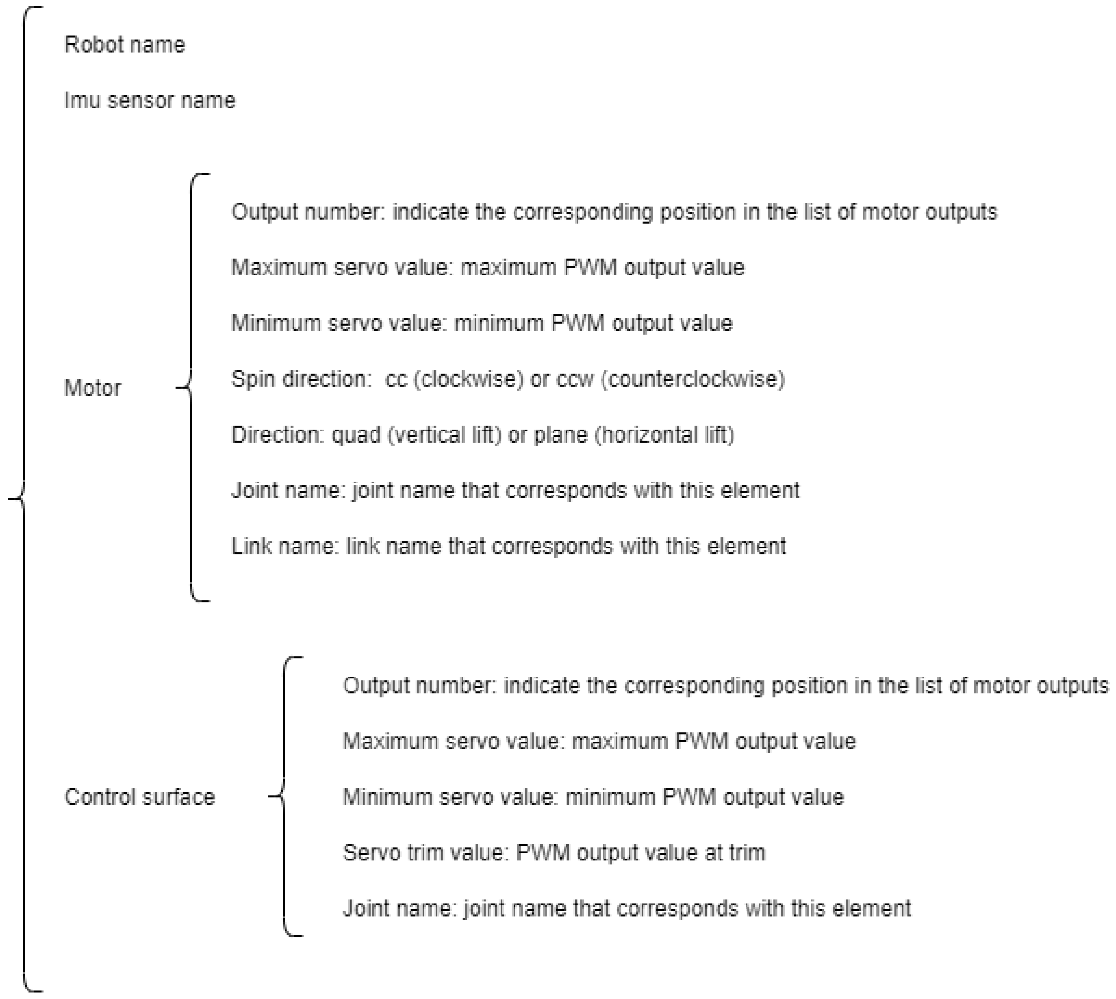

Robot name
Imu sensor name
Motor
Output number: indicate the corresponding position in the list of motor outputs
Maximum servo value: maximum PWM output value
Minimum servo value: minimum PWM output value
Spin direction: cc (clockwise) or ccw (counterclockwise)
Direction: quad (vertical lift) or plane (horizontal lift)
Joint name: joint name that corresponds with this element
Link name: link name that corresponds with this element
Control surface
Output number: indicate the corresponding position in the list of motor outputs
Maximum servo value: maximum PWM output value
Minimum servo value: minimum PWM output value
Servo trim value: PWM output value at trim
Joint name: joint name that corresponds with this element

Fig. 4.4 Plugin parameters and their explanation

As we can see, Fig. 4.4, defines all the parameters that the Plugin needs to work properly. As we know, in all UAVs there is more than one engine or control surface. To indicate this, it will be as easy as repeating the structure of the engine or control surface as many times as engines or control surfaces we have in our UAV.

4.3 Simulator operation

Once the elements involved in the simulation have been explained, we can now explain how the simulation works. First of all, we have to record what each part will do. As we said at the beginning, we will use Gazebo for the simulation, as this software allows simulating any type of vehicle in a 3D environment.

We also explained at the beginning that the Venturi V1 works with ArduPilot software and that it has a testing software, the SITL. The SITL is a tool for testing the behaviour of a UAV, but this simulation does not have a 3D graphical representation.

Finally, we use Mission Planner as a GCS, with which we will control the missions that the simulated UAV will perform, and the parameters and connection of a joystick to control the flight of the simulated UAV.

Now that we know all the parts involved, we can know what is necessary to carry out the desired simulations. We need to develop novel software that is able to understand the commands sent by the ArduPilot SITL, transform these commands so that Gazebo is able to understand them, and send them to Gazebo. In addition, the software must also be able to receive the GCS commands, which will be the commands generated by the control joystick and the generated missions.

As we explained before, due to the requirement imposed by Gazebo, the software developed by the author is written with C++ programming language. All software will be developed within a ROS project for easiness of compilation and subsequent use. Particularly, the software designed by the author will go in the form of a Plugin, as it is the way we can control a model in Gazebo.

To make everything clear and to make the work easier, Fig. 4.5 presents a blocks diagram showing the relationships between the various codes and components involved, as explained before. The coloured blocks correspond to the contributions developed by the author to achieve the desired simulation.

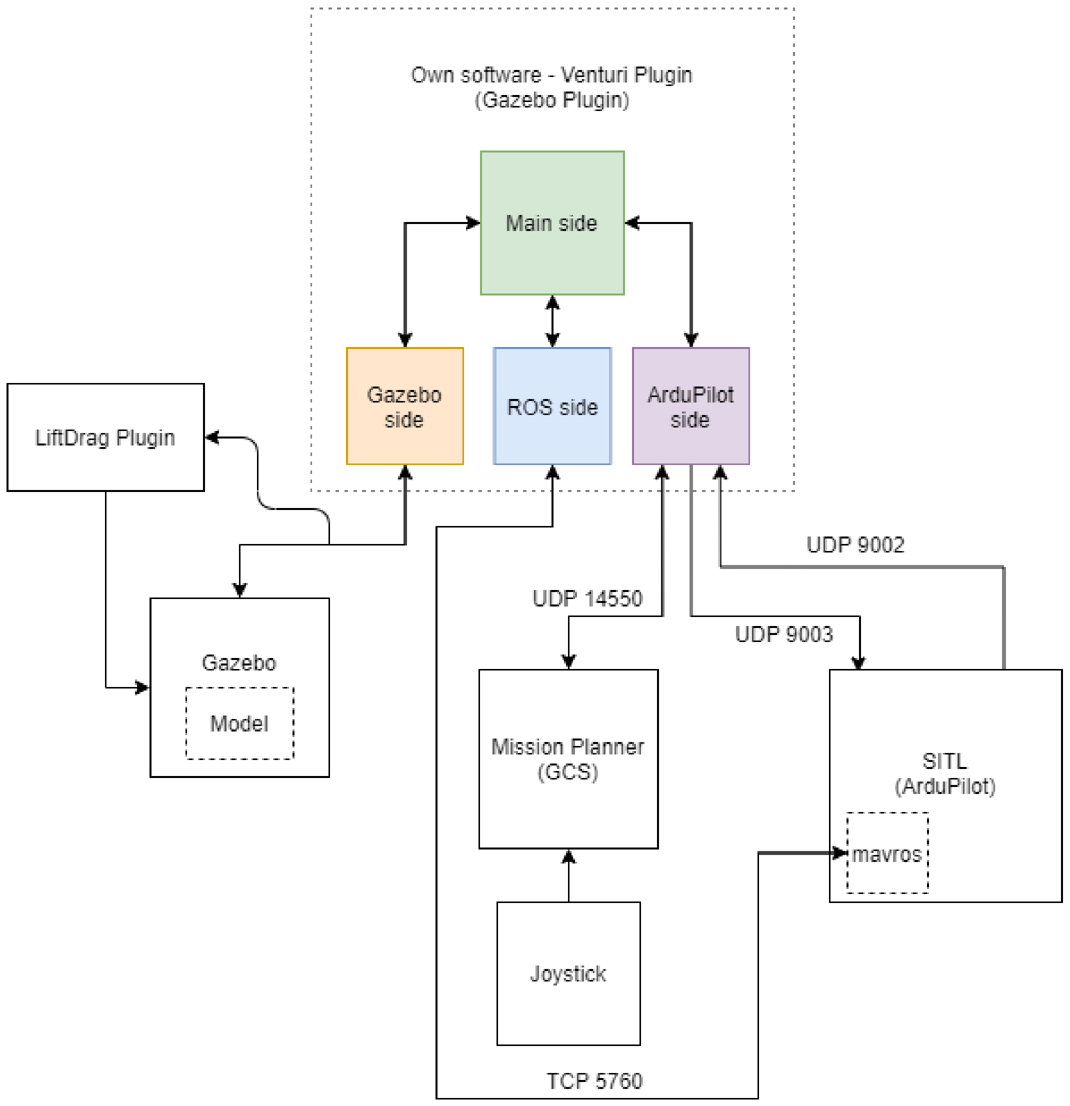


Fig. 4.5 Software blocks diagram

From Fig. 4.5 we can see that between the connection of some blocks appears UDP or TCP and some numbers. UDP is the acronym for User Data Protocol, and TCP is the acronym for Transmission Control Protocol, both are communication protocols. The numbers next to these acronyms are the ports used to make each connection.

As can be seen in Fig. 4.5, the software developed is divided into four parts. Three parts each of which is connected to the corresponding external part: Gazebo, ROS and ArduPilot. Then, there is a part that is the main code, which is the one that handles the overall operation of the designed software. In order to understand what each of the parts is doing, each will be explained separately in the following sections. Also, as we see on the left-hand side of Fig. 4.5, we use an external Plugin called LiftDrag that works in conjunction with the Gazebo part.

We will start by explaining the individual parts, and then the main file that is responsible for making each part work correctly.

4.3.1 ArduPilot side

This is one of the most important parts of the software, as it is the one that communicates with the SITL, which is the simulation tool of ArduPilot. As seen in the blocks diagram in Fig. 4.5, the connection between our software and ArduPilot is made through UDP, where data are sent and received through ports 9003 and 9002, respectively.

The most important issue in this part is to understand what we are sending and receiving. For this purpose, we need to look at the ArduPilot SITL code, specifically, at the file that corresponds to the connection of ArduPilot with Gazebo [24]. This file is called "SIM_Gazebo.cpp". In this file, we see that SITL sends us a package of the type servo_packet and we have to send a package of the type fdm_packet.

To find out the structure of each of the packages, we need to look for the header file that defines the packages [25]. This file is called "SIM_Gazebo.h". First of all, we analyse the packet we receive, that is, a packet of type servo_packet. According to the header, a package of the type servo_packet contains an array of 16 positions of floats, each of which gives the speed of a servo. As we saw in a previous section, in the ArduPilot documentation, up to 16 outputs for servos appear in the list of parameters that configure a VTOL (follow the scheme of the planes). That is why in a package of the type servo_packet there is an array with 16 servos speeds; each of the positions of the array corresponds to one of the possible servos that the UAV may have. It is important to note that servos can be either the rotors of the UAV or the control surfaces.

To find out what the structure of a package of type fdm_packet looks like, we need to look at the header file again. Recall that FDM stands for flight dynamic model. The structure of the package is as follows:

$$\left\{\begin{array}{c}\text{double timestamp}\\ \text{double angular\_velocity\_rpy}[3]\\ \text{double linear\_acceleration\_xyz}[3]\\ \text{double quaterion}[4]\\ \text{double velocity\_xyz}[3]\\ \text{double position\_xyz}[3]\end{array}\right.$$

As we can see, it asks for timestamp, a three-position array with the linear acceleration components in the x-, y-, and z-axis, a three-position array with the angular velocity components in the roll, pitch and yaw axis, a four-position array with the orientation quaternion, a three-position array with the velocity components in the x-, y-, and z-axis, and finally a three-position array with the position in the x-, y-, and z-axis.

It is very important to know the reference systems with which you are working. During the simulation, two different reference systems are used. The reference system of the world and the model is the same, they use a reference system where the x-axis is the longitudinal axis pointing forward, the z -axis is the vertical axis pointing up and the y-axis is the lateral axis pointing to the left. The inertial system uses the NED (North-East-Down) reference system. To better understand the differences between these reference systems, see Fig. 4.6.

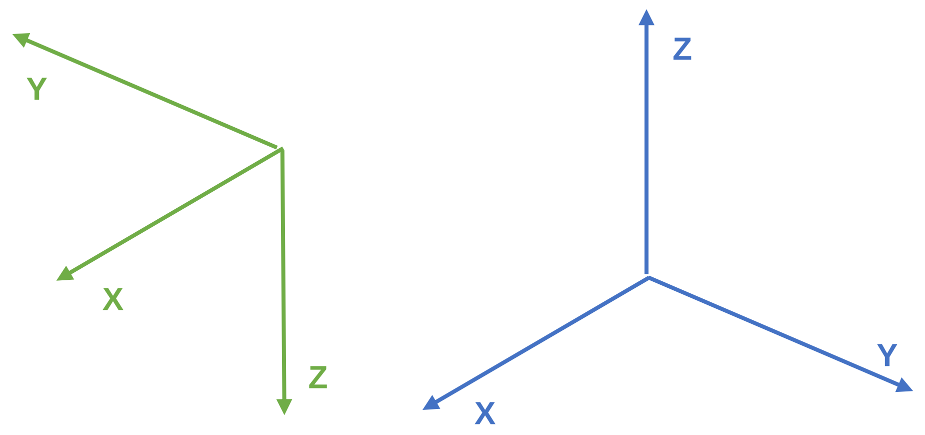


Fig. 4.6 Different reference systems: NED (left) and x-axis pointing forward, y-axis pointing left, and z-axis pointing up (right)

It is important that the order of the package is respected when sending this package, as otherwise the values that the SITL will receive will not make sense.

It is also important to understand that, in this part of the code, the data are only sent and received, that is, it is not in this piece of code where we operate the motors and control surfaces of our Gazebo model. This is done in the Gazebo part. The only action that is done in this section is a small calculation about the data we receive from the engines. SITL gives the information of the motors in Pulse Width Modulation (PWM) format. PWM modifies the duty cycle of a periodic signal to provide information. In Fig. 4.7, we can see how PWM works.

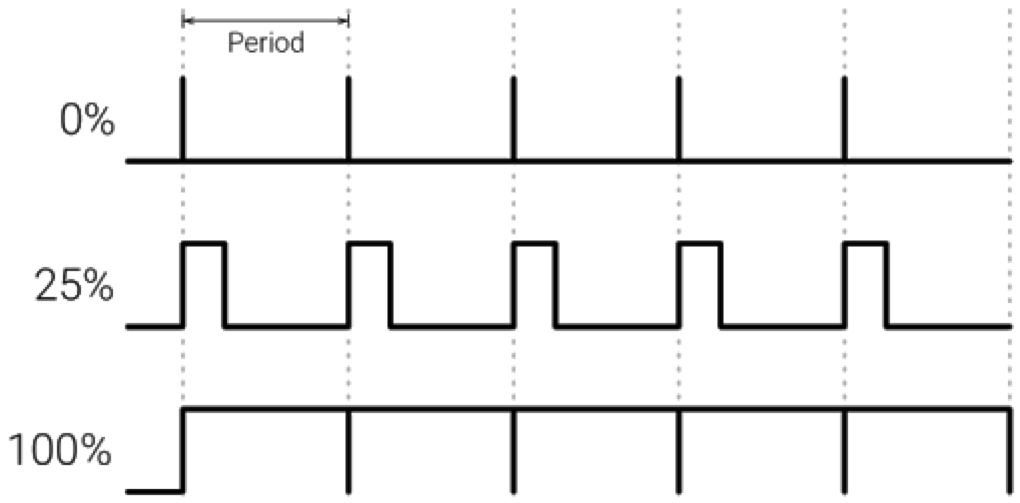


Fig. 4.7 PWM working principle, with the duty cycles of periodic signals

As can be seen in Fig. 4.7, the output corresponds to the percentage occupied by a duty cycle with respect to the period of the periodic signal. For example, if the duty cycle occupies 25% of the signal period, the output value is 25% and so on.

Then, to calculate the speed at which the motors are rotating or the position at which the control surfaces are deployed is very easy, since we have the maximum value and the minimum value that the SITL can send for each engine and control surface. Therefore, it is very simple to make these calculations with the following equations:

$$\text{Motor velocity} = \frac{\text{Current value} - \text{Min. PWM value}}{\text{Max. PWM value} - \text{Min. PWM value}} * 100 \quad \textbf{(4.1)}$$

$$\text{Ctrl. surface position} = \frac{\text{Current value} - \text{Min. PWM value}}{\text{Max. PWM value} - \text{Min. PWM value}} * 100 \quad \textbf{(4.2)}$$

The result of equations 4.1 and 4.2 is in percentage.

4.3.2 ROS side

As seen in the blocks diagram in Fig. 4.5, to use ROS it is necessary for SITL to activate a module called mavros. This module creates a master node that is posting messages to topics so that we can access them and is also subscribed to topics in which we can post to the SITL.

In this part of the code, we use ROS to subscribe to a single topic that gives us some information about where the SITL simulation is. The topic we subscribe to is called /mavros/state and we get from it whether we are connected to the SITL, whether the UAV is armed, in which flight mode it is, etc. It may seem that this information is not very important but in fact it is, as the SITL sends some meaningless values at the beginning of the connection and, thanks to the information we receive in this part, we avoid errors.

4.3.3 Gazebo side

This is the most important part of individual code, as it is the one that has a direct connection to the UAV model in Gazebo. First of all, it takes all the information we need from our model. This information is described in the parameters defined in the SDF file of the model.

When this code is executed, first it grabs all the information from the parameters. Because we have more than one motor and control surfaces, it makes a loop that collects information from all motors and all control surfaces. It stores all the information in an array, where the size of the array corresponds to the number of motors or control surfaces, and in each position there is a structure that stores all the internal parameters of the motors or control surfaces.

Once the necessary prior information has been saved, we are able to control the movements of the motors and control surfaces.

In this part, we basically have to transform the information coming from the SITL into speed for the motors or position for the control surfaces and calculate the information to be sent to the SITL.

We start by calculating the speed of the engines. We have taken as engines the engines used by the V1. The Venturi UAV has two types of engines: one for take-off and landing and another for horizontal flight. The engine for vertical movement is T-motor MN801S 150kv and the engine for horizontal flight is Hacker Q80-13S 28-Pole kv175. From the technical specifications of the vertical flight engine (attached in the appendices), we can extract the rotor speed values as a function of the throttle. These values are extracted experimentally and will not be met under all conditions. The datasheet does not contain the speeds for all throttle, so we made a linear interpolation. In the case of the vertical flight engine, we do not have data for throttle values inferior to 40%. Thus, a linear interpolation between 0% and 40% has been done to extract intermediate values. This approximation can be seen in Fig. 4.8. In the case of the horizontal flight engine, we only have values for 25%, 50%, 75% and 100% of the full throttle. These values do not correspond to the turning speed of the motor. To avoid making more approaches, we use the values of the vertical flight engines in horizontal flight for the V1 simulation.

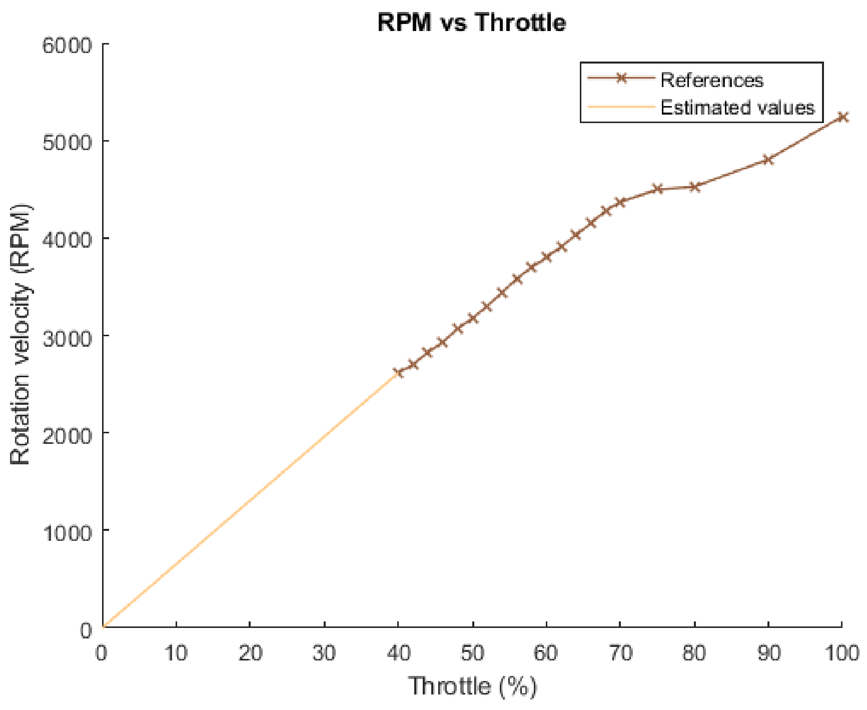


Fig. 4.8 Rotation speed of vertical flight motors with respect to the throttle

Once we have calculated the speed, we just have to send the command to the engine to turn at the indicated speed.

In the case of control surfaces, it is easier. From the real model of V1 we know that the control surfaces move in a range of +/- 20º. To calculate the position of the surface, we consider that -20º is equivalent to 0% and 20º is equivalent to 100%. It is therefore very easy to calculate the position of the control surface with the following equation:

$$Position\ (º) = \left(\frac{SITL\ control\ surface\ output}{100} * 40\right) - 20º \qquad \textbf{(4.3)}$$

It is very simple to calculate the required values in the fdm_packet package, as Gazebo has an inertial sensor that is built into the vehicle. It is possible set the type of noise, the standard deviation, the mean, the mean of bias and the standard deviation of bias. Linear acceleration and angular velocity can be extracted directly from this inertial sensor. The other parameters (orientation quaternion, velocity and position in the x-, y- and z- axis) can be extracted directly from values given by Gazebo, without the need to incorporate any sensors. In this section, we only save the information in a package of the type fdm_packet. This package is then sent in the part of ArduPilot, as explained before.

As we have seen in the Gazebo part, we move the engines and control surfaces, but neither the V1 or the Gazebo standard VTOL are able to fly only with this. To fly, an aerodynamic Plugin created by Gazebo is used. As we said before, this Plugin is called LiftDrag.

4.3.3.1 LiftDrag Plugin

This is a Plugin created by Gazebo, which implements the lift and drag effects on any Gazebo model [26]. When a body moves within a fluid, a fluid-dynamic force appears on the body (that is, a force that the fluid exerts on the body). This force can be decomposed in two components: the lift is the component in a direction perpendicular to the movement of the body, while the drag is the component aligned with the movement of the body but in the opposite direction (see Fig. 4.9).

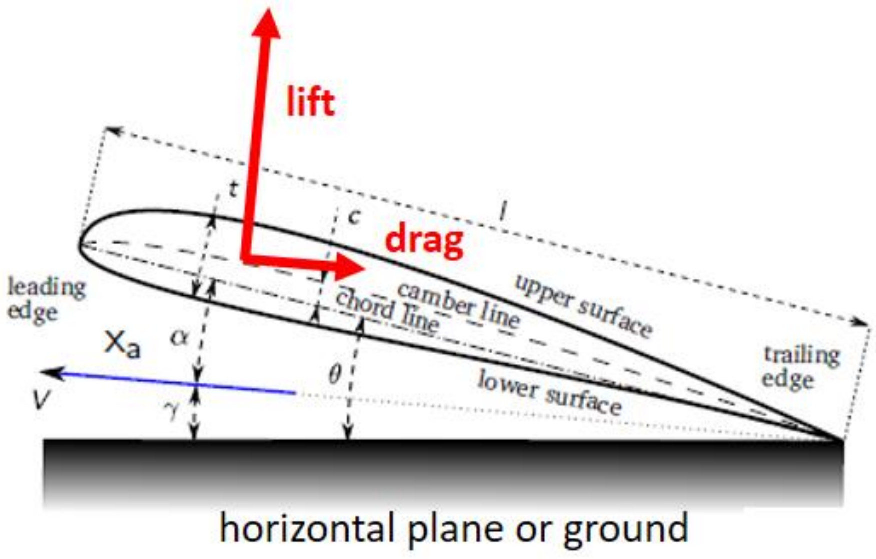


Fig. 4.9 Lift and drag forces acting on airfoil

Therefore, with this Plugin the UAV movement will be generated, as it calculates the forces generated by the rotation of the engines, the forces generated by the fuselage of the drone, and the movements of the control surfaces.

In order to calculate the aerodynamic effects of the propellers and the body of the UAV, we need some aerodynamic input parameters:

- Area
- α0 → initial alpha. Initial angle of attack. The angle of attack is the angle between the wind direction and the chord line of an airfoil.
- Cla → coefficient of lift before stall
- Cda → coefficient of drag before stall
- Cp → centre of pressure. Where forces are applied
- α stall → angle of attack at stall point
- Cla stall → Cla after stall
- Cda stall → Cda after stall
- ρ → air density

The values for cl and cd are extracted from the graph cl/cd vs alpha. The values of cl and cd depend on the slope of the curve of the lift coefficient or of the drag coefficient respectively. Cla and Cda correspond to the slope of the curve before entering stall zone. Cl stall and Cd stall correspond to the slope of the curve after stall point.

To better understand it, see Fig. 4.10, which shows the curves of lift and drag coefficient as a function of the angle of attack.

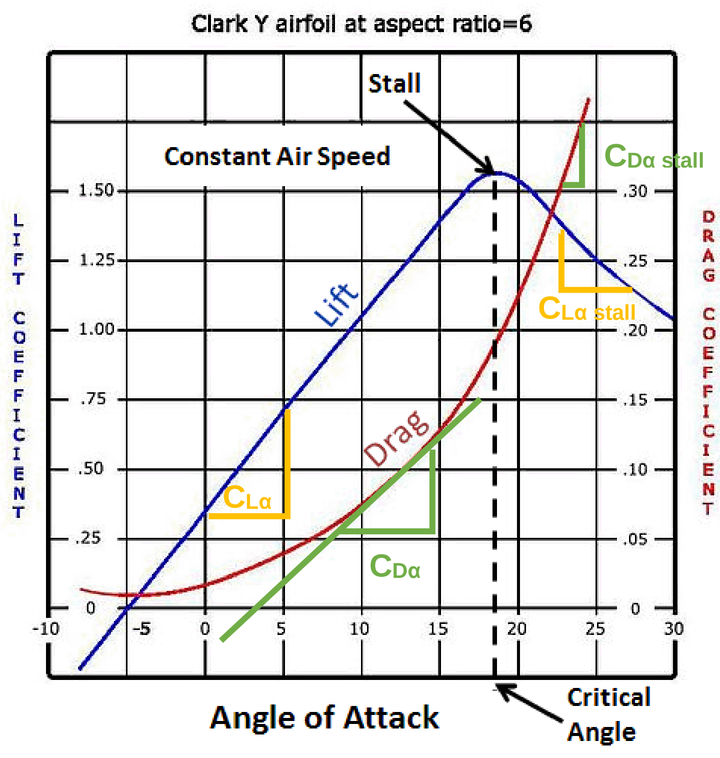


Fig. 4.10 Lift and Drag Coefficient vs Angle of attack

From these parameters and the real-time conditions of the flight, we calculate the aerodynamic forces generated by each element (lift and drag) and the moment.

The formulas used by the Plugin to calculate the lift and drag [27] for each UAV propeller and control surface are:

$$Lift = Cl * \frac{\rho * V^2}{2} * A \tag{4.4}$$

$$Drag = Cd * \frac{\rho * V^2}{2} * A \tag{4.5}$$

The values of Cl and Cd will vary according the angle of attack and the geometric shape of the vehicle. The results of equations 4.4 and 4.5 are the forces generated by the propellers, fuselage and control surfaces of the simulated UAV. The direction in which these forces are applied must be taken into account.

4.3.4 Main side

This is the central part of all the software designed. It is responsible for calling all the functions at the right time for the proper functioning of the Plugin. As expected, the first thing this part does is to initialize the other three parts and connections, that is, establish a connection to the Gazebo model, initialize the ROS node, and establish the connection with SITL.

Once the three parts have been initialized and a connection has been established, the execution of an infinite loop begins, in which the same actions are performed iteratively again and again. This loop will remain active as long as the SITL simulation is active.

The workflow is described in the diagram shown in Fig. 4.11.

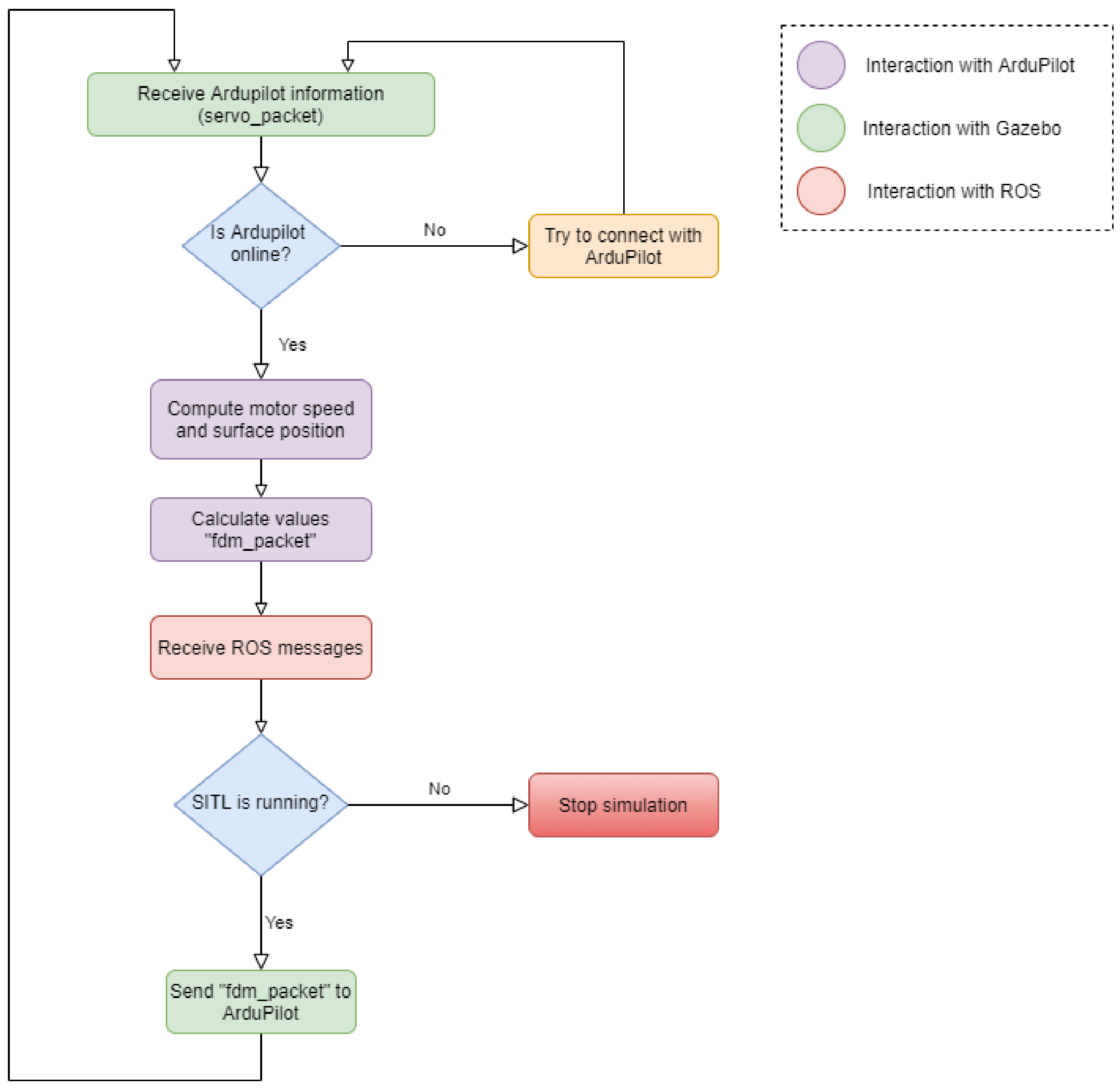

Receive Ardupilot information (servo_packet)
Is Ardupilot online?
No
Try to connect with ArduPilot
Yes
Compute motor speed and surface position
Calculate values "fdm_packet"
Receive ROS messages
SITL is running?
No
Stop simulation
Yes
Send "fdm_packet" to ArduPilot
Interaction with ArduPilot
Interaction with Gazebo
Interaction with ROS

Fig. 4.11 Developed software flowchart

4.4 Results

Once all the elements involved have been described, and the operation of the software that will control the simulation has been explained, the results can be presented. To see if the simulator works well, three tests have been planned for each of the existing QuadPlane models. The first test consists of a ground check: its purpose is to see that the motors rotate properly (through Mission Planner you can test the four motors that are responsible for the vertical movement) and that the control surfaces go to the starting position. The second test consists of the planning of a mission through Mission Planner and the consequent execution by the UAV. The last test consists of the execution of a commanded flight through a radio control.

With the three proposed tests, it will be possible to see if the operation of the simulator is correct or not. The simulator will work well if the simulated models are able to follow an operation just like a real QuadPlane would, that is, vertical take-off, transition to horizontal flight, then transition to vertical flight and landing. Since we have two QuadPlane models, we will start testing the Gazebo Standard VTOL model, as it will be error-free, in terms of physical properties such as inertia matrices and mass of the model, due to the fact that it was designed by Gazebo. Then, if the test results are good, we will repeat the tests with the Venturi V1 model. In the case of the planned mission, we will quantify the error between the desired trajectory and the trajectory that the simulated model follows. In the case of the mission controlled with the joystick we will see if the simulated UAV follows the commands that we send it with the joystick in the same way as a real UAV does.

4.4.1 Standard VTOL

4.4.1.1 Ground test

When the simulator is started, the control surfaces drop due to gravity. As soon as the SITL starts running and connects to the simulator, the control surfaces go to the initial position. As can be seen in Fig. 4.12, the control surfaces are placed in the initial position.

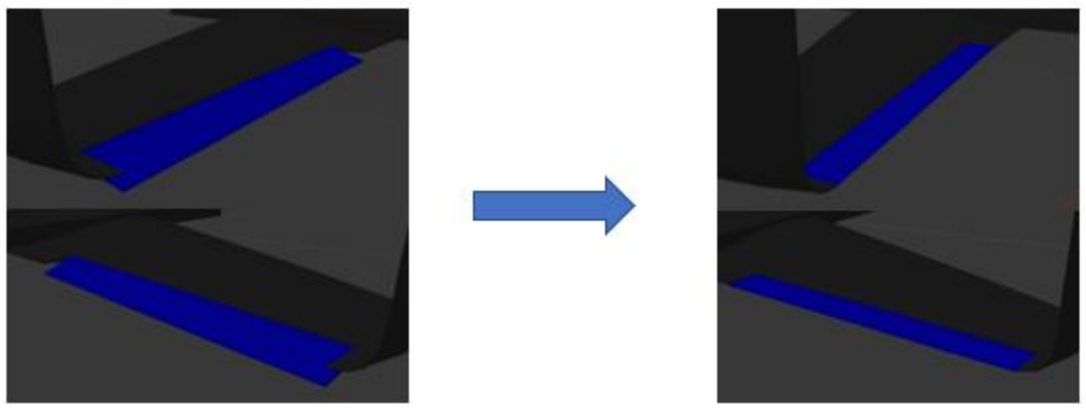


Fig. 4.12 Ailerons moving to initial position: before SITL runs (left) and during SITL simulation (right)

If we perform the engine rotation test through the Mission Planner, we see how each rotor rotates in the corresponding direction. To see the results, watch the video in section 2.1.1. of the annex.

Once we see that the control surfaces are in the initial position, we start the engine test. As each rotor responds to the spin request and spins in the correct direction, we consider that the motor test has been passed correctly. So, the ground test is given as satisfactory.

4.4.1.2 Planned Mission

For this test, we plan a simple mission, which is planned and sent through Mission Planner. A lay-out view of the trajectory of the drone is shown in Fig 4.13.

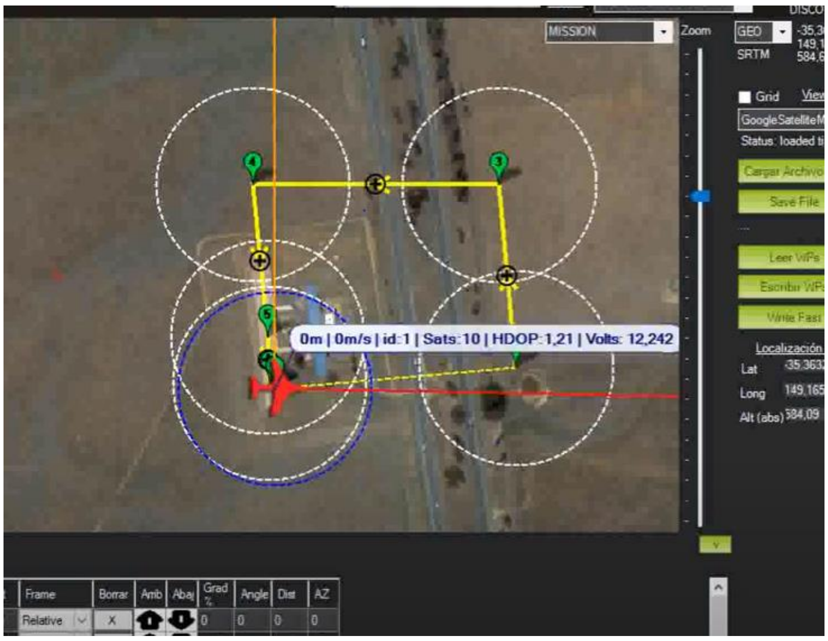


Fig. 4.13 Mission trajectory on Mission Planner

As can be seen in Fig. 4.13, the flight trajectory has approximately a square shape. The drone first rises to a height of 50 meters and then begins the transition to horizontal flight. After completing its square-shaped trajectory, it finally lands at the same point it took off.

As can be seen in the video section 2.1.2. of the annex, the drone follows the previously mentioned trajectory; particularly, it is able to take off, make the transition correctly, follow the planned trajectory, and finally land.

To see how the drone has made the transition, the speed of its five engines is analysed. In the video, you can see how the cruise motor starts to spin faster at the same time that the quad motors start to reduce their rpm. Finally, only the cruise motor works when the UAV goes from vertical flight to horizontal flight.

When going from horizontal to vertical flight it goes the other way around, the quad motors begin to rotate at the same time that the cruise motor stops rotating.

The trajectory of the drone in the planned mission can be seen in Fig. 4.14. From this figure, we can see that the trajectory is drawn in the geodesic coordinate system: the axes are longitude and latitude. To calculate the distance error between the actual trajectory followed by the UAV and the planned trajectory, we have to represent the trajectory in the local NED coordinate system, where the units are measured with meters.

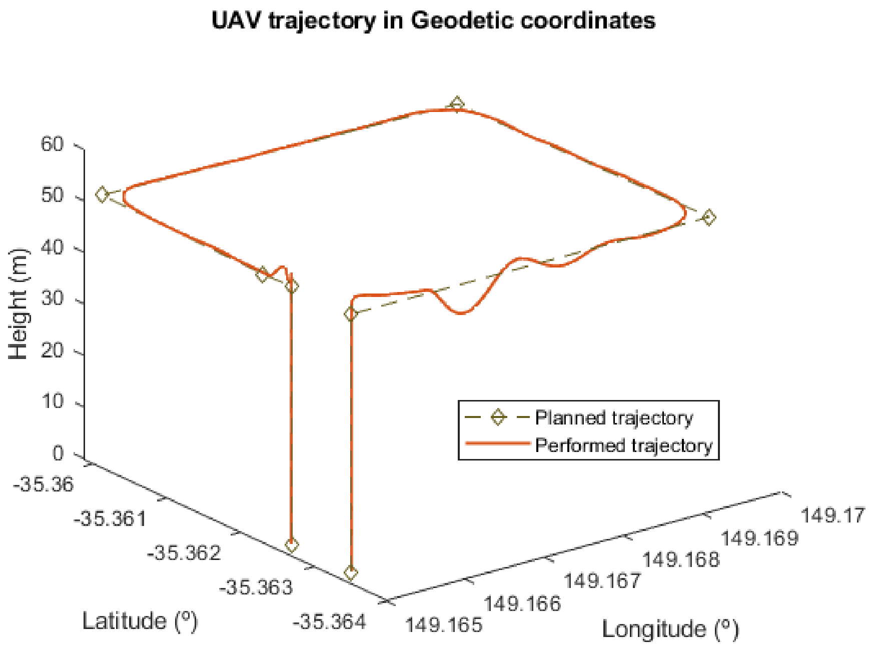


Fig. 4.14 UAV trajectory in planned mission

To calculate the distance error between the followed path and the planned one, we will calculate the distance between each point of the followed path and the segment formed by two consecutive waypoints, as can be seen in Fig. 4.15.

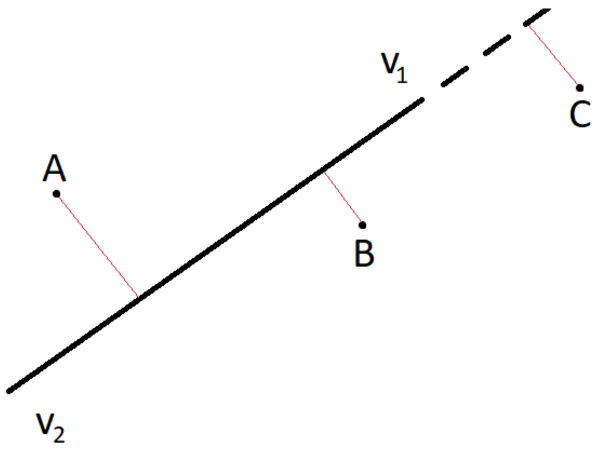


Fig. 4.15 Graphical visualization of distance between point and segment

In Fig. 4.16, we can see represented the path followed by the UAV in the NED coordinate system. The colour code of the path depends on the distance between each point and the corresponding segment. This distance is the error of the path at each point. As we can see, the largest errors are found at the beginning of the horizontal flight, in the turns, and at the end of the horizontal flight. The errors at the beginning and at the end are likely due to the transition (a critical moment of the flight of VTOLs, as mentioned earlier). The errors in the turns are due to the drone always turning before reaching a waypoint to search for the next one. The mean distance between the followed path and the planned path is 1.9 m.

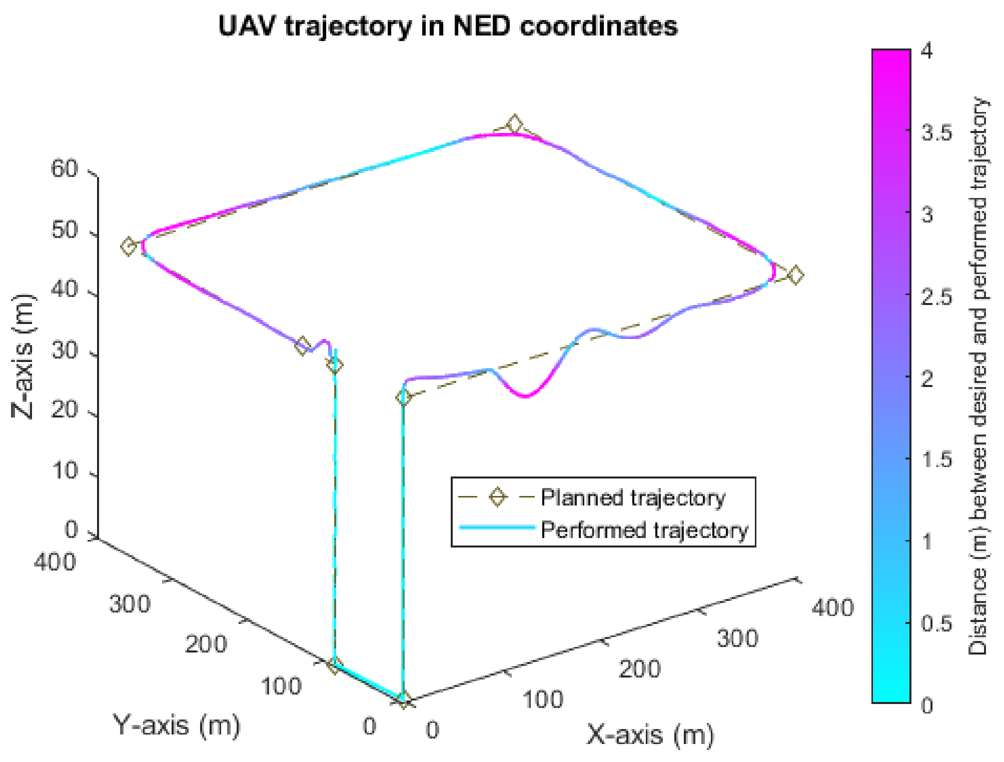


Fig. 4.16 UAV trajectory in planned mission with colour bar for distances

4.4.1.3 Controlled by a joystick

As in the previous mission, the UAV's goal now is to take off to a height of 50 m, fly a square-shaped trajectory, and finally land, with the only difference with respect to the previous case that this time the drone is controlled by a radio control.

As can be seen in the video in section 2.1.3. of the annex, the QuadPlane is able to follow the commands sent by the radio control, which is linked to it via Mission Planner. The trajectory of the UAV is not as accurate in this test as in the previous test, since we introduce human error, which is why the trajectory does not follow as accurately the square as in the previous test.

During this test the UAV has responded quickly and fluently to the joystick controls. The QuadPlane has understood the flight mode changes that I have made via radio control and has made the transitions well.

4.4.2 Venturi V1

4.4.2.1 Ground test

We now perform on the V1 the same tests as for the Standard VTOL. In the same way that happened before, when the model is loaded into Gazebo, its control surfaces appear dropped due to gravity (see Fig. 4.17 top). When we connect with the SITL, the control surfaces move to their initial position. The change in position of the control surfaces, once the Gazebo model has been connected to the SITL, can be clearly seen in Fig. 4.17 bottom.

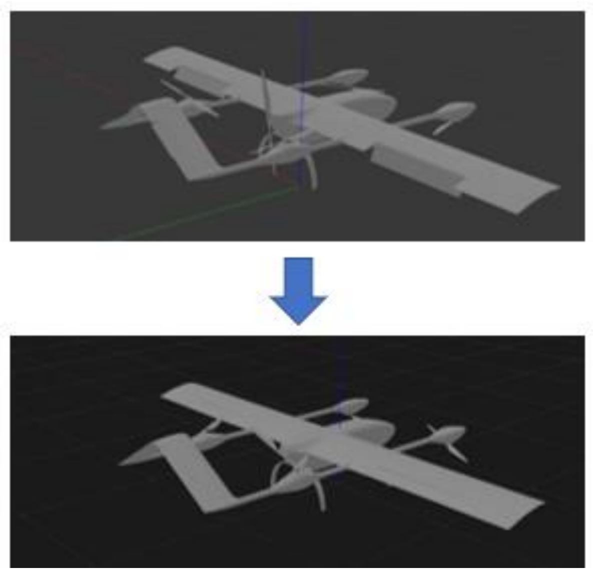


Fig. 4.17 Control surfaces moving to initial position for the V1 UAV model

If we perform the Mission Planner engine rotation test, the engines of the Gazebo model respond and rotate without any problems, in the same way as for the Standard VTOL model. This test can be seen in section 2.2.1. of the annex.

4.4.2.2 Planned mission

We plan a mission with a square flight trajectory, just as we did for the previous model. In this case, when the V1 QuadPlane tries to take off, it tilts too much, it is not able to take off in a stabilized way, and the mission is aborted. Many attempts are done to repeat the mission but unfortunately the same result is obtained always: in no occasion the SITL is able to stabilize the UAV and let it fly. The results of the flight test can be seen in section 2.2.2. of the annex.

Obviously, if the UAV is not able to pass this test, it will not be able to pass the mission test following the commands of a radio control due to problem it cannot be solved with joystick control as the flight controller acts in the same way as for a planned mission. As we have seen with the tests in the previous model, the simulator works correctly, the only difference between these tests and the previous ones is the model with which they are being performed.

The Gazebo model of the Venturi V1 was not made manually or natively in Gazebo, as explained earlier, it was made with a SolidWorks to Gazebo conversion tool. A Gazebo model must have its physics well defined, but in this case, as it is a UAV model with very complex geometries, it is difficult to correctly determine the matrix of moments of inertia, one of the key elements for the Gazebo simulation. Also, the calculation of the mass of the V1 model in SolidWorks does not correspond to the value of the mass of the V1 in reality. All of these reasons may cause the drone to tilt as it attempts to take off.

Before finding out what the problems were, it was thought that it might be a problem with the SITL stability control, particularly, that it was not able to control the UAV because of a problem with the UAV moments. The V1 has a tail, and this causes the CG of the drone to shift backwards. When the drone tries to take off, the CG is closer to the rear engines and therefore this generates a moment which may be responsible for the observed tilt, as it always tilts backwards. To explain all this, a study has been done in Matlab on the moments of the V1.

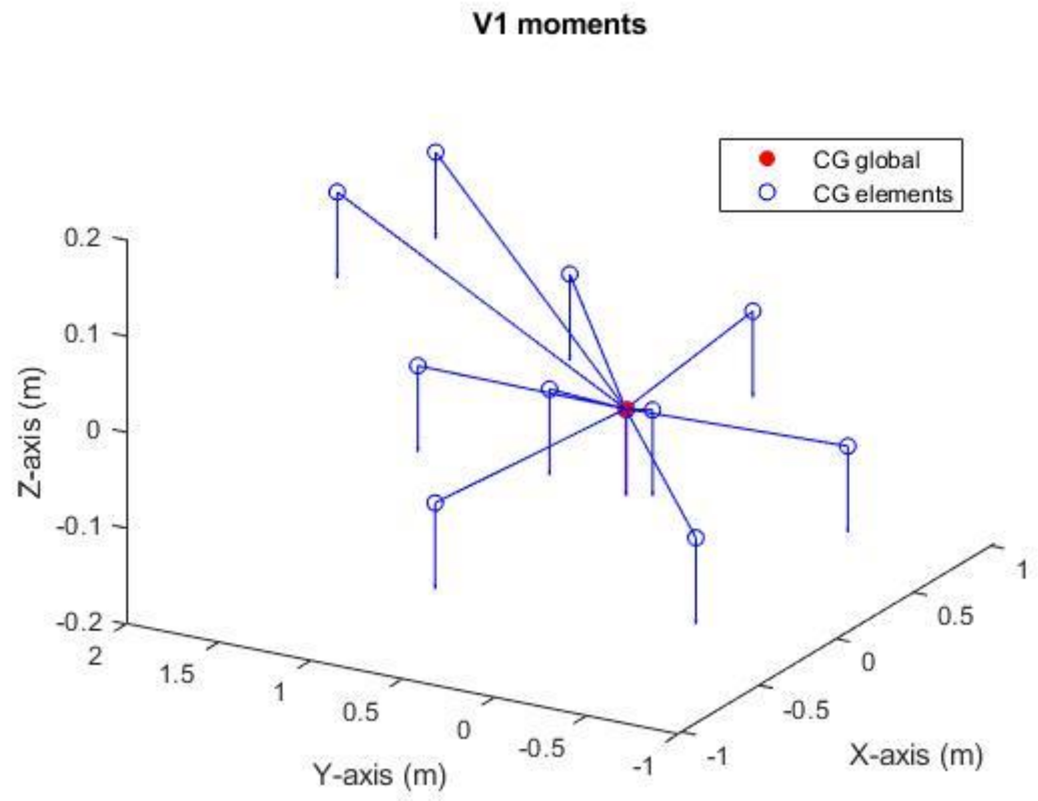


Fig. 4.18 Centre of gravity (CG) for each V1 element (motors, wings, tail) and global CG

As we see in Fig. 4.18, the global centre of gravity (CG) is located at the coordinates (7.58 × 10^{-5}, 0.1469, -0.0046) and the obtained moment respect to this point is (5.62 x 10^{-16}, 11.86 x 10^{-17}, 0), that is to say, 0, as expected in the CG.

If we add force in our motors, for example, a force of 6 N in the direction in which the motors create thrust, the sum of moments in the CG is (-0.9436, 0.0018, 0). The moments have increased considerably, especially on the x-axis. On the y-axis, the resulting moment is very close to 0, which may be the result of some decimal lost in the calculation.

In order to obtain null moment in the CG, the difference of the moments of the front and rear motors is calculated with respect to the CG. Once we have the difference of moments, and as we have the distance of the motors to the CG, we can find the extra strength that the rear engines need to compensate for the torque. The equations are as follows:

$$\Delta \mathrm{M_x} = \mathrm{M_{x,motor0}} - \mathrm{M_{x,motor1}} \tag{4.6}$$

$$\Delta M_y = M_{y,motor0} - M_{y,motor1} \tag{4.7}$$

$$\mathrm{F'_z} = \frac{\Delta \mathrm{M_x}}{\mathrm{d_{CG\rightarrow rotor1,y}}} \tag{4.8}$$

$$\mathrm{F'_z} = -\frac{\Delta \mathrm{M_y}}{\mathrm{d_{CG\rightarrow rotor1,x}}} \tag{4.9}$$

$$M\ total\ compensated_{\mathrm{x}} = Mtotal_x + F'_z * d_{CG\rightarrow rotor1,y} + F'_z * d_{CG\rightarrow rotor3,y} \tag{4.10}$$

$$M\ total\ compensated_{y} = Mtotal_y + F'_z * d_{CG\rightarrow rotor1,x} + F'_z * d_{CG\rightarrow rotor3,x} \tag{4.11}$$

In Equations 4.6 and 4.7 we calculate the difference in moments. Equations 4.8 and 4.9 are used to calculate the force on the z-axis generated by the difference in moments, that is, the force that motors 1 and 3 need to compensate for those moments. These equations come from the determinant of the matrix of moments.

$$\vec{M} = \begin{bmatrix} \vec{\imath} & \vec{\jmath} & \vec{k} \\ R_x & R_y & R_z \\ F_x & F_y & F_z \end{bmatrix} \tag{4.12}$$

$$\vec{\mathrm{M}} = \left(\mathrm{R_y F_z} - \mathrm{R_z F_y}\right)\vec{\mathrm{\imath}} - (\mathrm{R_x F_z} - \mathrm{R_z F_x})\vec{\mathrm{\jmath}} + (\mathrm{R_x F_y} - \mathrm{R_y F_x})\vec{\mathrm{k}} \tag{4.13}$$

As we can see in equations 4.12 and 4.13, we have the formula to calculate the components of the moment vector in each axis. Consider the forces on the x- and y-axes are null. Therefore, to calculate whether the moments are compensated with the new forces for motors 1 and 3, we use equations 4.10 and 4.11.

The resulting moment if we add force to the rear engines is (-0.0026,0.0017,0). As we see, the moment is virtually null. Therefore, it is verified that the method explained with equations 4.6-4.13 works. Please, see these calculations, in section 3.2. of the annex.

Unfortunately, if we do a flight test and rectify the forces of the rear engines following the method explained, still the V1 is unable to fly. Hence, we have shown that the problem is not that the SITL is not able to counterbalance these moments, as we have just fixed it ourselves and it has not worked. Therefore, the error likely comes from the generated model, that is, the inertia matrices generated in the transformation of the SolidWorks model to Gazebo are not good and this causes this error in the flight test. It is very difficult to fix this error because it is very difficult to calculate the inertia matrix of a body with such a difficult geometry.

CONCLUSIONS

The purpose of this work is to create a flight simulator for a VTOL. The choice to base a final master's thesis on the development of a simulator for a VTOL is due to the real needs of Venturi Unmanned Technologies. Venturi is a start-up that is designing drones for the inspection of power lines and pipelines. To develop a reliable drone, it is necessary to perform many configuration tests and do many hours of flight with the actual platform. One of the problems of the company is precisely when it comes to flying the drone: due to the laws, which are very restrictive, they have to fly the UAV in special conditions. The place where these conditions can be recreated is far from the location of the business. This makes it very expensive to perform flight tests in terms of logistics and economics.

The goals that were set at the beginning of the work were two. First of all, make a simulator that follows the flight mechanics of a VTOL drone, because, if you want to test special configurations, the simulator must be able to replicate the flight modes of a VTOL. The second goal was to develop a simulator that allows controlling the simulated UAV with a radio control or a joystick.

Thanks to this simulator, the company will reduce costs and time in testing new systems and equipment that a VTOL can carry, such as cameras and a LIDAR. Also, with the simulator, Venturi will be able to train new pilots without having to leave the office. Novice pilots can learn to configure missions, understand how the basic controls of the VTOL work and its flight modes. This will save the time it takes to travel to the location where drones can be flown in accordance with the law, and will save the cost of booking this airspace.

First of all, we have seen what types of VTOLs currently exist. There are three types, tailsitters, tiltrotors and QuadPlanes. We have seen the flight phases that a VTOL follows. We have analysed the advantages of using a VTOL against a multirotor or a fixed wing. Finally, we have seen the type of VTOL that Venturi V1 is and its features; particularly, Venturi's VTOL V1 can be classified as QuadPlane.

After this analysis, the current European regulation has been studied. Knowing the current law, we understand where the UAV designed by Venturi is and we understand what are the difficulties that Venturi has to be able to fly the drone and what are the conditions that allow us to do so.

Finally, in order to start developing the simulator, an in-depth study has been done of the technologies that are being used today to perform simulations. This analysis is one of the most important parts of this work, as the choice of technology must be correct to be able to develop the simulator and be able to meet the established objectives. As a result of this analysis, the software tool selected to develop the simulator is Gazebo, which is a tool for performing 3D simulations of any type of robot.

Up to five flight tests have been performed for two types of QuadPlanes models: a model created by Gazebo and a model that represents Venturi's V1. As a result of the flight tests on the first model, it is concluded that the simulator works correctly. Particularly, it has been possible to carry out two flight operations correctly, where the UAV behaved as a real QuadPlane. In these operations, the simulated UAV took off, made the transition to horizontal flight, made horizontal flight, made a transition to vertical flight and finally landed. The first operation was carried out autonomously by the UAV, following a pre-planned trajectory with an average error of 1.9 m. In the second mission, the drone was controlled by a joystick. The UAV responds to commands originated by the joystick without any problem, in the same way it should in a real operation.

The only problem with the flight tests was that it was not possible to fly with the simulated model of the Venturi V1. It has been determined that the error casuing this problem was originated in the process of transforming a SolidWorks model into a Gazebo model. Specifically, the error is found in the inertia matrices. An attempt has been made to solve this problem by correcting the forces generated by the V1's rear engines. Unfortunately, despite this engine power corrections, the SITL has not been able to take off the V1 in a stabilized manner.

However, as a result of this work and during the realization of this work, new project proposals have emerged for the company. First and foremost, after seeing how the simulator works, Venturi is considering the migration from ArduPilot V1 flight controller software to PX4. Another project that has emerged with the simulator is to integrate the simulation with a computer vision project that is being developed at Venturi. The computer vision system has two goals: the first one is the detection of people, and the second one is the detection of electrical towers and power lines. Pedestrian detection will be used for the landing phase, since landing is often autonomous and far from the take-off point. The aim is to detect people before landing, calculate the position of these people, calculate the safest landing zone, and make the UAV land in the safe zone. The detection of electrical towers is intended to detect the high voltage lines that join two electrical towers. Once the UAV detects these lines, it will be able to follow the power lines autonomously. In addition to following the power lines, it will take images of the elements of the tower that the potential users of the drone want to analyse to see if there is any failure or defect.

In order to integrate these systems into the simulation, a camera and gimbal have to be added to the simulated UAV. In addition, the world where the simulation is carried out must be modified so that pedestrians and electrical lines appear. Fig. C.1 shows an example of pedestrian and power lines models.

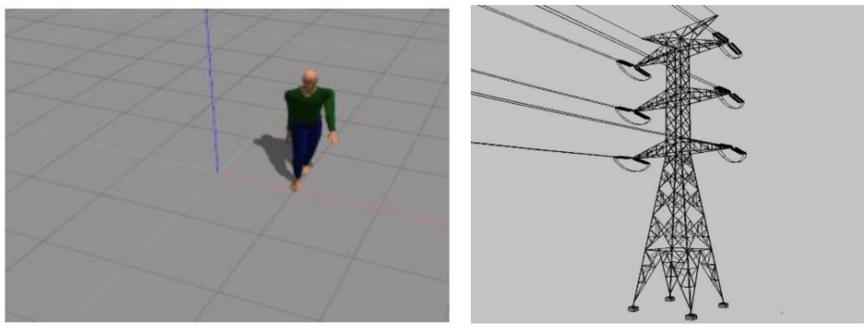


Fig. C.1 Pedestrian model (left) and Electric Tower model (right)

ACRONYMS

AC	Alternate Current
BVLOS	Beyond Visual Line of Sight
CG	Centre of Gravity
DC	Direct Current
EU	European Union
GCS	Ground Control Station
HITL	Hardware In the Loop
MTOM	Maximum Take Off Mass
ROS	Robot Operating System
RPAS	Remotely Piloted Aircraft System
SESAR JU	Single European Sky Air Traffic Management Research Joint Undertaking
SITL	Software In the Loop
UA	Unmanned Aircraft
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
VLOS	Visual Line Of Sight
VTOL	Vertical Take-Off and Landing

REFERENCES

[1] “History of ArduPilot – Dev documentation”, ArduPilot, https://ardupilot.org/dev/docs/common-history-of-ardupilot.html [last accessed: 2/9/2020].

[2] “Airframe Reference – PX4 v1.9.0 User Guide”, 30/03/2020, PX4, https://docs.px4.io/v1.9.0/en/airframes/airframe_reference.html#vtol [last accessed: 7/9/2020].

[3] “Tailsitter Planes – Plane documentation”, ArduPilot, https://ardupilot.org/plane/docs/guide-tailsitter.html#center-of-gravity [last accessed: 31/8/2020].

[4] “Tilt Rotor Planes – Plane documentation”, ArduPilot, https://ardupilot.org/plane/docs/guide-tilt-rotor.html#the-tilt-type [last accessed: 31/8/2020].

[5] “QuadPlane tips – Plane documentation”, ArduPilot, https://ardupilot.org/plane/docs/quadplane-tips.html#increasing-yaw-authority [last accessed: 1/9/2020].

[6] Official Journal of the European Union “COMMISSION DELEGATED REGULATION (EU) 2019/945”

[7] Official Journal of the European Union “COMMISSION IMPLEMENTING REGULATION (EU) 2019/947”

[8] “The history of Pixhawk | Auterion”, Auterion, https://auterion.com/company/the-history-of-pixhawk/ [last accessed: 1/9/2020].

[9] “HITL - Paparazzi UAV”, 24/11/2018, Paparazzi UAV, https://wiki.paparazziuav.org/wiki/HITL [last accessed: 7/9/2020].

[10] “Using SITL with X-Plane 10 – Dev documentation”, ArduPilot https://ardupilot.org/dev/docs/sitl-with-xplane.html# [last accessed: 9/9/2020].

[11] “RealFlight® 9.5 RC Flight Simulator - Now with more than 170 different aircraft to fly!”, Horizon Hobby, LLc, https://www.realflight.com/index.php [last accessed: 9/9/2020].

[12] “Gazebo”, Open Source Robotics Foundation, http://gazebosim.org/ [last accessed: 9/9/2020].

[13] “jMAVSim with SITL – PX4 v1.9.0 Developer Guide”, PX4, https://dev.px4.io/v1.9.0/en/simulation/jmavsim.html [last accessed: 9/9/2020].

[14] “JSBSim Open Source Flight Dynamics Model”, http://jsbsim.sourceforge.net/ [last accessed: 9/9/2020].

[15] “Introduction – FlightGear Flight Simulator”, https://www.flightgear.org/about/ [last accessed: 9/9/2020].

[16] AirSim, Microsoft, https://github.com/microsoft/AirSim [last accessed: 9/9/2020].

[17] “The most powerful real-time 3D creation platform - Unreal Engine”, Epic Games, https://www.unrealengine.com/en-US/ [last accessed: 9/9/2020].

[18] “Gazebo: Tutorial: Make a model”, Gazebo, http://gazebosim.org/tutorials?tut=build_model [last accessed: 1/9/2020].

[19] “Gazebo. Tutorial: Plugins 101”, Gazebo http://gazebosim.org/tutorials/?tut=plugins_hello_world [last accessed: 1/9/2020].

[20] Edpresso Team, “What is C++?”, Educative, https://www.educative.io/edpresso/what-is-cpp [last accessed: 7/9/2020].

[21] Amanda Dattalo, “ROS/Introduction – ROS Wiki”, 08/08/2018, ROS http://wiki.ros.org/ROS/Introduction [last accessed: 7/9/2020].

[22] “Complete parameter list – Plane documentation”, ArduPilot, https://ardupilot.org/plane/docs/parameters.html#servo-parameters [last accessed: 7/9/2020].

[23] “QuadPlane Frame setup – Plane documentation”, ArduPilot, https://ardupilot.org/plane/docs/quadplane-frame-setup.html#motor-ordering [last accessed: 7/9/2020].

[24] Github – SITL code from ArduPilot (SIM_Gazebo.cpp), 15/08/2019, https://github.com/ArduPilot/ardupilot/blob/master/libraries/SITL/SIM_Gazebo.cpp [last accessed: 7/9/2020].

[25] Github – SITL code from ArduPilot (SIM_Gazebo.h) , 15/08/2019, https://github.com/ArduPilot/ardupilot/blob/master/libraries/SITL/SIM_Gazebo.h [last accessed: 7/9/2020].

[26] “Gazebo: Tutorial: Aerodynamics”, Gazebo, http://gazebosim.org/tutorials?tut=aerodynamics&cat=physics [last accessed: 11/9/2020].

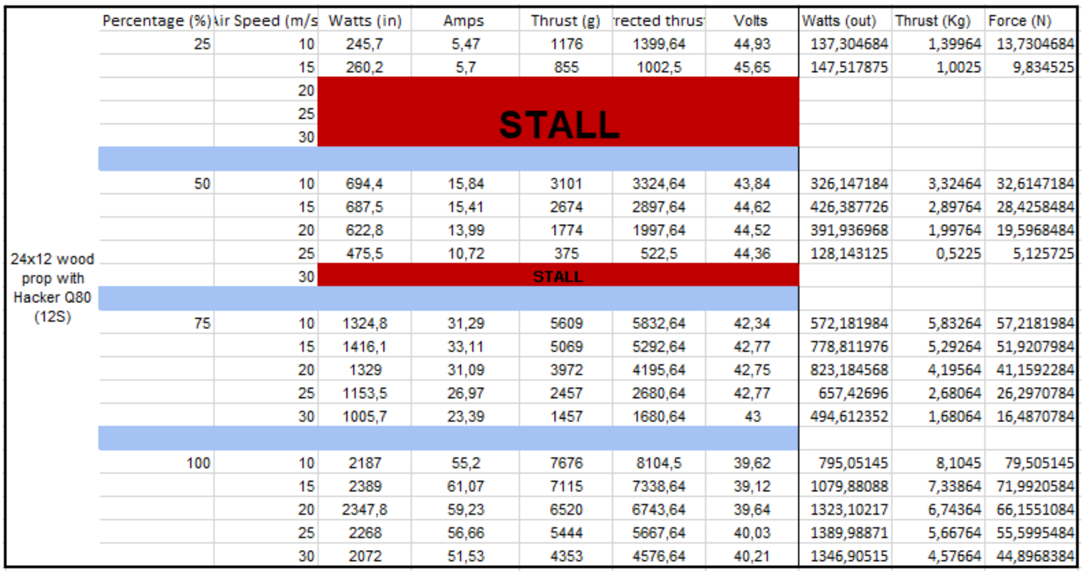
[27] Fadri Furrer, Michael Burri, Mina Kamel, Janosch Nikolic, Markus Achtelik, ROS – LiftDrag Plugin code, 6/06/2019, http://docs.ros.org/melodic/api/rotors_gazebo_plugins/html/liftdrag__plugin_8cpp_source.html [last accessed: 11/9/2020].

ANNEX

1. Motors specifications

1.1. Hacker Q80-13S 28-Pole kv175

Fig. A.1 shows the specifications of the engine Hacker Q80-13S 28-Pole kv175.

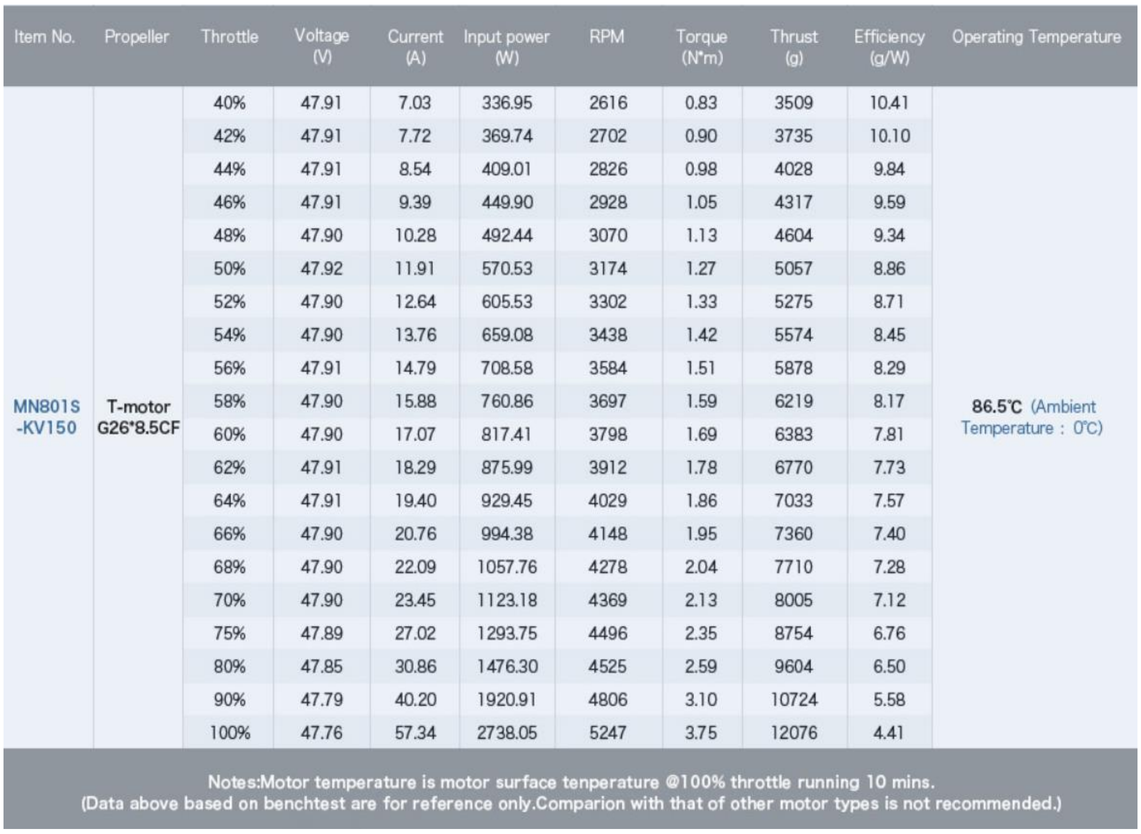


	Percentage (%)	Air Speed (m/s	Watts (in)	Amps	Thrust (g)	rected thrus	Volts	Watts (out)	Thrust (Kg)	Force (N)
	25	10	245,7	5,47	1176	1399,64	44,93	137,304684	1,39964	13,7304684
		15	260,2	5,7	855	1002,5	45,65	147,517875	1,0025	9,834525
		20	STALL							
		25								
		30								
	50	10	694,4	15,84	3101	3324,64	43,84	326,147184	3,32464	32,6147184
		15	687,5	15,41	2674	2897,64	44,62	426,387726	2,89764	28,4258484
		20	622,8	13,99	1774	1997,64	44,52	391,936968	1,99764	19,5968484
24x12 wood prop with Hacker Q80 (12S)		25	475,5	10,72	375	522,5	44,36	128,143125	0,5225	5,125725
		30	STALL							
	75	10	1324,8	31,29	5609	5832,64	42,34	572,181984	5,83264	57,2181984
		15	1416,1	33,11	5069	5292,64	42,77	778,811976	5,29264	51,9207984
		20	1329	31,09	3972	4195,64	42,75	823,184568	4,19564	41,1592284
		25	1153,5	26,97	2457	2680,64	42,77	657,42696	2,68064	26,2970784
		30	1005,7	23,39	1457	1680,64	43	494,612352	1,68064	16,4870784
	100	10	2187	55,2	7676	8104,5	39,62	795,05145	8,1045	79,505145
		15	2389	61,07	7115	7338,64	39,12	1079,88088	7,33864	71,9920584
		20	2347,8	59,23	6520	6743,64	39,64	1323,10217	6,74364	66,1551084
		25	2268	56,66	5444	5667,64	40,03	1389,98871	5,66764	55,5995484
		30	2072	51,53	4353	4576,64	40,21	1346,90515	4,57664	44,8968384

Fig. A.1 Hacker Q80-13S 28-Pole kv175 engine specifications

1.2. T-motor MN801S 150kv

Fig. A.2 shows the specifications of the engine T-motor MN801S 150kv.



Item No.	Propeller	Throttle	Voltage (V)	Current (A)	Input power (W)	RPM	Torque (N*m)	Thrust (g)	Efficiency (g/W)	Operating Temperature
MN801S -KV150	T-motor G26*8.5CF	40%	47.91	7.03	336.95	2616	0.83	3509	10.41	86.5℃ (Ambient Temperature : 0℃)
		42%	47.91	7.72	369.74	2702	0.90	3735	10.10	
		44%	47.91	8.54	409.01	2826	0.98	4028	9.84	
		46%	47.91	9.39	449.90	2928	1.05	4317	9.59	
		48%	47.90	10.28	492.44	3070	1.13	4604	9.34	
		50%	47.92	11.91	570.53	3174	1.27	5057	8.86	
		52%	47.90	12.64	605.53	3302	1.33	5275	8.71	
		54%	47.90	13.76	659.08	3438	1.42	5574	8.45	
		56%	47.91	14.79	708.58	3584	1.51	5878	8.29	
		58%	47.90	15.88	760.86	3697	1.59	6219	8.17	
		60%	47.90	17.07	817.41	3798	1.69	6383	7.81	
		62%	47.91	18.29	875.99	3912	1.78	6770	7.73	
		64%	47.91	19.40	929.45	4029	1.86	7033	7.57	
		66%	47.90	20.76	994.38	4148	1.95	7360	7.40	
		68%	47.90	22.09	1057.76	4278	2.04	7710	7.28	
		70%	47.90	23.45	1123.18	4369	2.13	8005	7.12	
		75%	47.89	27.02	1293.75	4496	2.35	8754	6.76	
		80%	47.85	30.86	1476.30	4525	2.59	9604	6.50	
		90%	47.79	40.20	1920.91	4806	3.10	10724	5.58	
		100%	47.76	57.34	2738.05	5247	3.75	12076	4.41	

Notes:Motor temperature is motor surface tenperature @100% throttle running 10 mins.
(Data above based on benchtest are for reference only.Comparion with that of other motor types is not recommended.)

Fig. A.2 T-motor MN801S 150kv engine specifications

2. Flight test videos

2.1. Standard VTOL

2.1.1. Ground test

In the following link you can find the video with the first flight test. As you can see in the video, a rotational test of the motors is done through Mission Planner. first one by one and then tested in sequence.

Link to video: https://youtu.be/vJbeKo1qtyc

2.1.2. Following a mission

In the following link you can find the video with the second flight test. As you can see in the video, a mission is planned through Mission Planner. Once planned, this mission is executed. The VTOL follows the trajectory set to perfection. The only critical moment of the flight is in the final transition, and the landing flight, which is a bit unstable.

Link to video: https://youtu.be/fwcL8f6n1hM

2.1.3. Controlled by a joystick

In the following link you can find the video with the third flight test. First we enable the joystick (it is connected to the computer) and then we configure the flight modes necessary to fly in quad mode and in fixed wing mode. It can be seen how we are able to carry out a trajectory similar to the previous mission.

Link to video: https://youtu.be/Jf0Z55Kuopw

2.2. Venturi V1

2.2.1. Ground test

In the following link you can find the video with the first flight test but in this case with the V1. The test is carried out in the same way as for the other model.

Link to video: https://youtu.be/7FDrj2aVsvE

2.2.2. Following a mission

In the following link you can find the video with the second flight test. The test is not performed well, when you try to execute the mission that we have planned the V1 leans. As it is not possible to take off in a stabilized way, the mission is aborted.

Link to video: https://youtu.be/muKVKo9Esto

3. Simulator codes

3.1. Plugin code

Due to the quantity and size of the simulator files, a link is attached to view all the code developed.

Link to code: https://github.com/slucasm/venturi_ardupilot_gazebo_plugin

3.2. Matlab code

This section presents the code that was written to find the balance of moments in the centre of gravity of the V1 to try to stabilize the flight.

```
% Storing data in struct
struct(1).name = 'imu';
struct(1).position = [0 0 0];
struct(1).mass = 0.015;

struct(2).name = 'base';

struct(2).position = [7.8e-05, 0.143099, -0.005197];
struct(2).mass = 1.12552;

struct(3).name = 'rotor_0';
struct(3).position = [-0.49257, (-0.62087-0.028277), -0.06048];
struct(3).mass = 0.000725188;

struct(4).name = 'rotor_1';
struct(4).position = [0.49257, (0.84087+0.023542), 0.06048];
struct(4).mass = 0.000725194;

struct(5).name = 'rotor_2';
struct(5).position = [0.49257, (-0.62087-0.028277), -0.06048];
struct(5).mass = 0.000725281;

struct(6).name = 'rotor_3';
struct(6).position = [-0.49257, (0.84087+0.023542), 0.06048];
struct(6).mass = 0.000725258;

struct(7).name = 'rotor_puller';
struct(7).position = [0, (0.54754 + 0.013947), 0];
struct(7).mass = 0.000725199;

struct(8).name = 'aileron_left';
struct(8).position = [(0.695+0.335), (0.32655-0.007343), (0.023592-0.032816)];
struct(8).mass = 0.00611893;

struct(9).name = 'aileron_right';
struct(9).position = [(-0.695-0.335), (0.32655-0.007343), (0.023592-
0.032816)];
struct(9).mass = 0.00611893;

struct(10).name = 'tail_left';
struct(10).position = [(0.1248+0.115253), (1.4491-0.066509), (0.21616-
0.024189)];
struct(10).mass = 0.00162948;

struct(11).name = 'tail_right';
struct(11).position = [(-0.1248-0.115225), (1.4491+0.066558), (0.21616-
0.024189)];
struct(11).mass = 0.00162948;

num_X = 0;
num_Y = 0;
num_Z = 0;

den_X = 0;
den_Y = 0;
den_Z = 0;

i = 1;
```

```
%Computing numerators and denominators to calculate GC

while (i <= length(struct))

    num_X = num_X + (struct(i).position(1) * struct(i).mass);
    num_Y = num_Y + (struct(i).position(2) * struct(i).mass);
    num_Z = num_Z + (struct(i).position(3) * struct(i).mass);

    den_X = den_X + struct(i).mass;
    den_Y = den_Y + struct(i).mass;
    den_Z = den_Z + struct(i).mass;

    i = i + 1;
end

%Compute GC
X_mc = num_X / den_X;
Y_mc = num_Y / den_Y;
Z_mc = num_Z / den_Z;

%Plotting all stuff
figure(1)
hold on;
i = 1;

scatter3(X_mc, Y_mc, Z_mc, 'r', 'filled');
while (i <= length(struct))

scatter3(struct(i).position(1),struct(i).position(2),struct(i).position(3),
'b');
    quiver3(struct(i).position(1),struct(i).position(2),struct(i).position(3),
0, 0, -0.1, 'b');
    plot3([X_mc struct(i).position(1)], [Y_mc struct(i).position(2)], [Z_mc
struct(i).position(3)], 'b');

    i = i + 1;
end
quiver3(X_mc, Y_mc, Z_mc, 0, 0, -0.1, 'r');

xlabel('X-axis (m)');
ylabel('Y-axis (m)');
zlabel('Z-axis (m)');
title('V1 moments');
view(-60,30);
legend('CG global', 'CG elements');

momentum_total = [0 0 0];
i = 1;



%computing sum of moments in GC
while (i <= length(struct))
```

```
    struct(i).line_cg = [struct(i).position(1)-X_mc, struct(i).position(2)-
Y_mc, struct(i).position(3)-Z_mc];

    struct(i).force = [0 0 -struct(i).mass * 9.81];
    struct(i).momentum = [struct(i).line_cg(2)*struct(i).force(3) -
struct(i).line_cg(3)*struct(i).force(2); -
(struct(i).line_cg(1)*struct(i).force(3) -
struct(i).line_cg(3)*struct(i).force(1));
struct(i).line_cg(1)*struct(i).force(2) -
struct(i).line_cg(2)*struct(i).force(1)];

    momentum_total(1) = momentum_total(1) + struct(i).momentum(1);
    momentum_total(2) = momentum_total(2) + struct(i).momentum(2);
    momentum_total(3) = momentum_total(3) + struct(i).momentum(3);
    i = i + 1;
end

force_rotors = [0 0 6];

%Compute distance from motors 1 & 3 to GC where sum of moments is zero
struct(4).position_cg_equilibrated = [struct(4).line_cg(1) (Y_mc +
abs((struct(3).position(2)-Y_mc))) struct(4).line_cg(3)];
struct(6).position_cg_equilibrated = [struct(6).line_cg(1) (Y_mc +
abs((struct(5).position(2)-Y_mc))) struct(6).line_cg(3)];

momentum_rotor0 = [struct(3).line_cg(2)*force_rotors(3) -
struct(3).line_cg(3)*force_rotors(2); -(struct(3).line_cg(1)*force_rotors(3) -
struct(3).line_cg(3)*force_rotors(1)); struct(3).line_cg(1)*force_rotors(2) -
struct(3).line_cg(2)*force_rotors(1)];
momentum_rotor1 = [struct(4).line_cg(2)*force_rotors(3) -
struct(4).line_cg(3)*force_rotors(2); -(struct(4).line_cg(1)*force_rotors(3) -
struct(4).line_cg(3)*force_rotors(1)); struct(4).line_cg(1)*force_rotors(2) -
struct(4).line_cg(2)*force_rotors(1)];
momentum_rotor2 = [struct(5).line_cg(2)*force_rotors(3) -
struct(5).line_cg(3)*force_rotors(2); -(struct(5).line_cg(1)*force_rotors(3) -
struct(5).line_cg(3)*force_rotors(1)); struct(5).line_cg(1)*force_rotors(2) -
struct(5).line_cg(2)*force_rotors(1)];
momentum_rotor3 = [struct(6).line_cg(2)*force_rotors(3) -
struct(6).line_cg(3)*force_rotors(2); -(struct(6).line_cg(1)*force_rotors(3) -
struct(6).line_cg(3)*force_rotors(1)); struct(6).line_cg(1)*force_rotors(2) -
struct(6).line_cg(2)*force_rotors(1)];

%compute new sum of moments with GC equilibrated
momentum_total(1) = momentum_total(1) + momentum_rotor0(1) +
momentum_rotor1(1) + momentum_rotor2(1) + momentum_rotor3(1);
momentum_total(2) = momentum_total(2) + momentum_rotor0(2) +
momentum_rotor1(2) + momentum_rotor2(2) + momentum_rotor3(2);
momentum_total(3) = momentum_total(3) + momentum_rotor0(3) +
momentum_rotor1(3) + momentum_rotor2(3) + momentum_rotor3(3);


%Compute force necessary for motors 1 & 3 to equilibrate moments in GC
difference_mom_x = abs(momentum_rotor0(1)) - abs(momentum_rotor1(1));
difference_mom_y = abs(momentum_rotor0(2)) - abs(momentum_rotor1(2));
```

```
force_to_equal_mom_y = difference_mom_x / struct(4).line_cg(2);

force_to_equal_mom_x = -difference_mom_y / struct(4).line_cg(1);

momentum_total(1) = momentum_total(1) +
(force_to_equal_mom_x+force_to_equal_mom_y)*struct(4).line_cg(2) +
(force_to_equal_mom_x+force_to_equal_mom_y)*struct(6).line_cg(2);

momentum_total(2) = momentum_total(2) +
(force_to_equal_mom_x+force_to_equal_mom_y)*struct(4).line_cg(1) +
(force_to_equal_mom_x+force_to_equal_mom_y)*struct(6).line_cg(1);
```