POLITECNICO DI TORINO

Dipartimento di Ingegneria Meccanica e Aerospaziale

Corso di Laurea Magistrale
in Ingegneria Aerospaziale
Aeromeccanica e Sistemi

Tesi di Laurea Magistrale

Development of an aircraft model for training flight simulation

Relatori
Prof. Giorgio Guglieri
. . . .
Ing. Alessio Beretta
. . . .
Ing. Luigi Mascolo
. . . .

Candidato
Lorenzo Petrozzino
. . . .

Aprile 2021

Abstract

This thesis work was developed in collaboration with TXT e-solutions® in order to improve a single aisle civil aircraft flight simulator mathematical model. The starting point was an existing model, referred to an Airbus A320, already available inside the company. To achieve this goal the procedure adopted was articulated in the following manner: first it was required an analysis of the existing assets in order to highlight the major lacks and the elements that prevented the model to be used with other aircraft. Then, it was necessary to learn to use some tools to produce the values which, inserted inside the simulator model, were able to reproduce the behaviour of other aircraft. This was done to acquire a benchmark capable to give some information about the improvements effectiveness. In order to facilitate these values production for different aircraft and flight conditions, some automation scripts, which interact with the existing programs, were written. Once all the tools required were gathered and well understood, it was possible to edit some sections of the model: to improve the versatility lots of parameters were brought outside the model and inserted in a configuration file specific for the aircraft; while to expand the capabilities and the fidelity the engine subsystem was improved and a landing gear system was added. Finally some tests, in compliance with the EASA regulation, were performed and their output was analysed to evaluate the impact of the changes made.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Acknowledgement

This work was possible thanks to the collaboration between the Politecnico di Torino and TXT e-solutions®. In particular thanks to the professor Giorgio Guglieri and his ability to create a bridge between the university world and a network of enterprises operating in all the ranges of aeronautic industry; and to Francesco Borgatelli which, understanding the value, for a student, of an experience in a work environment, allowed me to participate to this project. I also want to thanks Alessio Beretta for his precious advices. A special thanks goes to Marco Macarana for all the time spent together during this harsh months, and for his support and willingness along all this work, from the model explanation to the thesis revision. Finally I have the pleasure to thanks Luigi Mascolo for the countless thesis revisions but especially for the dedication lavished on the DRAFT POLITO [1] team to make it a vital element of the academic growth for lots of students.

[1]DRones Autonomous Flight Team https://www.draftpolito.it

1.2 Premise

Starting from the 1960s, thanks to the introduction of computers, the world assisted to a revolution that involved all branches of industry. These new capabilities offered by digital devices, soon started to be exploited by the aviation players, which integrated them both in airborne equipment and on ground systems. The introduction of digital components inside aircraft allowed to enhance their usability and flexibility, but it was on training segment that it caused a clean break from the past. Indeed, before the pilot training was an esoteric art based only on the sensitivity and the capability of the instructor, which with few rudimentary items was responsible for the life of the student when he first piloted a real plane. Starting from that moment it became possible to have some calibrated instruments which were unambiguously corresponding with the reference aircraft, with the same behaviour, or at least a similar one. It is worth remembering that until the introduction of digital systems in training simulations, the simulators (the devices which were supposed to allow the student to train) were a little more than children toys, with no guarantee of a coherent behaviour between different productions. Speaking of which it is curious to notice that from the dawn of the flight simulators the training and the recreational purpose always went at the same pace, aspect that we still can observe today, even if the presence of regulation is splitting the two categories, in particular for what concerns high fidelity systems.

The first digital flight simulator ever realized was made in 1954 [1] but as always in the technology fields it took some time for the regulatory authority to start to create a standard to define a common base of requirements that all the player involved had to respect. According to the first Advisory Circular published on the theme in 1983 [2] it is from the 1970s that the FAA recognised flight simulators as training instrument and started to regulate them.

In the 1990s the drop of the acquisition costs causes the adoption of digital calculus devices from a large portion of the developed country citizens. Even if this event may seem less important compared with the introduction of this kind of devices in the '60, it had a greater impact, because it empowered a significant number of people, allowing them to perform fast and cost effective computations, laying the groundwork for a completely new approach to science and industry, included the training simulation. The push given by hardware more and more powerful with affordable prices is not finished, quite the opposite, nowadays, even a mid-range laptop is able to run a simulation based on a simple aircraft model in real time. It is important to underline how the standard workflow for all fields of modern aerospace industry is based on the assumption that a cheap and fast digital calculus device is available to use.
Let's make some examples:

- Structure: today is inconceivable to project an aircraft without using a tool based on the Finite Element Method which allows to evaluate the strain and displacement under given loads.
- Aerodynamics: modern modelling techniques are base on the Computation Fluid Dynamics which is based on solving a set o differential equations with numerical methods.
- Propulsion: it makes use of both FEM and CFD techniques combined with thermal simulations.
- Systems Engineering: simulation are used to understand how systems respond to a given command in presence of external noise and how interacts between each other.

Once again it is worth to stress the fact that the level of performances in terms of power, efficiency but also reliability and maintainability and cost effectiveness reached by modern aircraft, it is all based on the digital technology which made possible to solve, in a numerical fashion, problems that otherwise would not have any practical solution.

1.3 Objectives and Approach

A little premise was needed to understand the framework in which this thesis was developed. The purpose of this work is to generalize a flight model specific for an aircraft (the Airbus A320), making it configurable for any aircraft belonging to the same class, with a view of a FNPT (Flight and Navigation Procedures Trainer) level II Simulator certification. The reference regulation for simulators inside the European Union is the CS-FSTD(A) [3], which defines an FNPT as follows: *'Flight and navigation procedures trainer (FNPT)' means a training device which represents the flight deck/cockpit environment including the assemblage of equipment and computer programmes* ***necessary to represent an aircraft or class of aeroplane*** *in flight operations to the extent that the systems appear to function as in an aircraft. It is in compliance with the minimum standards for a specific FNPT level of qualification.*. During this work the effort was focused on the flight model neglecting other elements such as the hardware-software integration or the command response. From now on, the term simulator will refer to the numerical model adopted to implement the different systems of the aircraft. The class targeted by the model developed is the single aisle, medium size civil jet aircraft, inside this class the aircraft selected to validate the work done on the simulator was the Embraer 190AR. The most significant solution adopted in this work consists in how the data from new aircraft were obtained in order to verify if the changes made enabled the model to be used for different aircraft.
Typically there are two ways to get the data:

- The first methodology is to rent an aircraft for a certain period of time during which a series of test are performed. This has the advantages to obtain accurate data that can be both used to characterize the aircraft and to validate the results produced by the simulator. However, the costs required to perform the test campaign are not negligible and they can hardly be justified for a preliminary evaluation such as this project.
- The second common way is to purchase the data from an institution or an industry player that are willing to sell their knowledge on the specific aircraft. This case too has advantages and disadvantages, the former are that once identified the seller it is straightforward to obtain the data, the latter are the costs to obtain the data which strongly depends on their accuracy. Also, it must be considered that when buying datasets, the seller may give the results for the aircraft without any information about the test performed, which often are not exactly the ones required by the regulator.

However, because the certification process for a specific aircraft was far beyond the scope of this work, which consists of a feasibility study, it was possible to lower the level of accuracy. This aspect, combined with the requirement for a cost effective solution, leads to the use of computer tools to generate the data to feed the model.
The flow in order to obtain the results for a test is the following:

- First a tool called AcBuilder is used to define the aircraft geometry creating a model.
- Then the model is given in input to another program called NeoCASS which gives in output the flight derivatives.
- After that the flight derivatives (in form of matrices) are used by the actual simulator, written in Matlab® and Simulink®, which performs the simulation and gives as output the time histories for the parameters listed in Appendix C, combined with related plots.

The major drawback of this procedure is the lack of real data to confirm the success of the simulation, on the contrary, the fact that flight derivatives are produced using computers tools add another layer of uncertainty. For these reason tests results can only be validated by the experience and the sensibility of the user.

2 State of the art

2.1 The Existing Model

As previously said the simulator was written using Matlab® and Simulink®. The users interaction, as visible in the flowchart of Figure 1, is limited to the first part of the code which is written in Matlab®. After the test selection starts an automatic routine that, unless unexpected behaviour, ends with the completion of the simulation. The Simulink® part of the program is used to compute the evolution of aircraft physic quantities during simulation run, so it is transparent to the final user.

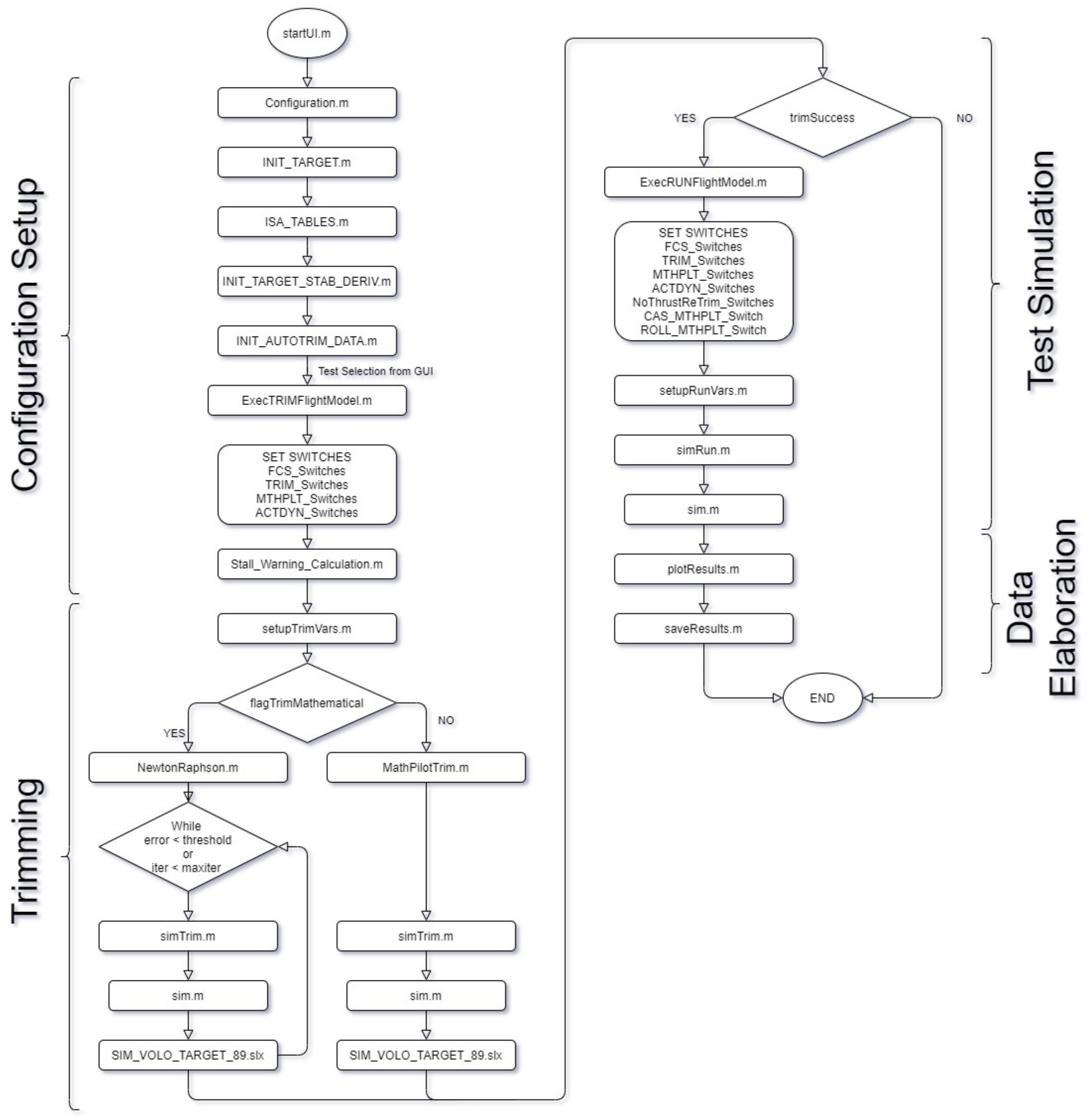


Figure 1: Flowchart of the simulator.

Figure 1 above shows the flow of actions of the simulator with some of the main functions used. Inside that flowchart are highlighted four high level procedures that the simulator has to perform in order to run the simulation, they are:

- Configuration Setup or Pre-processing
- Trimming
- Test Simulation
- Data Elaboration or Post-processing

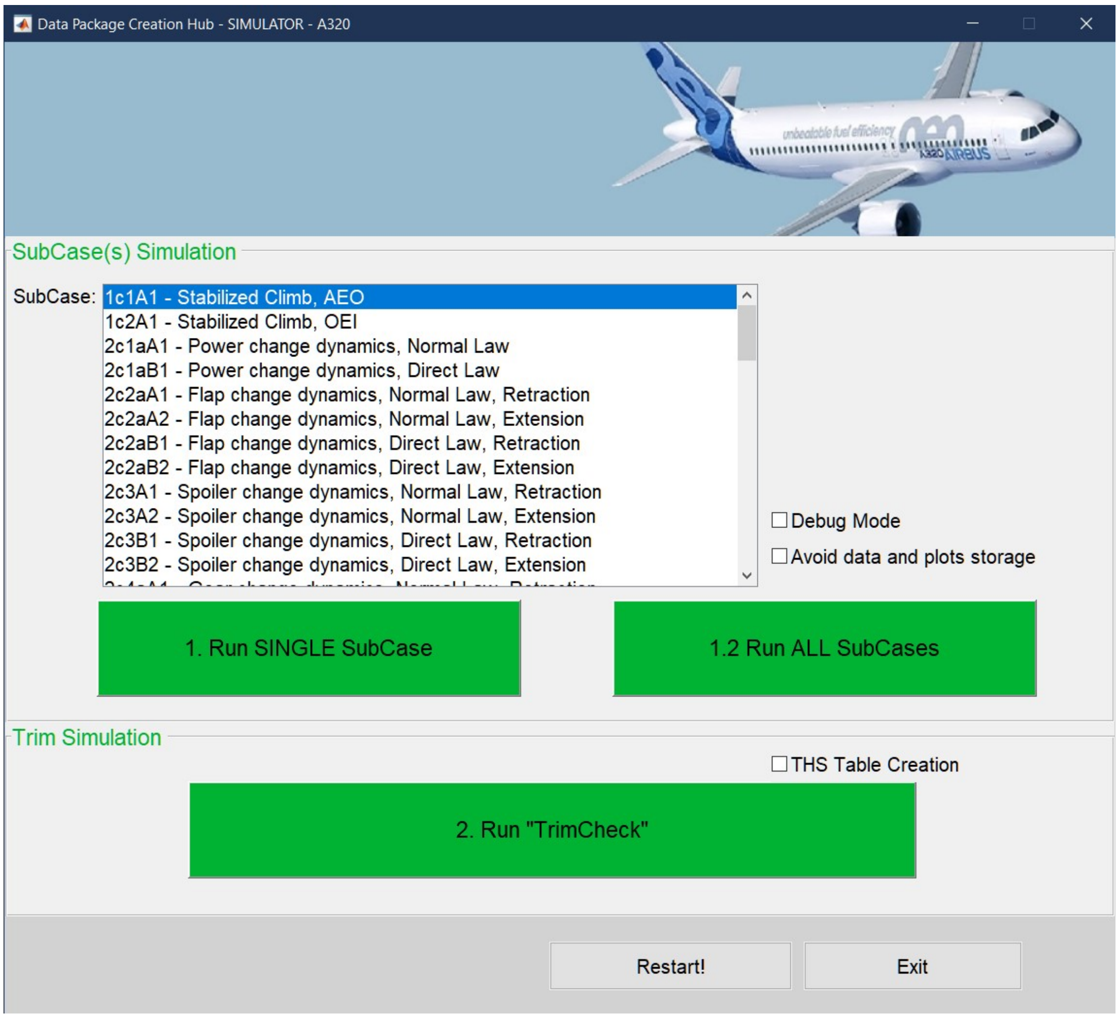
Let's see more in detail what each one of these is supposed to do.

2.1.1 Configuration Setup or Pre-processing

By running the Matlab® script called *startUI.m* the simulators starts. This script is responsible to create a GUI (Graphical User Interface) where the user can select the test to perform. Before printout the GUI it performs some other actions which are transparent to the user:

- First it calls *Configuration.m* this is the list filled with the name and ID (following the CS-FSTD(A) [3] name convention) for the aircraft tests available.
- After it calls *INIT_TARGET.m* which defines some variables related to the Earth geometrical and physical properties such as the radius and gravity acceleration, the environment, and also the aircraft. In its turns it calls three other scripts:
  - *ISA_TABLES*: it contains the tables related to standard atmosphere, air density and temperature in function of the altitude.
  - *INIT_TARGET_STAB_DERIV*: it loads and stores the flight derivatives from the matrices previously generated.
  - *INIT_AUTOTRIM_DATA*: it loads and stores the matrices containing the deflection of the THS in function of speed and altitude.

And finally the test selection window, visible in Figure 2 pops-up. This window is divided in two sections: *SubCase(s) Simulation* and *Trim Simulation.*
The main section is mostly occupied by the list of tests to run, next to this list there are two checkbox, the upper: *Debug Mode* which, if checked allows during execution to prompt more information about the simulation, such as the trim error magnitude, and also causes to pause the execution between each "main step" waiting for the user to proceed. The lower *Avoid data and plot storage* prevents the program to store the results produced. Of course these options are only useful for the developers of the simulator and not for the final user. At the bottom of this section there is the *1. Run SINGLE SubCase* button. Selecting the test by clicking on it and pushing this button will cause the simulation to start.
The second section is also used for debugging purposes only, it is made up of a single button *2. Run "TrimCheck"* which regardless of the test chosen causes the simulator to start trying to trim the aircraft in all the condition defined by the subcases. During typical execution the user wants to perform the simulation for a specific test so it will use the *1. Run SINGLE SubCase* button, doing so it calls the script *Run.m.*


Data Package Creation Hub - SIMULATOR - A320

SubCase(s) Simulation

SubCase:
1c1A1 - Stabilized Climb, AEO
1c2A1 - Stabilized Climb, OEI
2c1aA1 - Power change dynamics, Normal Law
2c1aB1 - Power change dynamics, Direct Law
2c2aA1 - Flap change dynamics, Normal Law, Retraction
2c2aA2 - Flap change dynamics, Normal Law, Extension
2c2aB1 - Flap change dynamics, Direct Law, Retraction
2c2aB2 - Flap change dynamics, Direct Law, Extension
2c3A1 - Spoiler change dynamics, Normal Law, Retraction
2c3A2 - Spoiler change dynamics, Normal Law, Extension
2c3B1 - Spoiler change dynamics, Direct Law, Retraction
2c3B2 - Spoiler change dynamics, Direct Law, Extension

☐ Debug Mode
☐ Avoid data and plots storage

1. Run SINGLE SubCase | 1.2 Run ALL SubCases

Trim Simulation

☐ THS Table Creation

2. Run "TrimCheck"

Restart! | Exit

Figure 2: A320 Test Selection Window.

Then it calls *ExecTRIMFlightModel.m*. This script sets some switches inside the Simulink® model in order to activate or bypass different execution logics by calling the following scripts:

- *FCS_Switches.m* changes the Flight Control System behaviour.
- *TRIM_Switches.m* selects if the commands are controlled by the system (during run) or forced from the outside (during trim).
- *MTHPLT_Switches.m* changes the MathPilot intervention behaviour.
- *ACTDYN_Switches.m* enable or disable the actuators dynamic.

After this procedure it calls *Stall_Warning_Calculation.m* where the stall speed for the current trimming condition is computed by way of tables which are defined in function of weight and flaps extension. At last it computes the environmental conditions for the current altitude and sets the starting conditions for the trim. Note that these starting conditions are a guess. From here the trimming phase starts.

2.1.2 Trimming

There are two ways to trim the aircraft inside the simulator: the first is the Newtown-Raphson method, which is faster but fails to trim asymmetrical conditions, the second one is called Math Pilot, which uses a series of PID controller connected to the command inputs to trim the aircraft in desired conditions. The selection of one rather than the other is defined in the Subcase script for the specific test. Hereafter are explained more in detail how they work.

- *NewtonRaphson.m* This is a mathematical method to find where a certain function crosses the zero. As previously said this method is fast but, because it is based on local considerations about the function, it doest not guarantee the convergence. In order to understand how it works let's first see it in a 1-D scenario. It approximates a generic non linear function as follows

$$f(c) = f(x_0 + h) \approx f(x_0) + hf'(x_0)$$

Because goal is to know where a given function is equal to zero, and assuming that $f'(x_0)$ is not very close to zero

$$h \approx -\frac{f(x_0)}{f'(x_0)}$$

Now improved estimated it is available

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

From here it is trivial to define in general

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

To better understand the method in Figure 3 is visualized with a geometrical perspective what previously discussed in a mathematical manner Once understood the one-dimensional

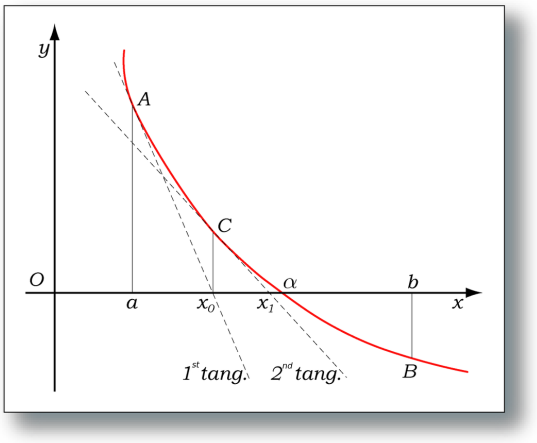


Figure 3: A visual implementation of the Newton-Raphson method

problem we can move to the multidimensional. In this case we can define a generic function with a Taylor expansion

$$f(x + h) = f(x) + J(x)h + O(||h||^2) = 0$$

Where J is the Jacobian Matrix. Let's assume that J is non singular resolving the linear system with h as unknown

$$J(x)h = -f(x)$$

It is possible to obtain

$$x_{k+1} = x_k + h$$

Where h is the solution to the linear system.
Now, because an explicit expression for the system is not available, it is not possible to evaluate the derivative of this function in an analytical fashion. To do so, it is necessary to make an additional approximation evaluating the derivative as the physical quantity difference in two consecutive instants, divided by the time which exists between these two moments (in this case the simulator has a time granularity of 100 Hz). Because this process requires to run a simulation for each physical quantity a little perturbed, it is quite expensive computationally speaking. For the reason exposed, the Jacobian is evaluated only at the first iteration. Because of this, the method used is called **Steady Newton-Raphson** using **Finite Difference**.
The method used has a limit of 50 iterations to converge to an error value which is smaller than 10^{-4}, otherwise it will report the inability to trim the aircraft causing the interruption of the program.

- *MathPilotTrim.m* In contrast to the Newton-Raphson method, which is mostly implemented in Matlab®, the so called MathPilot Trim is almost implemented using Simulink®. This system has been developed by TXT e-solutions® to reproduce the behaviour of a pilot controlling a given aircraft, because of this it offers a great versatility allowing to be used not only during the trimming phase but also to reproduce some manoeuvres during the subcase simulation. In this regard, here there is a list of what the Math Pilot is capable of:
  - Control roll, pitch, yaw and thrust independently from each other.
  - The roll control is capable of following a bank angle or a bearing angle, in both case the control is equipped with a normal and a direct law.
  - In the pitch control are available a direct and an alternate mode as well as an altitude following mode.
  - The yaw control offers a direct and an alternate mode to follow a given β angle

  Normal, Alternate and Direct mode are the autopilot working modes nomenclatures, used by Airbus to indicate the level of intervention from the most conservative to the most permissive. For more information about the active controls when each mode is enabled please refer to this link [1]. All of these functionalities are obtained by a series of PID controllers, this certainly gives to the system a wide versatility but at the same time it complicates in a significant manner the system. A visual consequence of this complexity is shown in Figure 4. The current main limitation of this system lies on its inability to check if the trim succeeded or if it failed. To understand why, it is first necessary to see how it works. The objectives to reach (they might be an altitude, a calibrated air speed or a given angle) are fed to the control system by enabling or disabling some switches inside the model. Next the simulation is started and ran for a certain amount of time (in this case is set to 80 seconds) that has been found as a trade-off between time required and accuracy reached. During the run, MathPilot tries to reach a balanced condition respecting the given objectives. When the time ends, it gives back to Matlab® the last status available

[1] http://www.airbusdriver.net/airbus_fltlaws.htm

for the control surfaces, these values are assumed to be the ones which trims the aircraft. As is understandable by this simple example, unlike the Newton-Raphson method, the Math Pilot does not check for any terminating condition to declare a success of the trim. So it is possible that the simulation starts not only from a unbalanced condition but from a totally different condition form the intended one nullifying the simulation results.

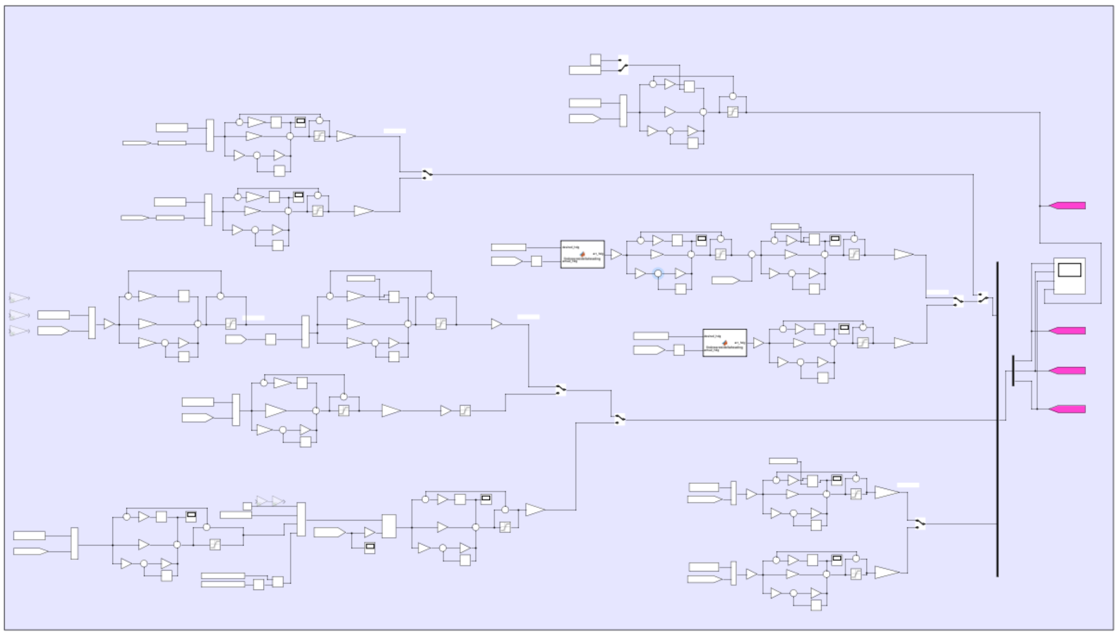


Figure 4: The MathPilot Trim system implemented in Simulink®.

2.1.3 Test Simulation

Once the trimming phase ended, the program checks if the trim was successful (because, as previously said, MathPilot trim does not perform any check to establish if the trim failed, the MathPilot trim is assumed to be always successful). In case the outcome is negative the program ends, otherwise it will proceed by calling *ExecRUNFlightModel.m.* As in the trim phase it first sets the switches for the simulation by calling the scripts listed below:

- The four scripts already presented in Paragraph 2.1.1
- *NoThrustReTrim_Switches.m* selects if the thrust command is controlled by the system or forced from the outside.
- *CAS_MTHPLT_Switch.m* forces the MatPilot to follow a Calibrated Air Speed rather than an Pressure Altitude.
- *ROLL_MTHPLT_Switch.m* forces the MatPilot to follow a bank angle or a bearing angle.

Then, the simulation is performed by calling *simRun.m* which in its turn calls *sim.m.* Let's see more in detail how the Simulink® model is organized and works. Inside the main model there are several sub-models, see Figure 5. They are listed below.

- TARGET_INPUTS_F_M

- TRANSL_DYN
- ROT_DYN
- REL_QUATERNION_PARAM
- CG_MOTION
- FLIGHT_ANGLES
- WIND_TRIANGLE
- FLIGHT_ANGLES
- AIR_DATA
- POSITION_COORDINATES

The first three items are the most important, in particular the first one: *TARGET_INPUTS_F_M* is responsible to define the command laws for the aircraft and also for computing the forces and moments, given the conditions for the current instant, in terms of CAS, altitude, flight angles, aircraft configuration, surface deflection and so on. The two others receive as input the forces and moments computed inside the former, and using the flight equations of motions, give as output the linear and angular speeds and accelerations. These values on their turn will be used to compute the forces for the next instant. This process is repeated until the end of the simulation. The other blocks are used to provide some auxiliary information or to add features to the simulator, that's the case of *WIND_TRIANGLE* sub-model which, by the way, was not used during this thesis work. Anyway, more details about the internal functioning mechanisms will be discussed in chapter 3.

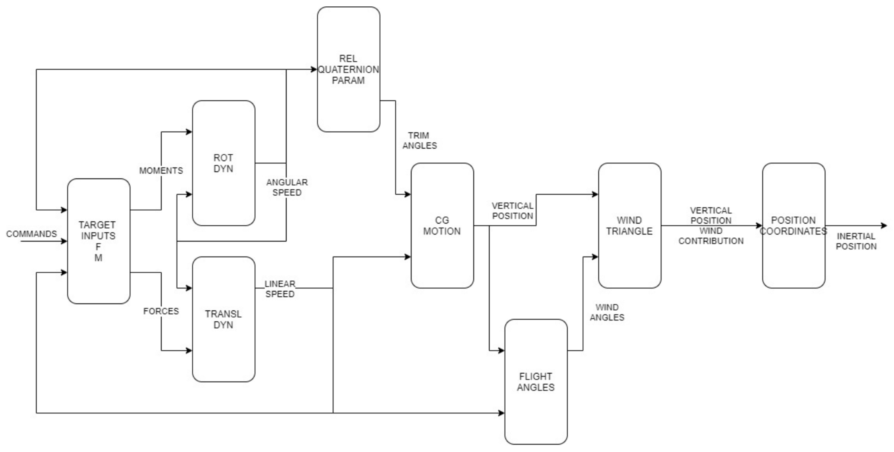


Figure 5: A comprehensive view of the higher level of the simulator model

2.1.4 Data Elaboration or Post-processing

Finally, after the simulation is completed two scripts are run

- *plotResults.m* produces some plots to analyse the outcome of the simulation.
- *saveResults.m* gathers all the data produced by the simulation and stores each physical quantity (see Appendix C) in a .csv file linking it with the time it is referred to.

2.2 Tools

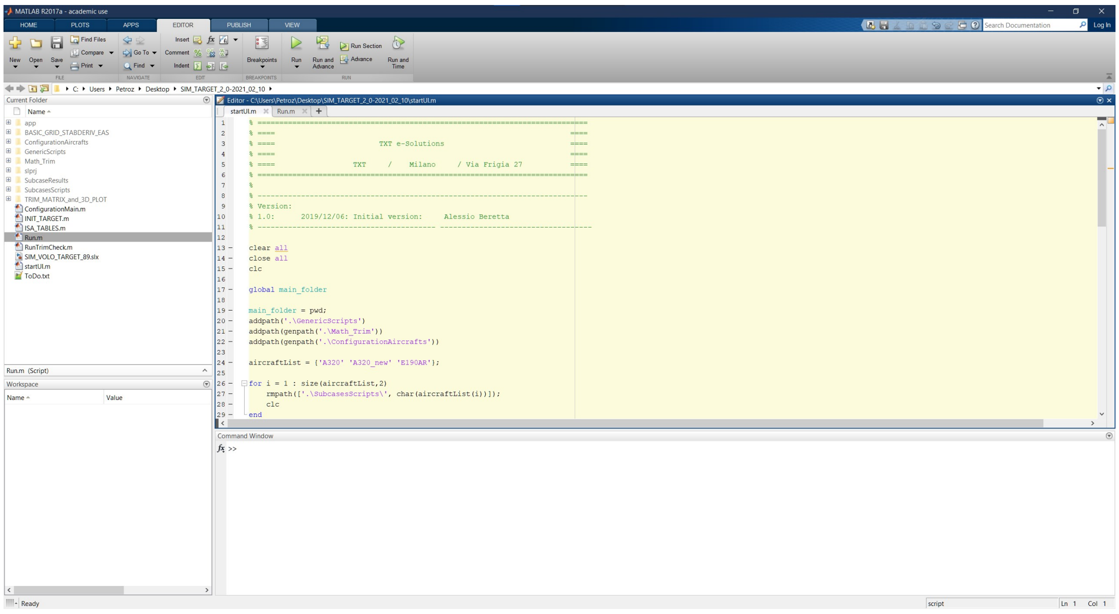
Hereafter there is a brief introduction to each one of the tools used during this thesis work.

2.2.1 Matlab®

Matlab®, which stands for MATrix LABoratory, is the name for a proprietary multi paradigm programming language and a developing environment which is proprietary too [4]. The main focus of Matlab is the numeric computing, in particular as the name suggests, it is the manipulation of matrices and arrays. During the years lot of new capabilities were added, such as the ability to generate graphical interactive interfaces, to interact with other programming language but also to manipulate images and lots of more.
Matlab was firstly developed in 1970s by Cleve Moler, a professor at the New Mexico University, to make his students able to use some numerical computation libraries written in Fortran without knowing the programming language. The tool build consensus in the academic environment, so Steve Bangert and Jack Little reached him in the effort to rewrite Matlab in C instead of Fortran. Together in 1984, to ensure the Matlab development and growth, they founded the MathWorks company, which still today is in charge of spreading Matlab® and making it profitable. The whole suite (Matab + Simulink® + Add-ons) is released biyearly. The recent releases are called with the year of release followed by an a or a b, the a version refers to the first semester release the b for the second one. For this thesis work the version used was the 2017a.
The working environment, as shown in Figure 6, is structured in 5 parts:

- The top bar, as the name suggests it is located on the top of the screen. It is used to switch between the different modes, and contains all the icons to interact with the Matlab® environment by the mouse.
- The editor, in Figure 6 is located on top right under the top bar. It is where the programs are written, it is possible to have more than one program opened at the same time, they are arrange in tab.
- The command window, in Figure 6 is located on bottom right. This prompt is used to interact with the Matlab® environment via text. The text output of Matlab® programs are showed in this window. It also can be used as input method to write some simple scripts, in the same way they are written in the editor window, with the only exception that they will not survive to a restart of the program.
- The workspace, in Figure 6 is located on bottom left. It is a list of all the variables which are available to be used and all the values stored inside.
- The file explorer, in Figure 6 is located on top left under the top bar. It is a simple file explorer showing informations about the current directory and the subfolders and files added to the path which therefore can be executed.

The main strength of Matlab®, that is also the reason why it is so used in industry and university, is the simplicity with which the matrices can be manipulated, combined with a set of functions well written and documented, that can solve all the major problems that an user can encounter in mathematical programming. These elements make writing programs in Matlab® lot easier and quicker compared to other programming language such as C or C++, although these last guarantee a better execution speed.



```
% ====================================================================
% ====                                                            ====
% ====                      TXT e-Solutions                       ====
% ====                                                            ====
% ====                 TXT    /    Milano     / Via Frigia 27     ====
% ====================================================================
%
% --------------------------------------------------------------------
% Version:
% 1.0:      2019/12/06: Initial version:     Alessio Beretta
% ------------------------------------ -------------------------------

clear all
close all
clc

global main_folder

main_folder = pwd;
addpath('.\GenericScripts')
addpath(genpath('.\Math_Trim'))
addpath(genpath('.\ConfigurationAircrafts'))

aircraftList = {'A320' 'A320_new' 'E190AR'};

for i = 1 : size(aircraftList,2)
    rmpath(['.\SubcasesScripts\', char(aircraftList(i))]);
    clc
end
```

```
>>
```

Figure 6: A view of Matlab® development environment.

2.2.2 Simulink®

Simulink® is a software that allows the modelling, simulation and analysis of dynamical systems. It is strictly integrated in the Matlab® developing environment. The main advantage of Simulink® is that while it is a versatile and powerful systems modelling environment, it is completely based on a graphical approach, potentially without the need to write a single line of code. This element allows the user to understand how the model works by a glance, and often consent to present it to non-technical people that will have an overall comprehension. However, due to an inherent lack of a defined structure to follow, and of a specific starting point, increasing the complexity of the system to reproduce, it became more and more complex to debug. The Simulink® programming workflow is based on the use of elementary blocks, which can be picked from a customizable library, and linked together to create more complex functions (Figure 7). In addition to the elementary blocks there are others more complex for specific applications such as aerospace, communication system and neural networks which can be purchased separately from Simulink® core product. After the model has been created it can be easily simulated by pushing the green play button located on the top bar. The simulation parameters like the duration, time step (which can be fixed or variable), the solver (e.g. Runge-Kutta, Stiff), and lot of others parameters can be set. During the simulation but also at the end of the process the signal analysis can be performed using especially designed blocks or via built-in functionalities of Simulink®. Other useful Simulink® features are listed below.

- Define different time granularity for sub-models. This is particularly useful when is necessary to simulate how different components interact between each others to deliver a specific function. Let's have an example. In a robot there is a sensor which scans the surrounding environment with a frequency of 100 Hz, and feed the data to a processor operating in the range of MHz, which has to perform some calculations, and command an electrical engine to allow the robot to move forward. The electrical engine and the robot itself have their own dynamics, which of course are slower than the ones of sensor and processor. All these elements combined together can give unexpected behaviours, that with a classical programming methodology, it is not possible to predict before the tests are performed on real hardware. But thanks to this Simulink® feature is trivial to test the integration of all these elements together.

- Export the model in C or C ++. While the Simulink® environment is rich of features and easy to use, it is not really speed to perform simulations, and it can only be used on generic hardware (such as AMD64 or ARM64) with a set of specific operative systems. All this means that the model created can not be run on dedicated hardware such as that of an on board processors, strongly affecting his versatility. To cope with this limitations there is a Model builder o Coder which automatically converts the Simulink® model in a set of C functions that potentially could be run everywhere. It is important to note that not all the Simulink® features are supported by this conversion, therefore the model has to be developed from its first stages keeping in mind the final usage.

- Because Simulink® is a sort of ”toolbox” of Matlab®, it is easy to integrate Matlab® code. This can be done in both sides: inside Simulink® a block can contain a Matlab® function which receives some inputs, elaborates them following the algorithm, and gives outputs in a transparent way for the Simulink® user. The opposite can also be done by calling a Simulink® model from Matlab® code, receiving the output of the simulation and performing some other elaboration on those data like an automatic analysis or plotting some related informations.

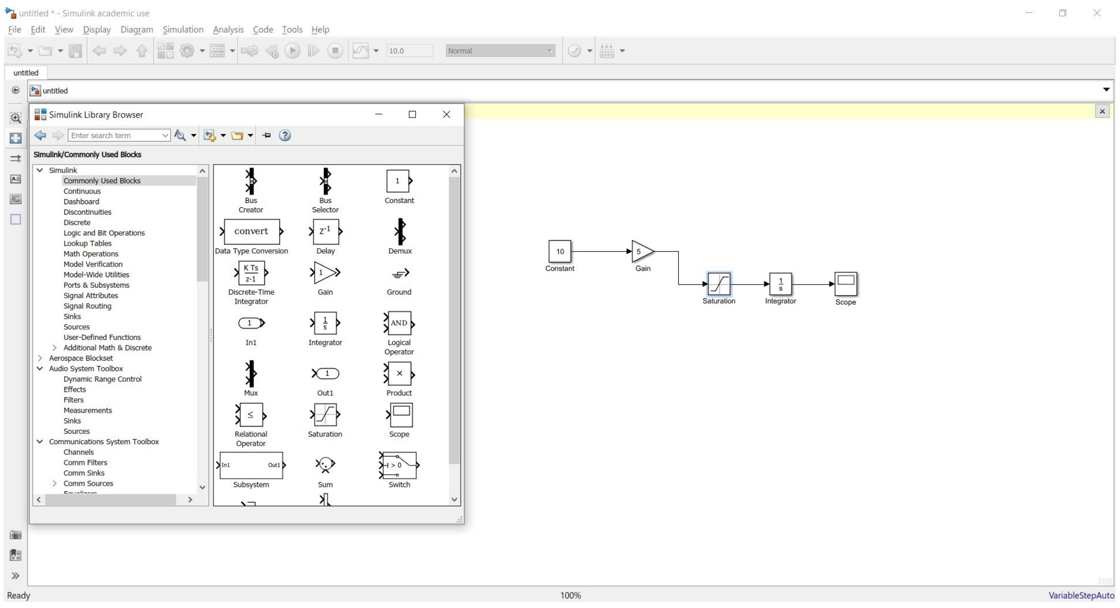
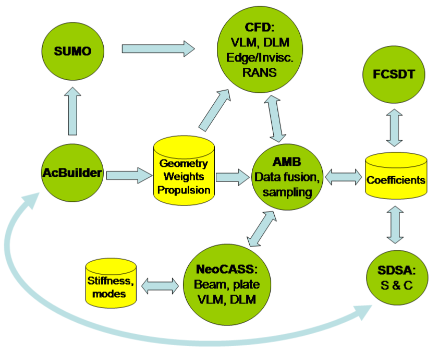


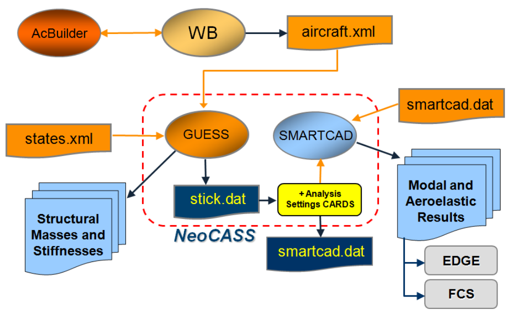
Figure 7: A view of Simulink® workspace and library.

2.2.3 AcBuilder & NeoCASS

Both AcBuilder and NeoCASS (Figure 8b), are open-source programs (released under GNU's GPL 2.1.) developed by the Politecnico di Milano to be part of a wider program called CEASIOM (Figure 8a), funded by the European Union, with the intent to create an affordable and effective tool capable to perform a preliminary evaluation project for the development of a generic aircraft. Both software are written using Matlab® and can run on any Windows or Linux platforms with a recent version of Matlab®. The eldest suitable Matlab® release is the R2008a.


SUMO
CFD: VLM, DLM Edge/Invisc. RANS
FCSDT
AcBuilder
Geometry Weights Propulsion
AMB Data fusion, sampling
Coefficients
Stiffness, modes
NeoCASS: Beam, plate VLM, DLM
SDSA: S & C

(a) CEASIOM modules structure.



(b) NeoCASS module structure.

Figure 8: A wider look at CEASIOM project and a focused view on NeoCASS.

If the scope of the work is not to perform a complete preliminary analysis, but only to make a small part of it, then depending on what is the output desired, it may be possible to use NeoCASS only, without the need of the whole CEASIOM suite. This was the case for this thesis work, where the outputs desired were the flight derivatives matrices only, and any other information concerning the aircraft, although useful, was not indispensable The flow of programs to use is the following :

- AcBuilder
  - Geometry Definition
  - Weight & Balance
- NeoCASS
  - GUESS (Generic Unknowns Estimator in Structural Sizing)
  - SMARTCAD (Simplified Models for Aeroelasticity in Conceptual Aircraft De-sign)

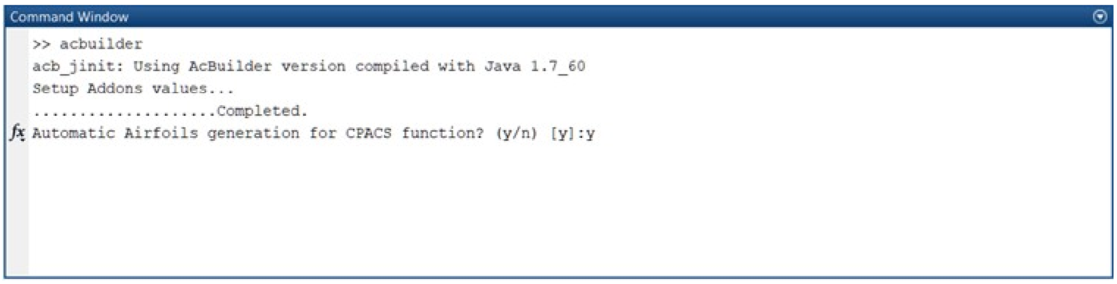
In the following sections the programs functionalities will be presented by explaining how to obtain the flight derivatives. The following procedures were extracted from more detailed tutorials written during this thesis work.

2.2.3.1 AcBuilder

AcBuilder allows to define the aircraft geometry. Inside its environment it is possible to select the components of the aircraft with their specifications. It is also possible to define the structural and aerodynamic discretization to be used by the following analysis. This tool also computes a

rough first estimate of the weight distribution. At the end of the process all the data related to the aircraft geometry are merged in a .xml file.
Hereafter will be presented the main AcBuilder functionalities, mentioning the process to obtain the aircraft model (the .xml file previously introduced).

- Before to start, it is important to notice that AcBuilder, in order to work, requires some software:
  - A supported operative system.
  - A supported Matlab® version.
  - The NeoCASS software package downloadable at this link [1].
- To start AcBuilder it is necessary, within the Matlab environment, to move to the NeoCASS - AcBuilder directory and digit inside the prompt *acbuilder*, as shown in Figure 9.

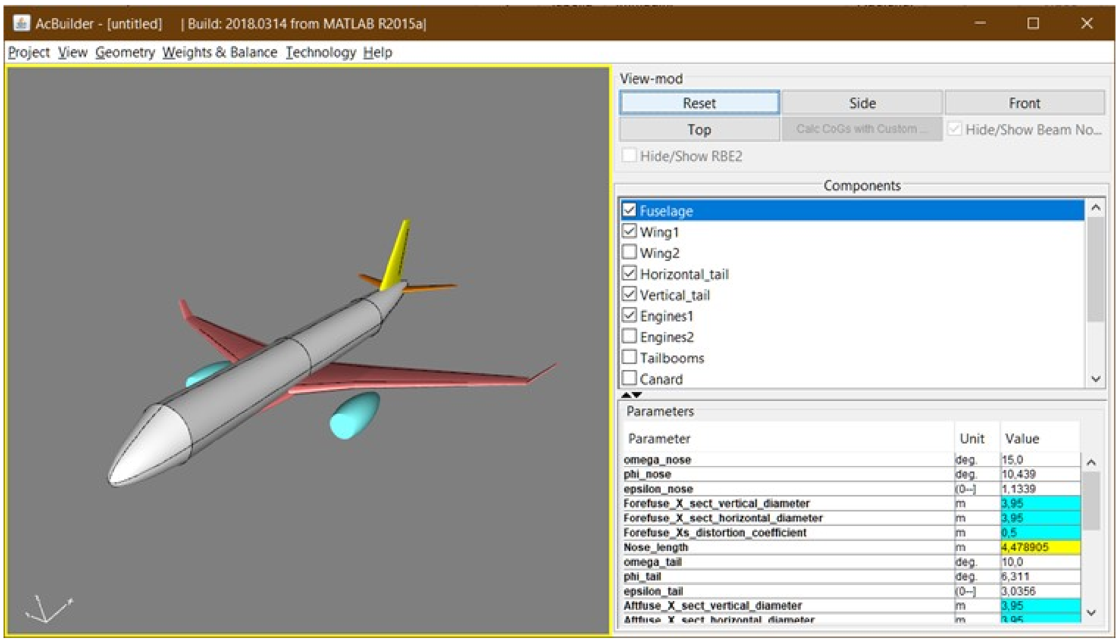


```
>> acbuilder
acb_jinit: Using AcBuilder version compiled with Java 1.7_60
Setup Addons values...
....................Completed.
fx Automatic Airfoils generation for CPACS function? (y/n) [y]:y
```

Figure 9: Matlab® command window.

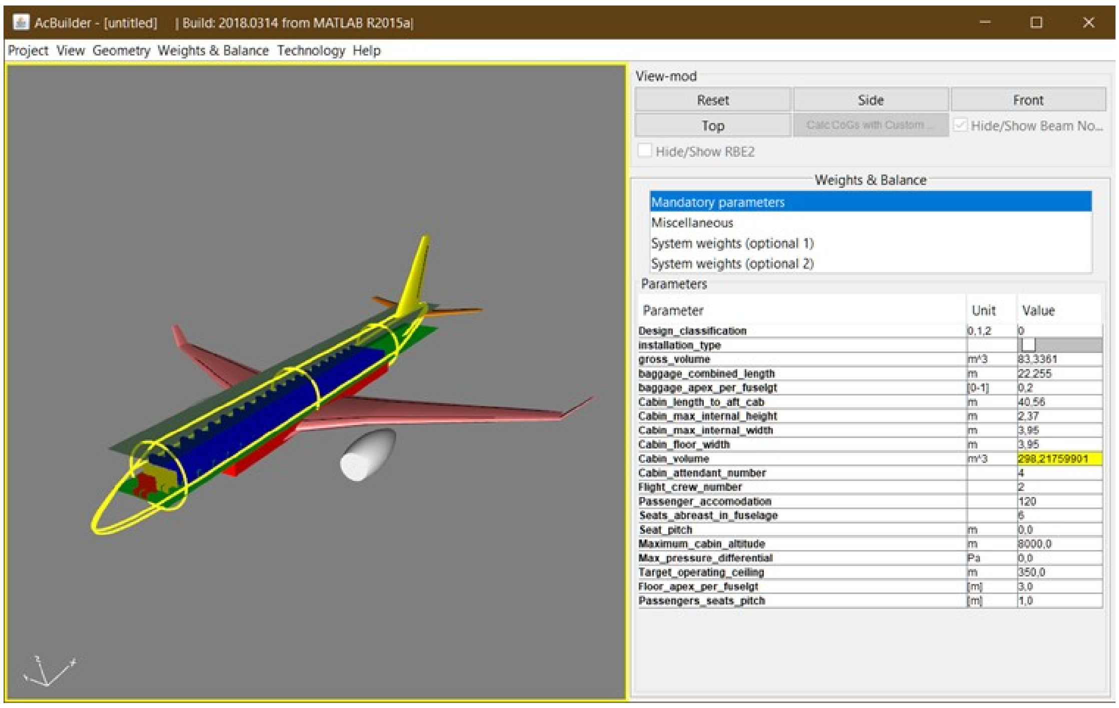
- The coordinates of the profiles available will be printed in the Matlab® command window, then the AcBuilder main window will open as in Figure 10.
- From here it is possible to start modelling the aircraft by editing the default model or another model previously defined.
- In *Geometry - Components* section, it is necessary to check the items which are present on the aircraft and define their specifications.
- The voices are differentiated by these colors:
  - White: editable value.
  - Cyan: multiple selection value.
  - Yellow: locked value (it is the result of other values inserted).
- More information about the editable parameters can be found at *ceasiom-xmlfiledefinition.pdf* which is downloadable at the following link [2].
- After editing *Geometry* section it is possible to go to *Weights & Balance - Weights & Balance* (Figure 11) and edit all the entries under *Mandatory parameters* (to define information about the the payload) and *Miscellaneous items*. Two other tabs are available but optional, this means that if nothing is defined, the values will be automatically computed.

[1]https://www.neocass.org/
[2]https://ceasiom.com/?wpdmpro=ceasiom-xml-definition/



Parameter	Unit	Value
omega_nose	deg.	15,0
phi_nose	deg.	10,439
epsilon_nose	(0--]	1,1339
Forefuse_X_sect_vertical_diameter	m	3,95
Forefuse_X_sect_horizontal_diameter	m	3,95
Forefuse_Xs_distortion_coefficient	m	0,5
Nose_length	m	4,478905
omega_tail	deg.	10,0
phi_tail	deg.	6,311
epsilon_tail	(0--]	3,0356
Aftfuse_X_sect_vertical_diameter	m	3,95
Aftfuse_X_sect_horizontal_diameter	m	3,95

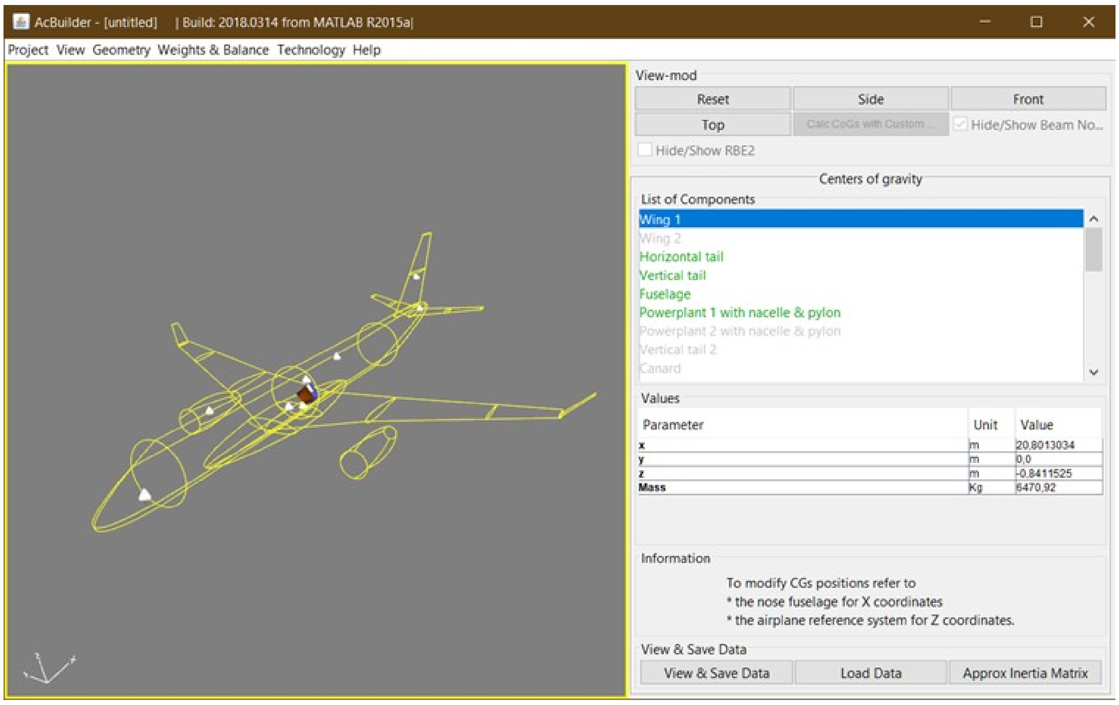
Figure 10: AcBuilder GUI, Geometry - Components window.


AcBuilder - [untitled] | Build: 2018.0314 from MATLAB R2015a|
Project View Geometry Weights & Balance Technology Help
View-mod
Reset
Side
Front
Top
Calc CoGs with Custom ...
Hide/Show Beam No...
Hide/Show RBE2
Weights & Balance
Mandatory parameters
Miscellaneous
System weights (optional 1)
System weights (optional 2)
Parameters

Parameter	Unit	Value
Design_classification	0,1,2	0
installation_type		
gross_volume	m^3	83,3361
baggage_combined_length	m	22,255
baggage_apex_per_fuselgt	[0-1]	0,2
Cabin_length_to_aft_cab	m	40,56
Cabin_max_internal_height	m	2,37
Cabin_max_internal_width	m	3,95
Cabin_floor_width	m	3,95
Cabin_volume	m^3	298,21759901
Cabin_attendant_number		4
Flight_crew_number		2
Passenger_accomodation		120
Seats_abreast_in_fuselage		6
Seat_pitch	m	0,0
Maximum_cabin_altitude	m	8000,0
Max_pressure_differential	Pa	0,0
Target_operating_ceiling	m	350,0
Floor_apex_per_fuselgt	[m]	3,0
Passengers_seats_pitch	[m]	1,0

Figure 11: Weight & Balance window.

- After entering the weights, going to *Weights & Balance - Centers of Gravity List of Components* (Figure 12) it will be possible the see the weight and the CoG position for each subsystem and for the whole aircraft.



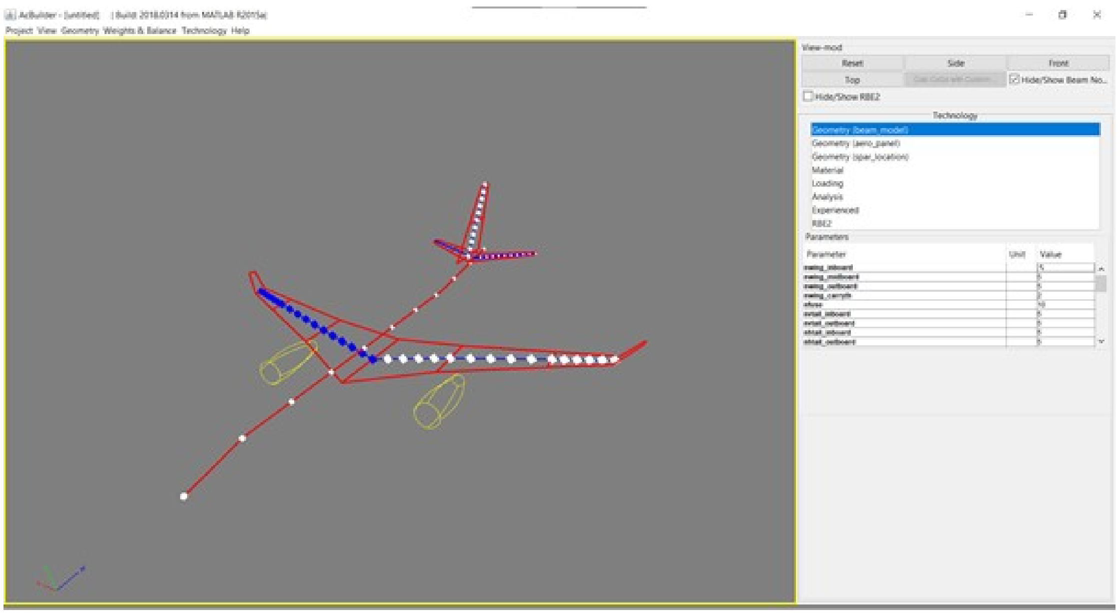
Parameter	Unit	Value
x	m	20,8013034
y	m	0,0
z	m	-0,8411525
Mass	Kg	6470,92

To modify CGs positions refer to
* the nose fuselage for X coordinates
* the airplane reference system for Z coordinates.

Figure 12: Center of Gravity window.

- In the *Technology* section (Figure 13) it is possible to define some parameters for the aerodynamic and structural analysis. During this thesis work, since the scope was to create a civil transport jet such as the one in default options, the *Technology* section has not been edited.
  The Technology section is articulated in the following subsections:
  - *Geometry (beam_model)*: here it is possible to define in how many segments each structural main element is divided, these information are used by FEM analysis.
  - *Geometry (aero_panel)*: here it is possible to define in how many panels each surface is divided, these information are used by CFD analysis.
  - *Geometry (spar_location)*: here it is possible to define the structural main elements positions referred to the respective chord (e.g., root wing, mid horizontal stabilizer, ecc.), this information is used by FEM analysis.
  - *Material*: here it is possible to define the materials (specifying their mechanical properties) of which structural elements are made.
  - *Loading*: here it is possible to define loading factors for various conditions of flight, these factors will be used to size structural elements.
  - *Analysis*: here it is possible to select the kind of analysis to be performed also defining the interpolation method for each element.

– *Experienced*: here it is possible to define other parameters for the structural analysis (e.g., number of nodes, buckling resistance coefficient, minimum gage thickness).
– *RBE2*: here it is possible to connect two nodes of different elements to guarantee structural continuity.



Parameter	Unit	Value
nwing_inboard		5
nwing_midboard		5
nwing_outboard		5
nwing_carryth		2
nfuse		10
nvtail_inboard		5
nvtail_outboard		5
nhtail_inboard		5
nhtail_outboard		5

Figure 13: Technology (beam_model) window.

- Once finished editing the model it is possible to export the model by going to *Weights & Balance - Centers of Gravity* and clicking on *Project – Export XML.*

2.2.3.2 NeoCASS

NeoCASS is a software to perform the sizing of structural elements of an aircraft. To do so it takes advantage of two modules: the first one is called GUESS which uses both simple analytical (like the beam model) and numerical (based on statistical data on common aircraft) methods to obtain a strongly approximated solution. The second one is called SMARTCAD which takes in input the results from GUESS and performs a more accurate analysis based on aeroelastic simplified models. The analysis requires two inputs: the .xml file, produced by AcBuilder, and a set of flight conditions the aircraft must be able to fly. By appropriately selecting these conditions it is possible to obtain the flight derivatives needed by the flight simulator.
Hereafter the procedure to follow in order to obtain the flight derivatives.

1. To use NeoCASS, first it is necessary to start the Matlab® environment, and similarly to what done for AcBuilder, start the program by running *NeoCASS* in the command prompt.

2. A window as shown in Figure 14 will be displayed.

3. In the NeoCASS GUI window (Figure 14), by clicking on *Open aircraft* button it is possible to load the .xml file previously generated by AcBuilder.

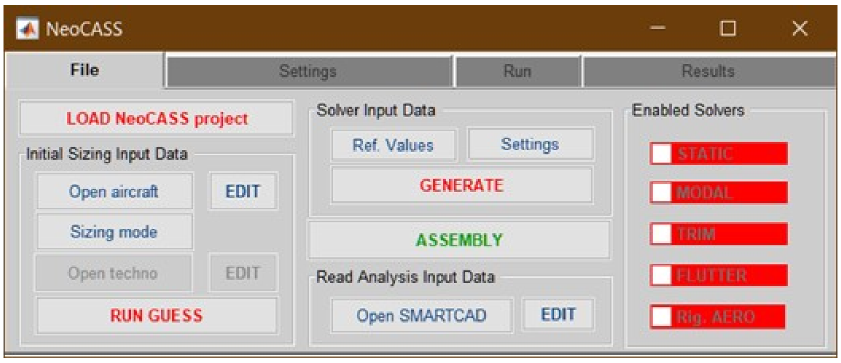
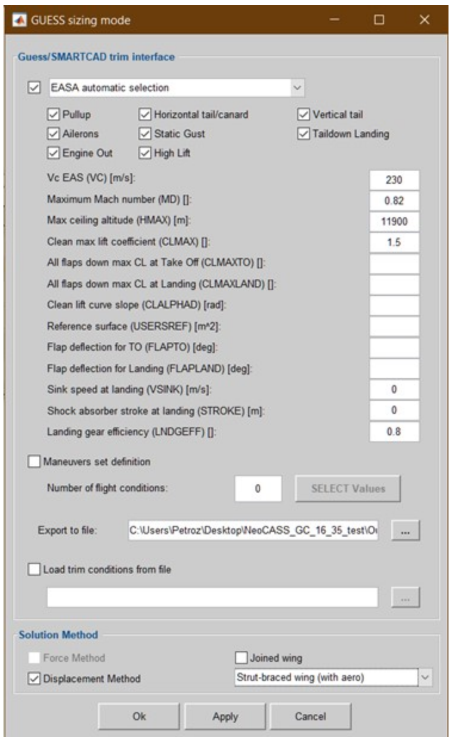


Figure 14: NeoCASS GUI window.

4. By clicking on *Sizing mode* button located in the NeoCASS GUI window, a new window as the one in Figure 15 will open. Here it is possible to define the tests specifications used by the GUESS module to size the aircraft.



GUESS sizing mode

Guess/SMARTCAD trim interface

☑ EASA automatic selection

☑ Pullup ☑ Horizontal tail/canard ☑ Vertical tail
☑ Ailerons ☑ Static Gust ☑ Taildown Landing
☑ Engine Out ☑ High Lift

Vc EAS (VC) [m/s]:	230
Maximum Mach number (MD) []:	0.82
Max ceiling altitude (HMAX) [m]:	11900
Clean max lift coefficient (CLMAX) []:	1.5
All flaps down max CL at Take Off (CLMAXTO) []:	
All flaps down max CL at Landing (CLMAXLAND) []:	
Clean lift curve slope (CLALPHAD) [rad]:	
Reference surface (USERSREF) [m^2]:	
Flap deflection for TO (FLAPTO) [deg]:	
Flap deflection for Landing (FLAPLAND) [deg]:	
Sink speed at landing (VSINK) [m/s]:	0
Shock absorber stroke at landing (STROKE) [m]:	0
Landing gear efficiency (LNDGEFF) []:	0.8

☐ Maneuvers set definition

Number of flight conditions: 0 SELECT Values

Export to file: C:\Users\Petroz\Desktop\NeoCASS_GC_16_35_test\O

☐ Load trim conditions from file

Solution Method

☐ Force Method ☐ Joined wing
☑ Displacement Method Strut-braced wing (with aero)

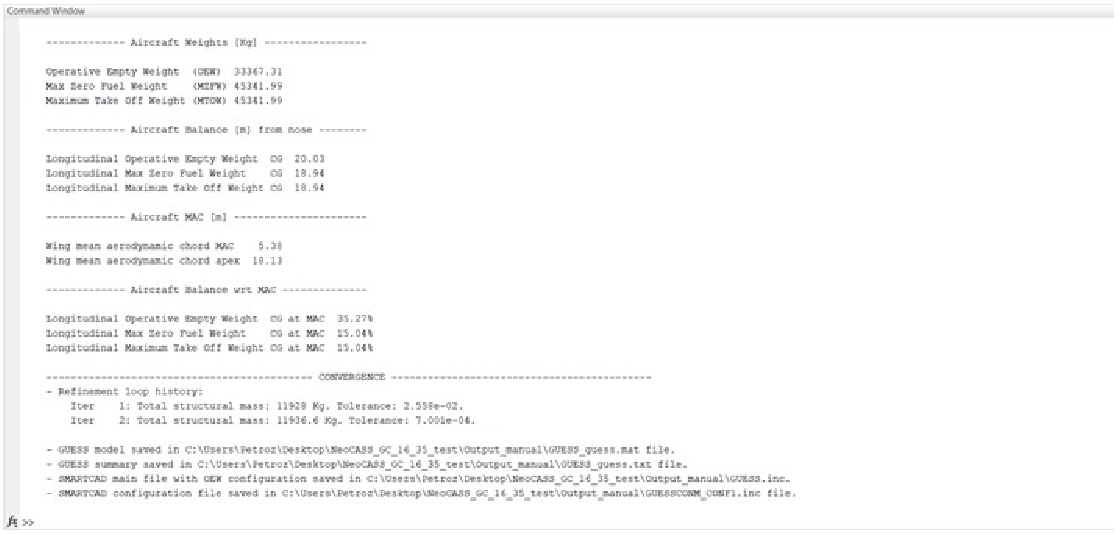
Ok Apply Cancel

Figure 15: GUESS sizing window.

5. Pressing the *Ok* button, will cause Matlab® to start a short elaboration, as shown in Matlab® command window, which will be over once 3 plot charts will be displayed.

6. In order to perform the GUESS computation is necessary to click on *RUN GUESS* button, and in the window that will open click on *Run* button .

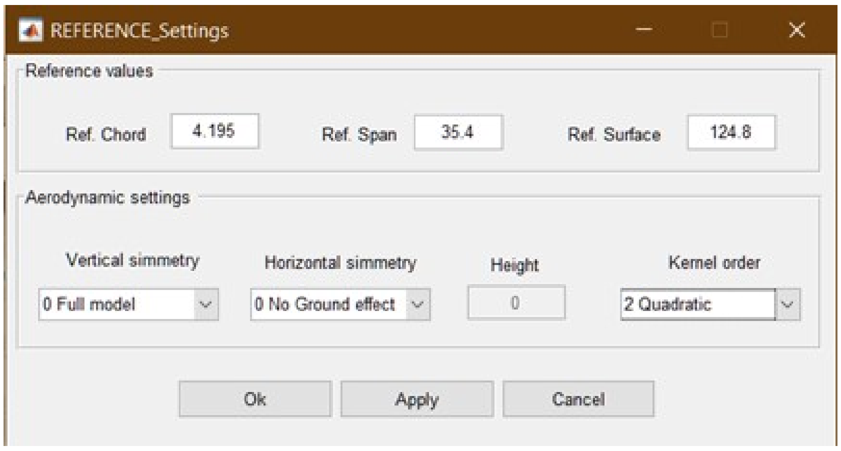
7. To converge to a result for each condition the script will elaborate for approx. 15 minutes. At the end of the process, Matlab® command window should appear as from Figure 16



```
Command Window

    ------------- Aircraft Weights [Kg] -----------------

    Operative Empty Weight  (OEW)  33367.31
    Max Zero Fuel Weight    (MZFW) 45341.99
    Maximum Take Off Weight (MTOW) 45341.99

    ------------- Aircraft Balance [m] from nose --------

    Longitudinal Operative Empty Weight  CG  20.03
    Longitudinal Max Zero Fuel Weight    CG  18.94
    Longitudinal Maximum Take Off Weight CG  18.94

    ------------- Aircraft MAC [m] ----------------------

    Wing mean aerodynamic chord MAC    5.38
    Wing mean aerodynamic chord apex  18.13

    ------------- Aircraft Balance wrt MAC --------------

    Longitudinal Operative Empty Weight  CG at MAC  35.27%
    Longitudinal Max Zero Fuel Weight    CG at MAC  15.04%
    Longitudinal Maximum Take Off Weight CG at MAC  15.04%

    -------------------------------------------- CONVERGENCE -------------------------------------------
    - Refinement loop history:
        Iter    1: Total structural mass: 11928 Kg. Tolerance: 2.558e-02.
        Iter    2: Total structural mass: 11936.6 Kg. Tolerance: 7.001e-04.

    - GUESS model saved in C:\Users\Petroz\Desktop\NeoCASS_GC_16_35_test\Output_manual\GUESS_guess.mat file.
    - GUESS summary saved in C:\Users\Petroz\Desktop\NeoCASS_GC_16_35_test\Output_manual\GUESS_guess.txt file.
    - SMARTCAD main file with OEW configuration saved in C:\Users\Petroz\Desktop\NeoCASS_GC_16_35_test\Output_manual\GUESS.inc.
    - SMARTCAD configuration file saved in C:\Users\Petroz\Desktop\NeoCASS_GC_16_35_test\Output_manual\GUESSCONM_CONF1.inc file.

fx >>
```

Figure 16: Matlab® command window at GUESS completion.

8. To proceed with the SMARTCAD module, it is required to click on the *Ref. Values* button in the NeoCASS GUI window, and insert the three requested parameters values. See example in Figure 17.



REFERENCE_Settings

Reference values

Ref. Chord 4.195 | Ref. Span 35.4 | Ref. Surface 124.8

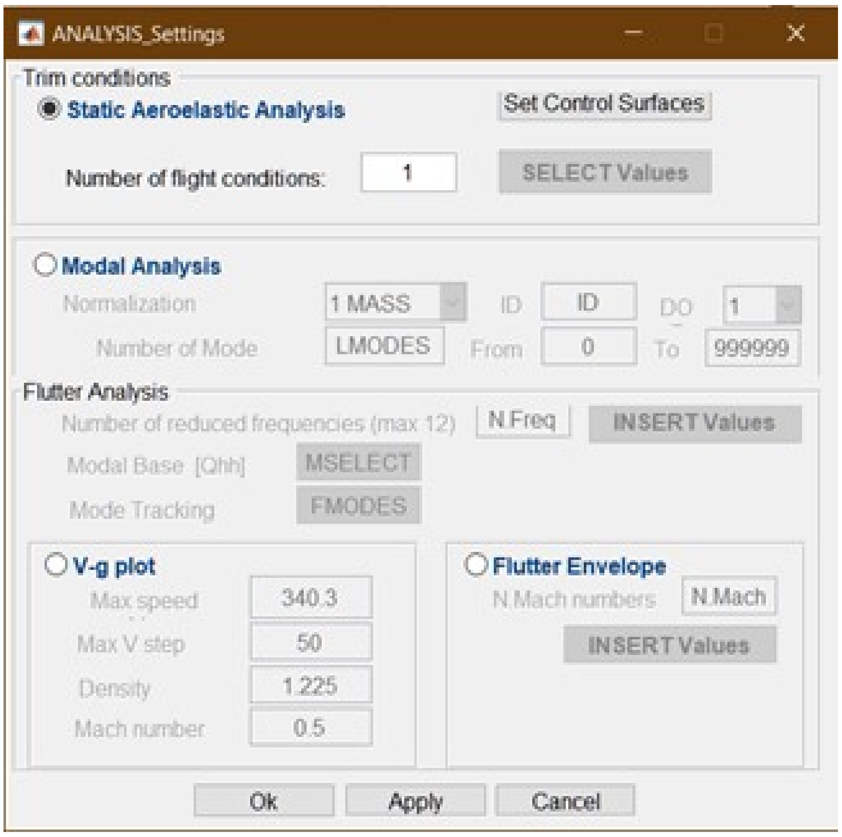
Aerodynamic settings

Vertical simmetry: 0 Full model | Horizontal simmetry: 0 No Ground effect | Height: 0 | Kernel order: 2 Quadratic

Figure 17: Reference setting window.

9. Clicking on the *Settings* button in the NeoCASS GUI window, will display a window as shown in Figure 18.

10. From here it is necessary to select *Static Aeroelastic Analysis*, and insert the number of different conditions to be evaluated during trim analysis. The conditions are the one for which the flight derivatives should be computed. Then press *SELECT Values* button.


ANALYSIS_Settings

Trim conditions

● Static Aeroelastic Analysis — Set Control Surfaces

Number of flight conditions: 1 — SELECT Values

○ Modal Analysis

Normalization: 1 MASS — ID: ID — DO: 1

Number of Mode: LMODES — From: 0 — To: 999999

Flutter Analysis

Number of reduced frequencies (max 12): N.Freq — INSERT Values

Modal Base [Qhh]: MSELECT

Mode Tracking: FMODES

○ V-g plot

Max speed: 340.3

Max V step: 50

Density: 1.225

Mach number: 0.5

○ Flutter Envelope

N.Mach numbers: N.Mach — INSERT Values

Ok — Apply — Cancel

Figure 18: NeoCASS trim condition selection window.

11. A window titled *Maneuver Definition* will open as shown in Figure 19. Inside this window, it is possible to define the conditions used by SMARCAD to size the aircraft and compute the flight derivatives. To proceed with NeoCASS the user must enter the data to define the manoeuvrer and click *Save* button.

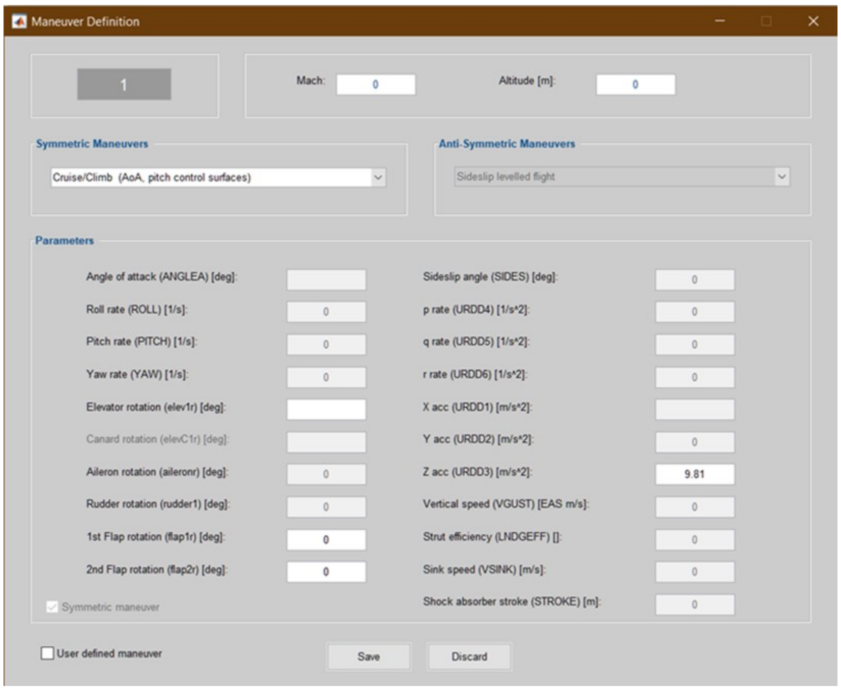
12. In the NeoCASS GUI window, clicking on *GENERATE* button will cause the program to ask where to store the solver file and how to name it.

13. Then, it is necessary to click on *ASSEMBLY* button, and save the .dat file. Inside the window that opens, the files to include in the assembler file must be selected:

    - *"GUESS file".inc*
    - *"GUESS file"CONM_CONF1.inc*
    - *"Solver file".inc*

14. In NeoCASS GUI window, press *Open SMARTCAD* button and select the .dat file previously generated with the *ASSEMBLY* command (step 13). It will unlock the analysis chosen during the solver setting (the corresponding entry, in the NeoCASS main window, will change colour from red to green with a tick on the side).

15. In NeoCASS GUI window upper menu clicking the *Run* tab, the window should look as in Figure 20, then click on *TRIM* button.


Maneuver Definition

1

Mach: 0 | Altitude [m]: 0

Symmetric Maneuvers: Cruise/Climb (AoA, pitch control surfaces)

Anti-Symmetric Maneuvers: Sideslip levelled flight

Parameters

Angle of attack (ANGLEA) [deg]:		Sideslip angle (SIDES) [deg]:	0
Roll rate (ROLL) [1/s]:	0	p rate (URDD4) [1/s^2]:	0
Pitch rate (PITCH) [1/s]:	0	q rate (URDD5) [1/s^2]:	0
Yaw rate (YAW) [1/s]:	0	r rate (URDD6) [1/s^2]:	0
Elevator rotation (elev1r) [deg]:		X acc (URDD1) [m/s^2]:	
Canard rotation (elevC1r) [deg]:		Y acc (URDD2) [m/s^2]:	0
Aileron rotation (aileronr) [deg]:	0	Z acc (URDD3) [m/s^2]:	9.81
Rudder rotation (rudder1) [deg]:	0	Vertical speed (VGUST) [EAS m/s]:	0
1st Flap rotation (flap1r) [deg]:	0	Strut efficiency (LNDGEFF) []:	0
2nd Flap rotation (flap2r) [deg]:	0	Sink speed (VSINK) [m/s]:	0
✓ Symmetric maneuver		Shock absorber stroke (STROKE) [m]:	0

☐ User defined maneuver

Figure 19: NeoCASS manoeuvrer definition.

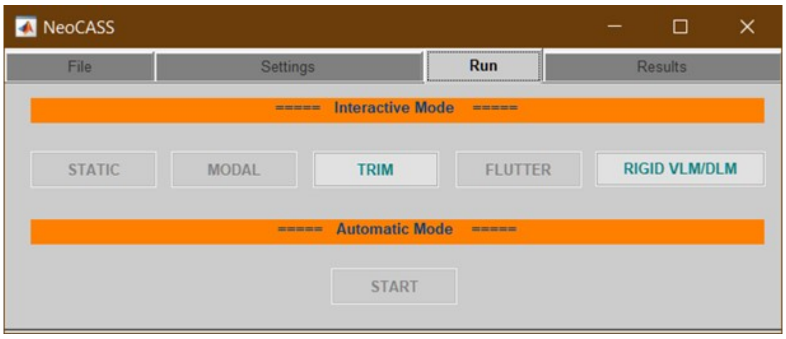


Figure 20: NeoCASS *RUN* tab.

16. In *Trim Condition Selection* window click *OK* button, the computation will start.

17. At the end of the process, in the same folder where the assembly file has been stored, there will be a file called *"The name given at step 13"_man_1.txt*. Inside this file it is possible to read the stability derivatives for the specified condition.

2.2.4 Pacelab APD®

APD® is a tool of the wider Pacelab® software, it is developed by Pace®, a TXT Group® company.
According to their site[5] Pacelab APD® is described as follows:

”*Built on the patented Pacelab Suite design platform, Pacelab APD allows evaluating conventional and non-conventional aircraft configurations in terms of performance, economics and technological risk and helps to assess the impact of conceptual and technical innovations early on in the design process.*
To fully leverage corporate engineering know-how and past IT investments, Pacelab APD supports the integration of commercial analysis tools, proprietary methods, legacy data and codes. This flexible, open software architecture also ensures that Pacelab APD can be customized to very specific or novel use cases.”

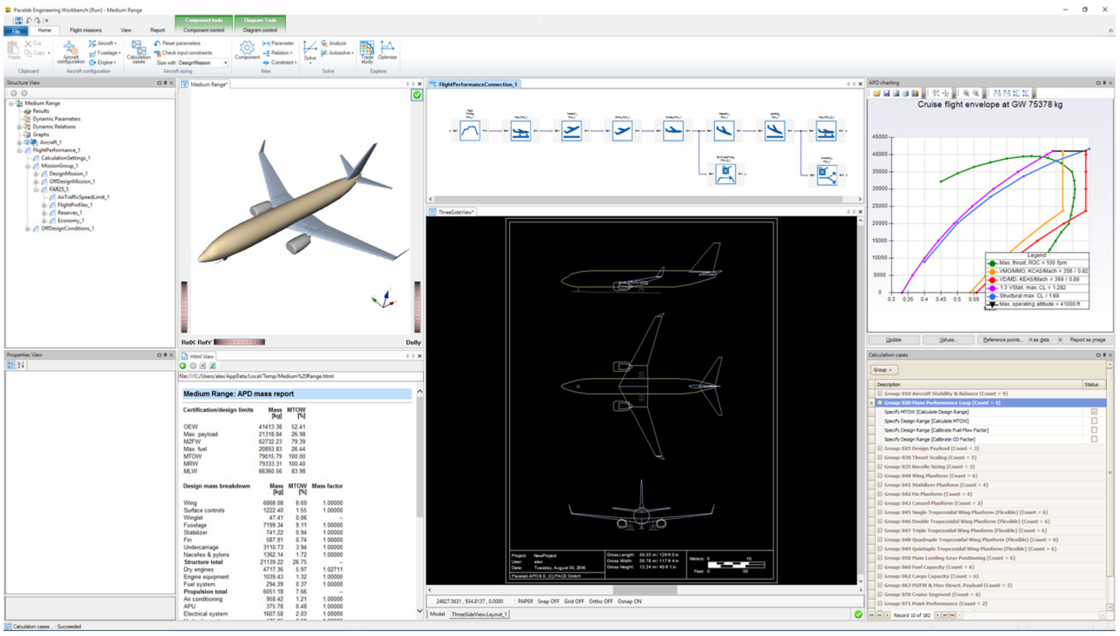

Pacelab Engineering Workbench [Run] - Medium Range
Home
Flight missions
View
Report
Component control
Diagram control
Structure View
Medium Range*
FlightPerformanceConnection_1
APD charting
Cruise flight envelope at GW 75378 kg
Properties View
Html View
Medium Range: APD mass report
ThreeSideView*
Calculation cases
Calculation cases - Succeeded

Figure 21: An APD® window capture.

In practise APD® allows the user to perform a preliminary analysis for a new aircraft starting from a known aircraft model or defining a totally new architecture. For this thesis work the software was only used because of two reasons:

- Among the aircraft already available in the catalogue, there were both the Airbus A320 and the Embraer E190AR with related engine performance data. To understand how these data were used please refer to paragraph 3.3
- Included with the aircraft model there is a CAD, which was used to measure some quantities to insert in AcBuilder.

3 Improvements to Model Architecture

In order to improve the simulator, which was specific for an aircraft, by enabling it to be configured with different models, many changes were needed. In this chapter the most significant ones will be exposed, alongside those which allowed to introduce new functionalities, approaching the behaviour to the requirements for an FNPT Level II.
It is worth empathizing how all the changes done to the simulator had to be compatible with some legacy tests. This means that, if a modification affected the results of a simulation performed with the original aircraft model (for which the simulator was initially developed), then, this change had to be optional, and could be activated only for new aircraft models. This was done to ensure the data consistency with the informations shared with partners, in this way the new simulator can be used to both further developments and to carry on previous project, without having to maintain two distinct versions.

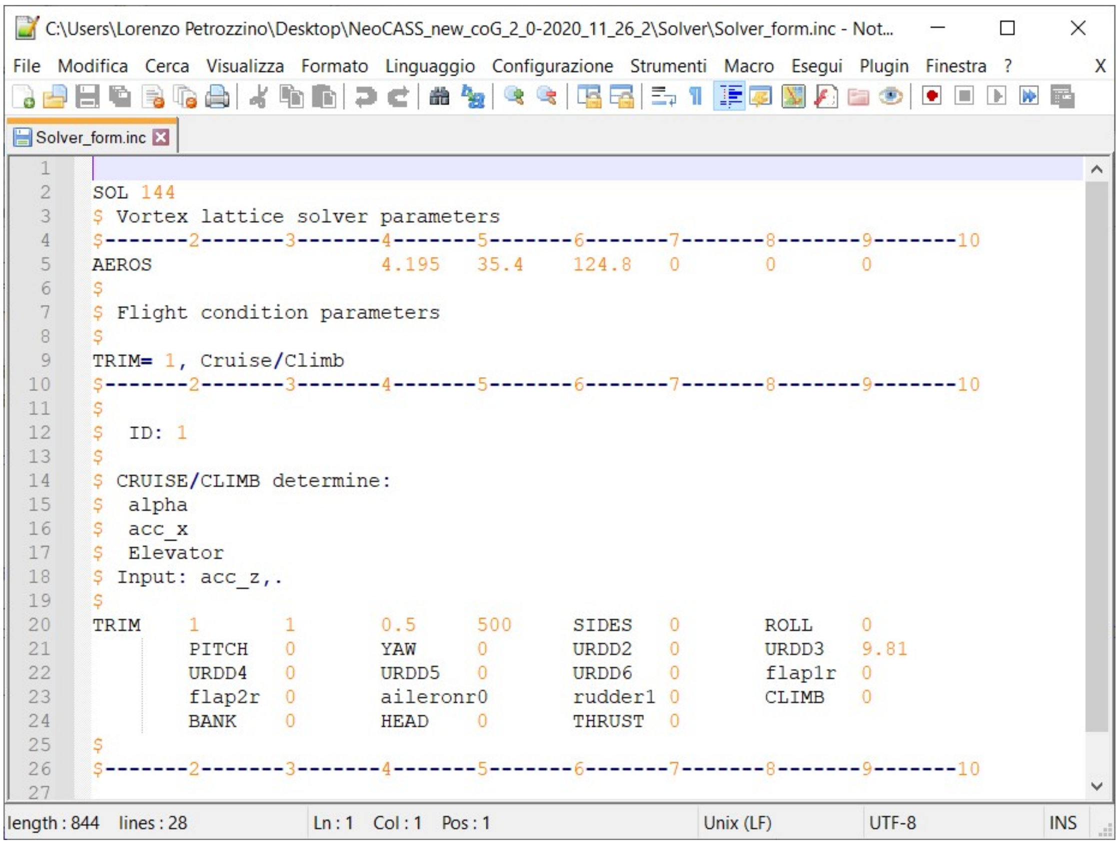
3.1 Generalization

3.1.1 Automatic Procedures

Looking at the NeoCASS procedures to generate the flight derivatives for a single flight condition, it is obvious that, if the objective is to map all the flight envelope for a given aircraft, even with coarse discretization, the operation will be extremely time consuming and error prone. From these considerations some scripts which interact with the NeoCASS code were written. These scripts in part were already available inside the TXT e-solutions®, but they were created with some fixed conditions in mind. The changes made to these script during this thesis work were focused to make the code more automated, requiring minor human interaction when the aircraft or the conditions are changed. Hereafter is presented the new procedure to follow, it differs from the previous one because it directly generates the flight derivatives matrices for all the specified conditions. This procedure is exactly the same as the one presented in paragraph 2.2.3.2 until step 12, the flight conditions specified at this step are not significant: the file created is only needed as a form that the following scripts will fill in. After that is sufficient to execute the script called Main.m, this in its turn calls five functions:

- **evaluate_mach.m** Inside this script the user defines the flight conditions in terms of altitude in feet and air speed in knots. Starting from these values the Mach is calculated for all the combinations available. The output is a Matrix N by M where N is the number of altitudes and M the number of speeds.

- **solver_gen.m** This script generates the solver files for the conditions specified in *evaluate_mach.m* replacing the steps from 9 to 12. As previously said it needs a generic solver file generated with the preceding NeoCASS procedure, starting from that single file it produces the files for any specified condition. The only constraints are that the "solver model" (Figure 22) must be called *Solver_form.inc* and it must refer to the same aircraft for which the automated script is executed. The latter limitation has been left because it is more convenient to define this parameters inside NeoCASS GUI than edit a line of text in a Matlab® file, bearing in mind that, in any case, the first steps of the NeoCASS procedure must be executed manually.
  The scripts acts in this way: it opens *Solver_form.inc* copies the first 19 lines, then edit the 20th with the values defined in *evaluate_mach.m*, and continues the copy until the bottom. The new file is called *SolverHXXXXXVYYY.inc* where XXXXX is the altitude in feet and YYY is the speed in knots. This name is vital for the compatibility with the next steps.


C:\Users\Lorenzo Petrozzino\Desktop\NeoCASS_new_coG_2_0-2020_11_26_2\Solver\Solver_form.inc - Not...
File Modifica Cerca Visualizza Formato Linguaggio Configurazione Strumenti Macro Esegui Plugin Finestra ?
Solver_form.inc
length : 844 lines : 28 Ln : 1 Col : 1 Pos : 1 Unix (LF) UTF-8 INS

```

SOL 144
$ Vortex lattice solver parameters
$-------2-------3-------4-------5-------6-------7-------8-------9-------10
AEROS                   4.195   35.4    124.8   0       0       0
$
$ Flight condition parameters
$
TRIM= 1, Cruise/Climb
$-------2-------3-------4-------5-------6-------7-------8-------9-------10
$
$  ID: 1
$
$ CRUISE/CLIMB determine:
$  alpha
$  acc_x
$  Elevator
$ Input: acc_z,.
$
TRIM    1       1       0.5     500     SIDES   0       ROLL    0
        PITCH   0       YAW     0       URDD2   0       URDD3   9.81
        URDD4   0       URDD5   0       URDD6   0       flap1r  0
        flap2r  0       aileronr0       rudder1 0       CLIMB   0
        BANK    0       HEAD    0       THRUST  0
$
$-------2-------3-------4-------5-------6-------7-------8-------9-------10

```

Figure 22: An example of Solver_form.inc.

- **smart_cad_gen.m** This script performs some trivial actions replacing step 13. It generates files *HXXXXXVYYY.dat*, where XXXXX is the altitude in feet and YYY is the speed in knots, The .dat file contains only three lines with the path of the files listed at step 13.
- **run_trim.m** This script simply calls *solve_free_lin_trim.m* replacing steps from 14 to the end, performing the computations to obtain flight derivatives.
- **stab_der_reader.m** This script performs some additional actions compared to the previous procedure, indeed, it takes the previously generated files, containing the flight derivatives, and fill the matrices that will be used by the simulator. N.B. As shown in Table 1 NeoCASS is able to compute the flight derivatives for all the surfaces except the tail horizontal stabilizer, which have to be obtained by other methods, and inserted manually inside the stability matrices. A workaround to this issue is presented in paragraph 3.1.2

	−	α	β	p	q	r	−	−	−	−	−	−	−	−	l_{ail}	r_{ail}	l_{flap}	r_{flap}	rud	$elev$
X	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
Y	−	−	x	x	−	x	−	−	−	−	−	−	−	−	x	x	−	−	x	−
Z	−	x	−	−	x	−	−	−	−	−	−	−	−	−	x	x	x	x	−	x
L	−	−	x	x	−	x	−	−	−	−	−	−	−	−	x	x	−	−	x	−
M	−	x	−	−	x	−	−	−	−	−	−	−	−	−	x	x	x	x	−	x
N	−	−	x	x	−	x	−	−	−	−	−	−	−	−	x	x	−	−	x	−

Table 1: An example of stability matrix produced.

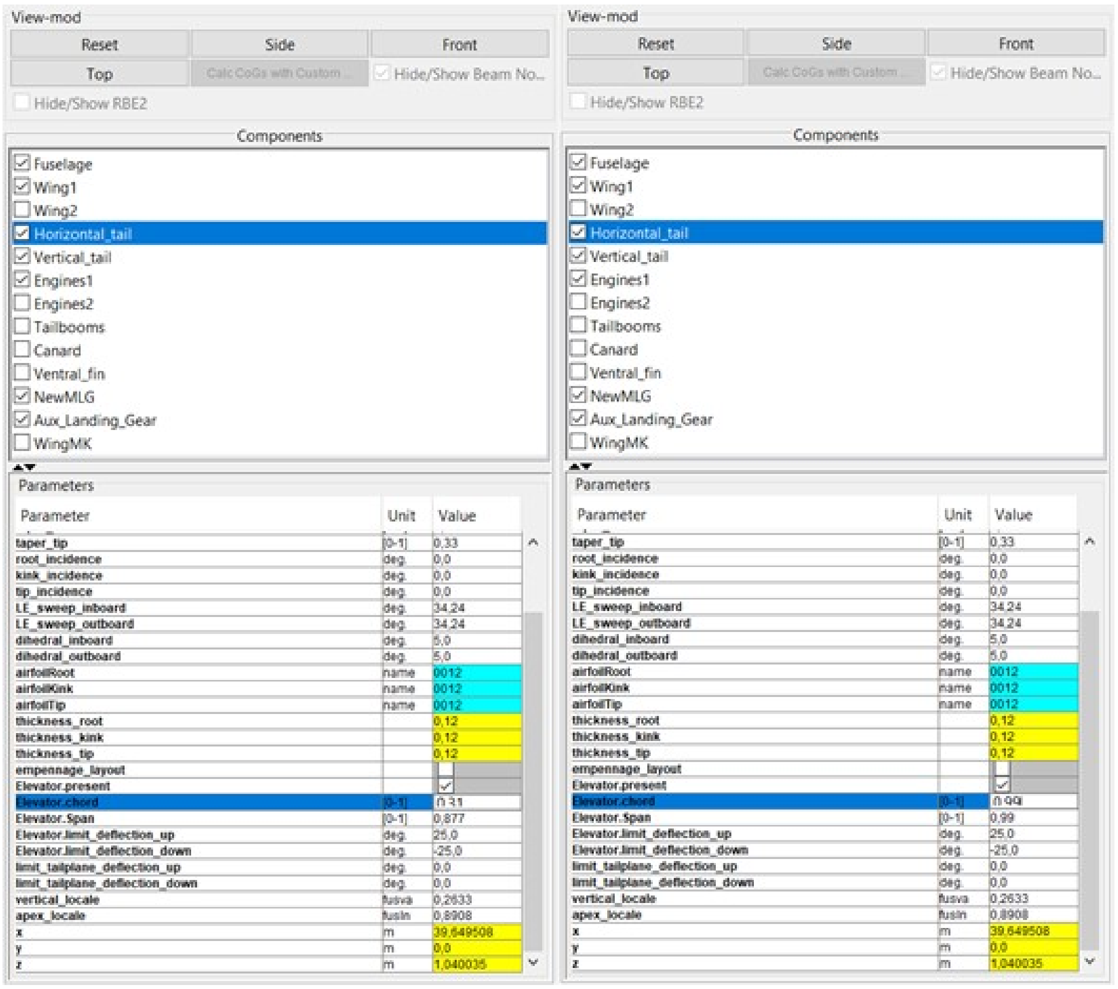
3.1.2 Tail Horizontal Stabilizer Derivatives

One of the strongest limitations linked to NeoCASS is the lack of flight derivatives related to the Tail Horizontal Stabilizer (from here on after, for shortness sake, it will be called THS). In order to cope with this constraint an alternative procedure was developed. The idea behind is: because NeoCASS is not able to evaluate the flight derivatives for the THS, but it is able to compute the derivatives for the elevator,then it is possible to define a "fake" elevator which takes up the entire surface of the horizontal stabilizer and compute the derivatives for this virtual surface; in a second moment the elevator derivatives so achieved will be replaced to the ones of the THS for the "real" aircraft (with the well sized elevator). Because NeoCASS performs the calculations on an empirical basis, the values computed for the THS were not compatible with the real ones, because of this, before being used they were multiplied for a coefficient extracted from real data referred to an Airbus A320. This means that the range of application for this methodology is limited to single aisle civil aircraft with a conventional architecture.

Let's quickly see what this new procedure is about: first is necessary to create, following AcBuilder tutorial (paragraph 2.2.3.1), the .xml file for the reference aircraft, setting both values of *Elevator.chord* and *Elevator.Span* to 0.99 (Figure 23). It follows the generation of stability matrices using the NeoCASS automatic procedure (paragraph 2.2.3.2).
It is here that the changes show off:

- The script *stab_der_reader.m* presented in chapter 3.1.1 was edited to take in input an ASCII matrix and use it to add another column referred to the THS. By default the ASCII matrix is filled with zeros, so the 21th column is empty too.
- A new function called *ASCII_Table_Creator* was introduced. It takes as input the matrices created by NeoCASS automated procedure, copies the 20th column in the 21th column of the ASCII matrix multiplied by a correction coefficient, and saves it.

The second time that NeoCASS is run (with the correct elevator dimensions), it will generate the matrices with the column related to the THS, as shown in Table 2.



Components:

- ✓ Fuselage
- ✓ Wing1
- ☐ Wing2
- ✓ Horizontal_tail
- ✓ Vertical_tail
- ✓ Engines1
- ☐ Engines2
- ☐ Tailbooms
- ☐ Canard
- ☐ Ventral_fin
- ✓ NewMLG
- ✓ Aux_Landing_Gear
- ☐ WingMK

Parameter	Unit	Value
taper_tip	[0-1]	0,33
root_incidence	deg.	0,0
kink_incidence	deg.	0,0
tip_incidence	deg.	0,0
LE_sweep_inboard	deg.	34,24
LE_sweep_outboard	deg.	34,24
dihedral_inboard	deg.	5,0
dihedral_outboard	deg.	5,0
airfoilRoot	name	0012
airfoilKink	name	0012
airfoilTip	name	0012
thickness_root		0,12
thickness_kink		0,12
thickness_tip		0,12
empennage_layout		☐
Elevator.present		✓
Elevator.chord	[0-1]	0,31
Elevator.Span	[0-1]	0,877
Elevator.limit_deflection_up	deg.	25,0
Elevator.limit_deflection_down	deg.	-25,0
limit_tailplane_deflection_up	deg.	0,0
limit_tailplane_deflection_down	deg.	0,0
vertical_locale	fusva	0,2633
apex_locale	fusln	0,8908
x	m	39,649508
y	m	0,0
z	m	1,040035

Components:

- ✓ Fuselage
- ✓ Wing1
- ☐ Wing2
- ✓ Horizontal_tail
- ✓ Vertical_tail
- ✓ Engines1
- ☐ Engines2
- ☐ Tailbooms
- ☐ Canard
- ☐ Ventral_fin
- ✓ NewMLG
- ✓ Aux_Landing_Gear
- ☐ WingMK

Parameter	Unit	Value
taper_tip	[0-1]	0,33
root_incidence	deg.	0,0
kink_incidence	deg.	0,0
tip_incidence	deg.	0,0
LE_sweep_inboard	deg.	34,24
LE_sweep_outboard	deg.	34,24
dihedral_inboard	deg.	5,0
dihedral_outboard	deg.	5,0
airfoilRoot	name	0012
airfoilKink	name	0012
airfoilTip	name	0012
thickness_root		0,12
thickness_kink		0,12
thickness_tip		0,12
empennage_layout		☐
Elevator.present		✓
Elevator.chord	[0-1]	0,99
Elevator.Span	[0-1]	0,99
Elevator.limit_deflection_up	deg.	25,0
Elevator.limit_deflection_down	deg.	-25,0
limit_tailplane_deflection_up	deg.	0,0
limit_tailplane_deflection_down	deg.	0,0
vertical_locale	fusva	0,2633
apex_locale	fusln	0,8908
x	m	39,649508
y	m	0,0
z	m	1,040035

Figure 23: Changes to elevator in AcBuilder.

	–	α	β	p	q	r	–	–	–	–	–	–	–	–	l_{ail}	r_{ail}	l_{flap}	r_{flap}	rud	$elev$	THS
X	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Y	–	–	x	x	–	x	–	–	–	–	–	–	–	–	x	x	–	–	x	–	–
Z	–	x	–	–	x	–	–	–	–	–	–	–	–	–	x	x	x	x	–	x	x
L	–	–	x	x	–	x	–	–	–	–	–	–	–	–	x	x	–	–	x	–	–
M	–	x	–	–	x	–	–	–	–	–	–	–	–	–	x	x	x	x	–	x	x
N	–	–	x	x	–	x	–	–	–	–	–	–	–	–	x	x	–	–	x	–	–

Table 2: The new stability matrix.

3.1.3 Aircraft Selection GUI

From this paragraph to the end of the chapter the changes made to the Matlab® and Simulink® code, in order to wider the range of aircraft compatible, will be presented. The first window the user run into when the simulator is started is the GUI for the selection of the Aircraft (Figure 24). Due to the presence of multiple aircraft models, now every aircraft has its own configuration folder. As first step the simulator checks if any of those folders are present in the PATH (because they were loaded on any previous simulation), if so it removes them to avoid to call the wrong configuration files. Then the GUI visible in Figure 24 appears, the program waits until the user pushes the *Select aircraft* button, when the event occurs it loads all the configuration files related to the aircraft selected. The configuration scripts executed are:

- *Configuration_"Aircraft Name"* where are defined all the aircraft specifications, for more information about it please refers to paragraph 3.1.4.
- *ISA_TABLES*.
- *INIT_TARGET_STAB_DERIV_"Aircraft Name"*.
- *INIT_AUTOTRIM_DATA_"Aircraft Name"*.

(These three files were already presented in paragraph 2.1.1).

- *Turbofan_table_reader* which reads the engine thrust and SFC from previous created tables. More information about the argument at paragraph 3.3.
- *ConfigurationTestList_"Aircraft Name"* which consist of a list with all the test available for the aircraft selected.

Finally, it loads the tests folder and prompt the test selection window, the same as Figure 2.

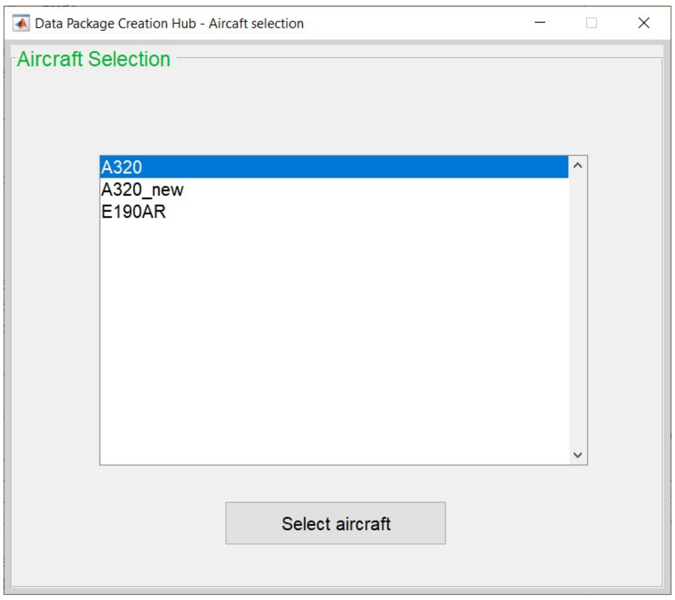


Figure 24: Aircraft selection window.

3.1.4 Aircraft Configuration Files

A quite significant work, which however stays hidden to the user's eyes, was the generalization of all the Simulink® model parameters which were specific for the aircraft. For the most part the changes consisted in taking the values inside the gain blocks, spread out over the model, and create new variables inside the aircraft configuration file to host these values, then insert the variables in the related blocks. This work mostly concerns the MathPilot section. But it also affected other areas of the simulator; a significant one was the surfaces control block, called *SAS & FBW C* CONTROL LAW "G"MANEUVER DEMAND*, where the aircraft control laws are defined. For example, on the Airbus A320, there is a law which causes both ailerons to be deflected downward if the flap are extracted, in this way when the aircraft needs more lift (typically when it is flying close to the ground) the ailerons can give their contribution. Of course this behaviour is not implemented in all the aircraft, or the deflection law is different, e.g. the Airbus law deflect of 5° the ailerons regardless of how much the flaps are extracted, on the other side another aircraft may have a law which controls the deflection in function of the flaps level, for this reason this laws must be optional. Indeed looking at Figure 25 the *FLAPS_AILERON_COORDINATOR* variable dictates if this behaviour has to be implemented or not, while the AIL_SHIFT_NOT_CLEAN and AIL_SHIFT_CLEAN variables define the magnitude of the deflection when the flaps are respectively extracted and retracted.

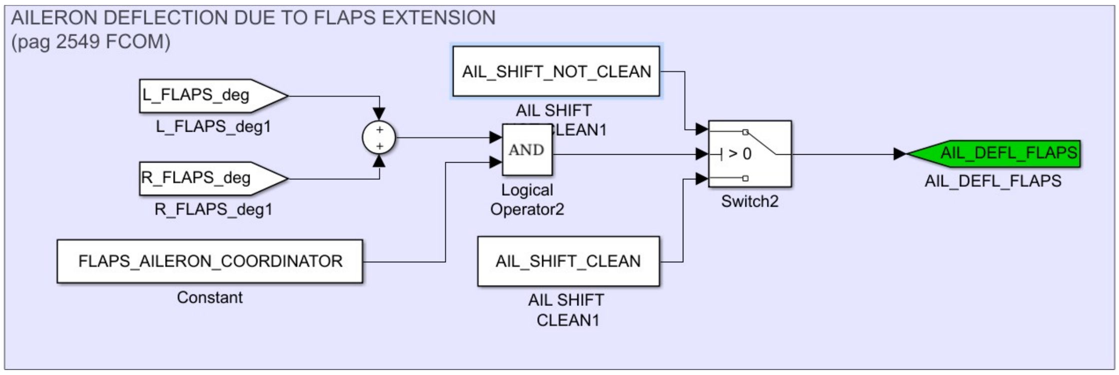


Figure 25: The aileron coordination law implemented in Simulink®.

3.1.5 MathPilot Stop Condition

A critical behaviour of the simulator was the inability, during the Math Pilot Trim, to inform the user if the operation ended successfully or not. Recalling paragraph 2.1.2, the Math Pilot Trim tries to trim the aircraft for 80 s then it stops, but it may happen that the trim reached a satisfactory magnitude of error before that time, causing the simulation to be longer than necessary, or it may be that it never reaches a trimmed condition, this is the worst case because the simulation will start with wrong data, vanishing its usefulness. In any case it represent a waste of time and energy. This is the reason because the system in Figure 26 was implemented. In order to work this system requires that the trim simulation is run for more than 80 s, for example it was set to 500 s, it may seems a value too big but it is important to remember that as the aircraft is trimmed the simulation will stop, so unless the aircraft cannot be trimmed at the given conditions that time will never be reached. That being said let's see how this block works, it compares the current variables values visible in the left side of Figure 26 with the values they had in the previous simulation instant, if these two measures differs less than a

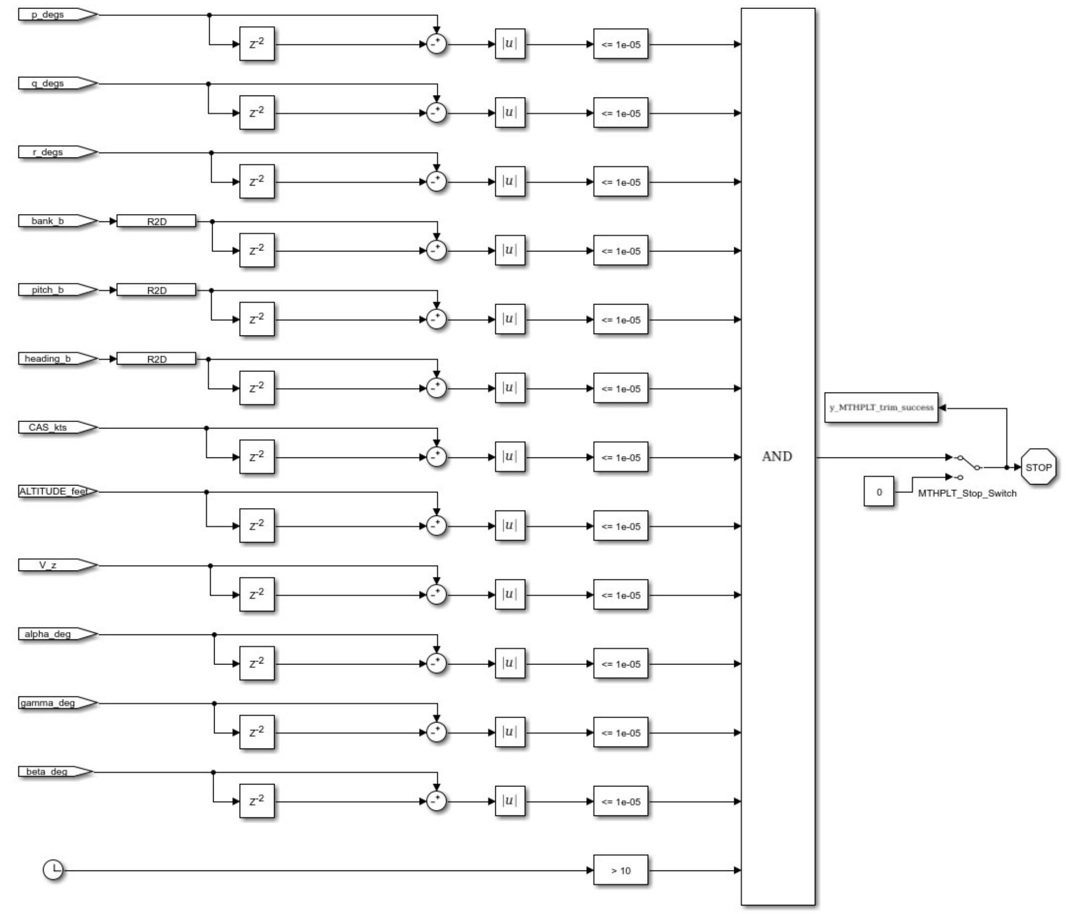


Figure 26: The Math Pilot successful trim implemented in Simulink®.

certain threshold (in this case 10^{-5}) then that physical quantity is considered trimmed. To have the condition of trimmed aircraft all the variables listed have to be trimmed in the same instant, and at least 10 s must have passed. The latter condition was introduced to avoid that the trim stopped at the first iteration when of course all the values are fixed. If all these conditions are respected at the same time the signal which leaves the *AND* block switch from 0 to 1 storing this information in *y_MTHPLT_trim_success* and activating the *STOP* block which causes the simulation to be interrupted. When the simulation finishes the Matlab® code check the last value from *y_MTHPLT_trim_success* if that value is 1, the trim was successful and proceeds, if it is 0, it was not, and the code execution is ended.

3.1.6 Stall Tables

Another effort to generalize the simulator behaviour was made when reading stall tables. The stall tables contain the stall speed, function of flaps extension and aircraft weight. If the speed during the simulation at a given weight and flaps extension reaches a value lower than the one inside the table, a stall flag is raised and the simulation is stopped after 2 s. To detect if a stall condition is reached, the simulator uses two stall tables: the first one is used when the flaps extension has a specific value, the second one has a wider flaps range and is used to obtain the stall speed for a generic condition, which is interpolated from this table. This was done because the first matrix contains data extracted by flight test, while the data inside the second one were obtained via other tools less reliable. By default these tables were defined as two matrices inside a Matlab® script, which didn't allowed an easy customization in case of a change in aircraft. To overcome this limitation the matrices were moved in *Stall_Matrix_"Aircraft Name".xlsx*, and the following script reported in Appendix A.1 was written to read one rather than another.

The function receives in input the name of the aircraft selected and a variable containing an integer number called *matrix_selector* which can be:

- 0 If the matrix to read is the first one, for specific values.
- 1 If the matrix to read is the second one, for generic values.

These values (0 and 1) were left to ensure the compatibility with the legacy code. After initializing the variables, the script tries to understand the dimension of the matrices. First, in order to determine the number of columns, it moves along the rows, looking for when each matrix starts and ends, then, it performs the same action but along columns to determine the number of rows. Next, it checks if *matrix_selector* value is compatible with the number of matrices founded by the search, e.g. if for other aircraft the stall data are reliable, and range all the possible weights and flaps extensions, there is no need to have two matrices. But it is possible to be in a condition where the flap extension does not match exactly one of the matrix values, so the function is called with *matrix_selector = 1*, this would cause the script to try to interpolate the second matrix which doesn't exist, raising an error. From here the need to check if the value passed is compatible with the number of matrices available, and if it is not the case, *matrix_selector* needs to be changed to point the right matrix. Finally, with the informations previously gathered, the script is able to generate the flaps and weight arrays together with the stall matrix.

3.2 Landing Gear

The CS-FSTD(A) [3] gives lot of importance to the aircraft interaction with the ground, giving several requirements for the landing gear behaviour. During this thesis work, where the goal was to throw the basis for a FNPT level II, and not to realize a simulator compatible with this standard, it was developed a simple landing system which allowed to have a basic interaction model.

3.2.1 Background

The model was based on the assignment done at university by a TXT e-solutions® employee [6].

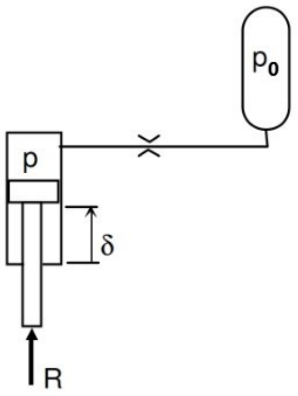


Figure 27: Shock absorber hydraulic scheme.

The shock absorber response was obtained by an hydraulic formulation (Figure 27) and divided in two components:

- The elastic force
$$R_e = p_0 \left(\frac{V_0}{V_0 - A\delta} - 1 \right)^{\gamma} * A \tag{1}$$
- The viscous force
$$R_v = \frac{1}{2} k \rho A^3 \left(\frac{\dot{\delta}|\dot{\delta}|}{{A_0}^2} \right)$$

  where

  - p_0 : initial landing gear accumulator pressure;
  - V_0 : initial landing gear chamber volume;
  - A : piston surface;
  - δ : leg excursion;
  - $\dot{\delta}$: leg excursion ratio;
  - γ : heat capacity ratio;
  - k : lost coefficient for orifices;
  - ρ : fluid density;
  - A_0 : orifices total area.

3.2.2 Implementation

In the following paragraph the landing gear integration inside Matlab®/Simulink® simulator, based on the above equation, will be discussed.
The architecture adopted for the landing gear system is the traditional tricycle with the rearward main landing gear equipped with two legs and the forward auxiliary/nose landing gear with a single leg. To understand how the system works, let's take as example the code for the main landing gear right leg reported on appendix A.2.

First, by means of trigonometric considerations based on the longitudinal and lateral aircraft attitude, the position of each leg apex is evaluated, these equation are valid under the assumption that the landing gear is perfectly aligned to aircraft z body axis
Once the theoretical apex position is known it is possible to evaluate the leg compression along its axis, this evaluation is done only when the theoretical apex position is negative because it means that the tyre, which is assumed to be perfectly rigid, has touched the ground.
Immediately after the script checks if the end of stroke condition is reached, if so, a warning message is prompted to the user by the Matlab® command window, and a signal to stop the simulation is sent. Then the compression rate is calculated using the finite difference method.
Now all the values to compute the reaction forces are available, and it is possible to proceed.
To take into account that the orifices are shaped to give different performance during the extension rather than compression, their "virtual section" is reduced during extension. During extension is also possible, because of the mathematics behind the model, that the landing gear produces a force which draws the aircraft to the ground, of course this is not physically feasible, in this case the force is zeroed.
At the end of the code listed is it possible to see a section where the current landing gear compression is compared to the compression at the previous instant, this check is performed to inform the user in case the aircraft leg touched the ground, or if it detached from the landing strip, event which can occur if the pilot performs a landing manoeuvre to hard which causes the aircraft to bounce.

The landing system carries out two main functions: the first one is to absorb the landing shock due to the impact with the ground, implemented in the code above, the second one is to slow down the aircraft when it has already touched the ground, implemented in a very simplistic fashion in the code reported in Appendix A.3. The brake starts when the *brake_status flag* is set to 1 and the speed of the aircraft is greater than 0, the latter condition is meant to avoid that once the aircraft is stopped the braking force continues to be applied causing the aircraft to move back, which of course does not replicate the real behaviour. The braking action is modelled as a constant force aligned with the X-body axis, these are strong assumptions which do not reproduce the real behaviour of the braking system. Indeed the force intensity depends on several different parameters, such as the tyres and ground materials, the pressure of the plant, the wear status of the disks brakes and so on. However, to take into account all these parameters would have made the model significantly more complicated, losing the intended simplicity. Also the direction of the force is not correct, but if the braking action starts only when all the legs has touched the ground, the error induced by this assumption becomes marginal. Finally the moment induced by the braking action is computed, the lever arm takes into account that the force is transmitted to the aircraft via the rim hub, indeed 0.5 m is an assumption about the wheel diameter. As happened for the touch condition, even in this case the user is informed about the braking status, also in this case the check is performed comparing the current status with the previous one.

The landing system addition required the stall behaviour to be changed. Indeed, looking at Figure 28, where the yellow blocks were added because of the landing gear integration, it

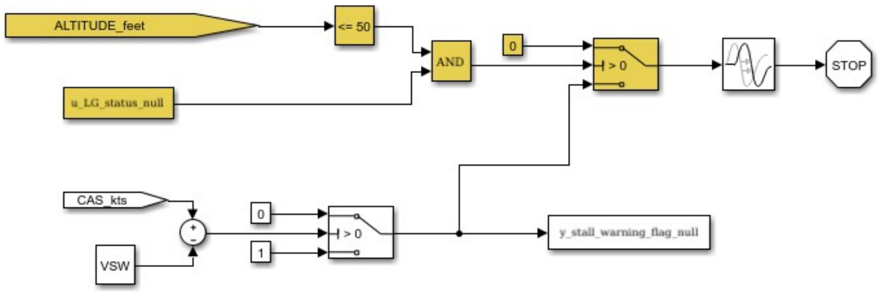


Figure 28: Stall system.

is possible to notice that before, the stall condition was detected if the air speed fell under the interpolated stall speed value. This behaviour would prevent the simulator to perform a complete landing. To allow such operations a check was added: the aircraft can encounter the stall condition only if the aircraft has an altitude higher than 50 ft or if the landing gear is retracted, if both these conditions are unmet, as occurs during landing or take-off, the stall check is suspended. The landing system integration required one last change, which affected the motion equations. These modifications were needed because when the speed approaches the zero value condition, some unexpected behaviours could happen. Before the introduction of the landing gear the aircraft couldn't reach such condition, so this issues never revealed. The problem is that the V CAS and its x component appear as the denominator in some equations, particularly when computing flight angles, this caused, as the speed approaches to 0, to obtain very high values, which on their turns, appeared in other equations causing unexpected behaviours linked exclusively with the finite mathematics behind the calculator, and not at all with the physics of the aircraft. These aspect were solved by forcing the values of the flight angles to 0 when the aircraft speed falls under a certain soil. This solution diverge from the real behaviour but it doesn't induce other errors because the speed is so low that the aerodynamics forces produced are negligible. Another issue associated with the landing gear integration is the forward speed which in some cases could become negative because of the braking system action, to avoid this event a check is performed: if the speed and the force computed along x body direction are both negative the x acceleration is forced to 0. A second check is performed for the x speed, its value must be positive otherwise it is set to 0. The solution adopted (visible in the upper side of Figure 29) prevents the aircraft to recede under the braking action, but allows in case of a propulsive action to start the forward motion. One last check is performed on the vertical equations, as shown at the bottom of Figure 29, if the forward speed reaches the 0 value, then the vertical speed is forced to the same value preventing the γ angle to assume unrealistic values. The only downside of this solution, is the presence of several 0, which required checks in the definition of some quantities to avoid NaN values, which would cause further issues.

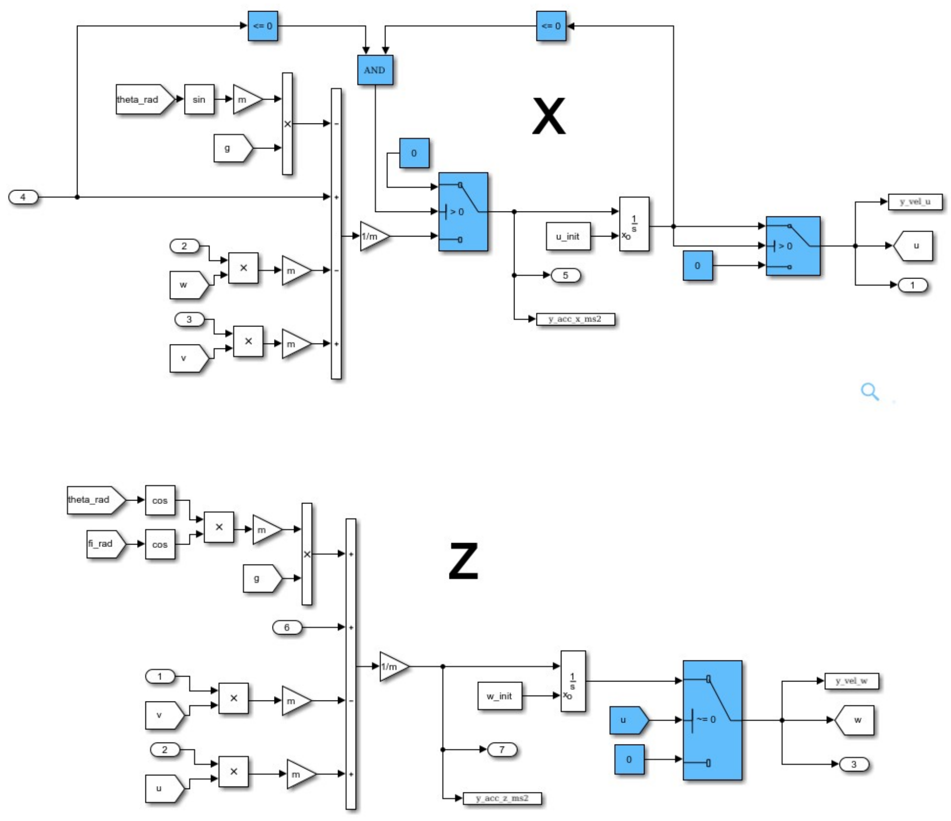


Figure 29: Motion equation.

3.3 Engine Model

Another element which required more fidelity was the engine system, indeed in the original model the thrust was exclusively function of throttle lever according to a direct relationship.

3.3.1 Background

To model this system two different approaches were available:

- **The physical approach**: with this method the system is decomposed in smaller subsystems until it consist of simple physical problems only, easily resolvable via known equations. E.g. an engine is decomposed in all its parts (the intake, the compressor, the combustion chamber, the turbine and so on), for each one of these part the conditions and the process involved are analysed, using as boundary relation that the exit condition from a stage are the initial condition for the following. In this way it is possible to obtain a model which fidelity ranges from low (the case of a steady working point with fixed ratios) to very high (if thermal aerodynamic and structural considerations are involved), depending on the computational budget, related to the complexity involved, and on the engine parts informations available.
- **The tabular approach**: using this method the output of the engine is not related to physical processes involved. As the name suggests, the output value is linked to one or more variables via one or more tables. These tables can be produced either via test measurements or using an analytical approach as described above. The advantages are the speed execution, because, in order to get the final value, the algorithm needs only to interpolate some values from a table rather than resolve several equations, as well as it is not needed to know all the the evolutions which take place inside the engine. The disadvantages are the other side of the same coin, because the only information available are the relations between the input and the output data, and all the middle processes are transparent. In case even a single part of the engine changes, or there is an input which was not considered before, it is no more possible to obtain a relationship for this new layout.

At first, in order to understand which was the best approach for this simulator, a trade-off analysis was made, but because it has not been possible to get the data related to the specific propulsion system, the physical approach quickly became unviable. To get the values of the thrust produced to insert inside the tables two possible path were available: the first one was to use Pacelab APD®, the tool presented in paragraph 2.2.4, the second one was to use the turbofan block form the aerospace pack available as a Matlab® extension. Both alternatives have three inputs (the coulours refers to Figure 30):

- Mach green.
- Altitude red.
- Throttle Lever blue.

In order to select one rather than the other, the "bench model" was created. The model is composed of three step counter, one for each model input, which range from zero to the maximum value available for the reference variable. Each signal source has its own time constant, which is the time in seconds after which the counter is increased, following the rule:

$$T_{curr.counter} = T_{prev.counter} * \#values_{prev.counter}$$

in such a way it is possible to reproduce all the conditions available, without have any repetition. For example if the Mach counter has time constant of 1 s, and can take 5 values the altitude counter will have time constant of 5, in this way for each altitude step the Mach counter can perform a complete range. The same procedure can be applied to the throttle respect to the altitude. Anyway, to understand more clearly how the counter works please look at the plot in Figure 30.

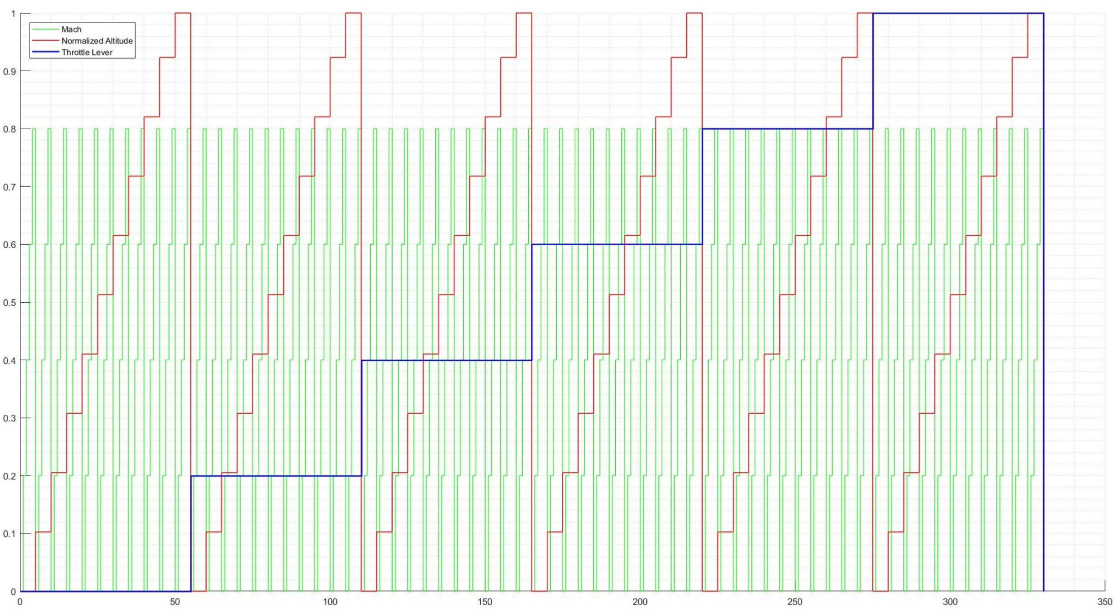


Figure 30: Turbofan input values plot.

The results produced by this test are shown in Figure 31, the lookup table model (from Pacelab APD®) is plotted in blue while the Turbofan Simulink® block is plotted in orange. From the plot it is possible to obtain some informations about the input influence on the models:

- In both models the throttle lever acts like a direct gain, indeed when it increases the graph are simply scaled proportionally.
- The effect of the Mach instead, is quite different. While an increase of the Mach value in the turbofan block always causes a decrease of thrust produced, in the lookup table model we assist at a reversal of behaviour. As the previous model when the engine is working at low altitudes an increase of the Mach causes a reduction of the thrust, but there is an height from which an increase of the Mach causes an increase of the thrust. This behaviour can be due to some analysis, made while obtaining the values, related to the compression effects which increases the density of the flow processed by the engine. However, as previously discussed, because of the inherent limitations which affect the tables, it is not possible to know with certainty which are the thought involved.
- The altitude effect is the one where the two models diverge the most. The lookup table model decreases in a linear way, while the turbofan Simulink® block seems to use an

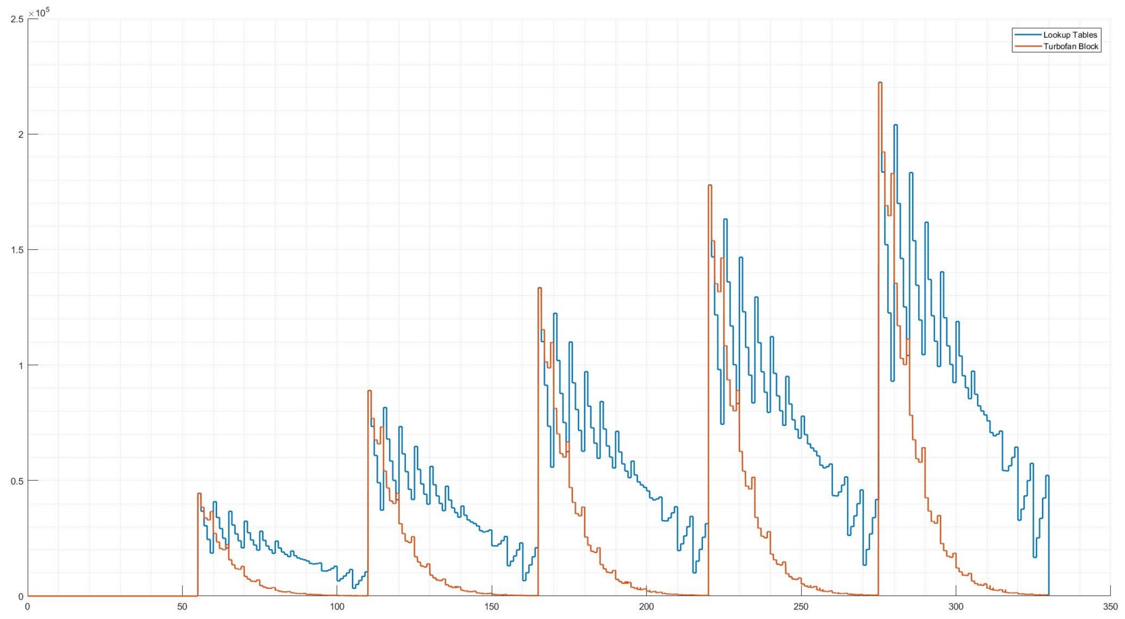


Figure 31: Turbofan thrust comparison plot.

exponential correlation causing the thrust to fall almost to 0 for high altitudes.

The behaviour of the Simulink® turbofan block is not compatible with the flight condition of an Airbus A320, for this reason the model was discarded focusing on the lookup tables obtained via Pacelab APD®.

The same process was used for the fuel consumption, but in this case the more accurate was the one linked to the Simulink® turbofan block, for this reason the table used by that block was extracted. The resulting model visible in Figure 32 receives as input the thrust produced together with the Mach, and gives in output a value which combined with other parameters produces the fuel flow.

3.3.2 Implementation

In the following paragraph the implementation of the model just presented will be discussed, focusing on the changes required by the trim process due to this integration. Looking at Figure 33 it is possible to notice in the upper side the legacy code, this as previously said, must be left to guarantee the compatibility with old tests. The engine model is selected by the switch *TLA_Thrust_Switch* at the top right end of the figure, the switch is controlled by a variable defined in the configuration file.

Focusing on the new model after the lookup table there is a violet section inside which a first order filter is implemented, this filter reproduces the engine dynamic behaviour, this means that the engine value does not switch instantly from one value to another, but it evolves according to its time constant. Of course an engine has a behaviour which is way more complex than a first order, but lacking the information about the specific model, and wanting to realize a system which is configurable, this solution is an acceptable trade-off. Another element which stands

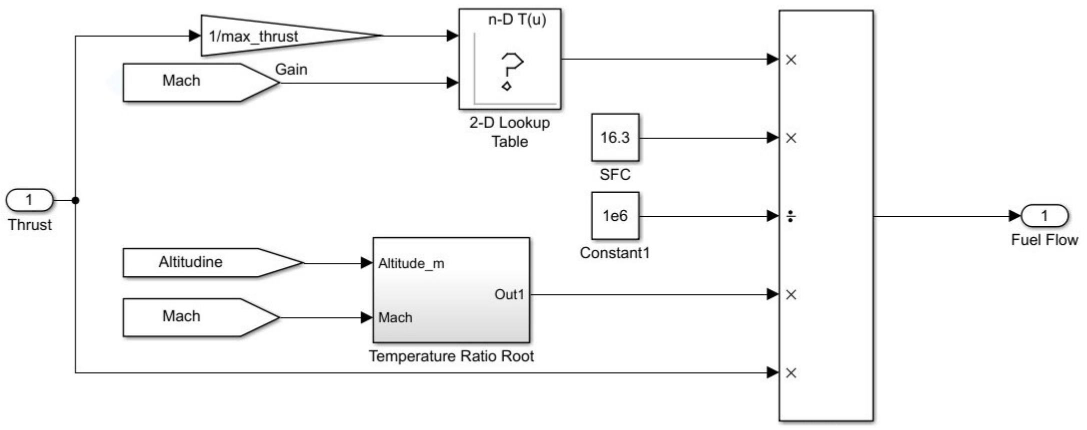


Figure 32: Turbofan fuel flow model.

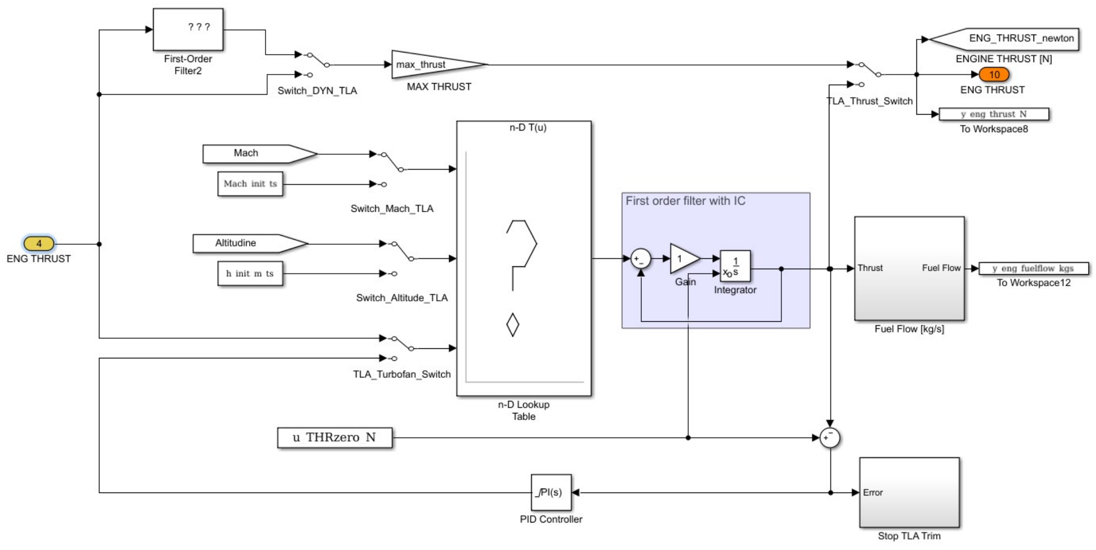


Figure 33: Turbofan engine model in Simulink®.

out are the switches before each input of the lookup table. They are used to switch the inputs between the trim and the simulation configuration. To understand how the trim procedure works let's recall what said in chapter 2.1, there are two ways to trim the aircraft: the Newton Raphson method and the Math Pilot. Let's assume to use the Newton Raphson method. First a normal trim is performed with the old engine model (on the top of Figure 33) exactly as it was before, if the trim succeeds the trim condition are found. However, the throttle lever is computed as a fraction of the sea level max thrust and not of the max thrust available in that condition, because of this it is required to perform another trim. This time the trim is performed on the new engine model, but because the only value to find is the throttle lever, the input switches are turned down. With the switch so configured the lookup table receives as input the Mach and altitude found during the previous trim, while the throttle lever is regulated by a PID controller in order to minimize the error. (The stop condition will be discussed later on in this paragraph). Then the simulation starts, the new model is used, but this time the switches are configured so that the lookup table are interpolated for the current values of all the three inputs. If the trim is performed using the Math Pilot the switches have the same layout as in the simulation. The

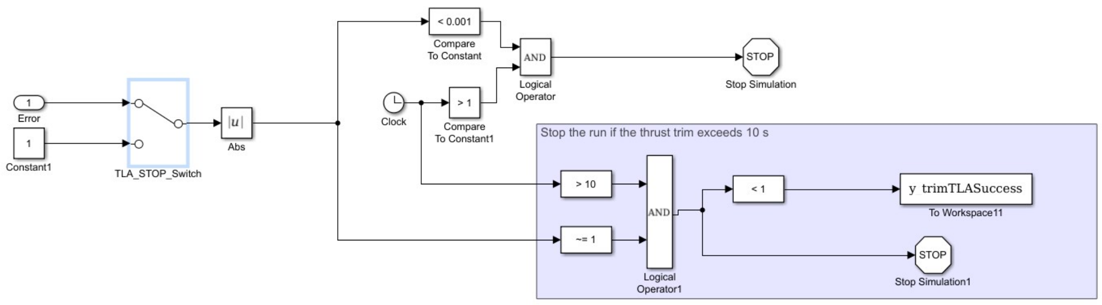


Figure 34: Turbofan trim stop condition model in Simulink®.

stop condition model implemented in Figure 34 is exclusively used by the throttle lever trim, and works as follows. The error signal sent to the PID is also sent to this block, where are implemented two different stop conditions:

- The first one occurs if the error goes under a certain thresholds (in this case 10^{-3}) and trim run was performed for at least one second.
- The second one was implemented to stop the simulation after a certain amount of time, even in case the thrust isn't enough to equilibrate the drag. To do so the time must be higher than a threshold (10 s), and the error must not be 1 (this is a check to deal with the solution adopted when the block is bypassed).

At the end of the trim, the Matlab® script checks the last value stored in *y_trimTLASuccess*: if it is 1 the trim was successful, if it is 0 it was not.

4 Embraer E190AR

The E190AR (shown in Figure 35) is the long range variant of the E190 produced by the brazilian company Embraer. This particular model was chosen because it belongs to the same A320 aircraft class, still being significantly smaller than the Airbus aircraft. This means that the two aircraft have both similar and different behaviours, representing a good test bench for the changes made to the simulator.

Figure 35: Embraer E190AR.

Hereafter, in table 3, is presented a specification comparison between the targeted aircraft.

	Embraer E190AR	Airbus A320	
Maximum Take-off Weight	51800	77000	kg
Operative Empty Weight	27820	42600	kg
Wing Area	92.53	122.6	m^2
Wing Span	28.72	34.09	m
Length Overall	36.24	37.57	m
Height Overall	10.57	11.76	m
Fuselage Width/Height	3.01/3.35	3.95/4.14	m
Sea Level Maximum Installed Thrust	82300	120000	N
Maximum Operating Mach	0.82	0.82	-
Service Ceiling	12500	12500	m

Table 3: Embraer E190AR specifications.

4.1 Embraer E190AR Derivatives

The main information source used to obtain the data required by AcBuilder and NeoCASS was a CAD model extracted from Pacelab APD®. As described in paragraph 2.2.3.1, AcBuilder requires some very specific parameters which are hard to find. For such parameters, the values were taken from the A320 ones or scaled starting from those.

Once all the data required were gathered, the procedure presented in paragraph 3.1.1 was performed. In table 4 are represented the Embraer flight derivatives matrices obtained for a given combination of altitude and speed , followed by table 5 showing the Airbus flight derivatives for the same conditions.

	–	α	β	p	q	r	$\cdots$	l_{ail}	r_{ail}	l_{flap}	r_{flap}	rud	$elev$	THS
X	–	–	–	–	–	–	$\cdots$	–	–	–	–	–	–	–
Y	–	–	-0.557	-0.338	–	0.595	$\cdots$	0.061	-0.061	–	–	-0.307	–	–
Z	–	-6.150	–	–	-19.786	–	$\cdots$	-0.196	-0.196	-0.980	-0.980	–	-0.645	2.019
L	–	–	-0.182	-0.551	–	0.096	$\cdots$	0.149	0.149	–	–	-0.035	–	–
M	–	-4.978	–	–	-62.016	–	$\cdots$	0.124	0.124	0.621	0.621	–	-3.102	-7.504
N	–	–	0.259	0.063	–	-0.331	$\cdots$	$6.2e^{-3}$	$-6.2e^{-3}$	–	–	0.188	–	–

Table 4: An example of stability matrix produced for the E190AR.

	–	α	β	p	q	r	$\cdots$	l_{ail}	r_{ail}	l_{flap}	r_{flap}	rud	$elev$	THS
X	–	–	–	–	–	–	$\cdots$	–	–	–	–	–	–	–
Y	–	–	-0.595	-0.203	–	0.760	$\cdots$	0.007	-0.007	–	–	-0.374	–	–
Z	–	-6.895	–	–	-21.956	–	$\cdots$	-0.233	-0.233	-1.164	-1.164	–	-0.493	2.019
L	–	–	-0.118	-0.602	–	0.103	$\cdots$	0.108	-0.108	–	–	-0.048	–	–
M	–	-5.207	–	–	-66.790	–	$\cdots$	0.188	0.188	0.938	0.938	–	-2.689	-7.504
N	–	–	0.335	0.068	–	-0.447	$\cdots$	$9.4e^{-4}$	$-9.4e^{-4}$	–	–	0.237	–	–

Table 5: An example of stability matrix produced for the A320.

Looking at table 6 it is possible to notice that the THS derivative as opposite compared with the elevator one. This is due to an error, linked with the convention adopted, made during the definition of the A320 THS derivative. This error was left in the A320 legacy model to keep the consistence for the results previously obtained. It was also kept in A320 new model in order to avoid its influence in the evaluation of the changes. The same way of thinking was kept in the E190AR model, where the THS derivatives values were copied from the A320 because of two reasons:

- To avoid that an assumption based on few data available (see paragraph 3.1.2), could compromise the result obtained.
- The procedure presented on paragraph 3.1.2 was developed after the whole model was tuned for the E190AR with the A320 THS derivatives, and the tests were produced. Changing the derivatives would require to retune the model and to run again all the tests.

	–	α	β	p	q	r	$\cdots$	l_{ail}	r_{ail}	l_{flap}	r_{flap}	rud	$elev$	THS
X	–	–	–	–	–	–	$\cdots$	–	–	–	–	–	–	–
Y	–	–	0.068	0.399	–	0.277	$\cdots$	0.884	0.884	–	–	0.221	–	–
Z	–	0.121	–	–	0.110	–	$\cdots$	0.188	0.188	0.188	0.188	–	0.240	0
L	–	–	0.351	0.091	–	0.078	$\cdots$	0.274	0.274	–	–	0.387	–	–
M	–	0.046	–	–	0.077	–	$\cdots$	0.510	0.510	0.510	0.510	–	0.133	0
N	–	–	0.291	0.086	–	0.351	$\cdots$	1.152	1.152	–	–	0.256	–	–

Table 6: Absolute relative deviation between the E190AR and the A320 flight derivatives.

Looking at the other derivatives it is possible to observe two recurring schemes:

- The control surfaces, save the elevator, are more effective in the Airbus than in the Embraer.
- The E190AR is less stable than the A320 in the lateral-directional plane, causing some tests to fail. Because of this some derivatives were changed. Hereafter the changes made:
  - $C_{L\beta}$ multiplied by 0.5.
  - $C_{N\beta}$ multiplied by 1.5.
  - C_{Nr} multiplied by 1.5.

As already anticipated it is not possible without real flight data to tell if these values are representative for the aircraft under development. However, in the next chapter it will be shown how these values affect the aircraft behaviour during the simulation. In particular, in paragraph 5.2 the effects of these changes in a specific test involving both the lateral and the longitudinal plane, will be presented.

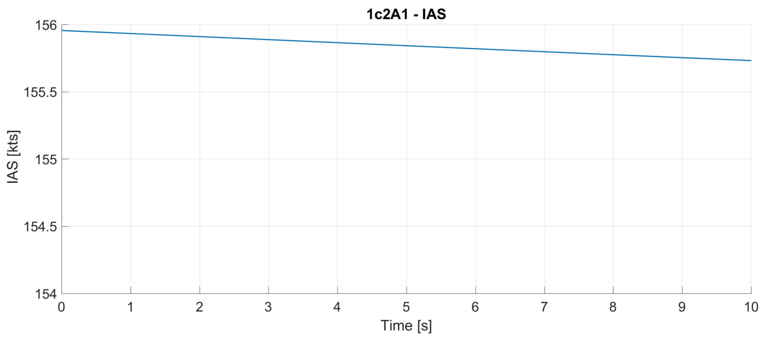
5 Tests results

Hereafter, the results of some tests performed will be presented. All the tests except for the landing one are based on the specifications given by the CS-FSTD(A). A list of all the tests required for an FNPT level II simulator can be found at Appendix B
In order to underline the changes made to the simulator, the chapter will be organized in the following way: First a small set of test for the A320 model with the new engine and landing gear system will be compared to the same test for the old A320 model, in such a way only the differences caused by the systems introduction will be highlighted. After, the tests results for the Embraer E190AR will be compared against the tests result from the A320 with the new system integration, this is done to appreciate only the differences produced by the new flight derivatives and different gains, rather than the changes caused by the introduction of new systems, behaviour already shown in the comparison between the A320 models.

5.1 Comparison between A320 old model and A320 new model

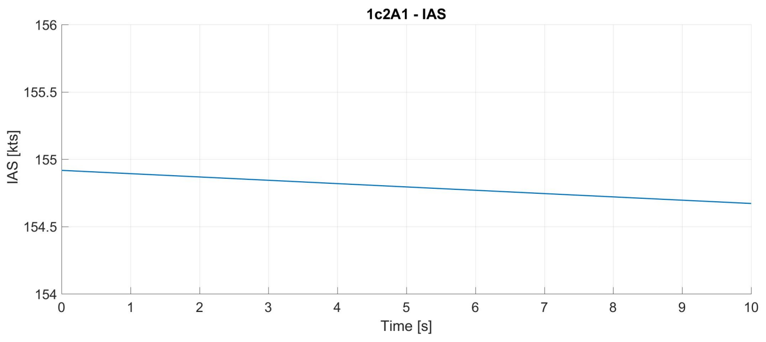
5.1.1 Stabilized Climb One Engine Inoperative (OEI)

During this test the aircraft must be able to perform a climb starting from an altitude of 1000 ft with only the right engine working. This test was chosen among the others because it shows the effects of two changes made to the simulator model: the new MathPilot trim end condition and the engine system.



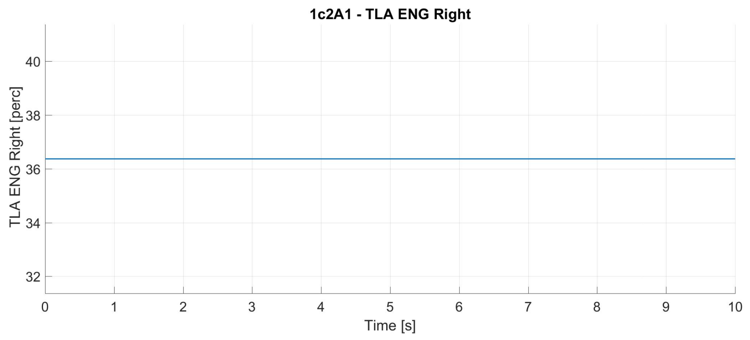
(a) A320.

Taking a look at Figure 36 it is possible to see that the the IAS for the A320 (Figure 36a) starts from a different value from the IAS for the A320_new (Figure 36b). This behaviour is due to the different end condition used by the tests, indeed for both test the trim was performed by the Math Pilot, but for the fist one the ending condition was only based on time, while for the second the trim stopped when the error fell under a certain threshold. On these graphs there is another change in behaviour which is barely visible here, but it will more clear presenting other tests; the dissimilarity is that the speed decrease more in the A320_new model than in the old A320 model. In this test because the altitude and speed variations are small, their effects on the thrust generation is marginal. Another behaviour strictly connected with the engine system which is emphasised in this test is the difference between the maximum thrust available at given conditions, and the sea level maximum thrust. Indeed, as shown in Figure 37, to produce the same thrust the old model needs only a fraction of the thrust available (Figure 37a). Quite the opposite happens for the new model where to trim the aircraft in desired conditions is necessary to provide all the thrust available (Figure 37b).

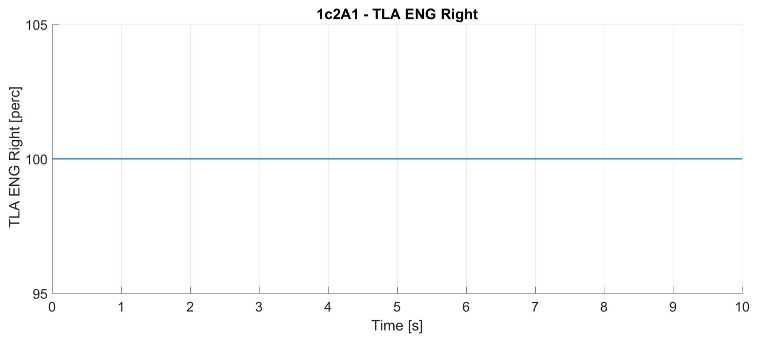


(b) A320_new.

Figure 36: Indicated Air Speed.



(a) A320.

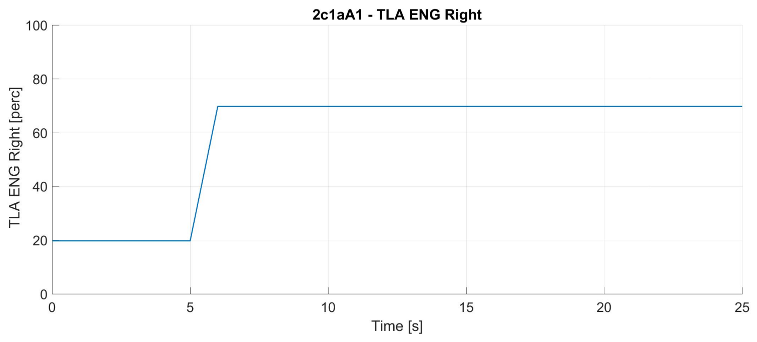


(b) A320_new.

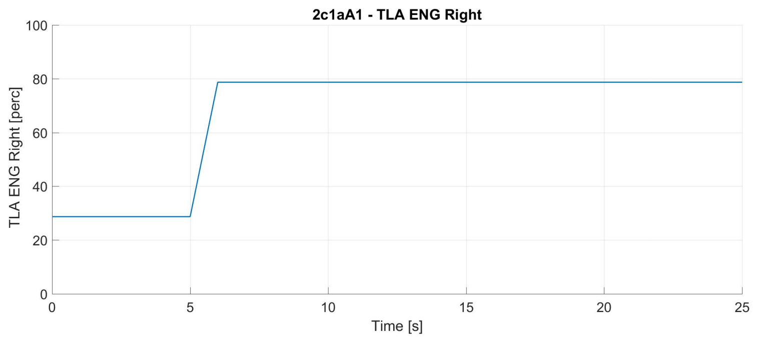
Figure 37: Throttle Level Percentage Right Engine.

5.1.2 Power Change Dynamics, Normal Law

This test consists in a throttle ramp performed during a a symmetrical flight, it was chosen because it highlights how the new engine model affects the whole aircraft behaviour. Let's start by looking at the throttle lever command history shown in Figure 38 where the graphs for the right engine are plotted (nothing would change if the graphs for the left one were plotted, since this manoeuvre is perfectly symmetrical). It is possible to notice how the starting throttle lever is different (because of the discrepancy between maximum sea level thrust, and maximum available thrust, already discussed in the previous test), but the absolute increase is the same for the two aircraft. Looking how the engine reacts to this command in Figure 39, as predictable, it



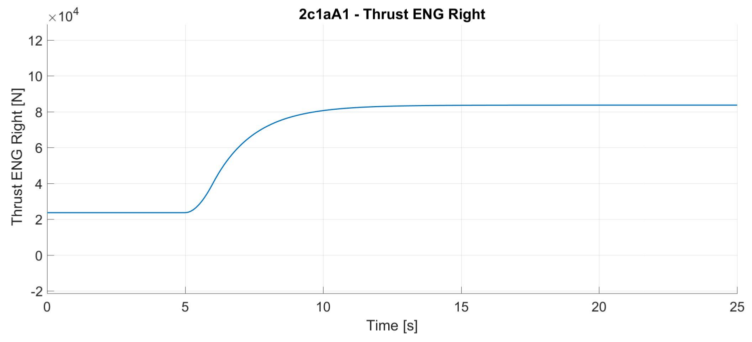
(a) A320.



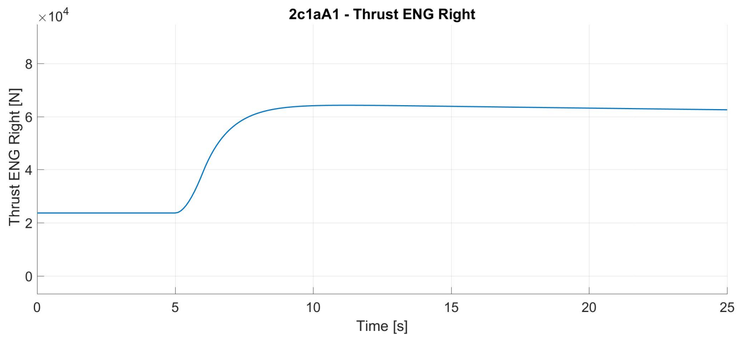
(b) A320_new.

Figure 38: Throttle Level Percentage Right Engine.

is possible to distinguish a first order behaviour for both the models, but there are also some differences: the old models reach an higher thrust value than the new one, and while the old model after a transition time settles on a fixed value, the new model continues to vary its output because of the variation in terms of altitude and speed.



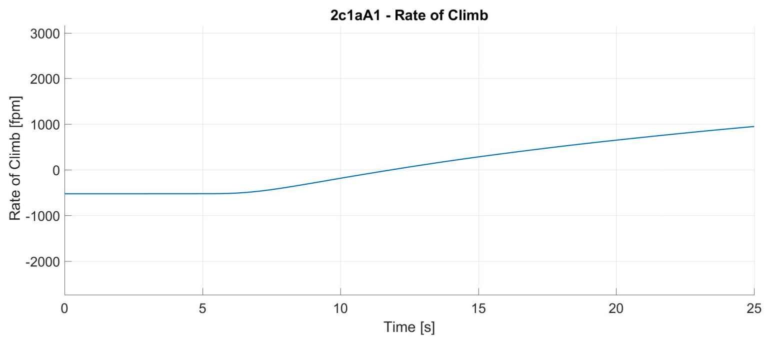
(a) A320.



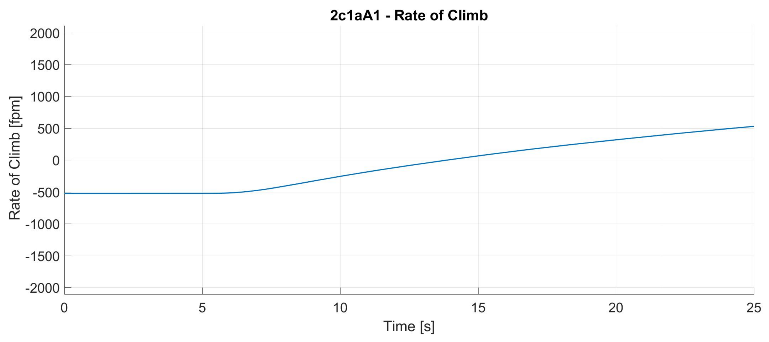
(b) A320_new.

Figure 39: Thrust Right Engine.

Of course the different thrust causes the aircraft to respond in different ways to the same throttle level variation, let's take as an example the Rate of Climb represented in Figure 40. Because in the old model the engine produce an higher thrust, the aircraft reaches an higher speed which causes the the aerodynamics actions to be more vigorous. All this translates in more lift which explains why the Rate of Climb is different between the two models.



(a) A320.



(b) A320_new.

Figure 40: Throttle Level Percentage Right Engine.

5.1.3 Phugoid Dynamics, Direct Law

The last manoeuvre to compare the "old" A320 model to the new one is the phugoid test. This test of course is performed with the flight control in *direct* mode, otherwise the control system would artificially damp the aircraft oscillations. To start the test a ramp command as the one in Figure 41 is given to the elevator, this will activate the phugoid dynamics whit its characteristic phase shift of approximately 90° between IAS and Pitch. Anyway this is not the behaviour which is interesting in this test, what is important to notice is the correlation between the Altitude (Figure 42) and the Thrust (Figure 43) (here applies the consideration to the right engine made in the previous test). Opposite to the old engine model thrust which stays still independently from the the altitude (consequently the density), the new model oscillates in phase with the altitude variation. It is curious to notice how these oscillation dampen slightly the phugoid oscillation, improving the flight characteristics of the aircraft.

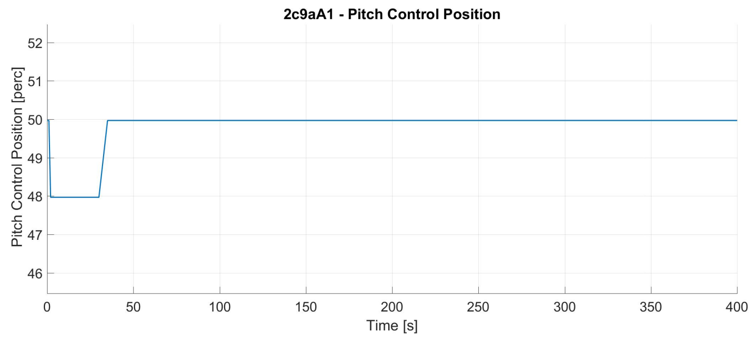
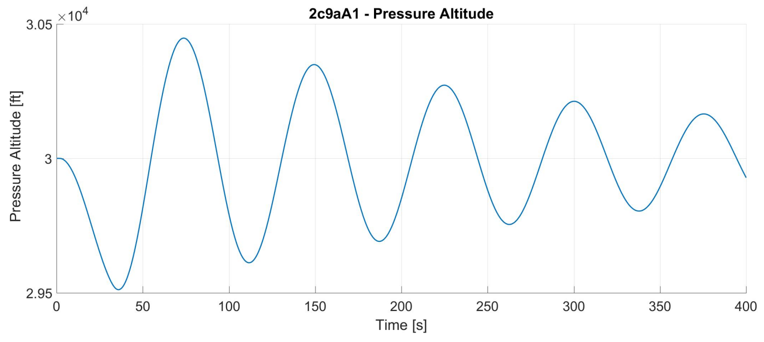
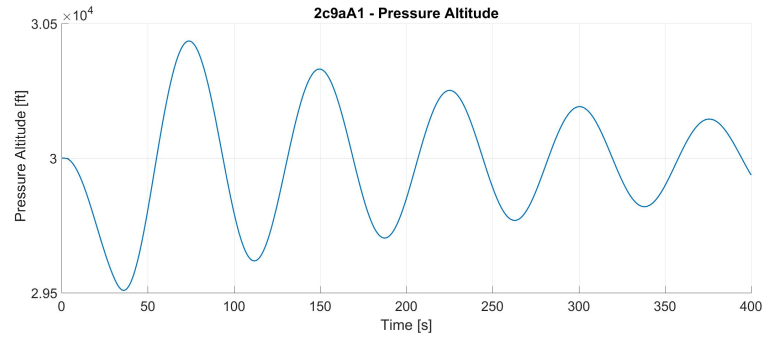


Figure 41: Pitch Command.

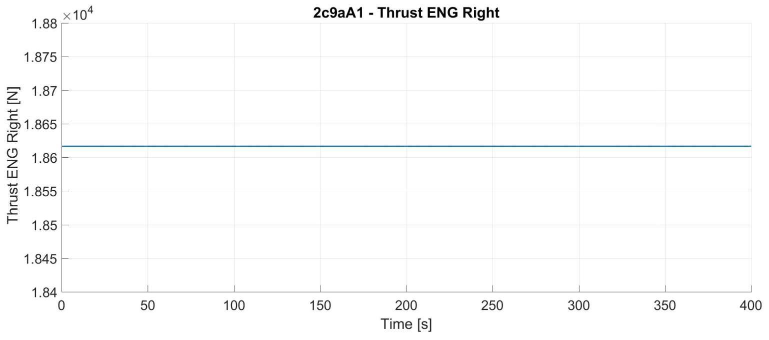


(a) A320.

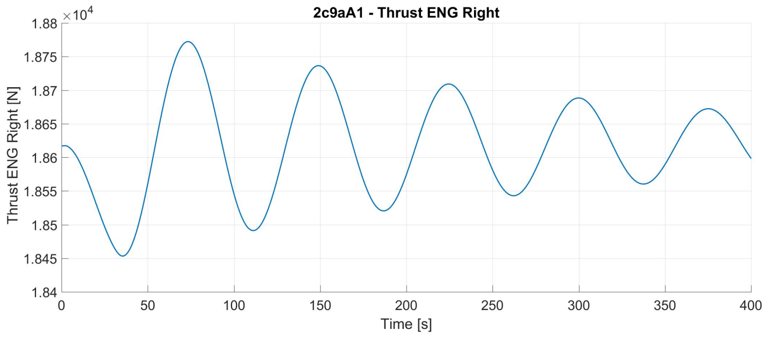


(b) A320_new.

Figure 42: Altitude.



(a) A320.



(b) A320_new.

Figure 43: Thrust Right Engine.

5.1.4 Landing and Take-Off

Differently from the others, this test was created without respecting any regulation, with the only intent to produce a proof of concept to demonstrate the landing gear system effectiveness. The test is entirely performed using the direct law. Indeed, once the aircraft is trimmed the throttle lever is the only command input to change, with all the surfaces staying identical to the trimmed condition.

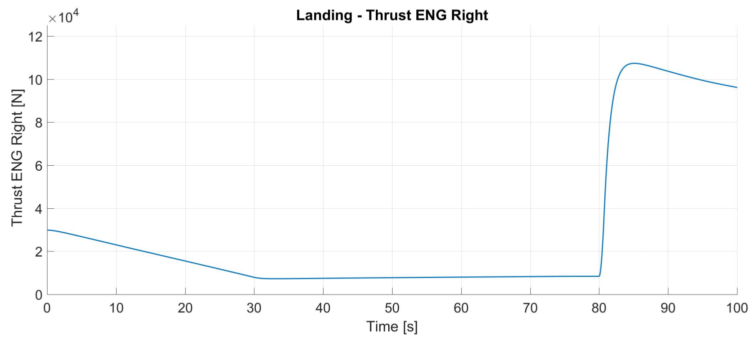


Figure 44: Right Engine Thrust.

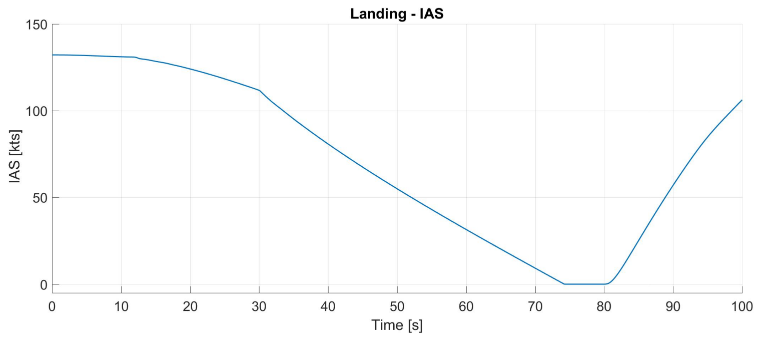


Figure 45: Indicated Air Speed.

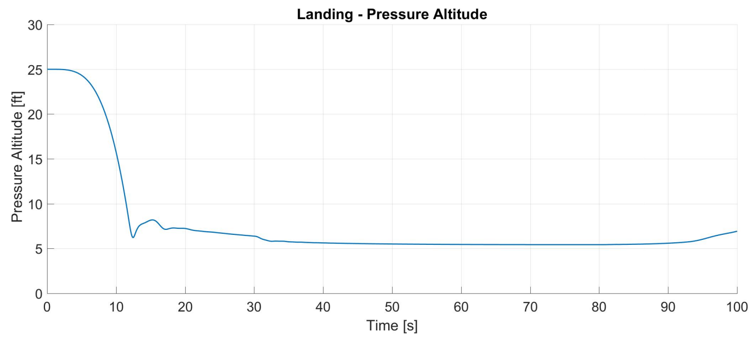


Figure 46: Pressure Altitude.

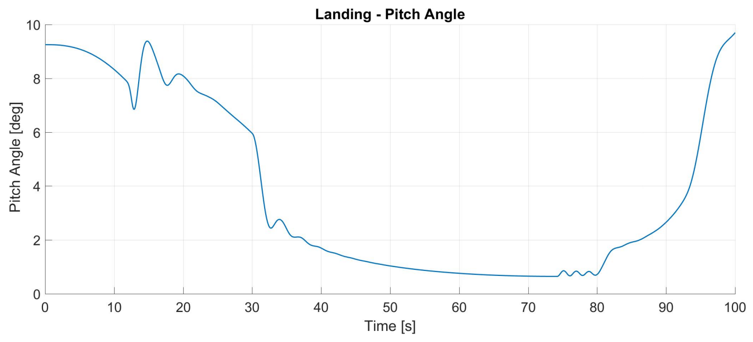


Figure 47: Pitch Angle.

Once the test is started the thrust is gradually reduced (first 30 s of Figure 44). The thrust reduction causes the aircraft to loose speed (Figure 45) and consequently to decrease the altitude (Figure 46). The aircraft falls until the main landing gear touches the ground (this is the first to touch the ground because, as shown in Figure 47, the aircraft is pitched up). The contact between the main landing gear and the landing strip have two effects: it causes the aircraft to bounce and to pitch down. This rotation causes the auxiliary landing gear to impact the ground, this justify the oscillations between 10 s and 25 s visible in Figure 47.

At 30 seconds the thrust has reached a minimum and the aircraft starts to brake, this action causes the aircraft to loose speed (Figure 45) and to pitch down, until it reaches the equilibrium, with the aircraft pitched less than 1°, so almost aligned with the ground. At around 75 s the aircraft stops, looking at Figure 47 it is possible to notice that the aircraft starts to oscillate around the y-body axis. This is not representative of a real behaviour but, as discussed at the end of paragraph 3.2.2, they are only numerical oscillations induced by how the angle is computed.

At 80 s the throttle is maxed, the aircraft starts to increase its speed and the aerodynamic surfaces become effective again, causing the aircraft to pitch up, and starting to take-off. Of course a real landing take less time (more or less half of the time), but it needs to control more commands requiring a significant pilot interaction. As previously said this test is meant only to demonstrate the landing system capability, without any claim of fidelity to a real landing procedure.

5.2 Comparison between E190AR default and tuned derivatives

In this section will be presented a test to show how the changes made to the derivatives values were able to stabilize the aircraft, which otherwise, would not have been able to perform all the tests producing significant values.

5.2.1 Rudder response, Direct Law, Approach

This test is meant to analyse the aircraft response to a rudder trapezoidal input, as show in Figure 48, Specifically, it shows how well the aircraft damps the oscillations which occurs in the lateral directional plane, highlighting the influence of the changes to the derivatives values.

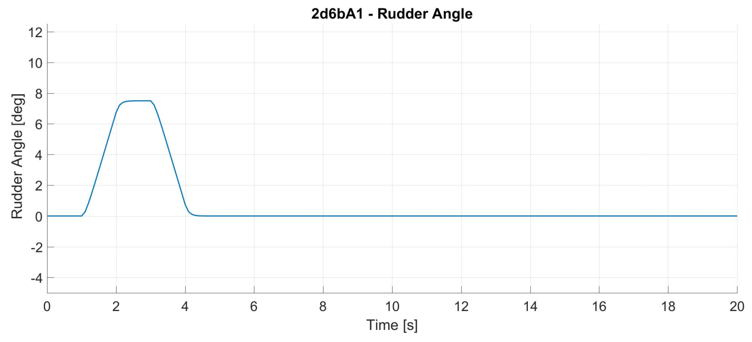
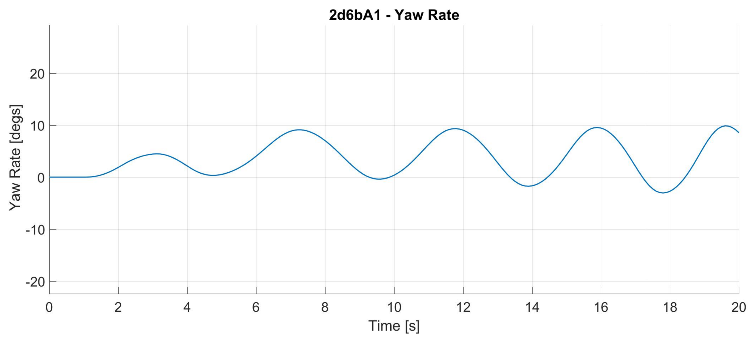
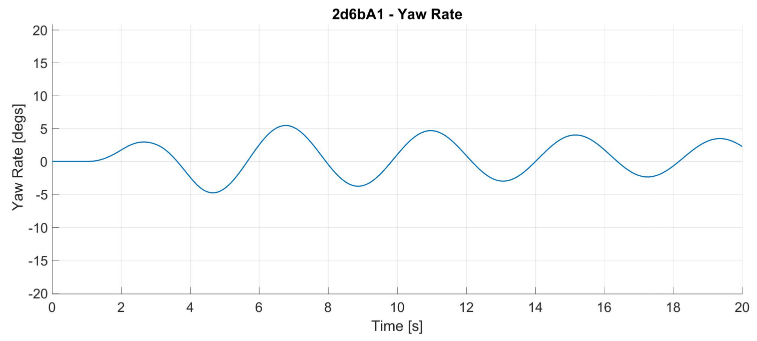


Figure 48: Rudder Angle.

Looking at the graphs for both the yaw rate and the roll angle is visible how the changes made to the flight derivatives were able to improve the aircraft stability passing from the unstable condition, with amplified oscillations of Figure 49a and Figure 50a to a stable condition visible in Figure 49b and Figure 50b where the oscillations decrease their magnitude every cycle. Other than that it is possible to notice how the aircraft response is to the perturbation is more limited. Indeed, the first peak is always greater for the model with default derivatives than for the model with the tuned ones. The derivatives values, presented in paragraph 4.1, are a trade-off between the unstable behaviour of the default model and an ideally damped condition which never oscillate, but is not representative of the reference aircraft.

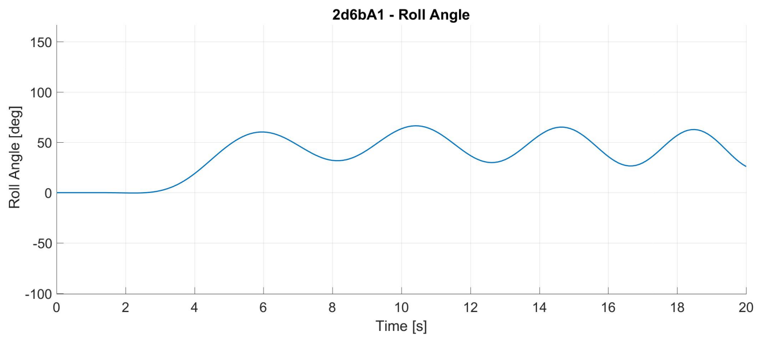


(a) E190AR stock.

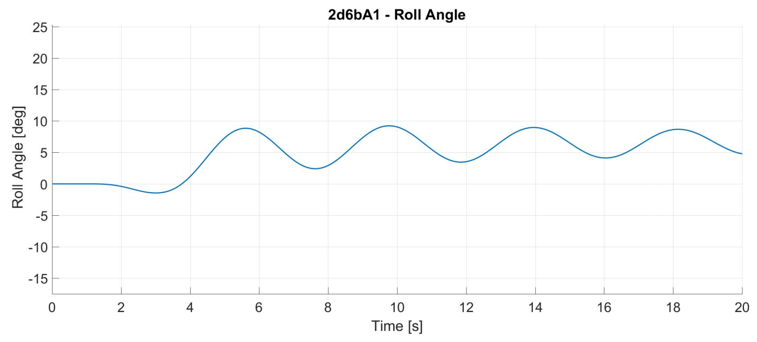


(b) E190AR stabilized.

Figure 49: Yaw Rate.



(a) E190AR stock.



(b) E190AR stabilized.

Figure 50: Roll Angle.

5.3 Comparison between E190AR and A320 new model

The criterion adopted for the test selection that will be presented in this paragraph, is to show the test where the E190AR model diverge the most from the A320, trying to understand and to explain those differences. Indeed, even if the large part of the test performed produced a result which is highly comparable between the two aircraft models, only the first test presented belongs to this category, all the others shows the "anomalies" produced, which are more significant to understand the simulator features and flaws.

5.3.1 Spoiler Change Dynamics, Direct Law, Extension

As the paragraph name suggests, and how it is possible to see in Figure 51, this test consists in a speed brakes extension (to its maximum) during a symmetrical flight. Notice that the speed brakes derivatives are not present in Table 4, neither in Table 5, they are computed inside the Simulink model, this means that they have the same values for both aircraft, so they are not related to any difference in behaviour between the two planes. Anyway, for the purpose of this work, this test is not meaningful because it provides information about the aircraft response to this specific surface actuation, rather than showing the typical correlation existing between these two aircraft models. Indeed, looking at Figure 52 and Figure 53, it is clear that the two aircraft respond in an analogous manner, what changes is the reaction magnitude which is usually greater for the Embraer than the Airbus, behaviour which can be linked to the smaller dimensions of the E190AR compared to the A320.

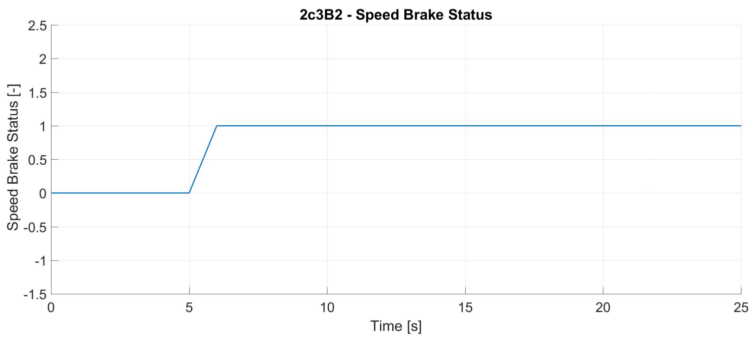
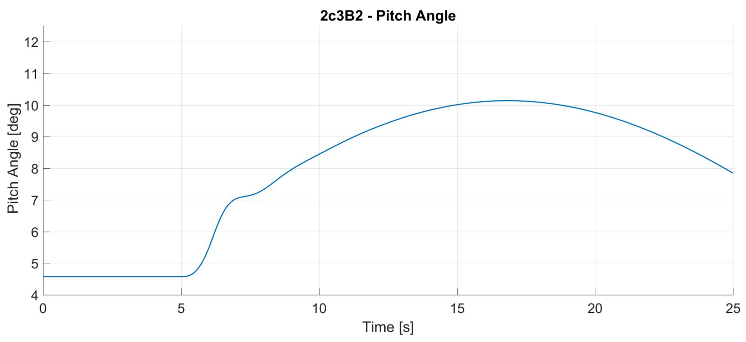
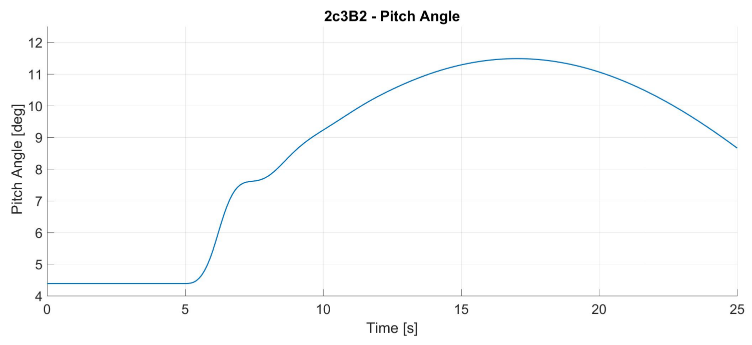


Figure 51: Speed Brake Status.

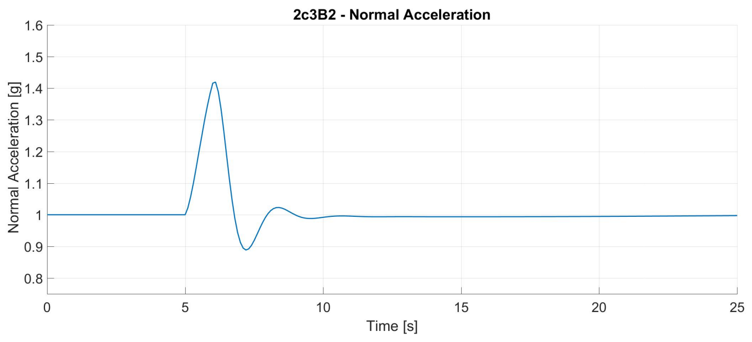


(a) A320_new.

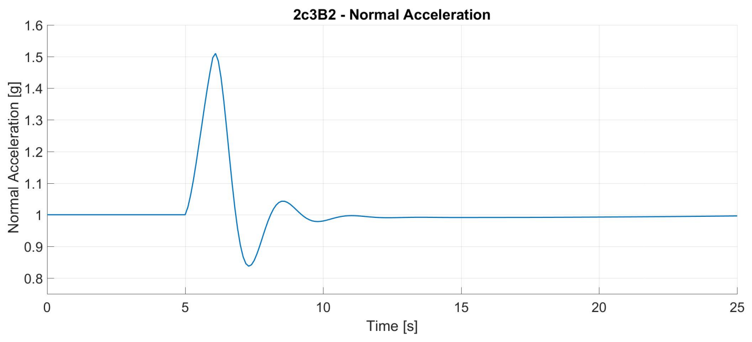


(b) E190AR.

Figure 52: Pitch Angle.



(a) A320_new.



(b) E190AR.

Figure 53: Z body acceleration.

5.3.2 Flap Change Dynamics, Normal Law, Retraction

This test shows the aircraft reaction to the flaps and slats retraction. Modern civil aircraft use Fowler flaps which can increase the lift without affecting too much the drag, by increasing the surface area. However this kind of flaps are very hard to model requiring complex and accurate aerodynamics computations. For this model which finds its strength in simplicity, they were modelled as plain flaps, adopting appropriate empirical coefficients to emulate the different kinds installed on the aircraft. As shown in Figure 54 the flaps are actuated with a linear law which takes 2 seconds to deflect from 15° to 10°, while in Figure 55 it is possible to notice how the slats are deflected with the same law applied for the same amount of time, but with a different excursion: from 22° to 18°. Despite both aircraft are subjected to the same deflection their reaction is significantly different. Looking at Figure 56a is evident how the first phases of the excursion does not impact in a significant manner the A320, which is able to damp the the rotation arround y body axis. It is only when the surfaces deflection stops that the aircraft starts visibly to pitch (Figure 57a). The E190AR instead, is not able to damp the effects of the surfaces deflection, the actuation start is already capable to induce a significant pitch rate (Figure 56b), and the end adds its effects to the previous perturbation causing the Embraer to pitch significantly more than the Airbus (Figure 57b). The different pitch angle causes a difference in lift production which in its turn is responsible for the opposite Rate of Climb shown in Figure 58

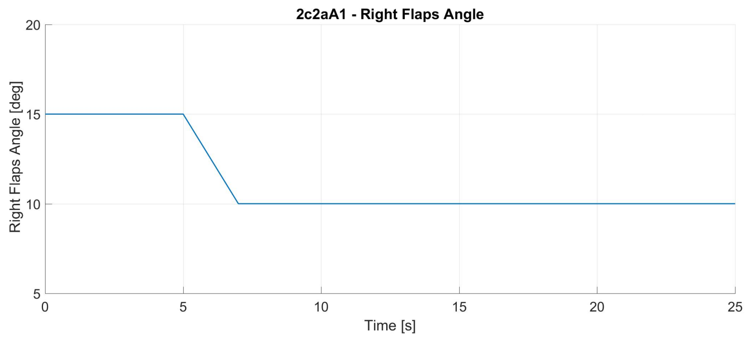


Figure 54: Flaps Deflection Angle.

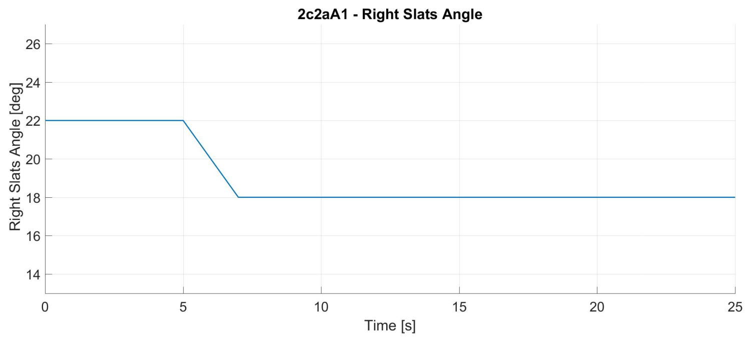
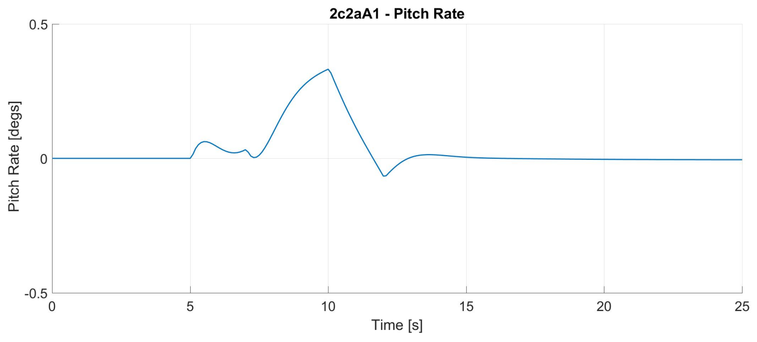
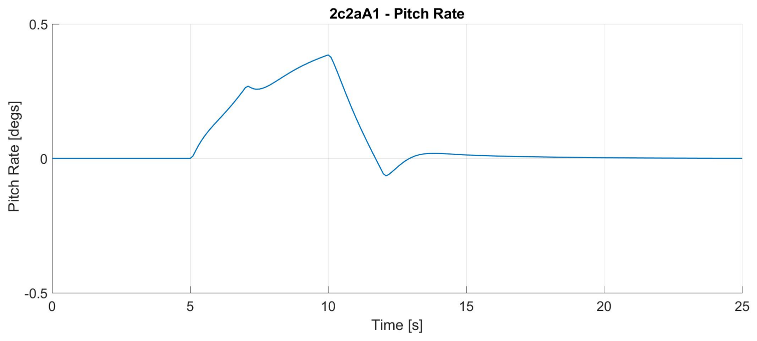


Figure 55: Slats Deflection Angle.

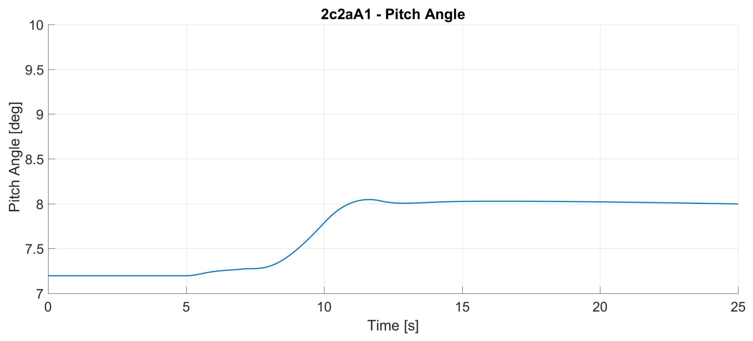


(a) A320_new.

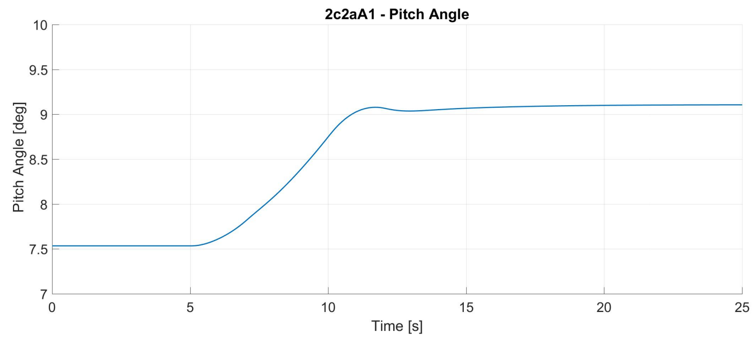


(b) E190AR.

Figure 56: Pitch Rate.

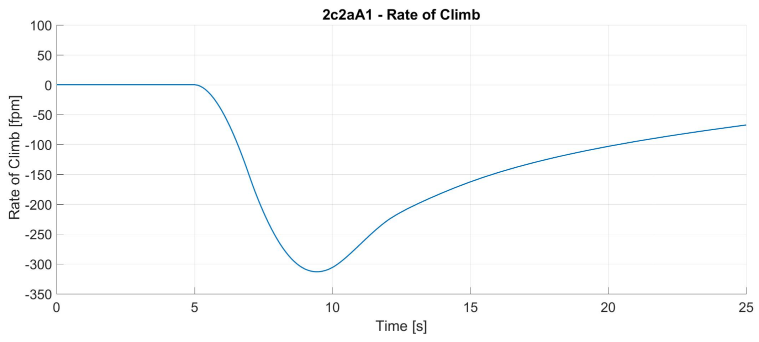


(a) A320_new.

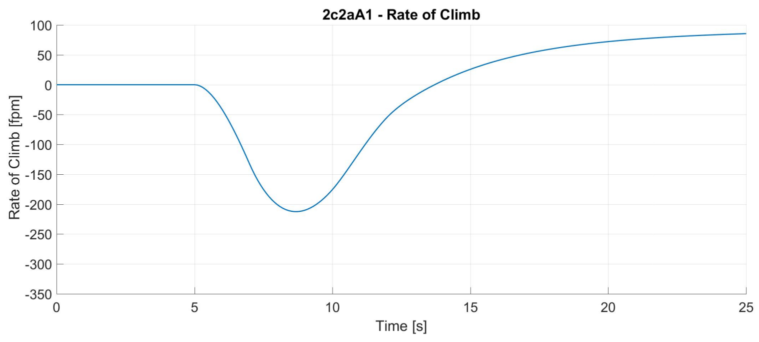


(b) E190AR.

Figure 57: Pitch Angle.



(a) A320_new.

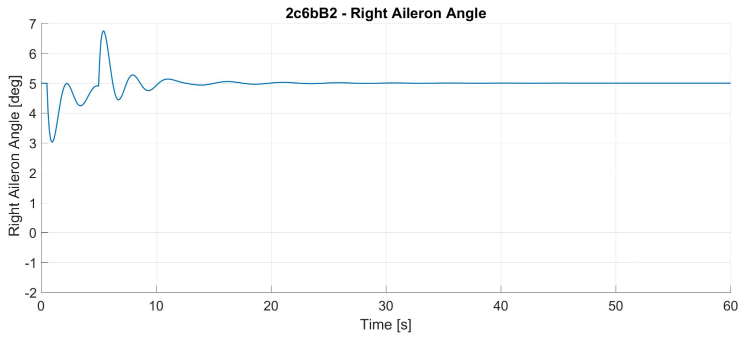


(b) E190AR.

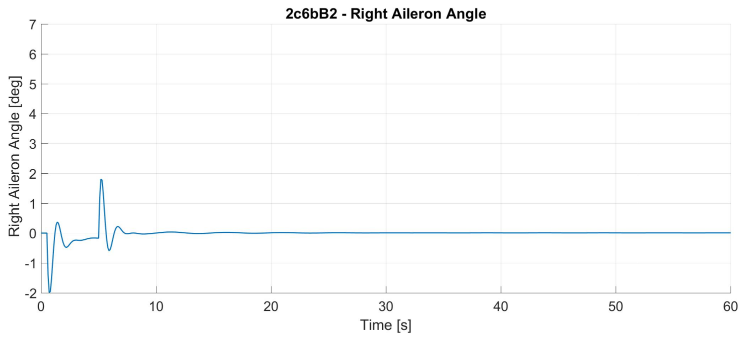
Figure 58: Rate of Climb.

5.3.3 Longitudinal Manoeuvring Stability, Direct Law

This test is meant to check the amount of elevator displacement to maintain the initial speed after a perturbation occurred. To perturb the equilibrium without using the pitch command, which measure is the scope of the test, a bank angle of 20° is commanded to the aircraft, the control system deflects the ailerons to achieve the desired angle (Figure 59). The ailerons deflection, as shown in table 4 and 5, gives a contribution to the pitching moment that the pilot has to compensate. The test is performed with the high lift surfaces extracted, it is possible to understand this detail by looking at Figure 59. As already explained in paragraph 3.1.4, there's an Airbus control law which deflects of 5° down both ailerons when the flaps and slats are extracted. The same law was not implemented in the Embraer model, that is why the two aircraft start from a different angle value, despite they are both initially flying symmetrically.



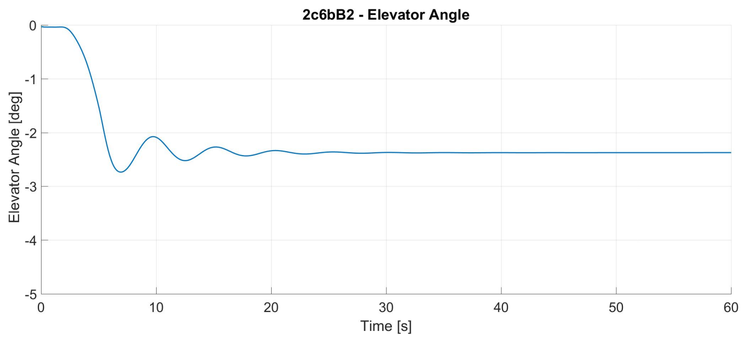
(a) A320_new.



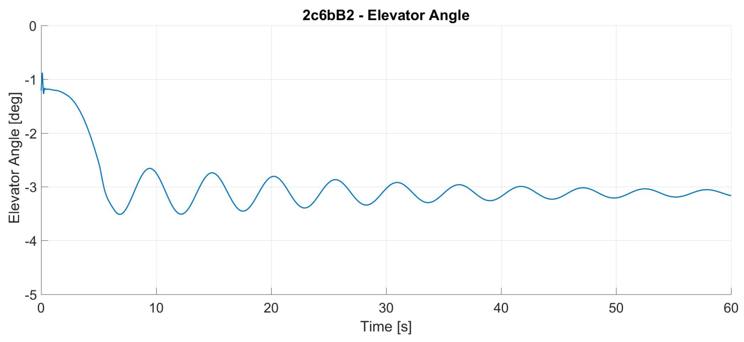
(b) E190AR.

Figure 59: Ailerons deflection angle.

As the title suggest, this test is performed using the direct command law, which means that all the pilot inputs to the pitching command are directly executed by the elevator (of course there is the actuator dynamics in between). The pilot was implemented via the Math Pilot by means of a serie of two PID controller with almost the same gains values: the gains were slightly tuned to guarantee the compatibility with the aircraft characteristics avoiding to run into limit cycles or instability caused by the command. The command output in terms of elevator deflection is visible in Figure 60, where is possible to appreciate how the Embraer due to its minor stability, requires more participation from the pilot in order to be trimmed. Finally in Figure 61, the pitch angle is shown, even if the two aircraft start from almost identical conditions, the Airbus requires less than 30 seconds to be trimmed, while the Embraer can not reach a steady condition, even doubling the time.

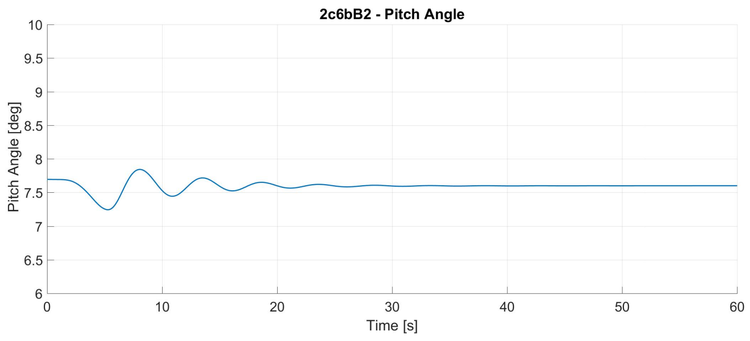


(a) A320_new.

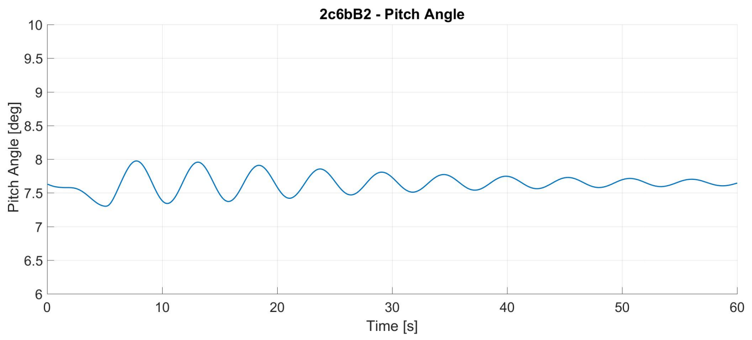


(b) E190AR.

Figure 60: Elevator deflection angle.



(a) A320_new.



(b) E190AR.

Figure 61: Pitch angle.

5.3.4 Roll Response, Left Displacement, Cruise

This test is meant to evaluate how the aircraft reacts to a roll manoeuvre. The simulation is performed with all the three control systems (roll, yaw and pitch) in ”normal” mode. The Math Pilot is set to control the pitch only, so that the lateral evolution is disconnected from what happens in the longitudinal plane. In Figure 62 is shown the manoeuvre commanded to obtain a constant roll rate for a period of time which is about 4 seconds.

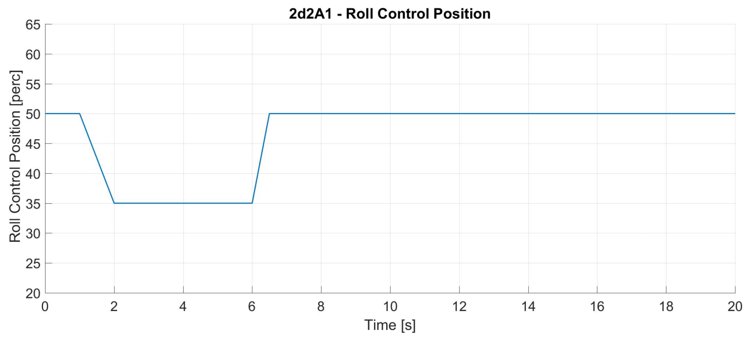
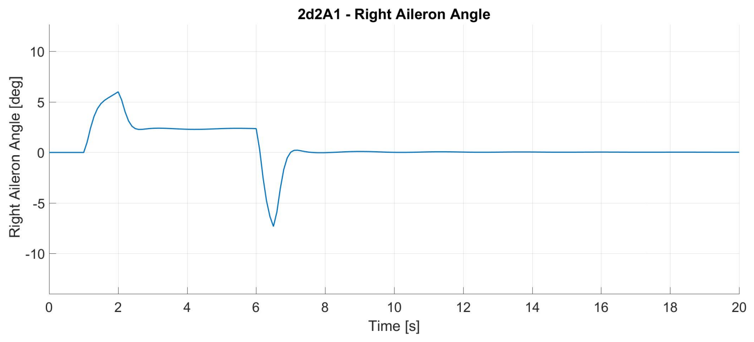


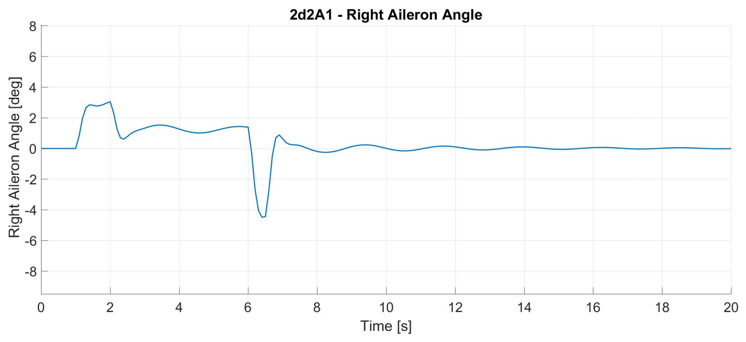
Figure 62: Roll command.

The control systems deflects the ailerons as in Figure 63 to follow the manoeuvre commanded. Comparing Figure 63a to Figure 63b it is evident that the former is significantly more stable than the latter, indeed while the Airbus control system deflects the ailerons only to start and stop the roll, the Embraer control system has to strongly intervene to damp the oscillations which rises after the ailerons perturb the starting equilibrium condition.

The same behaviour happens when the manoeuvre is finished, the Airbus stops to deflect the ailerons while the Embraer continues its activity to damp the oscillations. In order to keep the side-slip angle equal to zero, the yaw control is activated. Looking at Figure 64, the behaviour previously seen with the ailerons appears again but this time is even more amplified. Indeed while the Airbus A320 has to deflect the rudder only during the ailerons deflection, the Embraer control system has to intervene throughout all the simulation, with deflections which are significantly wider than the ones accomplished by the A320 rudder. All that means the Embraer is less stable than the Airbus about both the x and the z axes.

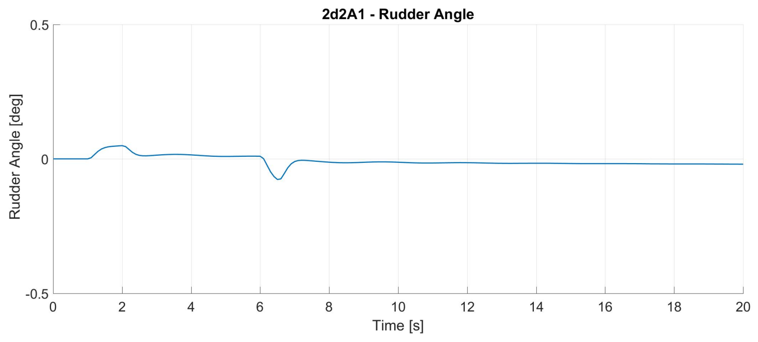


(a) A320_new.



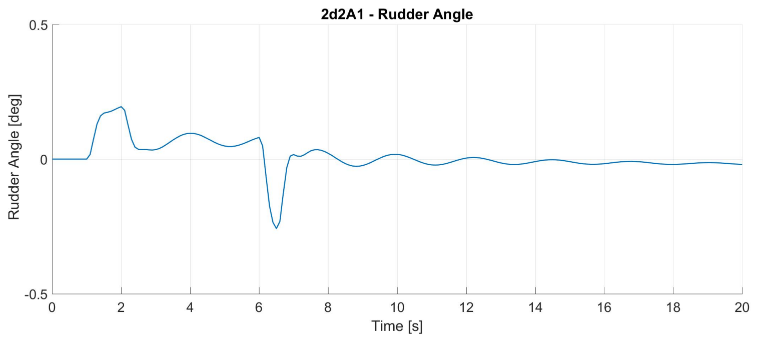
(b) E190AR.

Figure 63: Aileron deflection angle.



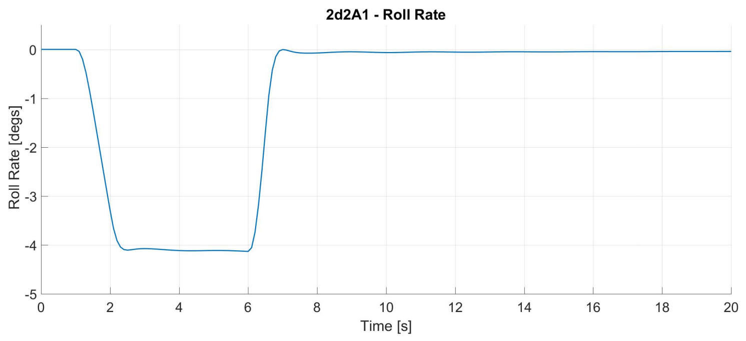
(a) A320_new.

Looking at Figure 66 it is possible to see that, despite the most changes to the flight derivatives were performed to stabilize the Embraer around the z axis, the aircraft is still not very stable. While looking at the roll rate in Figure 65, the change apported to the $C_{L\beta}$ has improved the stability of the aircraft , bringing the Embraer closer to a desirable behaviour.

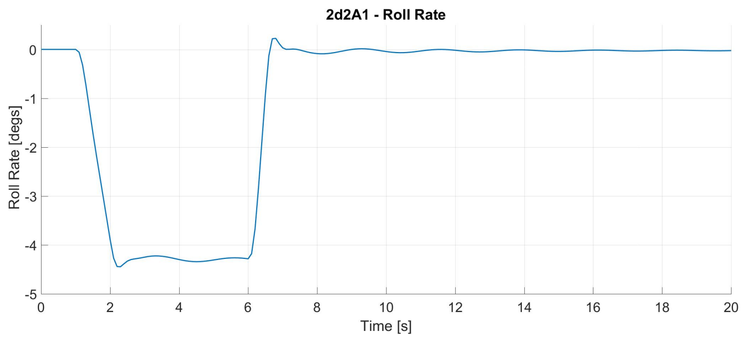


(b) E190AR.

Figure 64: Rudder deflection angle.

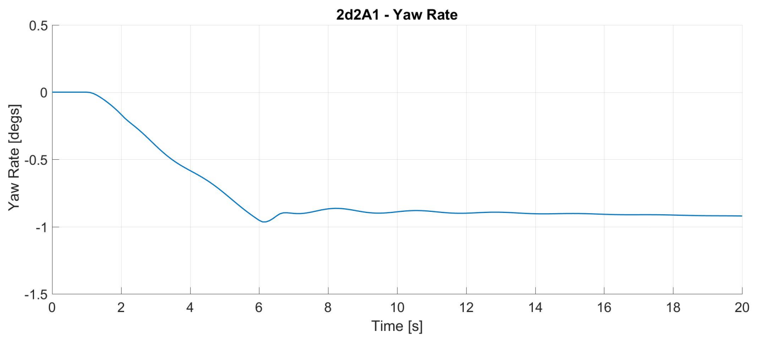


(a) A320_new.

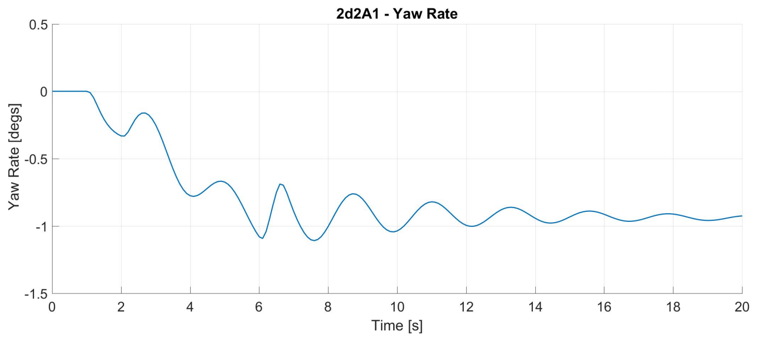


(b) E190AR.

Figure 65: Roll rate.



(a) A320_new.



(b) E190AR.

Figure 66: Yaw rate.

6 Conclusions & Future Developments

The work done during this thesis had two different purposes: the first one was to make the simulator customizable with a series of different aircraft belonging to the same category; the second one was to improve the model in order to perform the first step toward the FNPT level II certification.

Starting from the latter, the tests previously reported showed how the changes made improved the simulation fidelity. Indeed, the new engine model allowed a more accurate reproduction of the thrust trend, also providing more realistic values in term of throttle lever angle and aircraft capabilities. The landing system addition allowed to have a first basic interaction between the aircraft and the ground, providing a way to perform a simple landing and take-off, thus expanding significantly the model capabilities. Anyway, lot of landing gear elements and capabilities, such as a tyre model or a steering system, are still missing, preventing the aircraft to provide the experience of a complete flight as expected by the regulation for a certified training device. The major model flaw is the lack of a comprehensive aerodynamic formulation which, if inserted, can improve in a significant manner the simulation fidelity: first of all the model needs a stall prediction system which is not only taking in account of the flight speed but a series of other elements such as the angle of attack, the surfaces deflection, the atmosphere conditions. Secondary, the auxiliary surfaces effects on the aircraft could be implemented with a higher level of fidelity, taking into account how all their deflection change the airfoil polar. Then, other minor elements which affects only some manoeuvres could be implemented, the ground effect in the lift production is an example. Of course, all these additional elements could be implemented after performing a comparison between the simulation results and real aircraft data, aimed to check if the variance falls inside the ranges expected by the regulation, or the error is large enough to justify the time required and the complexity that these solutions bring with them.

The need of real data is the element which links the simulation model fidelity to its versatility to provide the same structure which can be customized for different aircraft (belonging to the same class). Whit this intent the model of an A320 was taken, and all the specific parameters and behaviours were brought out in various configuration file. Doing so, it was possible to define a development process able to adapt the current model to any single aisle civil aircraft. To demonstrate the feasibility of this operation, the model was customized to reproduce an Embraer E190AR, for which a series of test were performed. Referring to these tests, despite there still are some rough edges, the simulation model is able to produce results which are coherent with the data inserted. As far as a simulator can be a faithful reproduction of the real aircraft behaviour it does not reproduce all the physics which is behind the flight, but it uses a significant amount of workarounds such as gains, tables and simplifications, which does not have a counterpart in the reality, in order to fit the data produced by the simulation to the real ones. Because of this, in order to prove that the simulator can handle another aircraft, actual data related to that aircraft should be available. The greater is the data number, the greater will be the simulation fidelity. In the path pursued to obtain the flight derivatives there was uncertainty, both in the data provided to the computer tools, and in the procedure that these tools followed. This leads to a significant amount of uncertainty even before to start the simulation. All what said is to underline how the results produced, however favourable to the simulation model generalization, have no claim of fidelity to the real aircraft but only to the data supplied in input.

A data fidelity verification to the new aircraft will be the work for a further development. Another work which will be required for an effective certification is the integration of the software model in a physical environment, where the pilot can effectively interact with the simulation. This aspect was not touched at all during this thesis work, but it is vital, especially considering the emphasis that the regulation puts in the integration between the software and the hardware.

A Code

A.1 Stall Tables

```
function [ Flaps_range , Weight_range , V2_matrix ] = Stall_table_reader(
    selectedAircraft ,matrix_selector )

    temp = xlsread(['Stall_Matrix_' selectedAircraft '.xlsx']);

    matrix_selector = matrix_selector +1;

    % 1 = number; 0 = Nan
    previous_cell = 0;

    % Initialize Matrix column
    matrix_column = [0 0];

    % Initialize Matrix column
    matrix_row = [0 0];

    % Contuer to store matrix position
    a = 1;

    % Check long the row to find start and end of each matrix
    for i = 1 :  size(temp,2)
        if previous_cell == 0 && ~isnan(temp(2,i))
            matrix_column(a,1) = i;
        elseif previous_cell == 1 && isnan(temp(2,i))
            matrix_column(a,2) = i-1;
            a = a + 1;
        end
        if  isnan(temp(2,i))
            previous_cell = 0;
        else
            previous_cell = 1;
        end
    end

    % Check if a matrix ends in a 0 position and change it with the end of
    % temp
    for j = 1 : size(matrix_column, 1)
        if matrix_column(j,2) == 0
            matrix_column(j,2) = size(temp,2);
        end
    end


    % Check long the column to find the end of each matrix
    for j = 1 : size(matrix_column, 1)

        while ~isnan(temp(i,matrix_column(j,1)+1))  && i < size(temp,1)
            i = i + 1;
        end
        if i == size(temp,1)
            matrix_row(j) = i;
        else
            matrix_row(j) = i - 1;
        end
        i = 1;
    end

    if size(matrix_column, 1) == 1
        matrix_selector = 1;
    end


    Flaps_range = zeros(1,  matrix_column(matrix_selector,2) - matrix_column(
        matrix_selector,1));
    Weight_range = zeros(1, matrix_row(matrix_selector) - 1);
    V2_matrix = zeros(matrix_row(matrix_selector) - 1,matrix_column(matrix_selector,2)
        - matrix_column(matrix_selector,1));

    for j = matrix_column(matrix_selector,1) : matrix_column(matrix_selector,2)
        for i = 1 : matrix_row(matrix_selector)
            if i == 1 && j ~= matrix_column(matrix_selector,1)
                Flaps_range(1,j - matrix_column(matrix_selector,1)) = temp(i,j);
            elseif i ~= 1 && j == matrix_column(matrix_selector,1)
                Weight_range(1,i - 1) = temp(i,j);
            elseif i ~= 1 && j ~= matrix_column(matrix_selector,1)
                V2_matrix(i-1, j - matrix_column(matrix_selector,1)) = temp(i,j);
            end
        end
    end
end
```

A.2 Landing Gear Suspension

```
% Main Landing Gear Dx
h_mlg_dx = h_cg - l_x_mlg * sin(theta) - l_y_mlg * sin(phi) - length_mlg * cos(theta) *
    cos(phi);
if h_mlg_dx <= 0
        % Evaluate landing gear compression
        delta_mlg_dx = - h_mlg_dx / (cos(theta) * cos(phi));
        if delta_mlg_dx > 1.0 % V0_ml/A_ml
                fprintf('Main Right compression limit, !!!CRASHED!!!\n');
                lg_crash_flag = 1;
                delta_mlg_dx = 1.0; % V0_ml/A_ml - 0.01;
        end
        % Evaluate landing gear compression speed
        delta_dot_mlg_dx = (delta_mlg_dx - delta_mlg_dx_old) / 0.01;
        % Evaluate landing gear reaction force
        if delta_dot_mlg_dx < 0
                A0_mlg_dx = A0_ml/5;
        else
                A0_mlg_dx = A0_ml;
        end
        R_M_poly_mlg_dx = p0_ml*((V0_ml/(V0_ml-A_ml*delta_mlg_dx)-1)^gamma)*A_ml;
        R_M_visc_mlg_dx = 0.5*k*rho*A_ml^3*delta_dot_mlg_dx*abs(delta_dot_mlg_dx)/
            A0_mlg_dx^2;
        R_M_mlg_dx = R_M_poly_mlg_dx + R_M_visc_mlg_dx;
        if R_M_mlg_dx < 0
                R_M_mlg_dx = 0;
end
else
        delta_mlg_dx = 0;
        delta_dot_mlg_dx = 0;
        R_M_mlg_dx = 0;
end
% Check if landing gear touched
if delta_mlg_dx_old == 0 && delta_mlg_dx ~= 0
        fprintf('Right Main Landing Gear on ground \n');
elseif delta_mlg_dx_old ~= 0 && delta_mlg_dx == 0
        fprintf('Right Main Landing Gear Lift-off \n');
end
```

A.3 Landing Gear Brake

```
% Brake
if brake_status > 0 && V_CAS > 0
        X_brake = brake_status * 75000;
        M_brake = X_brake * (h_cg - 0.5);
else
        X_brake = 0;
        M_brake = 0;
        brake_status = 0;
end
if brake_status_old == 0 && brake_status ~= 0
        fprintf('BRAKING \n');
        elseif brake_status_old ~= 0 && brake_status == 0
        fprintf('STOP BRAKING \n');
end
```

B Tests

Code	Test	MathPilot Trim
1c1A1	Stabilized Climb, AEO	No
1c2A1	Stabilized Climb, OEI	Yes
2c1aA1	Power change dynamics, Normal Law	No
2c1aB1	Power change dynamics, Direct Law	No
2c2aA1	Flap change dynamics, Normal Law, Retraction	No
2c2aA2	Flap change dynamics, Normal Law, Extension	No
2c2aB1	Flap change dynamics, Direct Law, Retraction	No
2c2aB2	Flap change dynamics, Direct Law, Extension	No
2c3A1	Spoiler change dynamics, Normal Law, Retraction	No
2c3A2	Spoiler change dynamics, Normal Law, Extension	No
2c3B1	Spoiler change dynamics, Direct Law, Retraction	No
2c3B2	Spoiler change dynamics, Direct Law, Extension	No
2c4aA1	Gear change dynamics, Normal Law, Retraction	No
2c4aA2	Gear change dynamics, Normal Law, Extension	No
2c4aB1	Gear change dynamics, Direct Law, Retraction	No
2c4aB2	Gear change dynamics, Direct Law, Extension	No
2c5bA1	Longitudinal Trim, Cruise	No
2c5bA2	Longitudinal Trim, Approach	No
2c6bA1	Longitudinal manoeuvring stability, Normal Law, Trim	No
2c6bA2	Longitudinal manoeuvring stability, Normal Law, 20deg Left	No
2c6bA3	Longitudinal manoeuvring stability, Normal Law, 30deg Left	No
2c6bB1	Longitudinal manoeuvring stability, Direct Law, Trim	No
2c6bB2	Longitudinal manoeuvring stability, Direct Law, 20deg Right	No
2c6bB3	Longitudinal manoeuvring stability, Direct Law, 30deg Right	No
2c7A1	Longitudinal Static Stability, Acc., Trim	No
2c7A2	Longitudinal Static Stability, Acc. Speed 1	No
2c7A3	Longitudinal Static Stability, Acc. Speed 2	No
2c7A4	Longitudinal Static Stability, Dec., Trim	No
2c7A5	Longitudinal Static Stability, Dec. Speed 1	No
2c7A6	Longitudinal Static Stability, Dec. Speed 2	No
2c8bA1	Approach to Stall Characteristics, Normal Law, 2nd seg. Climb	No
2c8bA2	Approach to Stall Characteristics, Normal Law, Cruise	No
2c8bA3	Approach to Stall Characteristics, Normal Law, Approach	No
2c8bB1	Approach to Stall Characteristics, Direct Law, 2nd seg. Climb	No
2c8bB2	Approach to Stall Characteristics, Direct Law, Cruise	No
2c8bB3	Approach to Stall Characteristics, Direct Law, Approach	No
2c9aA1	Phugoid dynamics, Acc., Direct Law	No
2c10A1	Shortperiod dynamics, Normal Law	No
2c10B1	Shortperiod dynamics, Direct Law	No
2d1A1	Minimum Control Speed, air, Direct Law	Yes
2d1B1	Minimum Control Speed, air, Normal Law	Yes
2d2A1	Roll response (rate), Left Displacement, Cruise	No
2d2A2	Roll response (rate), Left Displacement, Approach	No
2d3A1	Step input of cockpit roll controller, Approach, Normal Law	No
2d3B1	Step input of cockpit roll controller, Approach, Direct Law	No
2d4A1	Spiral stability, Direct Law, Right Direction	No
2d4A2	Spiral stability, Direct Law, Left Direction	No
2d5A1	Engine inoperative trim, 2nd seg. Climb	Yes
2d5B1	Engine inoperative trim, Approach	Yes
2d6bA1	Rudder response, Direct Law, Approach	No
2d6bB1	Rudder response, Alternate Law, Approach	No
2d7A1	Dutch roll, Direct Law, Cruise	No
2d7A2	Dutch roll, Direct Law, Approach	No
2d8A1	Steady state sideslip, Trim	No
2d8A2	Steady state sideslip, Sideslip 1	No
2d8A3	Steady state sideslip, Sideslip 2	No

C Simulator Output

Parameter	Unit of Measure
X acceleration	m/s^2
Y acceleration	m/s^2
Z acceleration	m/s^2
Angle of Attack α	deg
Sideslip Angle β	deg
Calibrated Air Speed	m/s
Mach	–
Outside Air Temperature	K
Pressure Altitude	ft
Pitch Angle	deg
Roll Angle	deg
Yaw Angle	deg
Pitch Command (normalized)	–
Yaw Command (normalized)	–
Roll Command (normalized)	–
Trottle Lever Angle (normalized)	–
Gross Weight	kg
Left Engine Thrust	N
Right Engine Thrust	N
Rotation arround X-Axis p	deg/s
Rotation arround Y-axis q	deg/s
Rotation arround Z-axis r	deg/s
Vertical speed	ft/s
Landing Gear Lever Status	–
Landing Gear Status	–
Speed-Brakes Status	–
Stall Warning Status	–
Left Aileron Deflection	deg
Right Aileron Deflection	deg
Elevator Deflection	deg
Left Flaps Deflection	deg
Right Flaps Deflection	deg
Rudder Deflection	deg
Left Slats Deflection	deg
Right Slats Deflection	deg
Tail Horizontal Stabilizer THS Deflection	deg
Turn Coordinator Status	–
Yaw Damper Status	–

References

[1] W. H. Dunn, C. Eldert, and P. V. Levonian, "A digital computer for use in an operational flight trainer," *IRE Transactions on Electronic Computers*, vol. EC-4, no. 2, pp. 55–63, 1955.

[2] Federal Aviation Administration, *AC 120-40 Airplane Simulator Qualification*, 1983.

[3] European Aviation Safety Agency, *Certification Specifications for Aeroplane Flight Simulation Training Devices*, issue 2 ed., 2018.

[4] "Matlab." `https://en.wikipedia.org/wiki/MATLAB`, December 2020. Accessed on 2021-01-07.

[5] TXTGroup, "Pacelab apd." `https://pace.txtgroup.com/products/preliminary-design/pacelab-apd/`, December 2020. Accessed on 2021-01-21.

[6] R. Fiore, M. Macarana, and A. Matrone, "Modeling and simulation of aircraft landing gear," December 2018.